CNML Developer Guide

Release 7.10.2

Apr 16, 2021

Cambricon 寒武纪

Table of Contents

1 Copyright

DISCLAIMER

CAMBRICON MAKES NO REPRESENTATION, WARRANTY (EXPRESS, IMPLIED, OR STATUTORY) OR GUARANTEE REGARDING THE INFORMATION CONTAINED HEREIN, AND EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, TITLE, NONINFRINGEMENT OF INTELLECTUAL PROPERTY OR FITNESS FOR A PARTICULAR PURPOSE, AND CAMBRICON DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR SERVICES. CAMBRICON SHALL HAVE NO LIABILITY RELATED TO ANY DEFAULTS, DAMAGES, COSTS OR PROBLEMS WHICH MAY BE BASED ON OR ATTRIBUTABLE TO: (I) THE USE OF THE CAMBRICON PRODUCT IN ANY MANNER THAT IS CONTRARY TO THIS GUIDE, OR (II) CUSTOMER PRODUCT DESIGNS.

LIMITATION OF LIABILITY

In no event shall Cambricon be liable for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption and loss of information) arising out of the use of or inability to use this guide, even if Cambricon has been advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever, Cambricon’s aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the Cambricon terms and conditions of sale for the product.

ACCURACY OF INFORMATION

Information provided in this document is proprietary to Cambricon, and Cambricon reserves the right to make any changes to the information in this document or to any products and services at any time without notice. The information contained in this guide and all other information contained in Cambricon documentation referenced in this guide is provided “AS IS.” Cambricon does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within this guide. Cambricon may make changes to this guide, or to the products described therein, at any time without notice, but makes no commitment to update this guide.

Performance tests and ratings set forth in this guide are measured using specific chips or computer systems or components. The results shown in this guide reflect approximate performance of Cambricon products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. As set forth above, Cambricon makes no representation, warranty or guarantee that the product described in this guide will be suitable for any specified use. Cambricon does not represent or warrant that it tests all parameters of each product. It is customer’s sole responsibility to ensure that the product is suitable and fit for the application planned by the customer and to do the necessary testing for the application in order to avoid a default of the application or the product.

Weaknesses in customer’s product designs may affect the quality and reliability of Cambricon product and may result in additional or different conditions and/or requirements beyond those contained in this guide.

IP NOTICES

Cambricon and the Cambricon logo are trademarks and/or registered trademarks of Cambricon Corporation in China and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

This guide is copyrighted and is protected by worldwide copyright laws and treaty provisions. This guide may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without Cambricon’s prior written permission. Other than the right for customer to use the information in this guide with the product, no other right or license, either express or implied, is granted by Cambricon under this guide. For the avoidance of doubt, Cambricon does not grant any right or license (express or implied) to customer under any patents, copyrights, trademarks, trade secret or any other intellectual property or proprietary rights of Cambricon.

- Copyright
- © 2021 Cambricon Corporation. All rights reserved.

2 Preface

2.1 Version

Table 2.1: Release Version

File Name	CNML Developer Guide
Version Number	V7.10.2
Aurthor	Cambricon
Revision Date	April 16, 2021

2.2 Documentation Update History

This section lists content changes on documentation that were made for each product release.

- V7.10.2
  Date: April 16, 2021
  Changes:
  - Updated description of cnmlSetOperationName to remove the illegal characters checking.
  - Updated description of cnmlCreateDeformConvOpParam.
- V7.10.1
  Date: April 2, 2021
  Changes:
  - Updated Deconv, ConvFirst relevant API description, fixed some mistakes.
  - Updated description of cnmlSetOperationName and cnmlSetTensorName, added illegal characters checking.
  - Updated description of cnmlCreateNdConcatOp.
  - Updated description of cnmlCreateDeconvOpParam_V4, cnmlCreateDeconvGroupOp, cnmlCreateConvFirstOp, cnmlSetTensorName.
- V7.10.0
  Date: February 25, 2021
  Changes:
  - Updated ConvDepthwise relevant API. Added quantization for filter tensor and input tensor, and channel quantization for filter tensor.
  - Updated BatchNorm, NdBatchNorm, Scale, NdScale, FrozenBatchNorm relevant API. Added constant tensors decription.
  - Supported new values in cnmlCoreVersion_t.
  - Added the following new APIs:
    * cnmlWhereOpParam
    * cnmlWhereOpParam_t
    * cnmlSetOperationName
    * cnmlGetOperationNameLength
    * cnmlGetOperationName
    * cnmlConfigAutotune
    * cnmlCreateWhereOpParam
    * cnmlDestroyWhereOpParam
    * cnmlCreateWhereOp_V2
    * cnmlCopyFusionOp
  - Revised supporting tensor data type of Conv、ConvFirst、ConvGroup、ConvDepthwise.
  - Revised special symbols in formula.
  - Corrected the formula for Tril operation.
  - Updated the supported data types for Max, Min, UnarySelect, and Where APIs.
  - Added limitation for MatrixMult op and Batchdot op.
- V7.9.4
  Date: January 04, 2021
  Changes:
  - When the cast_type of Cast operation is set to CAST_FLOAT32_TO_FLOAT16, CAST_INT16_TO_FLOAT16, or CAST_INT32_TO_FLOAT32, the Round Mode is changed to roundTiesToAway.
- V7.9.3
  Date: December 10, 2020
  Changes:
  - Added the following new APIs:
    * cnmlCreateNdGatherV2Op

    * cnmlComputeNdGatherV2OpForward
    * cnmlCreateDeconvDepthwiseOpParam_V2
    * cnmlCreateDeconvOpParam_V4
- V7.8.0
  Date: August 31, 2020
  Changes:
  – Added the cnmlGetLibVersion new API.
- V7.6.0
  Date: July 10, 2020
  Changes:
  – Added the new Deform Conv operation.
  – Added the cnmlCreateConvDepthwiseOpParam_V3 new API.
- V7.4.0
  Date: May 28, 2020
  Changes:
  – Added the following new APIs and datatype:
    * cnmlCreateInterpOpParam_V2
    * cnmlCreateQuantizedParamByChannel
    * cnmlCastType_t
  – Parameter changes on cnmlCreateQuantizedParamByChannel.
- V7.3.0
  Date: April 16, 2020
  Changes:
  – Added the new cnmlDyadicType_t data type.
  – Added the following new operations:
    * Nddyadic
    * NdTile
  – Updated the platform changes for Lstm Pro opreation and other operations.
  – Added a limitation that the value of the `kmode` parameter in cnmlCreateTopkOp_V2 and cnmlCreateNdTopkOp APIs only supports CNML_TOPK_OP_MODE_MAX.
- V7.2.1
  Date: February 24, 2020
  Changes:
  – Added the following new APIs:
    * cnmlSetQuantizedAlphaByChannel
    * cnmlCreateConvFirstOpParam_V2
  – Added the following new operations:
    * NdCrop
    * Decov Depthwise
  – Added the description about fomula, scale limitations, and so on on the Deconv, Crop, Power, Sqrt, and Boardcast Mult operations.
- V7.2.0
  Date: December 31, 2019
  Changes:
  – Added the following new APIs and data types:
    * cnmlDestroySequenceMaskOpParam
    * cnmlGetConstData
    * cnmlAddFusionOpCacheGraph
    * cnmlSetFusionOpCacheMode
    * cnmlReduce_andDim_t
    * cnmlReduce_orDim_t
  – Added the following new operations:
    * broadcast args
    * reduce and
    * reduce or
  – Removed APIs and data types due to the function changes.
- V7.1.1
  Date: December 15, 2019
  Changes:
  – Added the following new APIs:
    * cnmlComputeNdScaleOpForward_V2
    * cnmlGetFusionOpRequiredStackSize
    * cnmlAddFusionOpCacheGraph
    * cnmlSetFusionOpCacheMode
  – Added the following new operations:
    * Control Flow
    * while Loop
    * Cond
  – Changes on the followng APIs:
    * Parameter changes on cnmlComputeShuffleChannelOpForward_V4.
- V7.1.0

Date：November 30, 2019
Changes：
- Added the following new APIs:
  * cnmlCreatePoolOpParam_V3
  * cnmlCreateConvOpParam_V2
  * cnmlComputeSignOpForward_V4
  * cnmlCreateGrayNormalizeForKcfOp
  * cnmlComputeGrayNormalizeForKcfOpForward
- Added the following new operations:
  * Group Norm
  * Basic Div
  * Logloss
  * Control Flow
  * AsStride
- The following APIs and datatypes changed:
  * Softmax operation APIs：The type of the parameters are changed from `SoftmaxDim_t` to `cnmlDimension_t`.
  * Reduce Max operation APIs：The type of the parameters are changed from `ReduceMaxMode_t` to `cnmlDimension_t`.
  * Concat operation APIs：The type of the parameters are changed from `cnmlConcatMode_t` to `cnmlDimension_t`.
  * Split operation APIs：The type of the parameters are changed from `cnmlSplitMode_t` to `cnmlDimension_t`.
  * Argmax operation APIs：The type of the parameters are changed from `cnmlArgmaxAxis_t` to `cnmlDimension_t`.
  * Reverse operation APIs：The type of the parameters are changed from `cnmlReverseAxis_t` to `cnmlDimension_t`.
  * DeconvGroup operation APIs: The type of the parameters are changed from `cnmlDeconvGroupOpParam_t` to `cnmlDeconvOpParam_t`.
  * Removed the `sparse_mode` parameter from the cnmlCreateConvOpParam_V2, cnmlCreateConvOpParam, cnmlCreateConvFirstOpParam APIs.
  * Removed the `sparse_mode` parameter from the MLP operation APIs.
  * Changed APIs whose return datatype is `void` to `cnmlStatus_t`.
  * Added `group` parameter in cnmlCreateDeconvGroupOp API.
  * Added `size` parameter in cnmlGetTensorSize_V2 API.
  * Added `len` parameter in cnmlGetTensorLen API.
  * Added `dtype` parameter in cnmlGetTensorDataType API.
- The following APIs is renamed:
  * cnmlComputeDeConvOpForward_V3 is renamed to cnmlComputeDeconvOpForward_V3
  * cnmlComputeDeConvOpForward_V4 is renamed to cnmlComputeDeconvOpForward_V4
  * cnmlCreateAddPadOp4Param is renamed to cnmlCreateAddPadOpParam_V2
  * cnmlCreateAddPadChannelOp4Param is renamed to cnmlCreateAddPadChannelOpParam_V2
  * cnmlCreateGrepChannelOp2Param is renamed to cnmlCreateGrepChannelOpParam_V2
- The following datatypes are removed:
  * cnmlDataPreprocessStrategy_t
  * cnmlSparseMode_t
  * cnmlSoftmaxDim_t
  * cnmlReduceMaxMode_t
  * cnmlConcatMode_t
  * cnmlDeconvOptLevel_t
  * cnmlSplitMode_t
  * cnmlArgmaxAxis_t
  * cnmlReverseAxis_t
  * cnmlDeconvOptLevel_t
- The following operations are removed:
  * Basic RNN. You can use Basic RNN Pro operation instead.
  * TNC Reshape
  * Trigon
  * L2 Pooling Operation
  * Xw Plus B Operation
  * Coeff Add Operation
  * Mean Var Norm Operation
  * Resize Opreation
  * Scaled Tanh Operation
  * Maxtc Operation
  * Maxtt Operation
  * Mintc Operation
  * Mintt Operation
  * Lstm Operation
  * Svdf Operation
  * Svdfnn Operation
  * Ceil
  * RoundToZero
  * FakeQuant
  * Proposal
  * Roi Pooling
  * Image Detect
  * Ssd Detection Output

    * Ssd Detection
    * Ssd Detection Pose Output
    * Ssd Detection Pose
    * Roialign
    * Ps Roi Pooling
    * Yolo Detect
    * Thrs Relu
    * Elem Power
    * Cycle Bit And
    * Cycle Bit Or
    * Cycle Bit Xor
  - The following APIs are removed:
    * cnmlComputeFusionOpForward
    * cnmlComputeFusionOpForward_V2
    * cnmlCpuComputeScatterRefOpForward
- V7.0.1
  Date：November 15, 2019
  Changes：
  - None.
- V7.0.0
  Date：October 23, 2019
  Changes：
  - Supports new APIs.
- V6.8.0
  Date：September 23, 2019
  Changes：
  - Supports new APIs.
  - Added the missing description of formula and data type for APIs that are already supported.
- V6.7.0
  Date：August 15, 2019
  Changes：The first version of the document.

3 CNML Datatypes Reference

3.1 cnmlActiveFunction_t

typedef enum {

CNML_ACTIVE_NONE = 0,

CNML_ACTIVE_SIGMOID = 1,

CNML_ACTIVE_RELU = 2,

CNML_ACTIVE_TANH = 3,

CNML_ACTIVE_RELU1 = 4,

CNML_ACTIVE_RELU6 = 5,

CNML_ACTIVE_HARD_SIGMOID = 6,

} cnmlActiveFunction_t;

`enum cnmlActiveFunction_t`

An enum.

Which activation function is used by active operator in CNML. Enumeration cnmlActiveFunction_t has 4 enumerated values, indicating different activation functions in active operator. User can pass in it according to their actual needs when they create the active operator.

Values:

`enumerator CNML_ACTIVE_NONE` = 0

Do not use activation function.

`enumerator CNML_ACTIVE_SIGMOID` = 1

Use sigmoid activation function.

`enumerator CNML_ACTIVE_RELU` = 2

Use relu activation function.

`enumerator CNML_ACTIVE_TANH` = 3

Use tanh activation function.

`enumerator CNML_ACTIVE_RELU1` = 4

Use relu1 activation function.

`enumerator CNML_ACTIVE_RELU6` = 5

Use relu6 activation function.

`enumerator CNML_ACTIVE_HARD_SIGMOID` = 6

Use hard sigmoid activation function.

3.2 cnmlAddPadChannelOpParam

`struct cnmlAddPadChannelOpParam`

A struct.

cnmlAddPadChannelOpParam is a structure describing the param parameter of addpad channel operation, used to create addpad channel operation. cnmlCreateAddPadChannelOpParam() and cnmlCreateAddPadChannelOpParam_V2() is used to create an instance of cnmlAddPadChannelOpParam_t cnmlDestroyAddPadChannelOpParam() is used to destroy an instance of cnmlAddPadChannelOpParam_t.

3.3 cnmlAddPadChannelOpParam_t

`typedef struct` cnmlAddPadChannelOpParam `*cnmlAddPadChannelOpParam_t`

`cnmlAddPadChannelOpParam_t` is a pointer to *cnmlAddPadChannelOpParam* which is a structure holding the description of a addpad channel operation param.

3.4 cnmlAddPadOpParam

`struct cnmlAddPadOpParam`

A struct.

cnmlAddPadOpParam is a structure describing the param parameter of addpad operation, used to create addpad operation. cnmlCreateAddPadOpParam() and cnmlCreateAddPadOpParam_V2() is used to create an instance of cnmlAddPadOpParam_t. cnmlDestroyAddPadOpParam() is used to destroy an instance of cnmlAddPadOpParam_t.

3.5 cnmlAddPadOpParam_t

`typedef struct` cnmlAddPadOpParam `*cnmlAddPadOpParam_t`

`cnmlAddPadOpParam_t` is a pointer to *cnmlAddPadOpParam* which is a structure holding the description of a addpad operation param.

3.6 cnmlBaseOp

`struct cnmlBaseOp`

A struct.

The cnmlBaseOp is a structure describing base operation. Each base operation calls its corresponding function to create an instance of cnmlBaseOp_t. For example, add operation calls cnmlCreateAddOp() to create a specific instance of cnmlBaseOp_t. cnmlDestroyBaseOp() is used to destroy an instance of cnmlBaseOp_t.

3.7 cnmlBaseOp_t

`typedef struct` cnmlBaseOp `*cnmlBaseOp_t`

`cnmlBaseOp_t` is a pointer to *cnmlBaseOp* which is a structure holding the description of base operation.

3.8 cnmlCastType_t

typedef enum {

CNML_CAST_FLOAT32_TO_UINT8 = 0,

CNML_CAST_UINT8_TO_FLOAT32 = 1,

CNML_CAST_INT8_TO_FLOAT16 = 3,

CNML_CAST_FIX8_TO_FLOAT16 = 3,

CNML_CAST_FLOAT16_TO_FLOAT32 = 4,

CNML_CAST_FLOAT16_TO_FIX8 = 5,

CNML_CAST_FLOAT16_TO_INT8 = 5,

CNML_CAST_FLOAT32_TO_FLOAT16 = 6,

CNML_CAST_INT16_TO_FLOAT16 = 7,

CNML_CAST_FLOAT16_TO_INT16 = 8,

CNML_CAST_FLOAT16_TO_INT16_ROUND_ZERO = 9,

CNML_CAST_FLOAT16_TO_UINT8 = 10,

CNML_CAST_UINT8_TO_FLOAT16 = 11,

CNML_CAST_INT8_TO_FLOAT32 = 12,

CNML_CAST_FLOAT32_TO_INT8 = 13,

CNML_CAST_INT16_TO_FLOAT32 = 14,

CNML_CAST_FLOAT32_TO_INT16 = 15,

CNML_CAST_FLOAT16_TO_INT16_ROUND_EVEN = 16,

CNML_CAST_FLOAT16_TO_FLOAT16_ROUND_EVEN = 17,

CNML_CAST_FLOAT16_TO_INT32_ROUND_ZERO = 18,

CNML_CAST_FLOAT32_TO_INT32_ROUND_ZERO = 19

} cnmlCastType_t;

`enum cnmlCastType_t`

An enum.

Data conversion of cast operator in CNML.Enumeration cnmlCastType_t has 9 enumerated values, each of which represent a kind of data conversion mode. User can pass in it according to their actual needs when they create the cast operator.

Values:

`enumerator CNML_CAST_FLOAT32_TO_UINT8` = 0

Convert float32 to uint8.

`enumerator CNML_CAST_UINT8_TO_FLOAT32` = 1

Convert uint8 to float32.

`enumerator CNML_CAST_INT8_TO_FLOAT16` = 3

Convert int8 to float16.

`enumerator CNML_CAST_FIX8_TO_FLOAT16` = 3

Convert int8 to float16.

`enumerator CNML_CAST_FLOAT16_TO_FLOAT32` = 4

Convert float16 to float32.

`enumerator CNML_CAST_FLOAT16_TO_FIX8` = 5

Convert float16 to int8.

`enumerator CNML_CAST_FLOAT16_TO_INT8` = 5

Convert float16 to int8.

`enumerator CNML_CAST_FLOAT32_TO_FLOAT16` = 6

Convert float32 to float16.

`enumerator CNML_CAST_INT16_TO_FLOAT16` = 7

Convert int16 to float16.

`enumerator CNML_CAST_FLOAT16_TO_INT16` = 8

Convert float16 to int16.

`enumerator CNML_CAST_FLOAT16_TO_INT16_ROUND_ZERO` = 9

`enumerator CNML_CAST_FLOAT16_TO_UINT8` = 10

Convert float16 to uint8. Convert uint8 to float16.

`enumerator CNML_CAST_UINT8_TO_FLOAT16` = 11

Convert int8 to float32.

`enumerator CNML_CAST_INT8_TO_FLOAT32` = 12

Convert float32 to int8.

`enumerator CNML_CAST_FLOAT32_TO_INT8` = 13

Convert int16 to float32.

`enumerator CNML_CAST_INT16_TO_FLOAT32` = 14

Convert float32 to int16.

`enumerator CNML_CAST_FLOAT32_TO_INT16` = 15

Convert float16/32 to int16/32.

`enumerator CNML_CAST_FLOAT16_TO_INT16_ROUND_EVEN` = 16

`enumerator CNML_CAST_FLOAT16_TO_FLOAT16_ROUND_EVEN` = 17

Convert float16 to int32 in round zero mode.

`enumerator CNML_CAST_FLOAT16_TO_INT32_ROUND_ZERO` = 18

Convert float32 to int32 in round zero mode.

`enumerator CNML_CAST_FLOAT32_TO_INT32_ROUND_ZERO` = 19

3.9 cnmlConcatOpParam

`struct cnmlConcatOpParam`

A struct.

cnmlConcatOpParam is a structure describing the param parameter of concat operation, used to create concat operation. cnmlCreateConcatOpParam() is used to create an instance of cnmlConcatOpParam_t. cnmlDestroyConcatOpParam() is used to destroy an instance of cnmlConcatOpParam_t.

3.10 cnmlConcatOpParam_t

typedef struct cnmlConcatOpParam ***cnmlConcatOpParam_t**

cnmlConcatOpParam_t is a pointer to *cnmlConcatOpParam* which is a structure holding the description of a concat operation param.

3.11 cnmlConvDepthwiseOpParam

struct cnmlConvDepthwiseOpParam

A struct.

cnmlConvDepthwiseOpParam is a structure describing the param parameter of depthwise conv operation, used to create depthwise conv operation. cnmlCreateConvDepthwiseOpParam() and cnmlCreateConvDepthwiseOpParam_V2() is used to create an instance of cnmlConvDepthwiseOpParam_t. cnmlDestroyConvDepthwiseOpParam() is used to destroy an instance of cnmlConvDepthwiseOpParam_t.

3.12 cnmlConvDepthwiseOpParam_t

typedef struct cnmlConvDepthwiseOpParam ***cnmlConvDepthwiseOpParam_t**

cnmlConvDepthwiseOpParam_t is a pointer to *cnmlConvDepthwiseOpParam* which is a structure holding the description of a conv depthwise operation param.

3.13 cnmlConvFirstOpParam

struct cnmlConvFirstOpParam

A struct.

cnmlConvFirstOpParam is a structure describing the param parameter of first conv operation, used to create first conv operation. cnmlCreateConvFirstOpParam() is used to create an instance of cnmlConvFirstOpParam_t. cnmlDestroyConvFirstOpParam() is used to destroy an instance of cnmlConvFirstOpParam_t.

3.14 cnmlConvFirstOpParam_t

typedef struct cnmlConvFirstOpParam ***cnmlConvFirstOpParam_t**

cnmlConvFirstOpParam_t is a pointer to *cnmlConvFirstOpParam* which is a structure holding the description of a conv first operation param.

3.15 cnmlConvMode_t

typedef enum {

CNML_CONV = 0,

CNML_CONV_GROUP = 1,

CNML_CONV_DEPTHWISE = 2,

} cnmlConvMode_t;

enum cnmlConvMode_t

An enum.

In which mode of conv operator is calculated in CNML. Enumeration cnmlConvMode_t has 3 enumerated values, indicating the mode in which Ndconv operator is calculated. User can pass in it according to their actual needs when they create the Ndconv operator.

Values:

enumerator CNML_CONV = 0

The common conv operation.

enumerator CNML_CONV_GROUP = 1

The conv group operation.

enumerator CNML_CONV_DEPTHWISE = 2

The conv depthwise operation.

3.16 cnmlCpuTensor

`struct cnmlCpuTensor`

A struct.

cnmlCpuTensor is a structure describing tensor in CPU. cnmlCreateCpuTensor() is used to create a 4D instance of cnmlCpuTensor_t. cnmlCreateCpuTensor_V2() is used to create a ND instance of cnmlCpuTensor_t. cnmlDestroyCpuTensor() is used to destroy an instance of cnmlCpuTensor_t.

3.17 cnmlCpuTensor_t

`typedef struct` cnmlCpuTensor `*cnmlCpuTensor_t`

`cnmlCpuTensor_t` is a pointer to *`cnmlCpuTensor`* which is a structure holding the description of a tensor in CPU.

3.18 cnmlNdConvParam

`struct cnmlNdConvParam`

A struct.

cnmlNdConvParam is a structure holding the description of a nd-convolution operation param that is used to create a nd-convolution operation. cnmlCreateNdConvParam() is used to create an instance of cnmlNdConvParam_t. cnmlDestroyNdConvParam() is used to destroy an instance of cnmlNdConvParam_t.

3.19 cnmlNdConvParam_t

`typedef struct` cnmlNdConvParam `*cnmlNdConvParam_t`

`cnmlNdConvParam_t` is a pointer to *`cnmlNdConvParam`* which is a structure holding the description of a nd-convolution operation param.

3.20 cnmlConvOpParam

`struct cnmlConvOpParam`

A struct.

cnmlConvOpParam is a structure describing the param parameter of conv operation, used to create conv operation. cnmlCreateConvOpParam() is used to create an instance of cnmlConvOpParam_t. cnmlDestroyConvOpParam() is used to destroy an instance of cnmlConvOpParam_t.

3.21 cnmlConvOpParam_t

`typedef struct` cnmlConvOpParam `*cnmlConvOpParam_t`

`cnmlConvOpParam_t` is a pointer to *`cnmlConvOpParam`* which is a structure holding the description of a Conv operation param.

3.22 cnmlCoreVersion_t

typedef enum {

CNML_1H8 = 1,

CNML_1H16 = 2,

CNML_C10 = 3,

CNML_1H8MINI = 4,

CNML_MLU100 = 3,

CNML_MLU270 = 5,

CNML_MLU220 = 6,

CNML_1M20 = 7,

CNML_1M70 = 8,

} cnmlCoreVersion_t;

`enum cnmlCoreVersion_t`

An enum.

CNML external core version. Enumeration cnmlCoreVersion_t has 8 enumerated values, each of which represent a type of hardware. This enumeration variable is passed in at compile time to determine which version of hardware instructions are generated by compilation.

Values:

`enumerator CNML_1H8` = 1

1H8 hardware.

`enumerator CNML_1H16` = 2
1H16 hardware.

`enumerator CNML_C10` = 3
MLU100 hardware, deprecated, use CNML_MLU100 instead.

`enumerator CNML_1H8MINI` = 4
1H8MINI hardware.

`enumerator CNML_MLU100` = 3
MLU100 hardware, instead name of CNML_C10.

`enumerator CNML_MLU270` = 5
MLU270 hardware.

`enumerator CNML_MLU220` = 6
MLU220 hardware.

`enumerator CNML_1M20` = 7
1M20 hardware.

`enumerator CNML_1M70` = 8
1M70 hardware.

3.23 cnmlCropOpParam

`struct cnmlCropOpParam`
A struct.

cnmlCropOpParam is a structure describing the param parameter of crop operation, used to create crop operation. cnmlCreateCropOpParam() is used to create an instance of cnmlCropOpParam_t. cnmlDestroyCropOpParam() is used to destroy an instance of cnmlCropOpParam_t.

3.24 cnmlCropOpParam_t

`typedef struct` cnmlCropOpParam `*cnmlCropOpParam_t`
`cnmlCropOpParam_t` is a pointer to *cnmlCropOpParam* which is a structure holding the description of a crop operation param.

3.25 cnmlCustomizedActiveOpParam

`struct cnmlCustomizedActiveOpParam`
A struct.

cnmlCustomizedActiveOpParam is a structure describing the param parameter of customize active operation, used to create customize active operation. cnmlCreateCustomizedActiveOpParam() is used to create an instance of cnmlCustomizedActiveOpParam_t. cnmlDestroyCustomizedActiveOpParam() is used to destroy an instance of cnmlCustomizedActiveOpParam_t.

3.26 cnmlCustomizedActiveOpParam_t

`typedef struct` cnmlCustomizedActiveOpParam `*cnmlCustomizedActiveOpParam_t`
`cnmlCustomizedActiveOpParam_t` is a pointer to *cnmlCustomizedActiveOpParam* which is a structure holding the description of a customized active operation param.

3.27 cnmlDataOrder_t

typedef enum {
CNML_NCHW = 0,
CNML_NCWH = 1,
CNML_NHCW = 2,
CNML_NHWC = 3,
CNML_NWHC = 4,
CNML_NWCH = 5,
CNML_CNHW = 6,
CNML_CNWH = 7,
CNML_CHNW = 8,
CNML_CHWN = 9,
CNML_CWHN = 10,

CNML_CWNH = 11,

CNML_HCNW = 12,

CNML_HCWN = 13,

CNML_HNCW = 14,

CNML_HNWC = 15,

CNML_HWNC = 16,

CNML_HWCN = 17,

CNML_WCHN = 18,

CNML_WCNH = 19,

CNML_WHCN = 20,

CNML_WHNC = 21,

CNML_WNHC = 22,

CNML_WNCH = 23,

CNML_TNC = 24,

CNML_ARRAY = 25,

CNML_NTC = 31,

CNML_NC = 32

} cnmlDataOrder_t;

`enum cnmlDataOrder_t`

An enum.

Enumeration Variables of data layout in CNML. Enumeration cnmlDataOrder_t has 25 enumerated values, each of which represent a kind of data layout. There are 24 4-D data layout types and 1 3D data layout type. This enumerated variable can be passed in when Tensor is created.

Values:

`enumerator CNML_NCHW` = 0

Data layout order: NCHW.

`enumerator CNML_NCWH` = 1

Data layout order: NCWH.

`enumerator CNML_NHCW` = 2

Data layout order: NHCW.

`enumerator CNML_NHWC` = 3

Data layout order: NHWC.

`enumerator CNML_NWHC` = 4

Data layout order: NWHC.

`enumerator CNML_NWCH` = 5

Data layout order: NWCH.

`enumerator CNML_CNHW` = 6

Data layout order: CNHW.

`enumerator CNML_CNWH` = 7

Data layout order: CNWH.

`enumerator CNML_CHNW` = 8

Data layout order: CHNW.

`enumerator CNML_CHWN` = 9

Data layout order: CHWN.

`enumerator CNML_CWHN` = 10

Data layout order: CWHN.

`enumerator CNML_CWNH` = 11

Data layout order: CWNH.

`enumerator CNML_HCNW` = 12

Data layout order: HCNW.

`enumerator CNML_HCWN` = 13

Data layout order: HCWN.

`enumerator CNML_HNCW` = 14

Data layout order: HNCW.

`enumerator CNML_HNWC` = 15

Data layout order: HNWC.

`enumerator CNML_HWNC` = 16
Data layout order: HWNC.

`enumerator CNML_HWCN` = 17
Data layout order: HWCN.

`enumerator CNML_WCHN` = 18
Data layout order: WCHN.

`enumerator CNML_WCNH` = 19
Data layout order: WCNH.

`enumerator CNML_WHCN` = 20
Data layout order: WHCN.

`enumerator CNML_WHNC` = 21
Data layout order: WHNC.

`enumerator CNML_WNHC` = 22
Data layout order: WNHC.

`enumerator CNML_WNCH` = 23
Data layout order: WNCH.

`enumerator CNML_TNC` = 24
Data layout order: TNC.

`enumerator CNML_ARRAY` = 25
Data layout order: Array.

`enumerator CNML_NDHWC` = 26
Data layout order: NDHWC.

`enumerator CNML_NCDHW` = 27
Data layout order: NCDHW.

`enumerator CNML_DHWCN` = 28
Data layout order: DHWCN.

`enumerator CNML_HW` = 29
Data layout order: HW.

`enumerator CNML_CHW` = 30
Data layout order: CHW.

`enumerator CNML_NTC` = 31
Data layout order: NTC.

`enumerator CNML_NC` = 32
Data layout order: NC.

3.28 cnmlDataType_t

```
typedef enum {
    CNML_DATA_INVALID = 0,
    CNML_DATA_FLOAT16 = 1,
    CNML_DATA_FLOAT32 = 2,
    CNML_DATA_DOUBLE = 3,
    CNML_DATA_FIX8 = 4,
    CNML_DATA_INT8 = 4,
    CNML_DATA_INT16 = 6,
    CNML_DATA_INT32 = 7,
    CNML_DATA_UINT8 = 8,
    CNML_DATA_UINT16 = 9,
    CNML_DATA_UINT32 = 10,
    CNML_DATA_QUANT8 = 11,
    CNML_DATA_BINARY = 12,
    CNML_DATA_BOOL = 13,
    CNML_DATA_INT4 = 14,
} cnmlDataType_t;
```

`enum cnmlDataType_t`

An enum.

Enumeration Variables of datatype in CNML. Enumeration cnmlDataType_t has 14 enumerated values, each of which represent a kind of datatype. This enumerated variable can be passed in when Tensor is created.

Values:

`enumerator CNML_DATA_INVALID` = 0
Invalid datatype.

`enumerator CNML_DATA_FLOAT16` = 1
FLOAT16 datatype.

`enumerator CNML_DATA_FLOAT32` = 2
FLOAT32 datatype.

`enumerator CNML_DATA_DOUBLE` = 3
DOUBLE datatype.

`enumerator CNML_DATA_FIX8` = 4
FIX8 datatype.

`enumerator CNML_DATA_INT8` = 4
INT8 datatype.

`enumerator CNML_DATA_INT16` = 6
INT16 datatype.

`enumerator CNML_DATA_INT32` = 7
INT32 datatype.

`enumerator CNML_DATA_UINT8` = 8
UINT8 datatype.

`enumerator CNML_DATA_UINT16` = 9
UINT16 datatype.

`enumerator CNML_DATA_UINT32` = 10
UINT32 datatype.

`enumerator CNML_DATA_QUANT8` = 11
QUANT8 datatype.

`enumerator CNML_DATA_BINARY` = 12
BINARY datatype.

`enumerator CNML_DATA_BOOL` = 13
BOOL datatype.

`enumerator CNML_DATA_INT4` = 14

3.29 cnmlDeconvDepthwiseOpParam

`struct cnmlDeconvDepthwiseOpParam`

A struct.

cnmlDeconvDepthwiseOpParam is a structure describing the param parameter of depthwise deconv operation, used to create depthwise deconv operation. cnmlCreateDeconvDepthwiseOpParam() is used to create an instance of cnmlConvDepthwiseOpParam_t. cnmlDestroyConvDepthwiseOpParam() is used to destroy an instance of cnmlConvDepthwiseOpParam_t.

3.30 cnmlDeconvDepthwiseOpParam_t

`typedef struct` cnmlDeconvDepthwiseOpParam `*cnmlDeconvDepthwiseOpParam_t`

`cnmlDeconvDepthwiseOpParam_t` is a pointer to *cnmlDeconvDepthwiseOpParam* which is a structure holding the description of a deconv depthwise operation param.

3.31 cnmlDeconvOpParam

`struct cnmlDeconvOpParam`

A struct.

cnmlDeconvOpParam is a structure describing the param parameter of deconv operation, used to create deconv operation. cnmlCreateDeconvOpParam() and cnmlCreateDeconvOpParam_V2() is used to create an instance of cnmlDeconvOpParam_t. cnmlDestroyDeconvOpParam() is used to destroy an instance of cnmlDeconvOpParam_t.

3.32 cnmlDeconvOpParam_t

`typedef struct` cnmlDeconvOpParam `*cnmlDeconvOpParam_t`

`cnmlDeconvOpParam_t` is a pointer to *cnmlDeconvOpParam* which is a structure holding the description of a deconv operation param.

3.33 cnmlDeformConvOpParam

`struct cnmlDeformConvOpParam`

A struct.

cnmlDeformConvOpParam is a structure describing the param parameter of deformable convolution operation, used to create deformable convolution operation. cnmlCreateDeformConvOpParam() is used to create an instance of cnmlDeformConvOpParam_t. cnmlDestroyDeformConvOpParam() is used to destroy an instance of cnmlDeformConvOpParam_t.

3.34 cnmlDeformConvOpParam_t

`typedef struct` cnmlDeformConvOpParam `*cnmlDeformConvOpParam_t`

`cnmlDeformConvOpParam_t` is a pointer to *cnmlDeformConvOpParam* which is a structure holding the description of a deformable convolution operation param.

3.35 cnmlDimension_t

typedef enum {

CNML_DIM_N = 0,

CNML_DIM_C = 1,

CNML_DIM_H = 2,

CNML_DIM_W = 3,

} cnmlDimension_t;

`enum cnmlDimension_t`

An enum.

Dimension of CNML.Enumeration cnmlDimension_t has 4 enumerated values, each of which represent a dimension. This enumerated variable can be used for function variable passing in.

Values:

`enumerator CNML_DIM_N = 0`

Dimension N.

`enumerator CNML_DIM_C = 1`

Dimension C.

`enumerator CNML_DIM_H = 2`

Dimension H.

`enumerator CNML_DIM_W = 3`

Dimension W.

`enumerator CNML_DIM_HW = 4`

Dimension HW.

`enumerator CNML_DIM_T = 5`

Dimension T.

3.36 cnmlDyadicType_t

typedef enum {

CNML_DYADIC_ADD = 0,

CNML_DYADIC_SUB = 1,

CNML_DYADIC_MULT = 2,

} cnmlDyadicType_t;

`enum cnmlDyadicType_t`

An enum.

It is an enumerated type passed to `cnmlCreateNdDyadicOp` to select the dyadic operate type.

Values:

`enumerator CNML_DYADIC_ADD` = 0

The dyadic type for the add operation.

`enumerator CNML_DYADIC_SUB` = 1

The dyadic type for the sub operation.

`enumerator CNML_DYADIC_MULT` = 2

The dyadic type for the mult operation.

3.37 cnmlDynamicRWParam

`struct cnmlDynamicRWParam`

A struct.

cnmlDynamicRWParam is a structure created for dynamicRead and dynamicWrite Ops, cnmlCreateDynamicRWParam() is used to create an instance of cnmlDynamicRWParam_t, cnmlDestroyDynamicRWParam() is used to destroy an instance of cnmlDynamicRWParam_t

3.38 cnmlDynamicRWParam_t

`typedef struct` cnmlDynamicRWParam `*cnmlDynamicRWParam_t`

`cnmlDynamicRWParam_t` is a pointer to *`cnmlDynamicRWParam`* which is a structure holding the description of a dynamicRead or dynamicWrite operation param.

3.39 cnmlFusionOp

`struct cnmlFusionOp`

A struct.

The cnmlFusionOp is a structure describing fusion operation. cnmlDestroyFusionOp() is used to create an instance of cnmlFusionOp_t. cnmlDestroyFusionOp() is used to destroy an instance of cnmlFusionOp_t.

3.40 cnmlFusionOp_t

`typedef struct` cnmlFusionOp `*cnmlFusionOp_t`

`cnmlFusionOp_t` is a pointer to *`cnmlFusionOp`* which is a structure holding the description of fusion operation.

3.41 cnmlGrepChannelOpParam

`struct cnmlGrepChannelOpParam`

A struct.

cnmlGrepChannelOpParam. is a structure describing the param parameter of grep channel operation, used to create grep channel operation. cnmlCreateGrepChannelOpParam() and cnmlCreateGrepChannelOpParam_V2() is used to create an instance of cnmlGrepChannelOpParam_t. cnmlDestroyGrepChannelOpParam() is used to destroy an instance of cnmlGrepChannelOpParam_t.

3.42 cnmlGrepChannelOpParam_t

typedef struct cnmlGrepChannelOpParam ***cnmlGrepChannelOpParam_t**

`cnmlGrepChannelOpParam_t` is a pointer to *cnmlGrepChannelOpParam* which is a structure holding the description of a grep channel operation param.

3.43 cnmlGrepOpParam

struct cnmlGrepOpParam

A struct.

cnmlGrepOpParam is a structure describing the param parameter of grep operation, used to create grep operation. cnmlCreateGrepOpParam() is used to create an instance of cnmlGrepOpParam_t. cnmlDestroyGrepOpParam() is used to destroy an instance of cnmlGrepOpParam_t.

3.44 cnmlGrepOpParam_t

typedef struct cnmlGrepOpParam ***cnmlGrepOpParam_t**

`cnmlGrepOpParam_t` is a pointer to *cnmlGrepOpParam* which is a structure holding the description of a grep operation param.

3.45 cnmlGRUMode_t

typedef enum {

CNML_GRU_MODE_V1 = 0,

CNML_GRU_MODE_V2 = 1,

} cnmlGRUMode_t;

enum cnmlGRUMode_t

Values:

enumerator CNML_GRU_MODE_V1 = 0

design formulas of GRU_MODE_V1:

i. rt = (w1*x + b1 + w2*h-1 + b2)
ii. zt = (w3*x + b3 + w4*h-1 + b4)
iii. ~ht = tanh(rt * (w5*x + b5) + w6*ht-1 + b6)
iv. ht = (1 - zt) * ht-1 + zt * ~ht

enumerator CNML_GRU_MODE_V2 = 1

design formulas of GRU_MODE_V2:

i. rt = (w1*x + b1 + w2*h-1 + b2)
ii. zt = (w3*x + b3 + w4*h-1 + b4)
iii. ~ht = tanh(rt * (w6*ht-1 + b6) + w5*x + b5)
iv. ht = (1 - zt) * ~ht + zt * ht-1

3.46 cnmlInterpOpParam

struct cnmlInterpOpParam

A struct.

cnmlInterpOpParam is a structure describing the param parameter of interp operation, used to create interp operation. cnmlCreateInterpOpParam() and cnmlCreateInterpOpParamByRatio() is used to create an instance of cnmlInterpOpParam_t. cnmlDestroyInterpOpParam() is used to destroy an instance of cnmlInterpOpParam_t.

3.47 cnmlInterpOpParam_t

typedef struct cnmlInterpOpParam ***cnmlInterpOpParam_t**

`cnmlInterpOpParam_t` is a pointer to *cnmlInterpOpParam* which is a structure holding the description of a interp operation param.

3.48 cnmlLrnOpParam

`struct cnmlLrnOpParam`

A struct.

cnmlLrnOpParam is a structure describing the param parameter of lrn operation, used to create lrn operation. cnmlCreateLrnOpParam() is used to create an instance of cnmlLrnOpParam_t. cnmlDestroyLrnOpParam() is used to destroy an instance of cnmlLrnOpParam_t.

3.49 cnmlLrnOpParam_t

`typedef struct` cnmlLrnOpParam `*cnmlLrnOpParam_t`

`cnmlLrnOpParam_t` is a pointer to *cnmlLrnOpParam* which is a structure holding the description of a LRN operation param.

3.50 cnmlLrnType_t

typedef enum {

CNML_LRN_V1,

CNML_LRN_V2,

CNML_LRN_V3

} cnmlLrnType_t;

`enum cnmlLrnType_t`

An enum.

Mode of LRN operator in CNML.Enumeration cnmlLrnType_t has 3 enumerated values, each of which represent a kind of computing mode in LRN operator. User can pass in it according to their actual needs when they create the LRN operator.

Values:

`enumerator CNML_LRN_V1`

LRN operator v1 mode. Specific meaning is Yi = Xi / [(alpha * sum(Xj^2) / m + k) ^ beta], m = min(local_size, 2*ci-1)

`enumerator CNML_LRN_V2`

LRN operator v2 mode. Specific meaning is Yi = Xi / [(alpha * sum(Xj^2) + k) ^ beta]

`enumerator CNML_LRN_V3`

LRN operator v3 mode. Specific meaning is Yi = Xi / [(alpha * sum(Xj^2) / local_size + k) ^ beta], j = i - local_size / 2 ~~ i + local_size / 2, (i and j are in C dimention)

3.51 cnmlLSTMClipParam

`struct cnmlLSTMClipParam`

A struct.

cnmlLSTMClipParam is a structure describing the param parameter of LSTM operation, used to create LSTM operation. cnmlCreateLSTMClipParam() is used to create an instance of cnmlLSTMClipParam_t. cnmlDestroyLSTMClipParam() is used to destroy an instance of cnmlLSTMClipParam_t.

3.52 cnmlLSTMClipParam_t

`typedef struct` cnmlLSTMClipParam `*cnmlLSTMClipParam_t`

`cnmlLSTMClipParam_t` is a pointer to *cnmlLSTMClipParam* which is a structure holding the descriptiomln of a LSTM operation param.

3.53 cnmlLSTMPeepholeParam

`struct cnmlLSTMPeepholeParam`

A struct.

cnmlLSTMPeepholeParam is a structure describing the param parameter of LSTM operation, used to create LSTM operation. cnmlCreateLSTMPeepholeParam() is used to create an instance of cnmlLSTMPeepholeParam_t. cnmlDestroyLSTMPeepholeParam() is used to destroy an instance of cnmlLSTMPeepholeParam_t.

3.54 cnmlLSTMPeepholeParam_t

`typedef struct` cnmlLSTMPeepholeParam `*cnmlLSTMPeepholeParam_t`

`cnmlLSTMPeepholeParam_t` is a pointer to *cnmlLSTMPeepholeParam* which is a structure holding the description of a LSTM operation param.

3.55 cnmlLSTMProjectionParam

`struct cnmlLSTMProjectionParam`

A struct.

cnmlLSTMProjectionParam is a structure describing the param parameter of LSTM operation, used to create LSTM operation. cnmlCreateLSTMProjectionParam() is used to create an instance of cnmlLSTMProjectionParam_t. cnmlDestroyLSTMProjectionParam() is used to destroy an instance of cnmlLSTMProjectionParam_t.

3.56 cnmlLSTMProjectionParam_t

`typedef struct` cnmlLSTMProjectionParam `*cnmlLSTMProjectionParam_t`

`cnmlLSTMProjectionParam_t` is a pointer to *cnmlLSTMProjectionParam* which is a structure holding the description of a LSTM operation param.

3.57 cnmlLSTMProParam

`struct cnmlLSTMProParam`

A struct.

cnmlLSTMProParam is a structure describing the param parameter of LSTM operation, used to create LSTM operation. cnmlCreateLSTMProParam() is used to create an instance of cnmlLSTMProParam_t. cnmlDestroyLSTMProParam() is used to destroy an instance of cnmlLSTMProParam_t.

3.58 cnmlLSTMProParam_t

`typedef struct` cnmlLSTMProParam `*cnmlLSTMProParam_t`

`cnmlLSTMProParam_t` is a pointer to *cnmlLSTMProParam* which is a structure holding the description of a LSTM operation param.

3.59 cnmlMaskZeroOpParam

`struct cnmlMaskZeroOpParam`

A struct.

cnmlMaskZeroOpParam is a structure describing the param parameter of computing datatype on device, using to specify the data type when Tensor participates in the computation on the chip. cnmlCreateMaskZeroOpParam() is used to create an instance of cnmlMsakZeroLayerParam_t. cnmlDestroyMaskZeroOpParam() is used to destroy an instance of cnmlMaskZeroLayerParam_t.

3.60 cnmlMaskZeroLayerParam_t

`typedef struct` cnmlMaskZeroOpParam `*cnmlMaskZeroLayerParam_t`

`cnmlMaskZeroLayerParam_t` is a pointer to the structure *cnmlMaskZeroOpParam* that descibes MaskZero operation parameters.

3.61 cnmlModel

`struct cnmlModel`

A struct.

cnmlModel is a structure describing offline model, used to generate and use offline model. cnmlCreateModel() is used to create an instance of cnmlModel_t. cnmlDestroyModel() is used to destroy an instance of cnmlModel_t.

3.62 cnmlModel_t

`typedef struct` cnmlModel `*cnmlModel_t`

`cnmlModel_t` is a pointer to *`cnmlModel`* which is a structure holding the description of a offline model.

3.63 cnmlNdCropOpParam

`struct cnmlNdCropOpParam`

A struct.

cnmlNdCropOpParam is a structure describing the param parameter of crop operation, used to create crop operation. cnmlCreateNdCropOpParam() is used to create an instance of cnmlNdCropOpParam_t. cnmlDestroyNdCropOpParam() is used to destroy an instance of cnmlNdCropOpParam_t.

3.64 cnmlNdCropOpParam_t

`typedef struct` cnmlNdCropOpParam `*cnmlNdCropOpParam_t`

`cnmlNdCropOpParam_t` is a pointer to *`cnmlNdCropOpParam`* which is a structure holding the description of a crop operation param.

3.65 cnmlNearestNeighborOpParam

`struct cnmlNearestNeighborOpParam`

A struct.

cnmlNearestNeighborOpParam is a structure describing the param parameter of nearest neighbor operation, used to create nearest neighbor operation. cnmlCreateNearestNeighborOpParam() and cnmlCreateNearestNeighborOpParamByRatio() is used to create an instance of cnmlNearestNeighborOpParam_t. cnmlDestroyNearestNeighborOpParam() is used to destroy an instance of cnmlNearestNeighborOpParam_t.

3.66 cnmlNearestNeighborOpParam_t

`typedef struct` cnmlNearestNeighborOpParam `*cnmlNearestNeighborOpParam_t`

`cnmlNearestNeighborOpParam_t` is a pointer to *`cnmlNearestNeighborOpParam`* which is a structure holding the description of a nearest neighbor operation param.

3.67 cnmlNmsOpParam

`struct cnmlNmsOpParam`

A struct.

cnmlNmsOpParam is a structure describing the param parameter of nms operation, used to create nms operation. cnmlCreateNmsOpParam() is used to create an instance of cnmlNmsOpParam_t. cnmlDestroyNmsOpParam() is used to destroy an instance of cnmlNmsOpParam_t.

3.68 cnmlNmsOpParam_t

`typedef struct` cnmlNmsOpParam `*cnmlNmsOpParam_t`

`cnmlNmsOpParam_t` is a pointer to *`cnmlNmsOpParam`* which is a structure holding the description of a nms operation param.

3.69 cnmlNormalizeOpParam

`struct cnmlNormalizeOpParam`

A struct.

cnmlNormalizeOpParam is a structure describing the param parameter of normalize operation, used to create normalize operation. cnmlCreateNormalizeOpParam() is used to create an instance of cnmlNormalizeOpParam_t. cnmlDestroyNormalizeOpParam() is used to destroy an instance of cnmlNormalizeOpParam_t.

3.70 cnmlNormalizeOpParam_t

typedef struct cnmlNormalizeOpParam ***cnmlNormalizeOpParam_t**

cnmlNormalizeOpParam_t is a pointer to *cnmlNormalizeOpParam* which is a structure holding the description of a normalized operation param.

3.71 cnmlNormalizeLiteOpParam

struct cnmlNormalizeLiteOpParam

A struct.

cnmlNormalizeLiteOpParam is a structure describing the param parameter of normalize operation,used to create normalizeLite operation. cnmlCreateNormalizeLiteOpParam() is used to create an instance of cnmlNormalizeLiteOpParam_t. cnmlDestroyNormalizeLiteOpParam() is used to destroy an instance of cnmlNormalizeLiteOpParam_t.

3.72 cnmlNormalizeLiteOpParam_t

typedef struct cnmlNormalizeLiteOpParam ***cnmlNormalizeLiteOpParam_t**

cnmlNormalizeLiteOpParam_t is a pointer to *cnmlNormalizeLiteOpParam* ,which is a structure holding the description of an normalized operation param.

3.73 cnmlNdPoolOpParam

struct cnmlNdPoolOpParam

A struct.

cnmlNdPoolOpParam is a structure holding the description of a pooling nd operation param. Use cnmlCreateNdPoolOpParam to create an instance and cnmlDestroyNdPoolOpParam to destroy this instance.

3.74 cnmlNdPoolOpParam_t

typedef struct cnmlNdPoolOpParam ***cnmlNdPoolOpParam_t**

cnmlNdPoolOpParam_t is a pointer to a structure (cnmlNdPoolOpParam) holding the description of a pooling ND operation param.

3.75 cnmlNdTransposeOpParam

struct cnmlNdTransposeOpParam

A struct.

cnmlNdTransposeOpParam is a structure describing the param parameter of transpose nd operation, used to create transpose nd operation. cnmlCreateNdTransposeOpParam() is used to create an instance of cnmlNdTransposeOpParam_t. cnmlDestroyNdTransposeOpParam() is used to destroy an instance of cnmlNdTransposeOpParam_t.

3.76 cnmlNdTransposeOpParam_t

typedef struct cnmlNdTransposeOpParam ***cnmlNdTransposeOpParam_t**

cnmlNdTransposeOpParam_t is a pointer to *cnmlNdTransposeOpParam* which is a structure holding the description of a transpose nd operation param.

3.77 cnmlPoolMode_t

typedef enum {

CNML_POOL_AVG = 0,

CNML_POOL_MAX = 1,

CNML_POOL_MAXINDEX = 2,

CNML_POOL_SUM = 3,

CNML_POOL_BLEND = 4,

} cnmlPoolMode_t;

enum cnmlPoolMode_t

An enum.

It is an enumerated type passed to cnmlCreatePoolOpParam to select the pooling value-selecting method to be used by cnmlCreatePoolOp.

Values:

`enumerator CNML_POOL_AVG` = 0
The average value inside the pooling window is used.

`enumerator CNML_POOL_MAX` = 1
The maximum value inside the pooling window is used.

`enumerator CNML_POOL_MAXINDEX` = 2

`enumerator CNML_POOL_SUM` = 3
The sum of values inside the pooling window is used. The average and maximum value inside the pooling window are used.

`enumerator CNML_POOL_BLEND` = 4

3.78 cnmlPoolOpParam

`struct cnmlPoolOpParam`
A struct.

cnmlPoolOpParam is a structure holding the description of a pooling operation param.

3.79 cnmlPoolOpParam_t

`typedef struct` cnmlPoolOpParam `*cnmlPoolOpParam_t`
`cnmlPoolOpParam_t` is a pointer to *cnmlPoolOpParam* which is a structure holding the description of a pooling operation param.

3.80 cnmlPoolStrategyMode_t

typedef enum {
CNML_POOL_KFULL = 0,
CNML_POOL_KVALID = 1,
} cnmlPoolStrategyMode_t;

`enum cnmlPoolStrategyMode_t`
An enum.

It is an enumerated type passed to `cnmlCreatePoolOpParam` to select pooling strategy method to be used by `cnmlCreatePoolOp`, which may cause different output size.

Values:

`enumerator CNML_POOL_KFULL` = 0
The pooing window can be out of bounds.

`enumerator CNML_POOL_KVALID` = 1
The pooling window must be within the bounds.

3.81 cnmlQuantizedParam

`struct cnmlQuantizedParam`
A struct.

The cnmlQuantizedParam is a structure describing the param parameter of computing datatype on device, using to specify the data type when Tensor participates in the computation on the chip by calling cnmlSetOperationComputingDataType(). cnmlCreateQuantizedParam() is used to create an instance of cnmlQuantizedParam_t. cnmlDestroyQuantizedParam() is used to destroy an instance of cnmlQuantizedParam_t.

3.82 cnmlQuantizedParam_t

`typedef struct` cnmlQuantizedParam `*cnmlQuantizedParam_t`
`cnmlQuantizedParam_t` is a pointer to the structure *cnmlQuantizedParam* that describes quantization parameters.

3.83 cnmlRandomUniformOpParam

`struct cnmlRandomUniformOpParam`

A struct.

cnmlRandomUniformOpParam is a structure describing the param parameter of random_uniform operation, used to create random_uniform operation. cnmlCreateRandomUniformOpParam() is used to create an instance of cnmlRandomUniformOpParam_t. cnmlDestroyRandomUniformOpParam() is used to destroy an instance of cnmlRandomUniformOpParam_t.

3.84 cnmlRandomUniformOpParam_t

`typedef struct` cnmlRandomUniformOpParam *`cnmlRandomUniformOpParam_t`

`cnmlRandomUniformOpParam_t` is a pointer to *cnmlRandomUniformOpParam* which is a structure holding the description of a Conv operation param.

3.85 cnmlReduce_andDim_t

typedef enum {

CNML_REDUCE_AND_DIM_C = 3,

CNML_REDUCE_AND_DIM_W = 1,

CNML_REDUCE_AND_DIM_H = 2,

CNML_REDUCE_AND_DIM_N = 4

} cnmlReduce_andDim_t;

`enum cnmlReduce_andDim_t`

An enum.

In which dimension of reduce_and operator is calculated in CNML. Enumeration cnmlReduce_orDim_t has 4 enumerated values, indicating the dimension in which reduce_and operator is calculated. User can pass in it according to their actual needs when they create the reduce_and operator. At present only N dimension and C dimension is supported.

Values:

`enumerator CNML_REDUCE_AND_DIM_C = 3`

Compute reduce_and in dimension C.

`enumerator CNML_REDUCE_AND_DIM_W = 1`

Compute reduce_and in dimension W.

`enumerator CNML_REDUCE_AND_DIM_H = 2`

Compute reduce_and in dimension H.

`enumerator CNML_REDUCE_AND_DIM_N = 4`

Compute reduce_and in dimension N.

3.86 cnmlReduce_orDim_t

typedef enum {

CNML_REDUCE_OR_DIM_C = 3,

CNML_REDUCE_OR_DIM_W = 1,

CNML_REDUCE_OR_DIM_H = 2,

CNML_REDUCE_OR_DIM_N = 4

} cnmlReduce_orDim_t;

`enum cnmlReduce_orDim_t`

An enum.

In which dimension of reduce_or operator is calculated in CNML. Enumeration cnmlReduce_orDim_t has 4 enumerated values, indicating the dimension in which reduce_or operator is calculated. User can pass in it according to their actual needs when they create the reduce_or operator. At present, only N dimension and C dimension is supported.

Values:

`enumerator CNML_REDUCE_OR_DIM_C = 3`

Compute reduce_or in dimension C.

`enumerator CNML_REDUCE_OR_DIM_W = 1`

Compute reduce_or in dimension W.

`enumerator CNML_REDUCE_OR_DIM_H = 2`

Compute reduce_or in dimension H.

`enumerator CNML_REDUCE_OR_DIM_N` = 4
Compute reduce_or in dimension N.

3.87 cnmlReductionmode_t

typedef enum {

NONE = 1,

MEAN = 2,

SUM = 3,

} cnmlReductionmode_t;

`enum cnmlReductionmode_t`
Values:

`enumerator NONE` = 0

`enumerator MEAN` = 1

`enumerator SUM` = 2

3.88 cnmlReorgOpParam

`struct cnmlReorgOpParam`
A struct.

cnmlReorgOpParam is a structure describing the param parameter of reorg operation, used to create reorg operation. cnmlCreateReorgOpParam() is used to create an instance of cnmlReorgOpParam_t. cnmlDestroyReorgOpParam() is used to destroy an instance of cnmlReorgOpParam_t.

3.89 cnmlReorgOpParam_t

`typedef struct` cnmlReorgOpParam `*cnmlReorgOpParam_t`
`cnmlReorgOpParam_t` is a pointer to *cnmlReorgOpParam* which is a structure holding the description of a reorg operation param.

3.90 cnmlReshapeOpParam

`struct cnmlReshapeOpParam`
A struct.

cnmlReshapeOpParam is a structure describing the param parameter of reshape operation, used to create reshape operation. cnmlCreateReshapeOpParam() is used to create an instance of cnmlReshapeOpParam_t. cnmlDestroyReshapeOpParam() is used to destroy an instance of cnmlReshapeOpParam_t.

3.91 cnmlReshapeOpParam_t

`typedef struct` cnmlReshapeOpParam `*cnmlReshapeOpParam_t`
`cnmlReshapeOpParam_t` is a pointer to *cnmlReshapeOpParam* which is a structure holding the description of a reshape operation param.

3.92 cnmlRngType_t

typedef enum {

CNML_RNG_MT19937 = 0,

} cnmlRngType_t;

`enum cnmlRngType_t`
cnmlRngType_t is an enumeration type to describe underlying algorithm to pseudo random number.

Values:

`enumerator CNML_RNG_MT19937` = 0

3.93 cnmlRgbType_t

typedef enum {

CNML_RGB0 = 0,

CNML_BGR0 = 1,

CNML_ARGB = 2,

} cnmlRgbType_t;

`enum cnmlRgbType_t`

An enum.

The order of output channels in yuv to rgb operator in CNML.Enumeration cnmlRgbType_t has 3 enumerated values, each of which represent a kind of the order of output channels in yuv to rgb operator. User can pass in it according to their actual needs when they create the yuv to rgb operator.

Values:

`enumerator CNML_RGB0` = 0

The order of output channels is RGB0.

`enumerator CNML_BGR0` = 1

The order of output channels is BGR0.

`enumerator CNML_ARGB` = 2

The order of output channels is ARGB.

3.94 cnmlRNNInputMode_t

typedef enum {

CNML_RNN_LINEAR_INPUT = 0,

CNML_RNN_SKIP_INPUT = 1,

} cnmlRNNInputMode_t;

`enum cnmlRNNInputMode_t`

An enum.

Type of RNN operator input in CNML.Enumeration cnmlRNNInputMode_t has 3 enumerated values, each of which represent a type of RNN operator input. User can pass in it according to their actual needs when they create the RNN operator.

Values:

`enumerator CNML_RNN_LINEAR_INPUT` = 0

LINEAR type input.

`enumerator CNML_RNN_SKIP_INPUT` = 1

SKIP type input.

3.95 cnmlRNNOpParam

`struct cnmlRNNOpParam`

A struct.

cnmlRNNOpParam is a structure describing the param parameter of basic rnn pro operation, used to create basic rnn pro operation or LSTM operation. cnmlCreateRNNOpParam() is used to create an instance of cnmlRNNOpParam_t. cnmlDestroyRNNOpParam() is used to destroy an instance of cnmlRNNOpParam_t.

3.96 cnmlRNNOpParam_t

`typedef struct` cnmlRNNOpParam `*cnmlRNNOpParam_t`

`cnmlRNNOpParam_t` is a pointer to *cnmlRNNOpParam* which is a structure holding the description of a RNN operation param.

3.97 cnmlScatterOpParam

`struct cnmlScatterOpParam`

A struct.

cnmlScatterOpParam is a structure describing the parameters required by scatter. cnmlCreateScatterOpParam() and cnmlDestroyScatterOpParam are used to create and destroy an instance of cnmlScatterOpParam_t, respectively.

3.98 cnmlScatterOpParam_t

`typedef struct` cnmlScatterOpParam `*cnmlScatterOpParam_t`

`cnmlScatterOpParam_t` is a pointer to *cnmlScatterOpParam*, which is a struct describing the parameters of the scatter operator.

3.99 cnmlScatterRefOpParam

`struct cnmlScatterRefOpParam`

A struct.

The cnmlScatterRefOpParam is a structure describing the parameters for selecting scatter_add, scatter_sub, scatter_mul, scatter_div, scatter_max, scatter_min and scatter_update. The scatter_div is not supported for now.

The cnmlCreateScatterRefOpParam() and cnmlDestroyScatterRefOpParam are used to create and destroy an instance of cnmlScatterRefOpParam_t, respectively.

3.100 cnmlScatterRefOpParam_t

`typedef struct` cnmlScatterRefOpParam `*cnmlScatterRefOpParam_t`

`cnmlScatterRefOpParam_t` is a pointer to *cnmlScatterRefOpParam*, which is a struct for selecting scatter_add, scatter_sub, scatter_mul, scatter_div, scatter_max, scatter_min and scatter_update.

3.101 cnmlScatterType_t

typedef enum {

CNML_SCATTER_ADD = 0,

CNML_SCATTER_SUB = 1,

CNML_SCATTER_MAX = 2,

CNML_SCATTER_MIN = 3,

CNML_SCATTER_MUL = 4,

CNML_SCATTER_DIV = 5,

CNML_SCATTER_UPDATE = 6,

} cnmlScatterType_t;

`enum cnmlScatterType_t`

An enum.

It is an enumerated type passed to `cnmlCreateScatterRefOpParam` to select the scatter type, which is to be used by `cnmlCreateScatterRefOp`.

Values:

`enumerator CNML_SCATTER_ADD` = 0

`enumerator CNML_SCATTER_SUB` = 1

`enumerator CNML_SCATTER_MAX` = 2

`enumerator CNML_SCATTER_MIN` = 3

`enumerator CNML_SCATTER_MUL` = 4

`enumerator CNML_SCATTER_DIV` = 5

`enumerator CNML_SCATTER_UPDATE` = 6

3.102 cnmlSequenceMaskOpParam

`struct cnmlSequenceMaskOpParam`

A struct.

The cnmlSequenceMaskOpParam is a structure describing the param parameter of SequenceMask operation, used to create deconv operation. cnmlCreateSequenceMaskOpParam() is used to create an instance of cnmlSequenceMaskOpParam_t. cnmlDestroySequenceMaskOpParam() is used to destroy an instance of cnmlSequenceMaskOpParam_t.

3.103 cnmlSequenceMaskOpParam_t

`typedef struct` cnmlSequenceMaskOpParam `*cnmlSequenceMaskOpParam_t`

`cnmlSequenceMaskOpParam_t` is a pointer to *cnmlSequenceMaskOpParam* which is a structure holding the description of a SequenceMask operation param.

3.104 cnmlSplitOpParam

`struct cnmlSplitOpParam`

A struct.

cnmlSplitOpParam is a structure describing the param parameter of split operation, used to create split operation. cnmlCreateSplitOpParam() is used to create an instance of cnmlSplitOpParam_t. cnmlDestroySplitOpParam() is used to destroy an instance of cnmlSplitOpParam_t.

3.105 cnmlSplitOpParam_t

`typedef struct` cnmlSplitOpParam `*cnmlSplitOpParam_t`

`cnmlSplitOpParam_t` is a pointer to *cnmlSplitOpParam* which is a structure holding the description of a split operation param.

3.106 cnmlStridedSliceOpParam

`struct cnmlStridedSliceOpParam`

A struct.

cnmlStridedSliceOpParam is a structure describing the param parameter of Strided Slice operation, used to create Strided Slice operation. cnmlCreateStridedSliceOpParam() is used to create an instance of cnmlStridedSliceOpParam_t. cnmlDestroyStridedSliceOpParam() is used to destroy an instance of cnmlStridedSliceOpParam_t.

3.107 cnmlStridedSliceOpParam_t

`typedef struct` cnmlStridedSliceOpParam `*cnmlStridedSliceOpParam_t`

`cnmlStridedSliceOpParam_t` is a pointer to *cnmlStridedSliceOpParam* which is a structure holding the description of a strided slice operation param.

3.108 cnmlNdStridedSliceOpParam

`struct cnmlNdStridedSliceOpParam`

A struct.

cnmlNdStridedSliceOpParam is a structure describing the param parameter of Strided Slice nd operation, used to create Strided Slice nd operation. cnmlCreateNdStridedSliceOpParam() is used to create an instance of cnmlNdStridedSliceOpParam_t. cnmlDestroyNdStridedSliceOpParam() is used to destroy an instance of cnmlNdStridedSliceOpParam_t.

3.109 cnmlNdStridedSliceOpParam_t

`typedef struct` cnmlNdStridedSliceOpParam `*cnmlNdStridedSliceOpParam_t`

`cnmlNdStridedSliceOpParam_t` is a pointer to *cnmlNdStridedSliceOpParam* which is a structure holding the description of a strided slice nd operation param.

3.110 cnmlStatus_t

typedef enum {

CNML_STATUS_NODEVICE = -1,

CNML_STATUS_SUCCESS = 0,

CNML_STATUS_DOMAINERR = 1,

CNML_STATUS_INVALIDARG = 2,

CNML_STATUS_LENGTHERR = 3,

CNML_STATUS_OUTOFRANGE = 4,

CNML_STATUS_RANGEERR = 5,

CNML_STATUS_OVERFLOWERR = 6,

CNML_STATUS_UNDERFLOWERR = 7,

CNML_STATUS_INVALIDPARAM = 8,

CNML_STATUS_BADALLOC = 9,

CNML_STATUS_BADTYPEID = 10,

CNML_STATUS_BADCAST = 11,

CNML_STATUS_UNSUPPORT = 12

} cnmlStatus_t;

`enum cnmlStatus_t`

An enum.

Values:

`enumerator CNML_STATUS_NODEVICE` = -1

No device error.

`enumerator CNML_STATUS_SUCCESS` = 0

Success.

`enumerator CNML_STATUS_DOMAINERR` = 1

Domain error.

`enumerator CNML_STATUS_INVALIDARG` = 2

Invalid parameter error.

`enumerator CNML_STATUS_LENGTHERR` = 3

Length error.

`enumerator CNML_STATUS_OUTOFRANGE` = 4

Out of range error.

`enumerator CNML_STATUS_RANGEERR` = 5

Out of bounds error.

`enumerator CNML_STATUS_OVERFLOWERR` = 6

Overflow error.

`enumerator CNML_STATUS_UNDERFLOWERR` = 7

Underflow error.

`enumerator CNML_STATUS_INVALIDPARAM` = 8

Invalid param error.

`enumerator CNML_STATUS_BADALLOC` = 9

Memory alloc error.

`enumerator CNML_STATUS_BADTYPEID` = 10

TYPE ID error.

`enumerator CNML_STATUS_BADCAST` = 11

Type conversion error.

`enumerator CNML_STATUS_UNSUPPORT` = 12

Unsupport.

3.111 cnmlTensor

`struct cnmlTensor`

A struct.

The cnmlTensor_t is a structure describing tensor in MLU. cnmlCreateTensor() is used to create a 4D instance of cnmlTensor_t. cnmlCreateTensor_V2() is used to create a ND instance of cnmlTensor_t. cnmlDestroyTensor() is used to destroy an instance of cnmlTensor_t.

3.112 cnmlTensor_t

`typedef struct` cnmlTensor `*cnmlTensor_t`

`cnmlTensor_t` is a pointer to *cnmlTensor* which is a structure holding the description of a tensor in MLU.

3.113 cnmlTensorType_t

typedef enum {

CNML_TENSOR = 0,

CNML_FILTER = 1,

CNML_CONST = 2,

CNML_VARIABLE = 3,

} cnmlTensorType_t;

`enum cnmlTensorType_t`

An enum.

Tensor type of CNML.Enumeration cnmlTensorType_t has 3 enumerated values, each of which represent a kind of tensor type. Users pass in different types of tensor in the interface for creating tensor to create different types of tensor.

Values:

`enumerator CNML_TENSOR` = 0

Tensor type.

`enumerator CNML_FILTER` = 1

Filter type.

`enumerator CNML_CONST` = 2

Const type.

`enumerator CNML_VARIABLE` = 3

variable type.

3.114 cnmlTransposeOpParam

`struct cnmlTransposeOpParam`

A struct.

cnmlTransposeOpParam is a structure describing the param parameter of transpose operation, used to create transpose operation. cnmlCreateTransposeOpParam() is used to create an instance of cnmlTransposeOpParam_t. cnmlDestroyTransposeOpParam() is used to destroy an instance of cnmlTransposeOpParam_t.

3.115 cnmlTransposeOpParam_t

`typedef struct` cnmlTransposeOpParam `*cnmlTransposeOpParam_t`

`cnmlTransposeOpParam_t` is a pointer to *cnmlTransposeOpParam* which is a structure holding the description of a transpose operation param.

3.116 cnmlTrilOpParam

`struct cnmlTrilOpParam`

A struct.

cnml TrilOpParam is a struct describing the param parameter of tril used to create tril operation. cnmlCreateTrilOpParam() is used to create an instance of cnmlTrilOpParam_t, cnmlDestroyTrilOpParam() is used to destroy an instance of cnmlTrilOpParam_t

3.117 cnmlTrilOpParam_t

`typedef struct` cnmlTrilOpParam `*cnmlTrilOpParam_t`

```
``cnmlTrilOpParam_t `` is a pointer to ``cnmlTrilOpParam`` which is a
```

structure holding the description of a tril operation param

3.118 cnmlUnpoolMode_t

typedef enum {

CNML_UNPOOL = 0,

CNML_ROWWISE_UNPOOL = 1,

CNML_MAXPOOLBP = 2,

CNML_AVGPOOLBP = 3,

CNML_MIDUNPOOL = 4,

CNML_DIV = 5,

CNML_REP = 6

} cnmlUnpoolMode_t;

`enum cnmlUnpoolMode_t`

An enum.

Specific mode of unpool operator in CNML. Enumeration cnmlUnpoolMode_t has 7 enumerated values, each of which represent a kind of unpool mode. User can pass in it according to their actual needs when they create the unpool operator.

Values:

`enumerator CNML_UNPOOL` = 0

The unpool mode.

`enumerator CNML_ROWWISE_UNPOOL` = 1

The rowwise unpool mode.

`enumerator CNML_MAXPOOLBP` = 2

The maxpoolbp mode.

`enumerator CNML_AVGPOOLBP` = 3

The avgpoolbp mode.

`enumerator CNML_MIDUNPOOL` = 4

The midunpool mode.

`enumerator CNML_DIV` = 5

The div mode.

`enumerator CNML_REP` = 6

The rep mode.

`enumerator CNML_DILATION_UNPOOL` = 7

The dilation unpool mode.

3.119 cnmlUnpoolOpParam

`struct cnmlUnpoolOpParam`

A struct.

cnmlUnpoolOpParam is a structure describing the param parameter of unpooling operation, used to create unpool operation. cnmlCreateUnpoolOpParam() is used to create an instance of cnmlUnpoolOpParam_t. cnmlDestroyUnpoolOpParam() is used to destroy an instance of cnmlUnpoolOpParam_t.

3.120 cnmlUnpoolOpParam_t

`typedef struct` cnmlUnpoolOpParam `*cnmlUnpoolOpParam_t`

`cnmlUnpoolOpParam_t` is a pointer to *cnmlUnpoolOpParam* which is a structure holding the description of a unpooling operation param.

3.121 cnmlUnPoolStrategyMode_t

typedef enum {

CNML_UNPOOL_KSAME = 0,

CNML_UNPOOL_KVALID = 1,

} cnmlUnPoolStrategyMode_t;

`enum cnmlUnPoolStrategyMode_t`

An enum.

It is an enumerated type passed to `cnmlCreateUnPoolParam` to select unpool strategy method to be used by `cnmlCreateUnPoolOp`, which may cause different output size.

Values:

`enumerator CNML_UNPOOL_KSAME` = 0

The window can be out of bounds.

`enumerator CNML_UNPOOL_KVALID` = 1

The window must be within the bounds.

3.122 cnmlYuvType_t

typedef enum {

CNML_YUV420SP_NV12 = 0,

CNML_YUV420SP_NV21 = 1,

} cnmlYuvType_t;

`enum cnmlYuvType_t`

An enum.

Types of yuv in yuv to rgb operator in CNML.Enumeration cnmlYuvType_t has 2 enumerated values, each of which represent a type of yuv in yuv to rgb operator. User can pass in it according to their actual needs when they create the yuv to rgb operator.

Values:

`enumerator CNML_YUV420SP_NV12` = 0

YUV420SP_NV12 type, correspond to YCbCr.

`enumerator CNML_YUV420SP_NV21` = 1

YUV420SP_NV21 type, correspond to YCrCb.

3.123 cnmlWhereOpParam

`struct cnmlWhereOpParam`

A struct.

cnmlWhereOpParam is a structure describing the param parameter of where operation, used to create where operation. cnmlCreateWhereOpParam() is used to create an instance of cnmlWhereOpParam_t. cnmlDestroyWhereOpParam() is used to destroy an instance of cnmlWhereOpParam_t.

3.124 cnmlWhereOpParam_t

typedef struct cnmlWhereOpParam ***cnmlWhereOpParam_t**

`cnmlWhereOpParam_t` is a pointer to *cnmlWhereOpParam* which is a structure holding the description of a where operation param.

4 CNML API Reference

4.1 Common Function

4.1.1 cnmlAddBaseOpToModel

cnmlStatus_t `cnmlAddBaseOpToModel`(cnmlModel_t model, cnmlBaseOp_t op, `const` char *symbol)

A function.

Add a compiled base operator to the model.

Parameters

- `[in] model`: Input. A pointer pointing to the model.
- `[in] op`: Input. A pointer pointing to base operator.
- `[in] symbol`: Input. The name of this add operation (for future reference).

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.2 cnmlAddFusionInput

cnmlStatus_t `cnmlAddFusionInput`(cnmlFusionOp_t op, cnmlTensor_t input_tensor)

A function.

This function is used to increase the input of the fusion operation.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operator.
- `[in] input`: Input. A four-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.3 cnmlAddFusionOpCacheGraph

cnmlStatus_t `cnmlAddFusionOpCacheGraph`(cnmlFusionOp_t all_cache_graph, cnmlFusionOp_t basic_graph)

A function.

This function is used to add basic_graph to all_cache_graph. Before calling this API, make sure the cache mode is set to `true`. You can set the cache mode by calling the `cnmlSetFusionOpCacheMode` API.

Parameters

- `[in] all_cache_graph`: Input. A pointer pointing to fusion operator, should be created.
- `[in] basic_graph`: Input. A pointer pointing to fusion operator, should be set all basic graph ready and should not set cache mode true.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - All cache graph pointer or basic_graph is null.
  - All cache graph’ s cache mode is false.
  - Basic graph is empty or is set cache mode true.

4.1.4 cnmlAddFusionOpToModel

cnmlStatus_t **cnmlAddFusionOpToModel**(cnmlModel_t model, cnmlFusionOp_t op, const char *symbol)

A function.

Add a compiled fusion operator to the model.

Parameters

- [in] model: Input. A pointer pointing to the model.
- [in] op: Input. A pointer pointing to fusion operator.
- [in] symbol: Input. The name of this add operation (for future reference).

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.5 cnmlAddFusionOutput

cnmlStatus_t **cnmlAddFusionOutput**(cnmlFusionOp_t op, cnmlTensor_t output_tensor)

A function.

This function is used to increase the output of the fusion operation.

Parameters

- [in] op: Input. A pointer pointing to fusion operator.
- [in] output: Input. A four-dimensional MLU output tensor.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
  - Output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.6 cnmlBindConstData

cnmlStatus_t **cnmlBindConstData**(cnmlTensor_t tensor, cnmlCpuTensor_t cpu_tensor, void *cpu_tensor_ptr)

A function.

This function is used to bind the data information of the tensors at MLU and CPU end.

Parameters

- [in] tensor: Input. A pointer pointing to the tensor at MLU end, and the interface currently supports only CNML_FILTER and CNML_CONST-type Tensor.
- [in] cpu_tensor: Input. A pointer pointing to the tensor at CPU end, and the interface currently supports only CNML_FILTER and CNML_CONST-type Tensor.
- [in] cpu_data_ptr: Input. A pointer of void type pointing to the binding data address of the Tensor at CPU end.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointers of tensor, cpu_tensor, cpu_data_ptr are null.
  - Tensor type is not supported.

4.1.7 cnmlBindConstData_V2

cnmlStatus_t **cnmlBindConstData_V2**(cnmlTensor_t tensor, void *cpu_tensor_ptr, bool free_aftercompile)

A function.

This function is used to bind the data information of the tensors at MLU and CPU end. If tensor type is CNML_FILTER or CNML_CONST, this function should be called before compiling by cnmlCompileBaseOp_V2 or cnmlCompileFusionOp_V2.

Parameters

- [in] tensor: Input. A pointer pointing to the tensor at MLU end, and the interface currently supports only CNML_FILTER and CNML_CONST-type Tensor.
- [in] cpu_data_ptr: Input. A pointer of void type pointing to the binding data address of the Tensor at CPU end.
- [in] free_aftercompile: Input. A bool variable, the cpu_tensor_ptr will be freed automatically if the value is true, otherwise it should be freed by user manually.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointers of tensor, cpu_data_ptr are null.
  - Tensor type is not supported.

4.1.8 cnmlBindCpuDataInfo

cnmlStatus_t **cnmlBindCpuDataInfo**(cnmlTensor_t tensor, cnmlCpuTensor_t cpu_tensor)

A function.

When data at MLU end participates in some operations, it is necessary to get the information of corresponding data at CPU end. This function is used to bind the information of Tensor at the CPU to the MLU, so that MLU can get the necessary information at the CPU end when computing.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at MLU end.
- `[in] cpu_tensor`: Input. A pointer pointing to the tensor at CPU end.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - The pointer of cpu_tensor is null.

4.1.9 cnmlCheckBaseOpRunnable

cnmlStatus_t **cnmlCheckBaseOpRunnable**(cnmlBaseOp_t op, cnmlCoreVersion_t version)

A function.

Check whether the base operator can run.

Parameters

- `[in] op`: Input. A pointer pointing to base oeprator.
- `[in] version`: Input. Platform version.(MLU270)

Return Value

- `CNML_STATUS_UNSUPPORT`: This function is not complete.

4.1.10 cnmlCompileBaseOp

cnmlStatus_t **cnmlCompileBaseOp**(cnmlBaseOp_t op, cnmlCoreVersion_t version, int core_limit)

A function.

Compile base operators.

Parameters

- `[in] op`: Input. A pointer pointing to base oeprator.
- `[in] version`: Input. Platform version.
- `[in] core_num`: Input. Number of cores participating in the computation.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.11 cnmlCompileBaseOp_V2

cnmlStatus_t **cnmlCompileBaseOp_V2**(cnmlBaseOp_t op)

A function.

Compile base operators.

Parameters

- `[in] op`: Input. A pointer pointing to base oeprator.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.12 cnmlCompileFusionOp

cnmlStatus_t **cnmlCompileFusionOp**(cnmlFusionOp_t op, cnmlCoreVersion_t version, int core_limit)

A function.

This function is used to compile fusion operators.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operator.
- `[in] version`: Input. Kernel version.
- `[in] core_limit`: Input. Kernel restriction.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.

4.1.13 cnmlCompileFusionOp_V2

cnmlStatus_t `cnmlCompileFusionOp_V2`(cnmlFusionOp_t op)

A function.

The function compiles the corresponding operating instructions according to the fusion operation pointer given by the user and the operating parameters (the default value can be used) set by the user in the pointer.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operation.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.14 cnmlComputeFusionOpForward_V3

cnmlStatus_t `cnmlComputeFusionOpForward_V3`(cnmlFusionOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeFusionOpForward_V4 is recommended to use.

The interface is used to perform specified fusion operations on input in MLU.

Parameters

- `[out] outputs[]`: Output. A pointer pointng to the MLU address of the output position.
- `[in] op`: Input. A pointer pointing to fusion operator.
- `[in] inputs[]`: Input. A pointer pointing to the initial address of input data.
- `[in] input_num`: Input. The number of input.
- `[in] output_num`: Input. The number of output.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.15 cnmlComputeFusionOpForward_V4

cnmlStatus_t `cnmlComputeFusionOpForward_V4`(cnmlFusionOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

A function.

The interface is used to perform specified fusion operations on input in MLU.

Parameters

- `[out] outputs[]`: Output. A pointer pointng to the MLU address of the output position.
- `[in] op`: Input. A pointer pointing to fusion operator.
- `[in] inputs[]`: Input. A pointer pointing to the initial address of input data.
- `[in] input_num`: Input. The number of input.
- `[in] output_num`: Input. The number of output.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.16 cnmlConfigAutotune

cnmlStatus_t `cnmlConfigAutotune`(cnmlFusionOp_t op, cnmlCoreVersion_t version, int core_limit, void *inputs[], int input_num, int time_limit, `const` char *config_path)

A function.

This function traverses the value of the config variables with the given fusion operation and returns the config with the best computing performance in a config file after compiling and computing multiple times.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operation.
- `[in] version`: Input. Platform version. Supports CNML_MLU270 and CNML_MLU220.
- `[in] core_limit`: Input. Number of MLU cores participating in the computation.
- `[in] inputs[]`: Input. A pointer pointing to the initial address of input data.
- `[in] input_num`: Input. The number of inputs.
- `[in] time_limit`: Input. The expected running time for autotune, unit is minute. Recommended to be larger than 20.
- `[in] config_path`: Input. The relative path of fine tuned config file which will be saved by this API, such as "resnet50_config_opt.ini"

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.17 cnmlCopyFusionOp

cnmlStatus_t `cnmlCopyFusionOp`(cnmlFusionOp_t *dst_op, cnmlFusionOp_t src_op)

A function.

This function is used to create a fusion operator by deep copying another fusion operator. Src_op should be uncompiled.

Parameters

- `[out] dst_op`: Output. A pointer pointing to the address of the new fusion operator.
- `[in] src_op`: Input. A pointer pointing to the origin fusion operation.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - src_op pointer is null.
  - dst_op pointer is null.
  - src_op is compiled.

4.1.18 cnmlCreateCpuTensor

cnmlStatus_t `cnmlCreateCpuTensor`(cnmlCpuTensor_t *cpu_tensor, cnmlTensorType_t tensor_type, cnmlDataType_t data_type, cnmlDataOrder_t data_order, int n, int c, int h, int w)

A function.

This function initializes a tensor at CPU end according to the user-specified Tensor type.

Parameters

- `[out] cpu_tensor`: Output. A pointer pointing to the tensor at CPU end that has been created.
- `[in] tensor_type`: Input. An enumeration variable indicates the tensor type. The optional types are CNML_TENSOR, CNML_FILTER and CNML_CONST_TENSOR.
- `[in] data_type`: Input. A variable that indicates the type of data at CPU end.
- `[in] data_order`: Input. Variable indicating the placement format of data at CPU end, currently, the supported orders are: CNML_NCHW,CNML_NHWC,CNML_HWCN,CNML_TNC.
- `[in] n`: Input. An integer variable that indicates the size of n dimension.
- `[in] c`: Input. An integer variable that indicates the size of c dimension.
- `[in] h`: Input. An integer variable that indicates the size of h dimension.
- `[in] w`: Input. An integer variable that indicates the size of w dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_type,data_type,data_orderis not supported.

4.1.19 cnmlCreateCpuTensor_V2

cnmlStatus_t `cnmlCreateCpuTensor_V2`(cnmlCpuTensor_t *tensor, cnmlTensorType_t tensor_type)

A function.

This function initializes a tensor at CPU end according to the user-specified Tensor type.

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlCpuTensor_t.
- `[in] tensor_type`: Input. A variable indicating Tensor type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_type not supported.

4.1.20 cnmlCreateFusionOp

cnmlStatus_t `cnmlCreateFusionOp`(cnmlFusionOp_t *op)

A function.

This function is used to create a fusion operator pointer.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the fusion operator.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.21 cnmlCreateModel

cnmlStatus_t `cnmlCreateModel`(cnmlModel_t *model, `const` char *name)

A function.

This function creates an offline model.

Parameters

- `[in] model`: Input. A pointer pointing to an offline model.
- `[in] name`: Input. Name of the offline model.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.22 cnmlCreateQuantizedParam

cnmlStatus_t `cnmlCreateQuantizedParam`(cnmlQuantizedParam_t *param, int pos, float scale, float offset)

A function.

This function is used to create quant parameter of the tensor at MLU end when it is quantized to int precision.

Parameters

- `[in] tensor`: Input. A pointer pointing to the cnmlQuantizedParam.
- `[in] pos`: Input. A int variable indicating the position value.
- `[in] scale`: Input. A floating-point variable indicating the scale value.
- `[in] offset`: Input. A floating-point variable indicating the offset value.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of cnmlQuantizedParam is null.

4.1.23 cnmlCreateQuantizedParamByChannel

cnmlStatus_t `cnmlCreateQuantizedParamByChannel`(cnmlQuantizedParam_t *param, int *pos, float *scale, float *offset, int channel_num)

A function.

This function is used to create quant parameter of the tensor at MLU end when it is quantized to int precision.

Parameters

- `[in] tensor`: Input. A pointer pointing to the cnmlQuantizedParam.
- `[in] pos`: Input. An integer array that saves the value of the position parameter used for each channel quantization.
- `[in] scale`: Input. A floating-point array that saves the value of the scale parameter used for each channel quantization.
- `[in] offset`: Input. A pointer variable point to float type indicating the offset value.
- `[in] channel_num`: Input. An integer variable indicating the number of channels.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of cnmlQuantizedParam is null.

4.1.24 cnmlDestroyQuantizedParam

cnmlStatus_t **cnmlDestroyQuantizedParam**(cnmlQuantizedParam_t *param)

A function.

This function is used to destroy an instance of cnmlQuantizedParam_t.

Parameters

- [in] param: Input. A pointer instance pointing to the cnmlQuantizedParam.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer instance of cnmlQuantizedParam is null.

4.1.25 cnmlCreateTensor

cnmlStatus_t **cnmlCreateTensor**(cnmlTensor_t *tensor, cnmlTensorType_t tensor_type, cnmlDataType_t data_type, int n, int c, int h, int w)

A function.

This function initializes a multidimensional (1-N-dimensional) Tensor at the MLU end according to the user-specified Tensor type.

This interface is an older version and will not be maintained in the future. It is recommended to use cnmlCreateTensor_V2.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t
- [in] tensor_type: Input. A variable indicating the Tensor type
- [in] data_type: Input. A variable indicating the Tensor data type
- [in] n: Input. A variable indicating the size of the input Tensor in the n-dimension
- [in] c: Input. A variable indicating the size of the input Tensor in the c-dimension
- [in] h: Input. A variable indicating the size of the input Tensor in the h-dimension
- [in] w: Input. A variable indicating the size of the input Tensor in the w-dimension

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_typeis not supported.

4.1.26 cnmlCreateTensor_V2

cnmlStatus_t **cnmlCreateTensor_V2**(cnmlTensor_t *tensor, cnmlTensorType_t tensor_type)

A function.

This function initializes a multidimensional (1-N-dimensional) Tensor at the MLU end according to the user-specified Tensor type.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t
- [in] tensor_type: Input. A variable indicating the Tensor type

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_typeis not supported.

4.1.27 cnmlCreateTensor_V3

cnmlStatus_t **cnmlCreateTensor_V3**(cnmlTensor_t *tensor)

A function.

```
This function initializes a multidimensional (1-N-dimensional) Tensor at the MLU end
```

according to the user-specified Tensor type.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_typeis not supported.

4.1.28 cnmlDestroyBaseOp

cnmlStatus_t `cnmlDestroyBaseOp`(cnmlBaseOp_t *op)

A function.

Destroy base operator and free spce.

Parameters

- `[in] op`: Input. A pointer pointing to base operator.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.29 cnmlDestroyCpuTensor

cnmlStatus_t `cnmlDestroyCpuTensor`(cnmlCpuTensor_t *cpu_tensor)

A function.

According to the pointer given by the user, the pointer of the tensor at cpu end is freed.

Parameters

- `[in] cpu_tensor`: Input. A pointer pointing to the tensor address at the CPU end.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of cpu_tensor is null.
  - The pointer content pointed to by cpu_tensor has been freed.

4.1.30 cnmlDestroyFusionOp

cnmlStatus_t `cnmlDestroyFusionOp`(cnmlFusionOp_t *op)

A function.

This function is used to destroy the fusion operator pointer.

Parameters

- `[out] op`: Output. A pointer to the address of the fusion operator.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.31 cnmlDestroyModel

cnmlStatus_t `cnmlDestroyModel`(cnmlModel_t model)

A function.

This function destroys an offline model.

Parameters

- `[in] model`: Input. A pointer pointing to an offline model.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.32 cnmlDestroyTensor

cnmlStatus_t `cnmlDestroyTensor`(cnmlTensor_t *tensor)

A function.

According to the pointer given by the user, the pointer of tensor on device is freed.

This function is used to destroy the specified tenor on the mlu.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor address at the MLU end.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Pointer is null.
  - The pointer content pointed to by tensor has been freed.

Parameters

- `[in] tensor`: Input. a pointer pointing to the address to be destroyed

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.1.33 cnmlDiffFiles

cnmlStatus_t **cnmlDiffFiles**(const char *mlu_res, const char *cpu_res)

A function.

Find out the computation result difference between MLU and CPU.

Parameters

- [in] mlu_res: Input. The name of the file that stores the computation result of the MLU.
- [in] cpu_res: Input. The name of the file that stores the computation result of the CPU.

Return Value

- CNML_STATUS_INVALIDPARAM: Unable to open the file of MLU or CPU computation results.
- CNML_STATUS_OUTOFRANGE: The number of CPU and MLU results is inconsistent.
- CNML_STATUS_OVERFLOWERR: MLU results overflow, illegal.
- CNML_STATUS_SUCCESS: The function returns normally.

4.1.34 cnmlDumpTensorFromDevice

cnmlStatus_t **cnmlDumpTensorFromDevice**(cnmlTensor_t tensor, const char *filename, int opt_level)

A function.

Given a tensor, a file name, and an optimization level, the function copies the data from the specified Tensor device into the cnmldata / file in the current running directory.

This function has no return value, and if the inout tensor is a null pointer or the file name is illegal, it returns directly by throwing error.

opt_level input will not return if it is an illegal value, but a default value will be taken.

Parameters

- [in] tensor: Input. A pointer pointing to the tensor at the MLU end.
- [in] filename: Input. File name, which is used to save data in the device for this tensor.
- [in] opt_level: Input. Optimization level. An integer of 0 or 1, 0 means no optimization, 1 means computation optimization, some tensors may not be copied from the device after computation optimization.

4.1.35 cnmlDumpTensor2File

void **cnmlDumpTensor2File**(const char *filename, cnmlCpuTensor_t cpu_tensor, cnmlTensorType_t tensor_type, void *output, bool app)

A function.

This function is used to copy data of tensor at CPU end to file.

Parameters

- [in] filename: Input. A string pointer that specifies the file name.
- [in] cpu_tensor: Input. A pointer pointing to the tensor address at the CPU end, indicating which tensor data needs to be saved.
- [in] tensor_type: Input. An enumeration variable indicating the type of tensor at CPU end.
- [in] output: Input. A void* pointer pointing to the address allocated by the CPU to save data.
- [in] app: Input. A Boolean variable that specifies whether is written to a file in an additive manner.

4.1.36 cnmlDumpTensor2File_V2

cnmlStatus_t **cnmlDumpTensor2File_V2**(const char *filename, cnmlTensor_t tensor, void *output, bool app)

A function.

This function is used to copy data of tensor at CPU end to file.

Parameters

- [in] filename: Input. A string pointer that specifies the file name.
- [in] output: Input. A void* pointer pointing to the address allocated by the CPU to save data.
- [in] app: Input. A Boolean variable that specifies whether is written to a file in an additive manner.

4.1.37 cnmlFuseOp

cnmlStatus_t **cnmlFuseOp**(cnmlBaseOp_t op, cnmlFusionOp_t fusion_op)

A function.

This function is used to add base operators to fusion operators.

Parameters

- [in] op: Input. A pointer for adding base operators to fusion operators.
- [in] fusion_op: Input. A pointer pointing to fusion operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer or fusion_op is null.

4.1.38 cnmlExit

cnmlStatus_t **cnmlExit**()

A function.

This function is used to close hardware platform and LOG system.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.1.39 cnmlExtractFunctionFromFusionOp

cnmlStatus_t **cnmlExtractFunctionFromFusionOp**(cnmlFusionOp_t op, cnrtFunction_t function)

A function.

Extract Function (a data object of CNRT) from compiled fusion operators.

Parameters

- [out] function: Output. The function extracted from fusion operator
- [in] op: Input. A pointer pointing to fusion operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.40 cnmlExtractFunctionFromOp

cnmlStatus_t **cnmlExtractFunctionFromOp**(cnmlBaseOp_t op, cnrtFunction_t function)

A function.

Extract Function (a data object of CNRT) from compiled base operators.

Parameters

- [out] function: Output. The function extracted from base operator
- [in] op: Input. A pointer pointing to base operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.41 cnmlGetBaseOpRequiredStackSize

cnmlStatus_t **cnmlGetBaseOpRequiredStackSize**(cnmlBaseOp_t op, int64_t *size)

A function.

Get the size of stack space required by the operatop.

Parameters

- [out] size: Output. The size of stack space required by base operator.
- [in] op: Input. A pointer pointing to base operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends and returns normally.

4.1.42 cnmlGetConstData

cnmlStatus_t **cnmlGetConstData**(cnmlTensor_t tensor, void **cpu_ptr)

A function.

This function is used to get the const data from the binded tensor at MLU. This funciton should be used after cnmlCreateTensor API and cnmlBind-ConstData API.

Parameters

- [in] tensor: Input. A pointer pointing to the tensor at MLU end, and the interface currently supports only CNML_CONST-type Tensor.
- [in] cpu_ptr: Input. A pointer of void type pointing to the binding data address of the Tensor at CPU end.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointers of tensor, cpu_data_ptr are null.
  - Tensor type is not supported.

4.1.43 cnmlGetFusionMaxMemUsed

cnmlStatus_t **cnmlGetFusionMaxMemUsed**(cnmlFusionOp_t op, int64_t *totalmem, int64_t *sharemem, int64_t *privatemem)

A function.

This function is used to obtain the maximum memory used by the fusion operation.

Parameters

- [out] totalmem: Output. Total memory.
- [out] sharemem: Output. Sharing memory.
- [out] privatemem: Output. Private memory
- [in] op: Input. A pointer pointing to fusion operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.

4.1.44 cnmlGetFusionIOCount

cnmlStatus_t **cnmlGetFusionIOCount**(cnmlFusionOp_t op, int64_t *iocount)

A function.

This function is used to obtain the number of operators for fusion.

Parameters

- [out] iocount: Output. The number of operators for fusion.
- [in] op: Input. A pointer pointing to fusion operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.

4.1.45 cnmlGetFusionOpRequiredStackSize

cnmlStatus_t **cnmlGetFusionOpRequiredStackSize**(cnmlFusionOp_t op, int64_t *size)

A function.

This function is used to obtain the stack size required by the fusion operator operation.

Parameters

- [out] size: Output. A pointer pointing to the stack size.
- [in] op: Input. A pointer pointing to fusion operator.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
  - Output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.1.46 cnmlGetIOCount

cnmlStatus_t **cnmlGetIOCount**(cnmlBaseOp_t op, int64_t *iocount)

A function.

This function is used to obtain the space occupancy of the specified operator.

Parameters

- [out] iocount: Output. The IO number of the compiled operator.
- [in] op: Input. A pointer pointing to the base operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.1.47 cnmlGetLibVersion

cnmlStatus_t **cnmlGetLibVersion**(int *major, int *minor, int *patch)

A function.

This function is used to get the CNML library version.

Parameters

- [out] major: Output. return major version number
- [out] minor: Output. return minor version number
- [out] patch: Output. return patch version number

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are not met:
  - ver is a non-null pointer.

4.1.48 cnmlGetMaxMemUsed

cnmlStatus_t **cnmlGetMaxMemUsed**(cnmlBaseOp_t op, int64_t *totalmem, int64_t *sharemem, int64_t *privatemem)

A function.

This function is used to obtain the space occupancy of the specified operator.

Parameters

- [out] totalmem: Output. The size of the total space occupied by the operator.
- [out] sharemem: Output. The size of the shared memory occupied by the operator.
- [out] privatemem: Output. The size of the privated memory occupied by the operator.
- [in] op: Input. A pointer pointing to the base operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.1.49 cnmlGetModelSize

cnmlStatus_t **cnmlGetModelSize**(cnmlModel_t model, uint64_t *size)

A function.

Get the size of storage space needed to save the current model.

Parameters

- [out] size: Output. The size of storage space needed to save the current model.
- [in] model: Input. A pointer pointing to the current model.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.50 cnmlGetOperationName

cnmlStatus_t **cnmlGetOperationName**(cnmlBaseOp_t op, char *name)

A function.

Get the operation’ s name.

A new interface is provided to get the name of the operation.

Parameters

- [in] op: Input. A pointer to a cnmlBaseOp struct.
- [out] name: Output. A pointer to a char array allocated by user. The length of the array is get by function cnmlGetOperationNameLength. The char array only contains the valid part of the string - in other words, it is not null-terminated.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.51 cnmlGetOperationNameLength

cnmlStatus_t **cnmlGetOperationNameLength**(cnmlBaseOp_t op, int *length)

A function.

Get the length of the operation’ s name.

A new interface is provided to get the length of the name.

Parameters

- [in] op: Input. A pointer to a cnmlBaseOp struct.
- [out] length: Output. A pointer to an interger.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.52 cnmlGetTensorDataType

cnmlStatus_t **cnmlGetTensorDataType**(cnmlTensor_t tensor, cnmlDataType_t *dtype)

A function.

This function gets the data type of the input Tensor.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t

Return Value

- cnmlDataType_t: data type The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Data_typies not supported.

4.1.53 cnmlGetTensorLen

cnmlStatus_t **cnmlGetTensorLen**(cnmlTensor_t tensor, int *len)

A function.

This function gets the length of input Tensor.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t

Return Value

- int: type The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Data_typies not supported.

4.1.54 cnmlGetTensorShape

cnmlStatus_t **cnmlGetTensorShape**(cnmlTensor_t tensor, int *shape)

A function.

This function gets the shape of the input Tensor.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t.
- [in] shape: Input. A pointer pointing to an integer.

Return Value

- Null: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Data_typies not supported.

4.1.55 cnmlGetTensorSize_V2

cnmlStatus_t **cnmlGetTensorSize_V2**(cnmlTensor_t tensor, size_t *size)

A function.

This function gets the size of the space occupied by the input Tensor.

Parameters

- [in] tensor: Input. A pointer pointing to cnmlTensor_t

Return Value

- size_t: type The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Data_typies not supported.

4.1.56 cnmlGetQuantizedPosition

cnmlStatus_t **cnmlGetQuantizedPosition**(cnmlTensor_t tensor, int *position)

A function.

This function is used to obtain the position value of the tensor at MLU end when it is used for precision quantization.

Parameters

- [out] position: Output. A pointer pointing to an integer variable that saves the value of the position parameter.
- [in] tensor: Input. A pointer pointing to the tensor at the MLU end.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.57 cnmlGetQuantizedScale

cnmlStatus_t **cnmlGetQuantizedScale**(cnmlTensor_t tensor, float *scale)

A function.

This function is used to obtain the value of scale set by the tensor at MLU end for precision quantization.

Parameters

- [out] scale: Output. A pointer pointing to a floating-point variable that saves the value of the scale parameter.
- [in] tensor: Input. A pointer pointing to the tensor at the MLU end.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.58 cnmlGetQuant8Param

cnmlStatus_t **cnmlGetQuant8Param**(cnmlTensor_t tensor, float *scale, float *offset)

A function.

This function is used to obtain the scale and offset parameters required for Quant8 precision quantization of the tensor at MLU end.

Parameters

- [out] scale: Output. A pointer pointing to a floating-point variable that saves the value of the scale parameter.
- [out] float: Output. A pointer to a floating-point variable that saves the value of the offset parameter.
- [in] tensor: Input. A pointer pointing to the tensor at the MLU end.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.59 cnmlGetVersion

cnmlStatus_t **cnmlGetVersion**(unsigned int *ver)

A function.

This function is used to get the CNML version.

The cnmlGetVersion API is not recommended to use and will be deprecated in a future release. We recommend you use cnmlGetLibVersion API instead.

Parameters

- [out] ver: Output. return version number

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are not met:
  - ver is a non-null pointer.

4.1.60 cnmlInit

cnmlStatus_t **cnmlInit**(int flag)

A function.

This function is used to initialize hardware platform and LOG system.

Parameters

- [in] flag: Input. The input is integer variable. flag, the future reserved variable, is temporarily invalid.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.1.61 cnmlPrintTensor

cnmlStatus_t **cnmlPrintTensor**(cnmlTensor_t tensor, cnmlTensorType_t tensor_type)

A function.

This function is used to output the address information of the tensor at mlu end in memory.

Parameters

- [in] tensor: Input. A pointer pointing to the tensor at the MLU end.
- [in] type: Input. An enumeration variable that indicates tensor type.

4.1.62 cnmlSaveModel

cnmlStatus_t **cnmlSaveModel**(cnmlModel_t model, **const** char *fname)

A function.

Save the current model locally.

Parameters

- [in] model: Input. A pointer pointing to the current model.
- [in] fname: Input. The name of the local file.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.63 cnmlSaveModelToMem

cnmlStatus_t **cnmlSaveModelToMem**(cnmlModel_t model, void *ptr, uint64_t len, uint64_t *size)

A function.

Save the model in a specified space.

Parameters

- [out] size: Output. The actual size of the current model.
- [in] model: Input. A pointer pointing to the model
- [in] ptr: Input. A pointer pointing to storage space.
- [in] len: Input. Size of storage space.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.64 cnmlSetBaseOpCorenum

cnmlStatus_t **cnmlSetBaseOpCorenum**(cnmlBaseOp_t op, int core_num)

A function.

This function sets that the number of operation core is used at compile time according to the fusion operation pointer given by the user and calls it after the user creates the fusion operation pointer.

Parameters

- [in] op: Input. A pointer pointing to base operation.
- [in] core_num: Input. An int value, greater than or equal to 1, defaults to the number of core with maximum computation power at current platform, and each platform’ s core restrictions are different.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.65 cnmlSetBaseOpCoreVersion

cnmlStatus_t `cnmlSetBaseOpCoreVersion`(cnmlBaseOp_t op, cnmlCoreVersion_t version)

A function.

The function sets the platform version at compile time according to the fusion operation pointer given by the user, and calls it after the user creates the fusion operation pointer.

Parameters

- `[in] op`: Input. A pointer pointing to the base operation.
- `[in] version`: Input. Platform version.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.66 cnmlSetFusionOperationComputingLayout

cnmlStatus_t `cnmlSetFusionOperationComputingLayout`(cnmlFusionOp_t op, cnmlDataOrder_t layout)

4.1.67 cnmlSetFusionOpCacheMode

cnmlStatus_t `cnmlSetFusionOpCacheMode`(cnmlFusionOp_t all_cache_graph, bool cache_mode)

A function.

This function is used to set a fusion op cache mode. This function should be called before adding cache graphs and make sure the cache mode is set to `true` during compiling. Default cache mode value is false.

Parameters

- `[in] all_cache_graph`: Input. A pointer pointing to fusion operator.
- `[in] cache_mode`: Input. A bool value. Supported values are `true` and `false`. The default value is `false`. You need to set this parameter to `true` before adding basic graphs.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - all_cache_graph pointer is null.
  - set an all_cach_graph which has been set cache mode true cache mode false.

4.1.68 cnmlSetFusionThreadContext

cnmlStatus_t `cnmlSetFusionThreadContext`(bool fusion_mode)

A function.

This function is used in setting network running mode.It’ s used to solving conflict of different base op When run network with fusion many base op to fusion_op, please set fusion_mode true When run network with many base op one by one, please set fusion_mode false This func should be called before CompileOp,and should set once in one thread. When change running mode in one thread, this func should be call to change running mode

Parameters

- `[in] fusion_mode`: Input. A bool variable.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.1.69 cnmlSetFusionIO

cnmlStatus_t `cnmlSetFusionIO`(cnmlFusionOp_t op, cnmlTensor_t *inputs, int input_num, cnmlTensor_t *outputs, int output_num)

A function.

This function is used to set IO for the fusion operation.

Parameters

- `[out] outputs`: Output. A pointer pointing to a four-dimensional MLU output tensor.
- `[in] op`: Input. A pointer pointing to fusion operator.
- `[in] inputs`: Input. A pointer pointing to a four-dimensional MLU input tensor.
- `[in] input_num`: Input. The number of input.
- `[in] output_num`: Input. The number of output.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Task type is invalid at runtime.

4.1.70 cnmlSetFusionOpBatchsizeChangable

cnmlStatus_t `cnmlSetFusionOpBatchsizeChangable`(cnmlFusionOp_t op, bool changable)

A function.

This function sets whether to turn on Batchsize changable function according to the fusion operation pointer given by the user and calls it after the user creates the fusion operation pointer.

This function is not yet enabled.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operation.
- `[in] changable`: Input. A bool value, defaults to false. When it is true, batchsize is changeable.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.

4.1.71 cnmlSetFusionOpCorenum

cnmlStatus_t `cnmlSetFusionOpCorenum`(cnmlFusionOp_t op, int core_num)

A function.

This function sets that the number of operation core is used at compile time according to the fusion operation pointer given by the user and calls it after the user creates the fusion operation pointer.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operation.
- `[in] core_num`: Input. An int value, greater than or equal to 1, defaults to the number of core with maximum computation power at current platform, and each platform' s core restrictions are different.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.72 cnmlSetFusionOpCorenumChangable

cnmlStatus_t `cnmlSetFusionOpCorenumChangable`(cnmlFusionOp_t op, bool changable)

A function.

This function sets whether to turn on the number of operation core changable function according to the fusion operation pointer given by the user and calls it after the user creates the fusion operation pointer.

This function is not yet enabled.

Parameters

- `[in] op`: Input. A pointer pointing to fusion operation.
- `[in] changable`: Input. A bool value defaults to false. When it is true, the number of operation core is changeable at compile time.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.

4.1.73 cnmlSetFusionOpCoreVersion

cnmlStatus_t `cnmlSetFusionOpCoreVersion`(cnmlFusionOp_t op, cnmlCoreVersion_t version)

A function.

The function sets the platform version at compile time according to the fusion operation pointer given by the user, and calls it after the user creates the fusion operation pointer.

Parameters

- `[in] op`: Input. A pointer pointing to the fusion operation.
- `[in] version`: Input. Platform version.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
  - Platform version error.

4.1.74 cnmlSetFusionOpCoreVersionChangable

cnmlStatus_t **cnmlSetFusionOpCoreVersionChangable**(cnmlFusionOp_t op, bool changable)

A function.

The function sets whether to turn on the platform version changeable function according to the user's given fusion operation pointer, and calls it after the user creates the fusion operation pointer.

This function is not yet enabled.

Parameters

- [in] op: Input. A pointer pointing to fusion operation.
- [in] changable: Input. When it is true, the version is changeable at compile time.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.

4.1.75 cnmlSetFusionThreadContext

cnmlStatus_t **cnmlSetFusionThreadContext**(bool fusion_mode)

A function.

This function is used in setting network running mode.It's used to solving conflict of different base op When run network with fusion many base op to fusion_op, please set fusion_mode true When run network with many base op one by one, please set fusion_mode false This func should be called before CompileOp,and should set once in one thread. When change running mode in one thread, this func should be call to change running mode

Parameters

- [in] fusion_mode: Input. A bool variable.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.1.76 cnmlSetOperationComputingDataType

cnmlStatus_t **cnmlSetOperationComputingDataType**(cnmlBaseOp_t op, cnmlTensor_t tensor, cnmlDataType_t data_type, cnmlQuantizedParam_t quant_param)

A function.

Set Operation to specify the data type when Tensor participates in the computation on the chip.

A new interface is provided to set the data types and quantization parameters on the operator chip to OP separately. Tensor is opinput, output or weight tensor, off-chip data type is Tensor's own data type, and on-chip data type is input data type.

Parameters

- [in] op: Input. A pointer pointing to base oeprator.
- [in] tensor: Input. Tensor data.
- [in] data_type: Input. Data type
- [in] quant_param: Input. Decimal position.

Return Value

- CNML_STATUS_SUCCESS: The function returns normally.

4.1.77 cnmlSetOperationComputingLayout

cnmlStatus_t **cnmlSetOperationComputingLayout**(cnmlBaseOp_t op, cnmlDataOrder_t layout)

A function.

Set the layout for the operation. When running a network with many base op one by one, this interface should be called after cnmlCreateXXOp and before the first cnmlCompileBaseOp or cnmlCompileFusionOp of the entire computational diagram.

Parameters

- [in] op: Input. Pointer to a base operation
- [in] layout: Input. Pointer to a cnmlDataOrder_t

Return Value

- CNML_STATUS_SUCCESS: The function ends and returns normally.

4.1.78 cnmlSetOperationName

cnmlStatus_t `cnmlSetOperationName`(cnmlBaseOp_t op, `const` char *name)

A function.

Specify the operation’ s name.

A new interface is provided to set the name of the operation.

Parameters

- `[in] op`: Input. A pointer to a cnmlBaseOp struct.
- `[in] name`: Input. Operation name.

Return Value

- `CNML_STATUS_SUCCESS`: The function returns normally.

4.1.79 cnmlSetQuantizedAlphaByChannel

cnmlStatus_t `cnmlSetQuantizedAlphaByChannel`(cnmlTensor_t tensor, float *alphas, int alphas_size)

A function.

This function is used to set the alpha parameters of the data quantized by channel for the tensor at MLU end.

Formula

alpha = 1 / scale float_num = fix_num * 2^(position) * alpha = fix_num * 2^(position) / scale

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] alphas`: Input. A floating-point array that saves the value of the alpha parameter used for each channel quantization. If channel position is given, default alpha is 1.0.
- `[in] alphas_size`: Input. An integer variable indicating the number of channels, and the value should be consistent with Tensor’ s value in the N dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor, scales are null.

4.1.80 cnmlSetQuantizedPosition

cnmlStatus_t `cnmlSetQuantizedPosition`(cnmlTensor_t tensor, int position)

A function.

This function is used to set the position value of the tensor at MLU end when it is used for precision quantization.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] position`: Input. An integer variable that indicates the value of the position parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.81 cnmlSetQuantizedPositionByChannel

cnmlStatus_t `cnmlSetQuantizedPositionByChannel`(cnmlTensor_t tensor, int *positions, int positions_size)

A function.

This function is used to set the position parameters of the data quantized by channel for the tensor at the MLU end.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] positions`: Input. An integer array that saves the value of the position parameter used for each channel quantization. If channel scale or channel alpha is given, default value is 0.
- `[in] positions_size`: Input. An integer variable indicating the number of channels, and the value should be consistent with Tensor’ s value in the N dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor, positions are null.

4.1.82 cnmlSetQuantizedScale

cnmlStatus_t `cnmlSetQuantizedScale`(cnmlTensor_t tensor, float scale)

A function.

This function is used to set the value of scale required by the tensor at MLU end for precision quantization.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] scale`: Input. A floating-point variable that indicates the value of the scale parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.83 cnmlSetQuantizedScaleByChannel

cnmlStatus_t `cnmlSetQuantizedScaleByChannel`(cnmlTensor_t tensor, float *scales, int scales_size)

A function.

This function is used to set the scale parameters of the data quantized by channel for the tensor at MLU end.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] scales`: Input. A floating-point array that saves the value of the scale parameter used for each channel quantization. If channel position is given, default scale is 1.0.
- `[in] scales_size`: Input. An integer variable indicating the number of channels, and the value should be consistent with Tensor’ s value in the N dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor, scales are null.

4.1.84 cnmlSetQuantizedThreadContext

cnmlStatus_t `cnmlSetQuantizedThreadContext`(bool fix8_mode)

A function.

Set fix8 mode for the computation diagram under the current thread context. When running a network with many base op one by one, this interface should be called before the first cnmlCompileBaseOp or cnmlCompileFusionOp of the entire computational diagram.

Parameters

- `[in] fix8_mode`: Input. Open fix8 mode when the input is true; otherwise it will not open.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends and returns normally.

4.1.85 cnmlSetQuant8Param

cnmlStatus_t `cnmlSetQuant8Param`(cnmlTensor_t tensor, float scale, float offset)

A function.

This function is used to set the scale and offset parameters required for Quant8 precision quantization of the tensor at MLU end.

Parameters

- `[in] tensor`: Input. A pointer pointing to the tensor at the MLU end.
- `[in] scale`: Input. A floating-point variable that indicates the value of the scale parameter.
- `[in] offset`: Input. A floating-point variable that indicates the value of the offset parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.

4.1.86 cnmlSetTensorComputingLayoutInOperation

cnmlStatus_t **cnmlSetTensorComputingLayoutInOperation**(cnmlBaseOp_t op, cnmlTensor_t tensor, cnmlDataOrder_t layout)

A function.

Set the layout for the operation input. When running a network with many base op one by one, this interface should be called after cnmlCreateXXXOp and before the first cnmlCompileBaseOp or cnmlCompileFusionOp of the entire computational diagram.

Parameters

- `[in] op`: Input. Pointer to a base operation
- `[in] tensor`: Input. Pointer to a cnmlTensor_t
- `[in] layout`: Input. Pointer to a cnmlDataOrder_t

Return Value

- `CNML_STATUS_SUCCESS`: The function ends and returns normally.

4.1.87 cnmlSetTensorDataType

cnmlStatus_t **cnmlSetTensorDataType**(cnmlTensor_t tensor, cnmlDataType_t data_type)

A function.

According to the data type given by the user, the function sets the type of data when saved in MLU.According to the data type given by the user, the function sets the type of data when saved in MLU.

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlTensor_t.
- `[in] data_type`: Input. A variable indicating the Tensor data type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Data_typies not supported.

4.1.88 cnmlSetTensorName

cnmlStatus_t **cnmlSetTensorName**(cnmlTensor_t tensor, const char *name)

A function.

This function sets the name of the Tensor.

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlTensor_t
- `[in] name`: Input. The name string of tensor.

Return Value

- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_name is not supported.

4.1.89 cnmlSetTensorShape

cnmlStatus_t **cnmlSetTensorShape**(cnmlTensor_t tensor, int dim_nums, int dim_values[])

A function.

This function sets the shape of the Tensor at the MLU end according to the user-specified shape (the number of dimensions, and the specific value of each dimension).

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlTensor_t.
- `[in] dim_num`: Input. An integer variable indicating the number of dimensions.
- `[in] dim_values`: Input. An integer array storing specific value for each dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - The length of dim_num and dim_values arrays is not consistent.The length of dim_num and dim_values arrays is not consistent.

4.1.90 cnmlSetTensorShape_V2

cnmlStatus_t `cnmlSetTensorShape_V2`(cnmlTensor_t tensor, int dim_nums, int dim_values[], int dim_strides[])

A function.

This function sets the shape of the Tensor at the MLU end according to the user-specified shape (the number of dimensions, and the specific value of each dimension).

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlTensor_t.
- `[in] dim_num`: Input. An integer variable indicating the number of dimensions.
- `[in] dim_values`: Input. An integer array storing specific value for each dimension.
- `[in] dim_strides`: Input. An integer array storing specific value for each dimension’ s stride.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - The length of dim_num and dim_values arrays is not consistent.The length of dim_num and dim_values arrays is not consistent.

4.1.91 cnmlSetTensorType

cnmlStatus_t `cnmlSetTensorType`(cnmlTensor_t tensor, cnmlTensorType_t tensor_type)

A function.

This function sets the tensor type of the Tensor.

Parameters

- `[in] tensor`: Input. A pointer pointing to cnmlTensor_t

Return Value

- `cnmlTensorType_t`: tensor type The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer of tensor is null.
  - Tensor_typies not supported.

4.1.92 getPositionScaleByDataType

cnmlStatus_t `getPositionScaleByDataType`(float *buffer, int size, int *position, float *scale, cnmlDataType_t dataType)

A function.

Get position and scale by datatype

Parameters

- `[in] buffer`: Input. Header pointer of data array.
- `[in] size`: Input. Array length
- `[in] dataType`: Input. quantized dataType INT8/INT16
- `[out] position`: Output. pointer of quantized position.
- `[out] scale`: Output. pointer of quantized scale

4.2 Abs Operation

4.2.1 cnmlCreateAbsOp

cnmlStatus_t `cnmlCreateAbsOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateAbsOpForward is recommended to use.

According to the base operator pointer given by the user, create an absolute value operator.

Then creates a pointer to the base operator address, input output tensor, and introduce them into the function to create an absolute value operator.

The shapes of Input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

DataType

MLU270:

input_type : unlimited.

output_type : unlimited.

in_oc_type : float16/float32.

output_oc_type : the same as in_oc_type.

Parameters

- `[out] op`: Output. A pointer to the base operator address.

- [in] input_tensor: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The input pointer is null.
  - Reason3 The output pointer is null.

4.2.2 cnmlCreateAbsOpForward

This API has the same function as cnmlCreateAbsOp. For detailed information, see cnmlCreateAbsOp.

4.2.3 cnmlComputeAbsOpForward_V3

cnmlStatus_t **cnmlComputeAbsOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

For computing the user-specified absolute value operator on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeAbsOpForward_V4 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address pointing to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.2.4 cnmlComputeAbsOpForward_V4

cnmlStatus_t **cnmlComputeAbsOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified absolute value operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.3 Active Operation

4.3.1 cnmlCreateActiveOp

cnmlStatus_t `cnmlCreateActiveOp`(cnmlBaseOp_t *op, cnmlActiveFunction_t active_func, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Description

This function creates an activation operator according to the basic operator pointer given by the user.

After a pointer to the base operator, activation operator, input and output tensor are created, they are introduced into the function to create the activation operator.

The operator implements activation operation, `Out(i, j, k, l) = function ( In(i, j, k, l) )` The shapes of input and output should be exactly the same.

Formula

INVALID: out[n c h w] = in[n c h w];

SIGMOID: out[n c h w] = 1.0 / (1.0 + exp(0.0 - in[n c h w]));

TANH:

a = exp(in[n c h w]);

b = exp(0.0 - in[n c h w]);

out[n c h w] = (a - b) / (a + b);

RELU : out[n c h w] = (in[n c h w] < 0) ? 0 : in[n c h w];

RELU1: out[n c h w] = (in[n c h w] < -1) ? -1 : ((in[n c h w] < 1) ? in[n c h w] : 1);

RELU6: out[n c h w] = (in[n c h w] < 0) ? 0 : ((in[n c h w] < 6) ? in[n c h w] : 6);

HARD_SIGMOID:

out[n c h w] = (in[n c h w] < -2.5) ? 0 : ((in[n c h w] < 2.5) ? (0.2 * in[n c h w] + 0.5) : 1);

Datatype

MLU270:

-input_data_type: int8, int16, float16, float32

-compute_data_type: float16, float32

-output: int8, int16, float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The value of C dimension is a multiple of 128.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] function`: Input. An enumeration variable, the following can be selected:
  - `CNML_ACTIVE_NONE` represents identity.
  - `CNML_ACTIVE_SIGMOID`
  - `CNML_ACTIVE_RELU`
  - `CNML_ACTIVE_TANH`
  - `CNML_ACTIVE_RELU1`
  - `CNML_ACTIVE_RELU6`
- `[in] input_tensor`: Input. A 1 to n-dimensional tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid. For more information, see “Error Codes” section in this guide.

4.3.2 cnmlComputeActiveOpForward_V3

cnmlStatus_t `cnmlComputeActiveOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computing the activation operator specified by the user on the MLU. Deprecated. This interface will be deleted in next version and cnmlComputeActiveOpForward_V4 is recommended to use. After the activation operator, input, output and computational stream are created, they are introduced into the function to compute the activation operator.

Formula

INVALID: out[n c h w] = in[n c h w]; SIGMOID: out[n c h w] = 1.0 / (1.0 + exp(0.0 - in[n c h w])); TANH: float a = exp(in[n c h w]); float b = exp(0.0 - in[n c h w]); out[n c h w] = (a - b) / (a + b); RELU : out[n c h w] = (in[n c h w] < 0) ? 0 : in[n c h w]; RELU1: out[n c h w] = (in[n c h w] < -1) ? -1 : ((in[n c h w] < 1) ? in[n c h w] : 1); RELU6: out[n c h w] = (in[n c h w] < 0) ? 0 : ((in[n c h w] < 6) ? in[n c h w] : 6); HARD_SIGMOID: out[n c h w] = (in[n c h w] < -2.5) ? 0 : ((in[n c h w] < 2.5) ? (0.2 * in[n c h w] + 0.5) : 1);

DataType: MLU270: if RELU : input: int8, int16, float16, float32 compute: float16, float32 output: int8, int16, float16, float32 else : input_dt = output_dt float16, float32 Scale limitation: MLU270: Unlimited

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] output`: Output. An MLU address that specifies the position of the output.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.3.3 cnmlComputeActiveOpForward_V4

cnmlStatus_t `cnmlComputeActiveOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Computing the activation operator specified by the user on the MLU.

After the activation operator, input, output and computational queue are created, they are introduced into the function to compute the activation operator.

Formula

INVALID: out[n c h w] = in[n c h w];

SIGMOID: out[n c h w] = 1.0 / (1.0 + exp(0.0 - in[n c h w]));

TANH:

a = exp(in[n c h w]);

b = exp(0.0 - in[n c h w]);

out[n c h w] = (a - b) / (a + b);

RELU : out[n c h w] = (in[n c h w] < 0) ? 0 : in[n c h w];

RELU1: out[n c h w] = (in[n c h w] < -1) ? -1 : ((in[n c h w] < 1) ? in[n c h w] : 1);

RELU6: out[n c h w] = (in[n c h w] < 0) ? 0 : ((in[n c h w] < 6) ? in[n c h w] : 6);

HARD_SIGMOID: out[n c h w] = (in[n c h w] < -2.5) ? 0 : ((in[n c h w] < 2.5) ? (0.2 * in[n c h w] + 0.5) : 1);

DataType

MLU270:

- input_data_type: int8, int16, float16, float32
- compute_data_type: float16, float32
- output: int8, int16, float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70

Performance Optimization

The value of C dimension is a multiple of 128.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid. See “Error Codes” for more information.

4.4 Add Operation

4.4.1 cnmlCreateAddOp

cnmlStatus_t **cnmlCreateAddOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an add operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the add operator, pass them into the function to create the add operator.

Before creating the add operator, declare a pointer pointing to the struct address of operation parameters of the add operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Perform element-wise summation on the two inputs to obtain output.

The shapes of two inputs and one output should be exactly the same.

Formula

c[n c h w] = a[n c h w] + b[n c h w]

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.4.2 cnmlComputeAddOpForward_V3

cnmlStatus_t `cnmlComputeAddOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the add operator specified by users on the MLU. Deprecated. This interface will be deleted in next version and cnmlComputeAddOpForward_V4 is recommended to use.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the add operator.

Formula

c[n c h w] = a[n c h w] + b[n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.4.3 cnmlComputeAddOpForward_V4

cnmlStatus_t `cnmlComputeAddOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the add operator specified by users on the MLU.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the add operator.

Formula

c[n c h w] = a[n c h w] + b[n c h w]

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor_1`: Input. First input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. First MLU address pointing to input1 data.
- `[in] input_tensor_2`: Input. Second input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. Second MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.

- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.5 Add Pad Operation

4.5.1 cnmlCreateAddPadOpParam

cnmlStatus_t `cnmlCreateAddPadOpParam`(cnmlAddPadOpParam_t *param, int pad_h, int pad_w, float pad_value)

A function.

Description

According to the pointer given by the user, the function creates an AddPad operation parameter struct and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of AddPad operator operation parameter.
- `[in] pad_h`: Input. Pad length.
- `[in] pad_w`: Input. Pad width.
- `[in] pad_value`: Input. The data filled into input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.5.2 cnmlCreateAddPadOpParam_V2

cnmlStatus_t `cnmlCreateAddPadOpParam_V2`(cnmlAddPadOpParam_t *param, int pad_htop, int pad_hbottom, int pad_wleft, int pad_wright, float pad_value)

A function.

According to the pointer given by the user, the function creates an AddPad operation parameter struct and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of AddPad operator operation parameter.
- `[in] pad_htop`: Input. Above pad length.
- `[in] pad_hbottom`: Input. Below padmlength.
- `[in] pad_wleft`: Input. Left pad width.
- `[in] pad_wright`: Input. Right pad width.
- `[in] pad_value`: Input. The data filled into input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer. For more information, see “Error Codes” section in this guide.

4.5.3 cnmlDestroyAddPadOpParam

cnmlStatus_t `cnmlDestroyAddPadOpParam`(cnmlAddPadOpParam_t *param)

A function.

Description

According to the pointer given by the user, the struct pointer of AddPad operator operation parameter is freed.

After the operation of the AddPad operator is finished, the created struct pointer of AddPad operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of AddPad operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

- The content of the pointer pointed to by param has been freed. For more information, see “Error Codes” section in this guide.

4.5.4 cnmlCreateAddPadOp

cnmlStatus_t `cnmlCreateAddPadOp`(cnmlBaseOp_t *op, cnmlAddPadOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Description

According to the basic Operator pointer given by the user, an AddPad operator is created.

The addpad operation in the h, w direction can be used to fill the input h, w direction to a preset size.

Datatype

MLU270:

float16, float32

Scale Limitation

MLU270:

hi < 2^16

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The C dimension is a multiple of 128.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer pointing to the struct of AddPad operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi],supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni),supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null. For more information, see “Error Codes” section in this guide.

4.5.5 cnmlComputeAddPadOpForward_V3

cnmlStatus_t `cnmlComputeAddPadOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Description

Computing user-specified AddPad operator on MLU.

After the AddPad operator, Input. output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the AddPad operator.

Datatype

MLU270:

float16, float32

Scale Limitation

MLU270:

hi < 2^16

Deprecated. This interface will be deleted in next version and cnmlComputeAddPadOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null. For more information, see “Error Codes” section in this guide.

4.5.6 cnmlComputeAddPadOpForward_V4

cnmlStatus_t `cnmlComputeAddPadOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Computing user-specified AddPad operator on MLU.

After the AddPad operator, Input. output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the AddPad operator.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

hi < 2^16

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The value of C dimension is a multiple of 128.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid. For more information, see “Error Codes” section in this guide.

4.6 Add Pad Channel Operation

4.6.1 cnmlCreateAddPadChannelOpParam

cnmlStatus_t `cnmlCreateAddPadChannelOpParam`(cnmlAddPadChannelOpParam_t *param, int channel_, float pad_value)

A function.

According to the pointer given by the user, the function creates a struct of AddPadChannel operator operation parameter, and fills in the struct with parameters input by the user.

Supports MLU220 and MLU270.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of the AddPadChannel operator parameter.
- `[in] channel_`: Input. The size of filling in C direction.
- `[in] pad_value`: Input. The data filled into input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.6.2 cnmlCreateAddPadChannelOpParam_V2

cnmlStatus_t `cnmlCreateAddPadChannelOpParam_V2`(cnmlAddPadChannelOpParam_t *param, int c_front_, int c_back_, float pad_value)

A function.

According to the pointer given by the user, the function creates a struct of AddPadChannel operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220 and MLU270.

Parameters

- `[out] param`: Output. A pointer pointing address of struct of the AddPadChannel operator operation parameter.
- `[in] c_front_`: Input. The length of lower filling in C direction.
- `[in] c_back_`: Input. The length of upper filling in C direction.
- `[in] pad_value`: Input. The data filled into input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.6.3 cnmlDestroyAddPadChannelOpParam

cnmlStatus_t `cnmlDestroyAddPadChannelOpParam`(cnmlAddPadChannelOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of AddPadChannel operator operation parameter is freed.

After the operation of the AddPadChannel operator is completed, the created struct pointer of AddPadChannel operator operation parameter is freed.

Supports MLU220 and MLU270.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the AddPadChannel operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.6.4 cnmlCreateAddPadChannelOp

cnmlStatus_t `cnmlCreateAddPadChannelOp`(cnmlBaseOp_t *op, cnmlAddPadChannelOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the basic operator pointer given by the user, an AddPad Channel operator is created.

The addpad operation of feature direction can be used to fill the c direction of input to a preset size.

Related APIs

After calling this API, you may need to call the following APIs:

`cnmlFuseOp` or `cnmlComputeAddPadChannelOpForward`, and `cnmlDestroyBaseOp`.

DataType

MLU270:

- input: float16, float32
- output: float16, float32

MLU220:

- input: float16, float32
- output: float16, float32

Scale Limitation

MLU270:

Unlimited.

MLU220:

Unlimited.

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A struct pointer of AddPadChannel operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi].

- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.6.5 cnmlComputeAddPadChannelOpForward_V3

cnmlStatus_t `cnmlComputeAddPadChannelOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computing the AddPadChannel operator specified by the user on MLU. Deprecated. This interface will be deleted in next version and cnmlComputeAddPadChannelOpForward_v4 is recommended to use.

After the AddPadChannel operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the AddPadChannel operator.

Supports MLU220 and MLU270.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.6.6 cnmlComputeAddPadChannelOpForward_V4

cnmlStatus_t `cnmlComputeAddPadChannelOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing the AddPadChannel operator specified by the user on MLU.

After the AddPadChannel operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the AddPadChannel operator.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.7 And Operation

4.7.1 cnmlCreateAndOp

cnmlStatus_t `cnmlCreateAndOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Perform element-wise And operation on the two Tensors.

The shapes of two inputs and one output should be exactly the same.

Formula

c[n c h w] = (a[n c h w] !=0 && (b[n c h w] != 0)

DataType

MLU270:

float16, float32, bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.7.2 cnmlComputeAndOpForward_V3

cnmlStatus_t `cnmlComputeAndOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the And operator specified by users on the MLU.

After creating an And operator, input, output, and computation stream, pass them into the function to compute the And operator.

Formula

c[n c h w] = (a[n c h w] !=0 && (b[n c h w] != 0)

DataType

MLU270:

float16, float32, bool

Scale Limitation

MLU270:

Unlimited

Deprecated. This interface will be deleted in next version and cnmlComputeAndOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.7.3 cnmlComputeAndOpForward_V4

cnmlStatus_t `cnmlComputeAndOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the And operator specified by users on the MLU.

After creating an And operator, input, output, and computation stream, pass them into the function to compute the And operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. First input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. First MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Second input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. Second MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.8 Argmax Operation

4.8.1 cnmlCreateArgmaxOp

cnmlStatus_t `cnmlCreateArgmaxOp`(cnmlBaseOp_t *op, cnmlDimension_t argmax_mode, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

This function creates an argmax operator based on the base operator pointer given by the user.

The operator contains input, axis and output. The coordinates of the maximum value are found in the dimension selected by the axis. For the plurality of numbers with the same value in the cpu, the coordinates are arranged in ascending order, and the mlu is arranged in descending order of coordinates.

Support four-dimensional tensor. The scale restrictions include:

if axis == D_N, output has the shape of [1,c,h,w];

if axis == D_C, output has the shape of [n,1,h,w];

if axis == D_H, output has the shape of [n,c,1,w];

if axis == D_W, output has the shape of [n,c,h,1].

Summary

input[n, c, h, w],and compute the index of max value of given direction.

such as direction is n, output[1, c, h, w]

such as direction is c, output[n, 1, h, w]

…

DataType

MLU270:

value_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] output_tensor`: Input. A four-dimensional tensor, of which the shape is [batch,depth,height,width], supporting data of type float16.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, of which the shape is [batch,depth,height,width], supporting data of type float16.
- `[in] argmax_mode`: Input. An enumerator representing the dimension in which the Argmax is calculated. It's value can be taken from these: CNML_ARGMAX_AXIS_N = 0, CNML_ARGMAX_AXIS_C = 1, CNML_ARGMAX_AXIS_H = 2, CNML_ARGMAX_AXIS_W = 3.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - The operator pointer is empty.
  - The input and output is empty.

4.8.2 cnmlCreateNdArgmaxOp

cnmlStatus_t `cnmlCreateNdArgmaxOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateNdArgmaxOp.

This function creates an argmax operator based on the base operator pointer given by the user.

The operator contains input, axis and output. The coordinates of the maximum value are found in the dimension selected by the axis. For the plurality of numbers with the same value in the cpu, the coordinates are arranged in ascending order, and the mlu is arranged in descending order of coordinates.

Support non-four-dimensional and four-dimensional tensor.The scale restrictions include:

if axis == D_N, output has the shape of [1,c,h,w];

if axis == D_C, output has the shape of [n,1,h,w];

if axis == D_H, output has the shape of [n,c,1,w];

if axis == D_W, output has the shape of [n,c,h,1].

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] output_tensor`: Output. A n-dimensional MLU tensor, supporting data of type float16.
- `[in] input_tensor`: Input. A n-dimensional MLU tensor, supporting data of type float16.
- `[in] dim`: Input. specify different dimension directions to compute argmax, where all of four dimensions N, C, H, W can be specified.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - The operator pointer is empty.
  - The input and output is empty.

4.8.3 cnmlComputeArgmaxOpForward_V3

cnmlStatus_t `cnmlComputeArgmaxOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

It is used to compute the user-specified Argmax operator on the MLU.

After creating the argmax operator, related parameters and computation stream, pass them to the function to It is used to compute the Argmax operator.

Summary

input[n, c, h, w],and compute the index of max value of given direction.

such as direction is n, output[1, c, h, w]

such as direction is c, output[n, 1, h, w]

…

DataType

MLU270:

value_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Deprecated. This interface will be deleted in next version and cnmlComputeArgmaxOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address that points to the output data.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data tensor.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.8.4 cnmlComputeArgmaxOpForward_V4

cnmlStatus_t **cnmlComputeArgmaxOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

It is used to compute the user-specified Argmax operator on the MLU.

After creating the argmax operator, related parameters and computation stream, pass them to the function to It is used to compute the Argmax operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.8.5 cnmlComputeNdArgmaxOpForward

cnmlStatus_t **cnmlComputeNdArgmaxOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNdArgmaxOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeNdArgmaxOpForward_V2 is recommended to use.

Compute an argmax function that supports four or more dimensions tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output data.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data tensor.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.8.6 cnmlComputeNdArgmaxOpForward_V2

cnmlStatus_t `cnmlComputeNdArgmaxOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNdArgmaxOpForward.

Compute an argmax function that supports four or more dimensions tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.9 Argmin Operation

4.9.1 cnmlCreateArgminOp

cnmlStatus_t `cnmlCreateArgminOp`(cnmlBaseOp_t *op, cnmlDimension_t argmin_mode, cnmlTensor_t input, cnmlTensor_t output)

A function.

This function creates an argmin operator based on the base operator pointer given by the user.

The operator contains input, axis and output. The coordinates of the minimum value are found in the dimension selected by the axis. For the plurality of numbers with the same value in the cpu, the coordinates are arranged in ascending order, and the mlu is arranged in descending order of coordinates.

Support four-dimensional tensor. The scale restrictions include:

if axis == CNML_DIM_N, output has the shape of [1,c,h,w];

if axis == CNML_DIM_C, output has the shape of [n,1,h,w];

if axis == CNML_DIM_H, output has the shape of [n,c,1,w];

if axis == CNML_DIM_W, output has the shape of [n,c,h,1].

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] output_tensor`: Input. A four-dimensional tensor, of which the shape is [batch,depth,height,width], supporting data of type float16.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, of which the shape is [batch,depth,height,width], supporting data of type float16.
- `[in] argmin_mode`: Input. An enumerator representing the dimension in which the Argmin is calculated. It's value can be taken from these: CNML_DIM_N = 0, CNML_DIM_C = 1, CNML_DIM_H = 2, CNML_DIM_W = 3

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - The operator pointer is empty.
  - The input and output is empty.

4.9.2 cnmlComputeArgminOpForward_V3

cnmlStatus_t `cnmlComputeArgminOpForward_V3`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

It is used to compute the user-specified Argmin operator on the MLU.

After creating the Argmin operator, related parameters and computation stream, pass them to the function to It is used to compute the Argmin operator.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.10 AsStride Operation

4.10.1 cnmlCreateAsStrideOp

cnmlStatus_t `cnmlCreateAsStrideOp`(cnmlBaseOp_t *op, cnmlTensor_t input, cnmlTensor_t output, int k_value)

cnmlCreateAsStrideOp.

DataType:

MLU270: float16, float32

Creates an AsStride operator that resizes the channel dimension.

Before creating an AsStride operator, you need to declare a pointer pointing to the AsStride operator you will create later.

The shapes of the input and output tensor should be consistent.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the AsStride operator you have created.
- `[in] input`: Input. A 4-D MLU input tensor. The shape of the tensor is [batch, channel, height, width] The data type of this tensor descriptor must be float16. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.
- `[in] output`: Input. The descriptor of the 4-D MLU tensor. The shape of the tensor is [batch, channel, height, width]. The data type of this tensor descriptor must be float16. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.
- `[in] k_value`: Input. The size of the output channel. k_value is equal to the output channel size. Supported values are integers from 1 to 60,000.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions are met:
  - The operator pointer is NULL
  - The input pointer is NULL.
  - The output pointer is NULL.

4.10.2 cnmlComputeAsStrideOpForward_V3

cnmlStatus_t `cnmlComputeAsStrideOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeAsStrideOpForward_V3.

Resizes the channel dimension on MLU.

The cnmlComputeAsStrideOpForward_V2 API is not recommended to use and will be deprecated in a future release. We recommend you use cnmlComputeAsStrideOpForward_V3 API instead.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. A pointer to output data after the AsStride operator is applied.
- `[in] op`: Input. A pointer to the AsStride operator you have created.
- `[in] input`: Input. A pointer to the input data you want to compute.
- `[in] compute_forw_param`: Input. A pointer to the struct address that records the data parallelism and device affinity for runtime.
- `[in] queue`: Input. A pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions are met:
  - The operator pointer is NULL.
  - The output pointer is NULL.

4.10.3 cnmlComputeAsStrideOpForward_V4

cnmlStatus_t `cnmlComputeAsStrideOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeAsStrideOpForward_V4.

It is used to compute the user-specified selu activation function operator on the MLU.

After creating the selu activation function operator, Input, Output, runtime parameters, computation queue, pass them to the function to It is used to compute the selu activation function operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.11 Avg Operation

4.11.1 cnmlCreateAvgOp

cnmlStatus_t `cnmlCreateAvgOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an avg operator.

After creating a pointer to the base operator address and operator input output tensor, introduce them into the function to create an avg operator. The operator can be used to find the averages on the H and W dimensions, including an input and output.

Can only be used on the H or W dimension.

Deprecated.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[out] output`: Output. A four-dimensional MLU input tensor, the shape is [1, 1, 1, 1], supports data of float16 type.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape is [1, hi, wi, 1], supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

- Input tensor type is not CNML_TENSOR.

4.11.2 cnmlComputeAvgOpForward_V3

cnmlStatus_t `cnmlComputeAvgOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computethe user-specified Avg operator.

After creating Avg operator, input, output and computation stream, introduce them to the function to compute the Avg operator.

Deprecated.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Output. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.

4.11.3 cnmlComputeAvgOpForward_V4

cnmlStatus_t `cnmlComputeAvgOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computethe user-specified Avg operator.

After creating Avg operator, input, output and computation queue, introduce them to the function to compute the Avg operator.

Deprecated.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.12 Ax Opreation

4.12.1 cnmlCreateAxOp

cnmlStatus_t `cnmlCreateAxOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_A, cnmlTensor_t input_tensor_X, cnmlTensor_t output_tensor)

cnmlCreateAxOp.

Create an Ax operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create an Ax operator.

This function has the functionality of output = A Ã—X.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 65500

float32: < 16200

NHW direction: unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_A`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] input_tensor_X`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.12.2 cnmlComputeAxOpForward_V3

cnmlStatus_t `cnmlComputeAxOpForward_V3`(cnmlBaseOp_t op, void *input_A, void *input_X, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeAxOpForward_V3.

It is used to compute the user-specified Ax operator on the MLU.

After creating the Ax operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Ax operator.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 65500

float32: < 32750

NHW direction: unlimited

Deprecated. This interface will be deleted in next version and cnmlComputeAxOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input_A`: Input. An MLU address pointing to the input data tensor A.
- `[in] input_X`: Input. An MLU address pointing to the input data tensor X.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.12.3 cnmlComputeAxOpForward_V4

cnmlStatus_t `cnmlComputeAxOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *input_A, cnmlTensor_t input_tensor_X, void *input_X, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeAxOpForward_V4.

It is used to compute the user-specified Ax operator on the MLU.

After creating the Ax operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Ax operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensori_A`: Input. First input MLU tensor pointer. Pass NULL if not used.
- `[in] input_A`: Input. First MLU address pointing to input_A data.
- `[in] input_tensor_X`: Input. Second input MLU tensor pointer. Pass NULL if not used.
- `[in] input_X`: Input. Second MLU address pointing to input_X data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.13 AxpBy Opreation

4.13.1 cnmlCreateAxpByOp

cnmlStatus_t `cnmlCreateAxpByOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_A, cnmlTensor_t input_tensor_X, cnmlTensor_t input_tensor_B, cnmlTensor_t input_tensor_Y, cnmlTensor_t output_tensor)

cnmlCreateAxpByOp.

Create an AxpBy operator based on the base operator pointer given by user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create an AxpBy operator.

This function has the functionality of output = A * X + B * Y.

The input input_tensor_X and input_tensor_Y must have the same shape.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w] + B [n c 1 1] * Y [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 32500

float32: < 16200

NHW direction: unlimited

Parameters

- `[out] op`: Output. a pointer to the base operator address.
- `[in] input_tensor_A`: Input. a 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] input_tensor_X`: Input. a 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] input_tensor_B`: Input. a 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] input_tensor_Y`: Input. a 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output`: Input. a 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.13.2 cnmlComputeAxpByOpForward_V3

cnmlStatus_t `cnmlComputeAxpByOpForward_V3`(cnmlBaseOp_t op, void *input_A, void *input_X, void *input_B, void *input_Y, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeAxpByOpForward_V3.

It is used to compute the user-specified AxpBy operator on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeAxpByOpForward_V4 is recommended to use.

After creating the AxpBy operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the AxpBy operator.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w] + B [n c 1 1] * Y [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 32500

float32: < 16200

NHW direction: unlimited

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input_A`: Input. An MLU address pointing to the input data tensor A.
- `[in] input_X`: Input. An MLU address pointing to the input data tensor X.
- `[in] input_B`: Input. An MLU address pointing to the input data tensor B.
- `[in] input_Y`: Input. An MLU address pointing to the input data tensor Y.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.13.3 cnmlComputeAxpByOpForward_V4

cnmlStatus_t `cnmlComputeAxpByOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *input_A, cnmlTensor_t input_tensor_X, void *input_X, cnmlTensor_t input_tensor_B, void *input_B, cnmlTensor_t input_tensor_Y, void *input_Y, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeAxpByOpForward_V4.

It is used to compute the user-specified AxpBy operator on the MLU.

After creating the AxpBy operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the AxpBy operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor_A`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_A`: Input. MLU address pointing to input_A data.
- `[in] input_tensor_X`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input__X`: Input. MLU address pointing to input_X data.
- `[in] input_tensor_B`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_B`: Input. MLU address pointing to input_B data.
- `[in] input_tensor_Y`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_B`: Input. MLU address pointing to input_Y data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.14 Axpy Operation

4.14.1 cnmlCreateAxpyOp

cnmlStatus_t **`cnmlCreateAxpyOp`**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_A, cnmlTensor_t input_tensor_X, cnmlTensor_t input_tensor_Y, cnmlTensor_t output_tensor)

cnmlCreateAxpyOp.

Create an Axpy operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create an Axpy operator.

This function has the functionality of output = A Ã—X + Y.

The input input_tensor_X and input_tensor_Y must have the same shape.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w] + Y [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 43500

NHW direction: unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_A`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] input_tensor_X`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] input_tensor_Y`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.14.2 cnmlComputeAxpyOpForward_V3

cnmlStatus_t **`cnmlComputeAxpyOpForward_V3`**(cnmlBaseOp_t op, void *input_A, void *input_X, void *input_Y, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeAxpyOpForward_V3.

It is used to compute the user-specified Axpy operator on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeAxpyOpForward_V4 is recommended to use.

After creating the Axpy operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Axpy operator.

Formula

output X [n c h w] = A [n c 1 1] * X [n c h w] + Y [n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

C direction:

float16: < 43500

float32: < 21792

NHW direction: unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input_A`: Input. An MLU address pointing to the input data tensor A.
- `[in] input_X`: Input. An MLU address pointing to the input data tensor X.
- `[in] input_Y`: Input. An MLU address pointing to the input data tensor Y.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.14.3 cnmlComputeAxpyOpForward_V4

cnmlStatus_t `cnmlComputeAxpyOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *input_A, cnmlTensor_t input_tensor_X, void *input_X, cnmlTensor_t input_tensor_Y, void *input_Y, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeAxpyOpForward_V4.

It is used to compute the user-specified Axpy operator on the MLU.

After creating the Axpy operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Axpy operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor_A`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_A`: Input. MLU address pointing to input_A data.
- `[in] input_tensor_X`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_X`: Input. MLU address pointing to input_X data.
- `[in] input_tensor_Y`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_Y`: Input. MLU address pointing to input_Y data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.15 Basic Div Operation

4.15.1 cnmlCreateBasicDivOp

cnmlStatus_t `cnmlCreateBasicDivOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

The function Create basic division operator according to base operator pointers given by users on the MLU.

Formula

output[n c h w] = 1 / input[n c h w]

DataType

MLU270:

float16, float32

MLU220:

float16, float32

Scale Limitation

MLU270:

Unlimited

MLU220:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi].
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo] (no = ni, co = ci, ho = hi, wi = wo).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.15.2 cnmlComputeBasicDivOpForward

cnmlStatus_t `cnmlComputeBasicDivOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute basic division operator on the MLU.

After creating a basic division operator, input, output, computation type, and computation stream, pass them into the function to compute the basic division operator.

Formula

output[n c h w] = 1 / input[n c h w]

DataType

MLU270:

float16, float32

MLU220:

float16, float32

Scale Limitation

MLU270:

Unlimited

MLU220:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi].
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo] (no = ni, co = ci, ho = hi, wi = wo).
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM:`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.15.3 cnmlSetBasicDivHighPrecision

cnmlStatus_t `cnmlSetBasicDivHighPrecision`(cnmlBaseOp_t *op, bool high_precision_flag)

A function.

Set a high-precision mode according to the basic division operator given by users.

After creating a division operator, calling this interface will enable the high-precision mode.

If the high precision mode is set, the operator will support negative input and the precision will be improved to 1E-20.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] high_precision_flag`: Input. Set the operator as a high-precision identifier. The value "true" represents the mode is set to the high-precision mode.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM:`: At least one of the following conditions are met:
  - op is a null pointer.

4.16 Basic RNN Pro Operation

4.16.1 cnmlCreateRNNOpParam

cnmlStatus_t `cnmlCreateRNNOpParam`(cnmlRNNOpParam_t *param, bool is_bidirection, int batch_size, int vector_size, int hidden_size, int num_layers, cnmlRNNInputMode_t input_mode, int max_seqlength, int seqlength_array[])

A function.

According to the pointer given by the user, this function creates a struct of RNN operator operation parameters, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to another pointer, and the pointed pointer points to a struct describing the operation parameters of RNN operators.
- `[in] is_bidirection`: Input. When the input is true, the operator is bidirectional RNN; when the input is false, the operator is unidirectional RNN. For the time being, bidirectional RNN is not supported.
- `[in] batch_size`: Input. The number of sequences within the mini-batch.
- `[in] vector_size`: Input. The vector length in input and output tensor for each time step (that is, embedding size).
- `[in] hidden_size`: Input. The vector length in hidden layer state.
- `[in] num_layers`: Input. The number of layer of RNN.
- `[in] input_mode`: Input. When num_layers are greater than 1 and vector_size is not equal to hidden_size, input_mode must be RNN_LINEAR_INPUT.
- `[in] max_seqlength`: Input. The maximum seqlength of mini-batch.
- `[in] seqlength_array`: Input. The integer array with batchsize elements, and each element is seqlength.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.16.2 cnmlDestroyRNNOpParam

cnmlStatus_t `cnmlDestroyRNNOpParam`(cnmlRNNOpParam_t *param)

A function.

According to the pointer given by the user, the function creates a struct of RNN operator operation parameters to free space.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the struct of RNN operator operation parameters.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.16.3 cnmlCreateBasicRNNProOp

cnmlStatus_t `cnmlCreateBasicRNNProOp`(cnmlBaseOp_t *op, cnmlRNNOpParam_t param, cnmlTensor_t input, cnmlTensor_t state_input[], cnmlTensor_t output, cnmlTensor_t state_output[], cnmlTensor_t input_weight[], cnmlTensor_t state_weight[], cnmlTensor_t input_bias[], cnmlTensor_t state_bias[], cnmlActiveFunction_t active_func)

A function.

This function creates a basic RNN operator according to the pointer of base operator given by users. Its formula is:

ht = func(w1xt + w2ht-1 + b1 + b2).

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filter_type - filter_oc_type- out_oc_type - out_type

Supported combinations are:

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

Unlimited

If all parameters in the interface are arrays, the length of all arrays is num_layers, and the element with subscript 0 in arrays represents the initial value of hidden state of the first layer, the element with subscript 1 represents the second layer, and so on. The activation function func can be any activation function supported in CNML.

Supports MLU220 and MLU270

Parameters

- `[out] output`: Output. Tensor, the output data of RNN. This Tensor is a TNC or NTC type.

- [out] state_output: Output. The tensor array is the final result of the hidden state of RNN after T computations. This Tensor is a NCHW type with shape of [batchSize, hiddenSize, 1, 1].
- [in] param: Input. A pointer, pointing to the struct describing RNN operation parameters.
- [in] input: Input. Tensor, the input data of RNN. This Tensor is a TNC or NTC type.
- [in] state_input: Input. The Tensor array is the initial value of RNN’ s hidden state. And the length of arrays is num_layers. This Tensor is a NCHW type with shape of [batchSize, vectorSize, 1,1].
- [in] input_weight: Input. Tensor array is the weight of the matrix multiplied by the input data of RNN.
- [in] state_weight: Input. Tensor array is the weight of the matrix multiplied by the hidden state of RNN.
- [in] active_func: Input. Input specifies which activation function to be used. It can be any activation function that CNML already supports.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.16.4 cnmlComputeBasicRNNProOpForward

cnmlStatus_t **cnmlComputeBasicRNNProOpForward**(cnmlBaseOp_t op, void *input, void *state_input[], void *output, void *state_output[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

This interface is used to compute basic RNN on CPU;if all parameters in the interface are array type, all arrays include num_layers elements, and the element with subscript 0 in arrays corresponds to the first layer, the element with subscript 1 corresponds to the second layer, and so on.

Deprecated. This interface will be deleted in next version and cnmlComputeBasicRNNProOpForward_V2 is recommended to use.

Supports MLU220 and MLU270.

Parameters

- [out] output_tensor: Output. A 3-dimensional output tensor of CPU, the tensor is TNC type with shape of [seqLength,batchSize,hiddenSize], supporting the data of float16 type.
- [out] output: Output. A pointer pointing to the initial address of the output data of RNN.
- [out] state_output_tensor: Output. NumLayers 4-dimensional input tensor of CPU, the tensor is NCHW type with shape of [batchSize, hiddenSize,1,1],supporting the data of float16 type.
- [out] state_output: Output. A pointer array, and each element of the array is a pointer pointing to the initial address of the state output data of RNN.
- [in] input_weight_tensor: Input. NumLayers 4-dimensional constant tensor of CPU, the tensor is NCHW type with shape of [hiddenSize,vectorSize,1,1],supporting the data of float16 type.
- [in] input_weight: Input. Pointer array pointing to Tensor matrix multiplied by RNN input.
- [in] state_weight_tensor: Input. NumLayers 4-dimensional constant tensor of CPU, the tensor is NCHW type with shape of [hiddenSize,hiddenSize,1,1],supporting the data of float16 type.
- [in] state_weight: Input. Pointer array pointing to the Tensor matrix multiplied by the RNN state input.
- [in] input_bias_tensor: Input. NumLayers 4-dimensional constant tensor of CPU, the tensor is NCHW type with shape of [1,hiddenSize,1,1],supporting the data of float16 type.
- [in] input_bias: Input. Pointer array pointing to Bias matrix multiplied by RNN input.
- [in] state_bias_tensor: Input. NumLayers 4-dimensional constant tensor of, the tensor is NCHW type with shape of [1,hiddenSize,1,1],supporting the data of float16 type.
- [in] state_bias: Input. Pointer array pointing to Bias matrix multiplied by RNN state input.
- [in] active_func: Input. Input specifies which activation function to be used. It can be any activation function that CNML already supports.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.16.5 cnmlComputeBasicRNNProOpForward_V2

cnmlStatus_t **cnmlComputeBasicRNNProOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t state_input_tensors[], void *state_input[], cnmlTensor_t output_tensor, void *output, cnmlTensor_t state_output_tensors[], void *state_output[], cnrtQueue_t queue, void *extra)

A function.

This interface is used to compute basic RNN on CPU;if all parameters in the interface are array type, all arrays include num_layers elements, and the element with subscript 0 in arrays corresponds to the first layer, the element with subscript 1 corresponds to the second layer, and so on.

Supports MLU220 and MLU270.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] state_input_tensors: Input. MLU tensor pointer array. Pass NULL if not used.
- [in] state_input: Input. MLU address array pointing to state_input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] state_output_tensor: Input. State_Output MLU tensor array pointer. Pass NULL if not used.
- [out] state_output: Output. MLU address array pointing to state_output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.17 Batch2space Operation

4.17.1 cnmlCreateBatch2spaceOp

cnmlStatus_t **cnmlCreateBatch2spaceOp**(cnmlBaseOp_t *op, int w_block_size, int h_block_size, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateBatch2spaceOp.

Create a Batch2space operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, the zoom ratio in the h direction and the w direction, input and output tensors, pass them to the function to create a Batch2space operator.

Formula

this is a IO function, this is the reverse function of the space2batch.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. A pointer to the mlu end Tensor
- [in] op: Input. A pointer to the base operator
- [in] input: Input. A pointer to the mlu end
- [in] w_block_size[in] Input.: The zoom ratio representing the move from the w direction to the n direction, wo = wi * w_block_size.
- [in] h_block_size[in] Input.: The zoom ratio representing the move from the h direction to the n direction, ho = hi * h_block_size.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met: Op, input, and output are not empty.

4.17.2 cnmlComputeBatch2spaceOpForward_V3

cnmlStatus_t **cnmlComputeBatch2spaceOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeBatch2spaceOpForward_V3.

It is used to compute the user-specified Batch2space operator on the MLU.

After creating the Batch2space operator, Input, Output, runtime parameters, and computation stream, pass them to the function to It is used to compute the Batch2space operator.

Formula

this is a IO function, this is the reverse function of the space2batch.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Deprecated. This interface will be deleted in next version and cnmlComputeBatch2spaceOpForward_V4 is recommended to use.

Parameters

- [out] output: Output. An MLU address that points to the output location.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.

- [in] compute_forw_param: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue: Input. A computation stream pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.17.3 cnmlComputeBatch2spaceOpForward_V4

cnmlStatus_t **cnmlComputeBatch2spaceOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeBatch2spaceOpForward_V4.

It is used to compute the user-specified Batch2space operator on the MLU.

After creating the Batch2space operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Batch2space operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.18 Batch Norm Operation

4.18.1 cnmlCreateBatchNormOp

cnmlStatus_t **cnmlCreateBatchNormOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t mean_tensor, cnmlTensor_t var_tensor)

A function.

Description

Create a BatchNorm operator according to base operator pointers given by users.

Deprecated. This interface will be deleted in next version and cnmlCreateBatchNormOpForward is recommended to use.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a BatchNorm operator.

The var_tensor of cnmlCreateBatchNormOp: var_tensor = 1 / sqrt(variance + eps) The variance is variance of batch normalization layer. The eps is a positive floating point number added to the variance to avoid division by zero.

This function does not support scaling results. To scale results with gemma and beta factors, you can call cnmlCreateBatchNormOp and cnmlCreateScaleOp, or call cnmlCreateFrozenBatchNormOp instead.

Formula

output[n c h w] = (input[n c h w] - mean_tensor[1 c 1 1]) * var_tensor[1 c 1 1]

DataType

inputDataType, meanDataType, varDataType: float16, float32

outputDataType: float16, float32, int16, int8

Scale Limitation

MLU270:

c < 42000

Supports MLU220,MLU270,1M20,and 1M70.

Note The mean, var are constant data in batchnorm operation. The type of these tensors need to be CNML_CONST, and it need to bind data infomration by cnmlBindConstData_V2.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape of which is [n, h, w, c].
- `[in] output`: Input. A four-dimensional MLU output tensor, the shape of which is [n, h, w, c].
- `[in] mean`: Input. A four-dimensional mean value tensor of MLU, the shape of which is [1, 1, 1, c], supporting data type same as input tensor.
- `[in] var`: Input. A four-dimensional variance tensor of MLU, the shape of which is [1, 1, 1, c], supporting data type same as input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.
  - The mean value tensor is null.
  - The variance value is null. For more information, see “Error Codes” section in this guide.

4.18.2 cnmlCreateNdBatchNormOp

cnmlStatus_t `cnmlCreateNdBatchNormOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t mean_tensor, cnmlTensor_t var_tensor)

A function.

Description

Create a NdBatchNorm operator according to base operator pointers given by users.

Deprecated. This interface will be deleted in next version.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a NdBatchNorm operator.

The var_tensor of cnmlCreateNdBatchNormOp: var_tensor = 1 / sqrt(variance + eps) The variance is variance of batch normalization layer. The eps is a positive floating point number added to the variance to avoid division by zero.

This function does not support scaling results. To scale results with gemma and beta factors, you can call cnmlCreateNdBatchNormOp and cnmlCreateNdScaleOp.

Supports MLU220,MLU270,1M20,and 1M70.

Note The mean, var are constant data in NdBatchnorm operation. The type of these tensors need to be CNML_CONST, and it need to bind data infomration by cnmlBindConstData_V2.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input`: Input. A multi-dimensional MLU input tensor, supporting data of float16 type.
- `[in] output`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.
- `[in] mean`: Input. A multi-dimensional MLU mean tensor, supporting data type same as input tensor.
- `[in] var`: Input. A multi-dimensional MLU var tensor, supporting data type same as input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.
  - The mean value tensor is null.
  - The variance value is null.

4.18.3 cnmlCreateBatchNormOpForward

This API has the same function as cnmlCreateBatchNormOp. For detailed information, see cnmlCreateBatchNormOp.

4.18.4 cnmlComputeBatchNormOpForward_V3

cnmlStatus_t `cnmlComputeBatchNormOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Description

Compute the BatchNorm operator specified by users on the MLU.

After creating a Mult operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the BatchNorm operator.

Formula

output[n c h w] = (input[n c h w] - mean[1 c 1 1]) * var[1 c 1 1]

DataType

MLU270:

inputDataType: float16, float32

outputDataType: float16, float32, int16, int8

Scale Limitation

MLU270:

c < 42000

Deprecated. This interface will be deleted in next version and cnmlComputeBatchNormOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null. For more information, see “Error Codes” section in this guide.

4.18.5 cnmlComputeBatchNormOpForward_V4

cnmlStatus_t `cnmlComputeBatchNormOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Compute the BatchNorm operator specified by users on the MLU.

After creating a Mult operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the BatchNorm operator.

Formula

output[n c h w] = (input[n c h w] -mean[1 c 1 1])* var[1 c 1 1]

DataType

MLU270:

- inputDataType: float16, float32
- computeDataType: computeDataType = inputDataType
- outputDataType: float16, float32, int16, int8

Scale limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The value of C dimension is a multiple of 128.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid. For more information, see “Error Codes” section in this guide.

4.18.6 cnmlComputeNdBatchNormOpForward

cnmlStatus_t `cnmlComputeNdBatchNormOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Compute the NdBatchNorm operator specified by users on the MLU.

After creating a Mult operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the NdBatch-Norm operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid. For more information, see “Error Codes” section in this guide.

4.19 Batchdot Operation

4.19.1 cnmlCreateBatchDotOp

cnmlStatus_t `cnmlCreateBatchDotOp`(cnmlBaseOp_t *op, `const` cnmlTensor_t input_tensor_1, `const` cnmlTensor_t input_tensor_2, `const` cnml-Tensor_t output_tensor, bool trans_a, bool trans_b)

A function.

The function creates a BatchDot operator according to the base operator pointer given by users.

After creating a pointer pointing to base operators, matrix multiplication operator input tensor and output tensor, pass them into the function to create the BatchDot operator.

Realize dot product operation in batches.

the shape of input1 as(n,c1,h1,w1) and the shape of input2 as(n,c2,h2,w2),

set m = c1, p = h1 * w1, q = h2 * w2;

set the shape of input1 as(n,m,p,1)

set the shape of input2 as(n,p,q,1)

The output shape is output(n,m,p,1)

Compute Output:

output(n,m,p,q) = ${l}$input1(n,m,l,1) * input2(n,l,q,1)

If trans_a = true, transpose input1 in batches at first and then perform multiplication:input1(n,m,p,1)-> input1(n,p,m,1)

If trans_b = true, transpose input2 in batches at first and then perform multiplication:input2(n,p,q,1)-> input1(n,q,p,1)

Set the shape of two inputs as shape1 and shape2, and the output shape is: output (no,co,ho,wo).

shape1.h * shape1.w = shape2.c

shape1.n = shape2.n

Set p = shape1.w * shape1.h q =shape2.w * shape2.h

then no = shape1.n

co = trans_a ? p : shape1.c

ho = trans_b ? q : shape2.h

wo = 1

Notice: batchdot is not supported in single core situation. If running single op, the core_limit in cnmlCompileBaseOp and core_num in cnmlSet-BaseOpCoreNum should not equal to 1. And if running a network including this op, the core_limit in cnmlCompileFusionOp also should not equanl to 1.

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor1`: Input. A four-dimensional input tensor, the shape of which is [n1, c1, h1,w1], supporting data of float16 type.
- `[in] input_tensor2`: Input. A four-dimensional input tensor, the shape of which is [n2, c2, h2, w2](n2 = n1) , supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional output tensor, the shape of which is [no, co, ho, wo] (no = n1), supporting data of float16 type.
- `[in] trans_a`: Input. input_tensor1 transposing options, supporting input of bool type.
- `[in] trans_b`: Input. input_tensor2 transposing options, supporting input of bool type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.19.2 cnmlComputeBatchDotOpForward_V3

cnmlStatus_t **cnmlComputeBatchDotOpForward_V3**(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the BatchDot operator specified by users on the MLU.

After creating the BatchDot operator, Input. output, and computation stream, pass them into the function to compute the BatchDot operator.

Deprecated. This interface will be deleted in next version and cnmlComputeBatchDotOpForward_V4 is recommended to use.

Supports MLU220 and MLU270.

Parameters

- `[out] output`: Output. An MLU address specifying the output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] inputA`: Input. An MLU address which points to input data.
- `[in] inputB`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.19.3 cnmlComputeBatchDotOpForward_V4

cnmlStatus_t **cnmlComputeBatchDotOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the BatchDot operator specified by users on the MLU.

After creating the BatchDot operator, Input. output, and computation queue, pass them into the function to compute the BatchDot operator.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. First input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. First MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Second input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. Second MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.20 Broadcast Operation

4.20.1 cnmlCreateBroadcastOp

cnmlStatus_t `cnmlCreateBroadcastOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Descrption

Create a broadcast operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast operator.

Before creating a broadcast operator, declare a pointer pointing to the struct address of operation parameters of the broadcast operator, and pass the pointer and operator parameters required into the function to set operator parameters.

When one dimension of the input is equal to 1 and not equal to the corresponding dimension of the output, duplicate the dimension data to expand the dimension to the output dimension.

For example:

a. int ni = 1, ci = 1, hi = 4, wi = 4;

int no = 1, co = 256, ho = 4, wo = 4;

duplicate and expand data of dimension c.

a. int ni = 1, ci = 256, hi = 4, wi = 4;

int no = 4, co = 256, ho = 4, wo = 4;

duplicate and expand data of dimension n.

a. int ni = 1, ci = 1, hi = 4, wi = 1;

int no = 1, co = 256, ho = 4, wo = 4;

simultaneously duplicate and expand data of dimension c and n.

a. The shape of input cannot be 0.
b. Each dimension of the output shape is larger than or equal to the corresponding dimension of the input shape.
c. When the input dimension n is greater than 1, the output dimension n must be equal to the input dimension n.

Note

When the input dimension c is greater than 1, the output dimension c must be equal to the input dimension c.

When the input dimension h is greater than 1, the output dimension h must be equal to the input dimension h.

When the input dimension w is greater than 1, the output dimension w must be equal to the input dimension w.

DataType

MLU270:

input: int8, int16, float16, float32, int32, bool

output: same as input

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.20.2 cnmlCreateNdBroadcastOp

cnmlStatus_t `cnmlCreateNdBroadcastOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a n-dimensional broadcast operator according to base operator pointers given by users, which is the extension of broadcast operator. It supports tensor from one to any dimension. We recommend you use this interface to create the broadcast operator.

a. The shape of input cannot be 0.
b. Each dimension of the output shape is larger than or equal to the corresponding dimension of the input shape.
c. When the dimension in input is greater than 1, the same dimension in output must be equal to the input.

Note

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A one to any dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A one to any dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.20.3 cnmlCreateBroadcastOpForward

cnmlStatus_t `cnmlCreateBroadcastOpForward`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a broadcast operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast operator.

Before creating a broadcast operator, declare a pointer pointing to the struct address of operation parameters of the broadcast operator, and pass the pointer and operator parameters required into the function to set operator parameters.

When one dimension of the input is equal to 1 and not equal to the corresponding dimension of the output, duplicate the dimension data to expand the dimension to the output dimension.

For example:

a. int ni = 1, ci = 1, hi = 4, wi = 4;
   int no = 1, co = 256, ho = 4, wo = 4;
   duplicate and expand data of dimension c.
b. int ni = 1, ci = 256, hi = 4, wi = 4;
   int no = 4, co = 256, ho = 4, wo = 4;
   duplicate and expand data of dimension n.
c. int ni = 1, ci = 1, hi = 4, wi = 1;
   int no = 1, co = 256, ho = 4, wo = 4;
   simultaneously duplicate and expand data of dimension c and n.

a. The shape of input cannot be 0.
b. Each dimension of the output shape is larger than or equal to the corresponding dimension of the input shape.
c. When the input dimension n is greater than 1, the output dimension n must be equal to the input dimension n.

Note

When the input dimension c is greater than 1, the output dimension c must be equal to the input dimension c.

When the input dimension h is greater than 1, the output dimension h must be equal to the input dimension h.

When the input dimension w is greater than 1, the output dimension w must be equal to the input dimension w.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.20.4 cnmlComputeBroadcastOpForward_V3

cnmlStatus_t `cnmlComputeBroadcastOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the broadcast operator specified by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeBroadcastOpForward_V4 is recommended to use.

After creating a broadcast operator, input, output, runtime parameters, and computation queue, pass them into the function to compute broadcast operator.

DataType

MLU270:

input: int8, int16, float16, float32, bool

output: same as input

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.20.5 cnmlComputeBroadcastOpForward_V4

cnmlStatus_t `cnmlComputeBroadcastOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast operator specified by users on the MLU.

After creating a broadcast operator, input, output, runtime parameters, and computation queue, pass them into the function to compute broadcast operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.20.6 cnmlComputeNdBroadcastOpForward

cnmlStatus_t `cnmlComputeNdBroadcastOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdBroadcastOpForward_V2 is recommended to use.

Compute the extensional broadcast operator on the MLU.

After creating a NdBroadcast operator, input, output, and computation stream, pass them into the function to compute the broadcast operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] type`: Input. An enumeration constant specifying the computation mode on the MLU.
- `[in] stream`: Input. A computational stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.20.7 cnmlComputeNdBroadcastOpForward_V2

cnmlStatus_t `cnmlComputeNdBroadcastOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the extensional broadcast operator on the MLU.

After creating a NdBroadcast operator, input, output, and computation stream, pass them into the function to compute the broadcast operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.21 Broabcast Add Operation

4.21.1 cnmlComputeBroadcastAddOpForward_V3

cnmlStatus_t `cnmlComputeBroadcastAddOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the broadcast add operator given by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeBroadcastAddOpForward_V4 is recommended to use.

After creating a broadcast add operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast add operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.

- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.21.2 cnmlComputeBroadcastAddOpForward_V4

cnmlStatus_t **cnmlComputeBroadcastAddOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast add operator given by users on the MLU.

After creating a broadcast add operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast add operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. First input MLU tensor pointer. Pass NULL if not used.
- [in] input_1: Input. First MLU address pointing to input1 data.
- [in] input_tensor2: Input. Second input MLU tensor pointer. Pass NULL if not used.
- [in] input_2: Input. Second MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.21.3 cnmlCreateBroadcastAddOp

cnmlStatus_t **cnmlCreateBroadcastAddOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a broadcast add operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast add operator.

Before creating a broadcast add operator, declare a pointer pointing to the struct address of operation parameters of the broadcast add operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise addition on the input to obtain the output.

DataType

MLU270:

input : float16, float32, int32

output : the same as input

For example:

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 32, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension h and w of input2, and perform element-wise addition on the input.
b. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 1, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension c, h, and w of input2, and perform element-wise addition on the input.

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n1, c1, h1, w1], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [n2, c2, h2, w2], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.22 Broabcast Args Operation

4.22.1 cnmlCreateBroadcastArgsOp

cnmlStatus_t `cnmlCreateBroadcastArgsOp`(cnmlBaseOp_t *op, `const` cnmlTensor_t inputTensorA, `const` cnmlTensor_t inputTensorB, `const` cnmlTensor_t outputTensor)

A function.

According to the base operator pointer given by the user, create a broadcast args operator.

A broadcast args operator is given two one-dimension tensors that represent shapes, compute the broadcasted shape, output tensor is also one-dimension.

It’ s just the same as max equal in one-dimension and support different input size.

Formula

output[index] = input1[index] >= input2[index] ? input1[index] : input2[index]

if size of input1 larger than input2, compute max equal in same index(from high to low), the remain position in output just the same as input1, and vice versa.

for example:

input1 data shape: {2}, input2 data shape: {4}, output data shape: {4}

input1 data: [12, 1]

input2 data: [1, 14, 1, 9]

output data: [1, 14, 12, 9]

DataType

MLU270:

int16, int32

MLU220:

int16, int32

Scale Limitation

Only supports one-dimension tensor, and input size should follow limitation below.

MLU270:

int32: input1 size < 32768, input2 size < 32768

int16: input1 size < 65536, input2 size < 65536

MLU220:

int32: input1 size < 32768, input2 size < 32768

int16: input1 size < 65536, input2 size < 65536

Performance Optimization

The number of bytes in the C dimension is a multiple of 128

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] inputTensorA: Input. A one-dimensional MLU input tensor, supporting data of int16/int32 type.
- [in] inputTensorB: Input. A one-dimensional MLU input tensor, supporting data of int16/int32 type.
- [in] output_tensor: Input. A one-dimensional MLU output tensor, shape of which equals to max shape between input1 and input2, supporting data of int16/int32 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: The shape of output tensor is different from that of input tensor.

4.22.2 cnmlComputeBroadcastArgsOpForward

cnmlStatus_t **cnmlComputeBroadcastArgsOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast args operator given by users on the MLU.

After creating a broadcast args operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the division operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input_1: Input. An MLU address pointing to input data.
- [in] input_tensor2: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input_2: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.23 Broadcast lesser Operation

4.23.1 cnmlCreateBroadcastLesserOp

cnmlStatus_t **cnmlCreateBroadcastLesserOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a broadcast lesser operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast lesser operator.

Before creating a broadcast lesser operator, declare a pointer pointing to the struct address of operation parameters of the broadcast lesser operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise lesser than comparison operation on the input to obtain the output.

For example:

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 32, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension h and w of input2, and perform element-wise lesser than comparison operation on the input.
b. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 1, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension c, h, and w of input2, and perform element-wise lesser than comparison operation on the input.

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Summary

For input1[n1, c1, h1, w1], input2[n2, c2, h2, w2],

output[n, c, h, w] = (broadcast(input1)[n, c, h, w] < broadcast(input2)[n, c, h, w]) ? 1 : 0

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n1, c1, h1, w1], supporting data of float16 and float32 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [n2, c2, h2, w2], supporting data of float16 and float32 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo], supporting data of float16 and float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.23.2 cnmlComputeBroadcastLesserOpForward_V3

cnmlStatus_t `cnmlComputeBroadcastLesserOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the broadcast lesser operator given by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeBroadcastLesserOpForward_V4 is recommended to use.

After creating a broadcast lesser operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast lesser operator.

Summary

For input1[n1, c1, h1, w1], input2[n2, c2, h2, w2],

output[n, c, h, w] = (broadcast(input1)[n, c, h, w] < broadcast(input2)[n, c, h, w]) ? 1 : 0

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.23.3 cnmlComputeBroadcastLesserOpForward_V4

cnmlStatus_t `cnmlComputeBroadcastLesserOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast lesser operator specified by users on the MLU.

After creating a broadcast mlu operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast mlu operator.

Summary

For input1[n1, c1, h1, w1], input2[n2, c2, h2, w2],

output[n, c, h, w] = (broadcast(input1)[n, c, h, w] < broadcast(input2)[n, c, h, w]) ? 1 : 0

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.24 Broadcast Mult Operation

4.24.1 cnmlCreateBroadcastMultOp

cnmlStatus_t `cnmlCreateBroadcastMultOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a broadcast mult operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast mult operator.

Before creating a broadcast mult operator, declare a pointer pointing to the struct address of operation parameters of the broadcast mult operator, and pass the pointer and operator parameters required into the function to set operator parameters.

For example:

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
const int n2 = 1, c2 = 32, h2 = 1, w2 = 1;
const int no = 1, co = 32, ho = 4, wo = 4;
duplicate and expand dimension h and w of input2, and perform element-wise multiplication on the input.
b. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
const int n2 = 1, c2 = 1, h2 = 1, w2 = 1;
const int no = 1, co = 32, ho = 4, wo = 4;
duplicate and expand dimension c, h, and w of input2, and perform element-wise multiplication on the input.

Formula

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise multiplication on the input to obtain the output.

DataType

MLU270:

float16, float32, int32

Scale Limitation

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n1, h1, w1, c1], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [n2, h2, w2, c2], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, ho, wo, co], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.24.2 cnmlCreateBroadcastMultOpForward

cnmlStatus_t `cnmlCreateBroadcastMultOpForward`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a broadcast mult operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast mult operator.

Before creating a broadcast mult operator, declare a pointer pointing to the struct address of operation parameters of the broadcast mult operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise multiplication on the input to obtain the output.

For example:

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;

const int n2 = 1, c2 = 32, h2 = 1, w2 = 1;

const int no = 1, co = 32, ho = 4, wo = 4;

duplicate and expand dimension h and w of input2, and perform element-wise multiplication on the input.

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;

const int n2 = 1, c2 = 1, h2 = 1, w2 = 1;

const int no = 1, co = 32, ho = 4, wo = 4;

duplicate and expand dimension c, h, and w of input2, and perform element-wise multiplication on the input.

Formula

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise multiplication on the input to obtain the output.

DataType

MLU270:

float16, float32, int32

Scale Limitation

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n1, h1, w1, c1], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [n2, h2, w2, c2], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, ho, wo, co], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.24.3 cnmlComputeBroadcastMultOpForward_V3

cnmlStatus_t `cnmlComputeBroadcastMultOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the broadcast mlu operator specified by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeBroadcastMultOpForward_V4 is recommended to use.

After creating a broadcast mlu operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast mlu operator.

Formula

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise multiplication on the input to obtain the output.

DataType

MLU270:

float16, float32, int32

Scale Limitation

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.

- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.24.4 cnmlComputeBroadcastMultOpForward_V4

cnmlStatus_t cnmlComputeBroadcastMultOpForward_V4(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast mlu operator specified by users on the MLU.

After creating a broadcast mlu operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast mlu operator.

Formula

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise multiplication on the input to obtain the output.

DataType

MLU270:

float16, float32, int32

Scale Limitation

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.25 Broadcast Sub Operation

4.25.1 cnmlComputeBroadcastSubOpForward_V3

cnmlStatus_t `cnmlComputeBroadcastSubOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the broadcast sub operator given by users on the MLU.

After creating a broadcast sub operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast sub operator.

Deprecated. This interface will be deleted in next version and cnmlComputeBroadcastSubOpForward_V4 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.25.2 cnmlComputeBroadcastSubOpForward_V4

cnmlStatus_t `cnmlComputeBroadcastSubOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the broadcast sub operator given by users on the MLU.

After creating a broadcast sub operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the broadcast sub operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.25.3 cnmlCreateBroadcastSubOp

cnmlStatus_t `cnmlCreateBroadcastSubOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a broadcast sub operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a broadcast sub operator.

Before creating the broadcast sub operator, declare a pointer pointing to the struct address of operation parameters of the broadcast sub operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Expand each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise subtraction on the input to obtain the output.

DataType

MLU270:

input : float16, float32, int32

output : the same as input

For example:

a. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 32, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension h and w of input2, and perform element-wise subtraction on the input.
b. const int n1 = 1, c1 = 32, h1 = 4, w1 = 4;
   const int n2 = 1, c2 = 1, h2 = 1, w2 = 1;
   const int no = 1, co = 32, ho = 4, wo = 4;
   duplicate and expand dimension c, h, and w of input2, and perform element-wise subtraction on the input.

In fact, addition, subtraction, and multiplication of broadcast match different NG operators according to the shape of operands: scale, cycle operators, and ordinary addition, subtraction, and multiplication operators.

Input1 has the same shape as output.

The shape of input2 supports three settings:

(1)n2 = n1, c2 = c1, h2 = h1, w2 = w1;

(2)n2 = 1, c2 = c1, h2 = 1, w2 = 1;

(3)n2 = 1, c2 = 1, h2 = 1, w2 = 1;

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n1, c1, h1, w1], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [n2, c2, h2, w2], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.26 Cast Operation

4.26.1 cnmlCreateCastOp

cnmlStatus_t `cnmlCreateCastOp`(cnmlBaseOp_t *op, cnmlCastType_t cast_type, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A funciton.

Description

According to the base operator pointer given by the user, a cast operator is created.

After a pointer pointing to the address of base operator, the CastType and input-output tensor are created, they are introduced into the function to create the cast operator.

The supported round mode are roundTowardZero, roundTiesToAway and roundTiesToEven.

The round mode is set to roundTowardZero, when the cast_type is CNML_CAST_FLOAT32_TO_UINT8, CNML_CAST_FLOAT16_TO_FIX8, CNML_CAST_FLOAT16_TO_INT8, CNML_CAST_FLOAT16_TO_INT16, CNML_CAST_FLOAT16_TO_INT16_ROUND_ZERO, CNML_CAST_FLOAT16_TO_UINT8, CNML_CAST_FLOAT32_TO_INT8, CNML_CAST_FLOAT16_TO_INT16, CNML_CAST_FLOAT16_TO_INT32_ROUND_ZERO or CNML_CAST_FLOAT32_TO_INT32_ROUND_ZERO.

The round mode is set to roundTiesToAway, when the cast_type is CNML_CAST_UINT8_TO_FLOAT32, CNML_CAST_INT8_TO_FLOAT16, CNML_CAST_FIX8_TO_FLOAT16, CNML_CAST_FLOAT16_TO_FLOAT32, CNML_CAST_FLOAT32_TO_FLOAT16, CNML_CAST_INT16_TO_FLOAT16, CNML_CAST_UINT8_TO_FLOAT16, CNML_CAST_INT8_TO_FLOAT32 or CNML_CAST_INT16_TO_FLOAT16.

The round mode is set to roundTiesToEven, when the cast_type is CNML_CAST_FLOAT16_TO_INT16_ROUND_EVEN or CNML_CAST_FLOAT16_TO_FLOAT16_ROUND_EVEN.

Different data types correspond to different input restrictions, for example when the cast_type is CNML_CAST_FLOAT32_TO_UINT8, the output should be in [0,255].

The shapes of input and output should be the same.

Formula

Change the input data from input’s datatype to output’s datatype.

DataType

MLU270:

input: float32 uint8 float16 int8 int16

output: float32 uint8 float16 int8 int16 int32

Scale Limitation

MLU270:

input_shape’s c < 63000

data only support 22 width for int when the output datatype is int32.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[out] output`: Output. A four-dimensional MLU input tensor, the shape is [batch, height, width, depth], supports data of the following types: float32 uint8 float16 int8 int16.
- `[in] cast_type`: Input. An enumeration constant that represents the data type conversion method.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape is [batch, height, width, depth], supports data of the following types: float32 uint8 float16 int8 int16.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - input tensor type is not CNML_TENSOR

4.26.2 cnmlCreateCastOpForward

cnmlStatus_t `cnmlCreateCastOpForward`(cnmlBaseOp_t *op, cnmlCastType_t cast_type, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateCastOpForward.

Description

According to the base operator pointer given by the user, a cast operator is created.

After a pointer pointing to the address of base operator, the CastType and input-output tensor are created, they are introduced into the function to create the cast operator.

The operator can be used to realize data type conversion of tensor.

The supported round mode are roundTowardZero, roundTiesToAway and roundTiesToEven.

The round mode is set to roundTowardZero, when the cast_type is CNML_CAST_FLOAT32_TO_UINT8, CNML_CAST_FLOAT16_TO_FIX8, CNML_CAST_FLOAT16_TO_INT8, CNML_CAST_FLOAT16_TO_INT16, CNML_CAST_FLOAT16_TO_INT16_ROUND_ZERO, CNML_CAST_FLOAT16_TO_UINT8, CNML_CAST_FLOAT32_TO_INT8, CNML_CAST_FLOAT16_TO_INT16, CNML_CAST_FLOAT16_TO_INT32_ROUND_ZERO or CNML_CAST_FLOAT32_TO_INT32_ROUND_ZERO.

The round mode is set to roundTiesToAway, when the cast_type is CNML_CAST_UINT8_TO_FLOAT32, CNML_CAST_INT8_TO_FLOAT16, CNML_CAST_FIX8_TO_FLOAT16, CNML_CAST_FLOAT16_TO_FLOAT32, CNML_CAST_FLOAT32_TO_FLOAT16, CNML_CAST_INT16_TO_FLOAT16, CNML_CAST_UINT8_TO_FLOAT16, CNML_CAST_INT8_TO_FLOAT32 or CNML_CAST_INT16_TO_FLOAT16.

The round mode is set to roundTiesToEven, when the cast_type is CNML_CAST_FLOAT16_TO_INT16_ROUND_EVEN or CNML_CAST_FLOAT16_TO_FLOAT16_ROUND_EVEN.

Different data types correspond to different input restrictions, for example when the cast_type is CNML_CAST_FLOAT32_TO_UINT8, the output should be in [0,255].

The shapes of input and output should be the same.

Formula

Change the input data from input’s datatype to output’s datatype.

DataType

MLU270:

input: float32 uint8 float16 int8 int16

output: float32 uint8 float16 int8 int16 int32

Scale Limitation

MLU270:

input_shape’ s c < 63000

data only support 22 width for int when the output datatype is int32.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[out] output`: Output. A four-dimensional MLU input tensor, the shape is [batch, height, width, depth], supports data of the following types: float32 uint8 float16 int8 int16.
- `[in] cast_type`: Input. An enumeration constant that represents the data type conversion method.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape is [batch, height, width, depth], supports data of the following types: float32 uint8 float16 int8 int16.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - input tensor type is not CNML_TENSOR

4.26.3 cnmlComputeCastOpForward_V3

cnmlStatus_t **cnmlComputeCastOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeCastOpForward_V3.

Compute the user-specified Cast operator. After creating Cast operator, input, output and computation stream, introduce them to the function to compute the Cast operator.

Formula

Change the input data from input’ s datatype to output’ s datatype

DataType

MLU270:

input: float32 uint8 float16 int8 int16

output: float32 uint8 float16 int8 int16 int32

Scale Limitation

MLU270:

input_shape’ s c < 63000

Deprecated. This interface will be deleted in next version and cnmlComputeCastOpForward_V4 is recommended to use.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.

4.26.4 cnmlComputeCastOpForward_V4

cnmlStatus_t **cnmlComputeCastOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeCastOpForward_V4.

Compute the user-specified Cast operator. After creating Cast operator, input, output and computation queue, introduce them to the function to compute the Cast operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.27 Clip Operation

4.27.1 cnmlCreateClipOp

cnmlStatus_t `cnmlCreateClipOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, double lower_bound, double upper_bound)

A function.

According to the base operator pointer given by the user, create a clip operation operator.

The operator crops an input tensor and intercepts the upper and lower boundaries. The operator can be seen as a combination of maxtc and mintc, which inputs a tensor and two constants. Perform an upper clipping and a lower clipping on the tensor to obtain a clipped tensor.

The scale of input and output must be the same.

Formula

y = x < lower_bound ? lower_bound : (x < upper_bound ? x : upper_bound)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [no, co, ho, wo] (no = ni), supports data of float16 type.
- `[in] lower_bound`: Input. Lower boundary of interception, supports data of float16 type..
- `[in] upper_bound`: Input. Upper boundary of interception, supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: Successfully created a clipping operation. Return the corresponding error code when execution is failed.

4.27.2 cnmlComputeClipOpForward_V3

cnmlStatus_t `cnmlComputeClipOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Perform the user-specified clipping operation on the MLU.

After creating Clip operator, input, output and computation stream, introduce them to the function to compute the Clip operator.

Deprecated. This interface will be deleted in next version and cnmlComputeClipOpForward_V4 is recommended to use.

Formula

y = x < lower_bound ? lower_bound : (x < upper_bound ? x : upper_bound)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address pointing to input data.
- [in] computue_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.27.3 cnmlComputeClipOpForward_V4

cnmlStatus_t **cnmlComputeClipOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform the user-specified clipping operation on the MLU.

After creating Clip operator, input, output and computation queue, introduce them to the function to compute the Clip operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.28 Concat Operation

4.28.1 cnmlCreateConcatOpParam

cnmlStatus_t **cnmlCreateConcatOpParam**(cnmlConcatOpParam_t *param, int input_num, int output_num, cnmlDimension_t concat_mode)

A function.

Create parameters of the splice operator, which include the number of input tensor and output tensor, but the number of output tensor must be 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input and output. Pointing to a struct of splice operator parameters.
- [in] input_num: Input. The number of input tensor.
- [in] output_num: Input. The number of output tensor must be 1.
- [in] mode: Input. Specifying the splicing dimensions, and the value can be: CNML_CONCAT_FEAT, CNML_CONCAT_BATCH, CNML_CONCAT_HEIGHT, and CNML_CONCAT_WIDTH.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.28.2 cnmlCreateNdConcatOp

cnmlStatus_t `cnmlCreateNdConcatOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t *input_tensors, int input_num, cnmlTensor_t *output_tensors, int output_num)

A function.

Create an NdConcat operator according to base operator pointers given by users. After creating a pointer pointing to base operator address, the specified dimension of the input tensor, input and output tensor, pass them into the function to create an NdConcat operator.

The dim parameter starts from 0, and less than the number of dimensions of input tensor. Take input_tensor[i][j][k] as an example, the dimensions of this input_tensor is 3, that means the dim parameter must be in [0, 2].

Output_num must be 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] dim`: Input. Specifying the dimension of concat operation.
- `[in] input_tensors`: Input. A tensor array, each element is a n-dimensional MLU tensor, supporting data of float16 type.
- `[in] intput_num`: Input. The size of inputs array.
- `[in] output_tensors`: Input. A tensor array, each element is a n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_num`: Input. The size of outputs array.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.28.3 cnmlDestroyConcatOpParam

cnmlStatus_t `cnmlDestroyConcatOpParam`(cnmlConcatOpParam_t *param)

A function.

Release the struct of splice operator parameters.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. Pointing to a struct of splice operator parameters.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.28.4 cnmlCreateConcatOp

cnmlStatus_t `cnmlCreateConcatOp`(cnmlBaseOp_t *op, cnmlConcatOpParam_t param, cnmlTensor_t *input_tensors, int input_num, cnmlTensor_t *output_tensors, int output_num)

A function.

Create a splice operator, which can splice multiple tensors into one tensor.

Formula

if dim =N:

Output[sum(n1, n2, ⋯, nn), h, w, c] = input([n1, h, w, c], [n2, h, w, c], ⋯, [nn, h, w, c])

if dim = C:

Output[n, h, w, sum(c1, c2, ⋯, cn)] = input([n, h, w, c1], [n, h, w, c2], ⋯, [n, h, w, cn])

if dim = H:

Output[n, sum(h1, h2, ⋯, hh), w, c] = input([n, h1, w, c], [n, h2, w, c], ⋯, [n, hn, w, c])

if dim = W:

Output[n, h, sum(w1, w2, ⋯, wn), c] = input([n, h1, w1, c], [n, h2, w2, c], ⋯, [n, hn, wn, c])

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For best practices and higher performance, it is recommended that you set either the N dimension or C dimension of the input and output tensors with the following conditions:

- All of the following conditions are met in N dimension:

a. The input and output data is in the middle layer of the network.
b. data_size_input = ni * hi*wI *ci
c. data_size_out = no * ho * wo *co
d. The value of `data_size_out` and `data_size_input` are multiple of 64.

- Set the size of the C-dimension is greater than 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] outputs_ptr`: Output. A 4-dimensional MLU tensor.
- `[in] op`: Input. Pointing to an operator address.
- `[in] param`: Input. Pointing to a struct address of operator parameters.
- `[in] inputs_ptr`: Input. A 4-dimensional MLU tensor.
- `[in] intput_num`: Input. The number of input tensor.
- `[in] output_num`: Input. The number of output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.28.5 cnmlComputeConcatOpForward_V3

cnmlStatus_t **cnmlComputeConcatOpForward_V3**(cnmlBaseOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeConcatOpForward_V4 is recommended to use.

Compute the splice operator.

Formula

if dim =N:

Output[sum(n1, n2, ⋯, nn), h, w, c] = input([n1, h, w, c], [n2, h, w, c], ⋯, [nn, h, w, c])

if dim = C: Output[n, h, w, sum(c1, c2, ⋯, cn)] = input([n, h, w, c1], [n, h, w, c2], ⋯, [n, h, w, cn])

if dim = H:

Output[n, sum(h1, h2, ⋯, hh), w, c] = input([n, h1, w, c], [n, h2, w, c], ⋯, [n, hn, w, c])

if dim = W:

Output[n, h, sum(w1, w2, ⋯, wn), c] = input([n, h1, w1, c], [n, h2, w2, c], ⋯, [n, hn, wn, c])

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. Pointing to an operator address.
- `[in] inputs`: Input. The address of input tensor data.
- `[in] intput_num`: Input. The number of input tensor.
- `[in] outputs`: Input. The address of output tensor data.
- `[in] output_num`: Input. The number of output tensor.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] stream`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.28.6 cnmlComputeConcatOpForward_V4

cnmlStatus_t `cnmlComputeConcatOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

A function.

Compute the splice operator.

Formula

if dim =N:

Output[sum(n1, n2, ⋯, nn), h, w, c] = input([n1, h, w, c], [n2, h, w, c], ⋯, [nn, h, w, c])

if dim = C:

Output[n, h, w, sum(c1, c2, ⋯, cn)] = input([n, h, w, c1], [n, h, w, c2], ⋯, [n, h, w, cn])

if dim = H:

Output[n, sum(h1, h2, ⋯, hh), w, c] = input([n, h1, w, c], [n, h2, w, c], ⋯, [n, hn, w, c])

if dim = W:

Output[n, h, sum(w1, w2, ⋯, wn), c] = input([n, h1, w1, c], [n, h2, w2, c], ⋯, [n, hn, wn, c])

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For best practices and higher performance, it is recommended that you set either the N dimension or C dimension of the input and output tensors with the following conditions:

- All of the following conditions are met in N dimension:

a. The input and output data is in the middle layer of the network.
b. data_size_input = ni * hi*wI *ci
c. data_size_out = no * ho * wo *co
d. The value of `data_size_out` and `data_size_input` are multiple of 64.

- Set the size of the C-dimension is greater than 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. Pointing to an operator address.
- `[in] input_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[in] inputs`: Input. The address of input tensor data.
- `[in] intput_num`: Input. The number of input tensor.
- `[in] output_tensors`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[in] outputs`: Input. The address of output tensor data.
- `[in] output_num`: Input. The number of output tensor.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.28.7 cnmlComputeNdConcatOpForward

cnmlStatus_t `cnmlComputeNdConcatOpForward`(cnmlBaseOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdConcatOpForward_V2 is recommended to use.

Compute the operator specified by users on the MLU. After creating an Ndconcat operator, input, output, and computation stream, pass them into the function to compute the Ndconcat operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] outputs`: Output. A pointer array, each element pointing to the MLU address of output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] inputs`: Input. A pointer array, each element pointing to the MLU address of input data.
- `[in] intput_num`: Input. The number of elements in the inputs array.
- `[in] output_num`: Input. The number of elements in the outputs array.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.28.8 cnmlComputeNdConcatOpForward_V2

cnmlStatus_t `cnmlComputeNdConcatOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

A function.

Compute the operator specified by users on the MLU. After creating an Ndconcat operator, input, output, and computation stream, pass them into the function to compute the Ndconcat operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[in] input_tensors`: Input. A pointer array, each element pointing to the MLU address of input data.
- `[in] intput_num`: Input. The number of elements in the inputs array.
- `[in] output_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[out] outputs`: Output. A pointer array, each element pointing to the MLU address of output position.
- `[in] output_num`: Input. The number of elements in the outputs array.
- `[in] queue`: Input. A computation stream pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.29 Cond Operation

4.29.1 cnmlCreateCondOp

cnmlStatus_t `cnmlCreateCondOp`(cnmlBaseOp_t *op, cnmlTensor_t cond_input_val)

A function.

The interface is used for creating a cond operator.

According to the base operator pointer given by the user, create a condition operator.

The cond operator is used to build a operator that could realize the if/else condition function.

The cond operator should receive a tensor as the condition, and would switch into different branches according to the condition tensor value(True/False). Each cond operator should only have one condition input tensor.

DataType

cond_input_val: int8, int16

input datatype is not need to be set

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operator address.
- `[in] cond_input_var`: Input. A four-dimensional MLU tensor, the shape is [n, c, h, w] (n = 1, h = 1, w = 1, c = 1),

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.29.2 cnmlAddCondMerge

cnmlStatus_t **cnmlAddCondMerge**(cnmlBaseOp_t op, cnmlTensor_t inputs[], int in_num, cnmlTensor_t output)

A function.

The interface is used for adding the merge operator to a cond operator, and the cond operator should be already created by cnmlCreateCondOp function.

Cond merge operators are the operators that could merge different inputs into a output, which realize transfering the ready tensor from input to output.

Formula

output = merge(inputs[0], inputs[1], ⋯inputs[n - 1]) when inputs[i] is ready.

DataType

inputs: int8, int16, float16, float32

in_num: int

output: same as inputs

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype

2.input and output shape should be the same

input.shape = output.shape

3.in_num only support 2 now

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] inputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] in_num`: Input. The number of inputs.
- `[in] output`: Input. A four-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.29.3 cnmlAddTrueCondOperation

cnmlStatus_t **cnmlAddTrueCondOperation**(cnmlBaseOp_t op, cnmlBaseOp_t branch_op)

A function.

The interface is used for adding the operator in a true condition branch to the cond operator, and the cond operator should be already created by cnmlCreateCondOp function.

True condition operators are the operators created by the users and need to be used in the true branch in a cond operator.

All the true condition operators should be added into the cond operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] branch_op`: Input. A pointer pointing to base operator address.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.29.4 cnmlAddFalseCondOperation

cnmlStatus_t **cnmlAddFalseCondOperation**(cnmlBaseOp_t op, cnmlBaseOp_t branch_op)

A function.

The interface is used for adding the operator in a false condition branch to the cond operator, and the cond operator should be already created by cnmlCreateCondOp function.

True condition operators are the operators created by the users and need to be used in the false branch in a cond operator.

All the false condition operators should be added into the cond operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] branch_op: Input. A pointer pointing to base operator address.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.29.5 cnmlSetCondIO

cnmlStatus_t **cnmlSetCondIO**(cnmlBaseOp_t op, cnmlTensor_t inputs[], int in_num, cnmlTensor_t outputs[], int out_num)

A function.

The interface is used for setting the input and output tensors to a cond operator, and the cond operator should be already created by cnmlCreateCondOp function.

DataType

inputs: int8, int16, float16, float32

in_num: int

outputs: same as inputs

out_num: same as in_num

input and output onchip datatypes are not need to be set

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] inputs: Input. A array of four-dimensional MLU input tensors.
- [in] in_num: Input. The number of inputs.
- [in] outputs: Input. A array of four-dimensional MLU input tensors.
- [in] out_num: Input. The number of outputs.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.29.6 cnmlComputeCondOpForward_V3

cnmlStatus_t `cnmlComputeCondOpForward_V3`(cnmlBaseOp_t op, void **input, int in_num, void **output, int out_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t stream)

A function.

For computing the cond operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operator.
- `[in] inputs`: Input. An MLU address pointing to input data.
- `[in] in_num`: Input. The number of inputs.
- `[in] out_num`: Input. The number of outputs.
- `[in] stream`: Input. A computation stream pointer.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.30 Control Flow Operation

4.30.1 cnmlCreateControlFlowOp

cnmlStatus_t `cnmlCreateControlFlowOp`(cnmlBaseOp_t *op)

A function.

The interface is used for creating a ControlFlow operator.

According to the base operator pointer given by the user, create a controlflow operator.

The controlflow operator is used to build a operator that could realize the condition/while and other control flow functions.

The controlflow operator consists of five sub-operators, enter, merge, switch, next iteration and exit.

This operator provides a way to build a operator that could realize different functions according to the need of the users, by combining the five sub-operators in different way.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operator address.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.2 cnmlAddEnter

cnmlStatus_t `cnmlAddEnter`(cnmlBaseOp_t op, cnmlTensor_t input, cnmlTensor_t output)

A function.

The interface is used for adding an enter operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Enter operators should be used to transfer input tensors into the controlflow loop body.

All the enter operators should be added into the controlflow operator by this interface.

Formula

output = input

DataType

input: int8, int16, float16, float32

output: same as input

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype 2.input.shape = output.shape

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] input`: Input. A four-dimensional MLU input tensor.
- `[in] output`: Input. A four-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.3 cnmlAddMerge

cnmlStatus_t **cnmlAddMerge**(cnmlBaseOp_t op, cnmlTensor_t inputs[], int in_num, cnmlTensor_t output)

A function.

The interface is used for adding a merge operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Merge operators are used to merge different input tensors into one.

All the merge operators should be added into the controlflow operator by this interface.

Formula

output = merge(inputs[0], inputs[1], ⋯inputs[n - 1]) when inputs[i] is ready.

DataType

inputs: int8, int16, float16, float32

in_num: int

outputs: same as input

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype

2.all the inputs and output shape should be the same

inputs[i - 1].shape = inputs[i].shape

inputs[i].shape = output.shape

3.the in_num only support 2 now

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] inputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] in_num`: Input. The number of inputs.
- `[in] output`: Input. A four-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.4 cnmlAddSwitch

cnmlStatus_t `cnmlAddSwitch`(cnmlBaseOp_t op, cnmlTensor_t input, cnmlTensor_t cond, cnmlTensor_t outputs[], int out_num)

A function.

The interface is used for adding a switch operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Switch operators are used to choose which branch to run according to the conditon tensor.

There are two input tensors for the function, while the first is the input tensor, and the second is the condition tensor.

All the switch operators should be added into the controlflow operator by this interface.

Formula

output[0] = input if condition == true; else output[1] = input.

DataType

input: int8, int16, float16, float32

cond: int

outputs: same as input

out_num: int

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype

2.all the input and outputs shape should be the same

outputs[i - 1].shape = outputs[i].shape

outputs[i].shape = input.shape

3.out_num only support 2 now

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] input`: Input. A four-dimensional MLU input tensors.
- `[in] cond`: Input. A four-dimensional MLU input tensors.
- `[in] outputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] out_num`: Input. The number of outputs.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.5 cnmlAddExit

cnmlStatus_t `cnmlAddExit`(cnmlBaseOp_t op, cnmlTensor_t input, cnmlTensor_t output)

A function.

The interface is used for adding an exit operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Exit operators should be used to transfer output tensors out of the controlflow loop body.

All the exit operators should be added into the controlflow operator by this interface.

Formula

output = input.

DataType

input: int8, int16, float16, float32

output: same as input

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype 2.input and output shape should be the same input.shape = output.shape

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.

- [in] input: Input. A four-dimensional MLU input tensor.
- [in] output: Input. A four-dimensional MLU input tensor.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.6 cnmlAddBody

cnmlStatus_t **cnmlAddBody**(cnmlBaseOp_t op, cnmlBaseOp_t body_op)

A function.

The interface is used for adding the loop body operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Body operators are the operators created by the users and need to be used in the controlflow operator.

All the loop body operators should be added into the controlflow operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] body_op: Input. A pointer pointing to base operator address.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.7 cnmlAddCondition

cnmlStatus_t **cnmlAddCondition**(cnmlBaseOp_t op, cnmlBaseOp_t condition_op)

A function.

The interface is used for adding the condition operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Condition operators are the operators that decide whether to enter a loop branch(True/False).

All the condition operators should be added into the controlflow operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] condition_op: Input. A pointer pointing to base operator address.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.8 cnmlAddNextIteration

cnmlStatus_t **cnmlAddNextIteration**(cnmlBaseOp_t op, cnmlTensor_t input, cnmlTensor_t output)

A function.

The interface is used for adding a next iteration operator to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

Next iteration operators transfer the tensor from the loop body into a next iteration.

All the next iteration operators should be added into the controlflow operator by this interface.

Formula

output = input.

DataType

input: int8, int16, float16, float32

output: same as input

input and output onchip datatypes are not need to be set

Scale Limitation

1.input_datatype = output_datatype 2.input and output shape should be the same input.shape = output.shape

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] input`: Input. A four-dimensional MLU input tensor.
- `[in] output`: Input. A four-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.9 cnmlSetControlFlowIO

cnmlStatus_t **cnmlSetControlFlowIO**(cnmlBaseOp_t op, cnmlTensor_t inputs[], int in_num, cnmlTensor_t outputs[], int out_num)

A function.

The interface is used for setting the input and output tensors to a controlflow operator, and the controlflow operator should be already created by cnmlCreateControlFlowOp function.

DataType

inputs: int8, int16, float16, float32

in_num: int

outputs: same as input

out_num: same as in_num

input and output onchip datatypes are not need to be set

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operators address.
- `[in] inputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] in_num`: Input. The number of inputs.
- `[in] outputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] out_num`: Input. The number of outputs.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.30.10 cnmlComputeControlFlowOpForward_V3

cnmlStatus_t **cnmlComputeControlFlowOpForward_V3**(cnmlBaseOp_t op, void *inputs[], int in_num, void *outputs[], int out_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

For computing the controlflow operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] outputs`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operator.
- `[in] inputs`: Input. An MLU address pointing to input data.
- `[in] in_num`: Input. The number of inputs.
- `[in] out_num`: Input. The number of outputs.
- `[in] stream`: Input. A computation stream pointer.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.31 Conv Operation

4.31.1 cnmlCreateConvOpParam

cnmlStatus_t **cnmlCreateConvOpParam**(cnmlConvOpParam_t *param, int stride_height, int stride_width, int dilation_height, int dilation_width, int pad_height, int pad_width)

A function.

Description

This function fills cnmlConvOpParam_t struct with the convolution operation parameters input by the user, and return to the user.

This function allocates param memory, and after usage is done, the user needs to call cnmlDestroyConvOpParam destroy param parameters.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of the struct of the convolution operator operation parameter.
- `[in] stride_height`: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- `[in] stride_width`: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- `[in] dilation_height`: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in height dimension.
- `[in] dilation_width`: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in weight dimension.
- `[in] pad_height`: Input. Pad length.
- `[in] pad_width`: Input. Pad width.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer. For more information, see “Error Codes” section in this guide.

4.31.2 cnmlCreateConvOpParam_V2

cnmlStatus_t **cnmlCreateConvOpParam_V2**(cnmlConvOpParam_t *param, int stride_height, int stride_width, int dilation_height, int dilation_width, int pad_top, int pad_bottom, int pad_left, int pad_right)

A function.

Description

This function fills cnmlConvOpParam_t struct with the convolution operation parameters input by the user, and return to the user.

This function allocates param memory, and after usage is done, the user needs to call cnmlDestroyConvOpParam destroy param parameters.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of the struct of the convolution operator operation parameter.
- `[in] stride_height`: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- `[in] stride_width`: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- `[in] dilation_height`: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in height dimension.
- `[in] dilation_width`: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in weight dimension.
- `[in] pad_top`: Input. Pad_top the top of length, the default value is 0.
- `[in] pad_bottom`: Input. Pad_bottom the bottom of length, the default value is 0.
- `[in] pad_left`: Input. Pad_left the left of width , the default value is 0.
- `[in] pad_right`: Input. Pad_right the right of width, the default value is 0.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer. For more information, see “Error Codes” section in this guide.

4.31.3 cnmlCreateNdConvParam

cnmlStatus_t `cnmlCreateNdConvParam`(cnmlNdConvParam_t *param, int array_length, int dilations[], int strides[], int paddings[][2])

A function.

Creates a nd-convolution param object for holding the structure of the nd-convolution operator.

Notes:

- conv2d, conv_depthwise, conv_group support MLU220,MLU270,1M20 and 1M70.
- conv3d, convfirst3d only support MLU220 and MLU270.

Parameters

- `[out] param`: Output. A pointer to the struct of the nd-convolution operation parameter. operation parameter.
- `[in] dilations`: Input. An integer array specifies the dilation factor of kernel. The array values should be greater than or equal to 0, and values should be setted by NCDHW order.
- `[in] strides`: Input. An integer array specifies the stride factor of kernel. The array values should be greater than or equal to 0, and values should be setted by NCDHW order.
- `[in] paddings`: Input. An integer array. The dimensions of the tensor are integer tensors with shape [Dn, 2], where n is the ndims of the input, and the array value should be setted by NCDHW order. For each dimension D of the input, paddings[D, 0] specifies the number of extra zeros concatenated at the start of the input in that dimension. The paddings[D, 1] specifies the number of extra zeros concatenated at the end of the input in that dimension. The following formula specifies the padding size of each dimension D of the output. paddings(D, 0) + input.dim_size(D) + paddings(D, 1)

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The param pointer is NULL.

4.31.4 cnmlCreateNdConvParam_V2

cnmlStatus_t `cnmlCreateNdConvParam_V2`(cnmlNdConvParam_t *param, cnmlDataOrder_t data_order, int dilations[], int strides[], int paddings[][2])

A function.

Creates a nd-convolution param object according to data order.

Notes:

- conv2d, conv_depthwise, conv_group support MLU220,MLU270,1M20 and 1M70.
- conv3d, convfirst3d only support MLU220 and MLU270.

Parameters

- `[out] param`: Output. A pointer to the struct of the nd-convolution operation parameter. operation parameter.
- `[in] date_order`: input. An enum variable indicating the order of param’ s value, only NCHW, NHWC, NCDHW, and NDHWC are supported currently. And the length of dilations, strides and paddings should match the length of data_order.
- `[in] dilations`: Input. An integer array specifies the dilation factor of kernel. The array values should be greater than or equal to 0.
- `[in] strides`: Input. An integer array specifies the stride factor of kernel. The array values should be greater than or equal to 0.
- `[in] paddings`: Input. An integer array. The dimensions of the tensor are integer tensors with shape [Dn, 2], where n is the ndims of the input. For each dimension D of the input, paddings[D, 0] specifies the number of extra zeros concatenated at the start of the input in that dimension. The paddings[D, 1] specifies the number of extra zeros concatenated at the end of the input in that dimension. The following formula specifies the padding size of each dimension D of the output. paddings(D, 0) + input.dim_size(D) + paddings(D, 1)

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The param pointer is NULL.

4.31.5 cnmlDestroyConvOpParam

cnmlStatus_t `cnmlDestroyConvOpParam`(cnmlConvOpParam_t *param)

A function.

Description

Free the struct pointer of convolution operator operation parameter according to the pointer given by the user.

At the end of the convolution operator operation, the struct pointer of the convolution operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of the struct of the convolution operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed. For more information, see “Error Codes” section in this guide.

4.31.6 cnmlDestroyNdConvParam

cnmlStatus_t `cnmlDestroyNdConvParam`(cnmlNdConvParam_t *param)

A function.

Destroys the previously created nd-convolution operator param descriptor.

At the end of the nd-convolution operator operation, the struct pointer of the nd-convolution operator operation parameter is freed.

Notes:

- conv2d, conv_depthwise, conv_group support MLU220,MLU270,1M20 and 1M70.
- conv3d, convfirst3d only support MLU220 and MLU270.

Parameters

- `[in] param`: Input. A pointer to the nd-convolution operator param you want to destroy.

Return Value

- `CNML_STATUS_SUCCESS`: The descriptor destroyed successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions are met:
  - The param pointer is NULL.
  - The nd-convolution operator param that the pointer pointed has already destroyed.

4.31.7 cnmlCreateConvOp

cnmlStatus_t `cnmlCreateConvOp`(cnmlBaseOp_t *op, cnmlConvOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor)

A function.

Description

Deprecated. This interface will be deleted in next version and cnmlCreateConvOpForward is recommended to use.

According to the base operator pointer given by the user, a convolution operator is created.

After a pointer pointing to the address of base operator, the operation parameter of convolution operator and input-output tensor are created, they are introduced into the function to create the convolution operator.

Before the convolution operator is created, a pointer pointing to the address of the convolution operator parameter struct is declared, and the pointer and required operator parameter are introduced to the function to set the operator parameter.

A simple 2-dimensional convolution can be seen as a process that a 2-dimensional convolution kernel (weight matrix) slides on 2-dimensional input data, matrix multiplication is performed on some of the elements currently input, and then the results are summed into a single input pixel. The convolution kernel repeats this process until it traverses the entire picture and converts a 2-dimensional matrix into another. The special operation padding is equivalent to filling the edge of the input data with 0 (filling padding_height/2 0 in the height direction, padding_weight/2 0 in the weight direction), and stride refers to the sliding step of the convolution kernel.

The n-dimension and c-dimension generally refer to batch and channel. The general convolution operation (4-dimensional convolution) can be seen as a process that the input and weight of a batch is taken, and convolution operation is performed on the 2-dimensional inputs and convolution kernels of different channels, the computation results of different channels are added to get the output of the batch, after all the batches are completed, the bias is added to get the output.

hf <= hi,wf <= wi

if (dilation_height > 1 || dilation_width > 1) pad_height == 0 && pad_width == 0

The length of the weight in the Height direction must be less than or equal to the length of the input in the Height direction. The length of the weight in the Width direction must be less than or equal to the length of the input in the Width direction.

If dilation factor of Height dimension or dilation factor of Width dimension is greater than 1, the pad length in the Height direction and the Width direction is greater than 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer struct of convolution operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, hi, wi, ci],supporting data of int8, int16, float16 or float32 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, ho, wo, co](no = ni),supporting data of int8, int16, float16, float32 type. In channel quant, output’ s tensor data type should not be int8, int16.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, hf, wf, cf] (nf = co, cf = ci),supporting data of int8, int16, float16 or float32 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, hb, wb, cb] (nb = 1, hb = 1, wb = 1, cb = co), supporting data of float16, float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by the bias tensor is null.

4.31.8 cnmlCreateNdConvOp

cnmlStatus_t `cnmlCreateNdConvOp`(cnmlBaseOp_t *op, cnmlConvMode_t conv_mode, cnmlNdConvParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor, cnmlTensor_t mean_tensor, cnmlTensor_t stdt_tensor)

A function.

According to the base operator pointer given by the user, a nd-convolution operator is created.

After a pointer pointing to the address of base operator, the operation parameter of nd-convolution operator and input-output tensor are created, they are introduced into the function to create the nd-convolution operator.

Before the nd-convolution operator is created, a pointer pointing to the address of the nd-convolution operator parameter struct is declared, and the pointer and required operator parameter are introduced to the function to set the operator parameter.

A simple 3-dimensional convolution can be seen as a process that a 3-dimensional convolution kernel (weight matrix) slides on 3-dimensional input data, matrix multiplication is performed on some of the elements currently input, and then the results are summed into a single input pixel. The nd-convolution kernel repeats this process until it traverses the entire picture and converts a 3-dimensional matrix into another. The special operation padding is equivalent to filling the edge of the input data with 0 , and stride refers to the sliding step of the nd-convolution kernel.

The n-dimension and c-dimension generally refer to batch and channel, and d-dimension generally refer to depth. The general nd-convolution operation (5-dimensional convolution) can be seen as a process that the input and weight of a batch is taken, and nd-convolution operation is performed on the 3-dimensional inputs and nd-convolution kernels of different channels, the computation results of different channels are added to get the output of the batch, after all the batches are completed, the bias is added to get the output.

The length of the weight in the Height direction must be less than or equal to the length of the input in the Height direction. The length of the weight in the Width direction must be less than or equal to the length of the input in the Width direction.

Notes:

- conv2d, conv_depthwise, conv_group support MLU220,MLU270,1M20 and 1M70.
- conv3d, convfirst3d only support MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer struct of nd-convolution operation.
- `[in] input_tensor`: Input. A 5-dimensional MLU input tensor, the shape is [ni, di, hi, wi, ci].
- `[in] output_tensor`: Input. A 5-dimensional MLU output tensor, the shape is [no, do, ho, wo, co](no = ni). In channel quant, output' s tensor data type should not be int8, int16.
- `[in] filter_tensor`: Input. A 5-dimensional MLU weight tensor, the shape is [nf, df, hf, wf, cf] (nf = co, cf = ci).
- `[in] bias_tensor`: Input. A 5-dimensional MLU bias tensor, the shape is [nb, db, hb, wb, cb] (nb = 1, db, hb = 1, wb = 1, cb = co).
- `[in] mean_tensor`: Input. A 5-dimensional MLU mean tensor.
- `[in] stdt_tensor`: Input. A 5-dimensional MLU stdt tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by the bias tensor is null.

4.31.9 cnmlCreateConvOpForward

This API has the same function as cnmlCreateConvOp. For detailed information, see cnmlCreateConvOp.

4.31.10 cnmlComputeConvOpForward_V3

cnmlStatus_t `cnmlComputeConvOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeConvOpForward_V4 is recommended to use.

Computing user-specified convolution operators on MLU.

After convolution operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute convolution operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.31.11 cnmlComputeConvOpForward_V4

cnmlStatus_t cnmlComputeConvOpForward_V4(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Computing user-specified convolution operators on MLU.

After convolution operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute convolution operator.

Summary

If the shape of an input tensor is [batch, in_height, in_width, in_channels] and the shape of a filter

tensor or a kernel tensor is [filter_height, filter_width, in_channels, out_channels], with the default NHWC format.

output[b, i, j, k] = sum_{di, dj, q} input[b, strides[1] * i + di, strides[2] * j + dj, q] * filter[di, dj, q, k]

Ph = p_top + p_botton

Pw = p_left + p_right

Ho = (Hi + Ph - dh(Kh - 1) -1) / Sh + 1

Wo = (Wi + Pw - dw(Kw - 1) -1)/ Sw + 1

co = K

where di is the dialation in h dimension, dj is the dialation in w dimension, q is control variable in ci dimension, k is the variable in co dimension, Ph is the size of the padding in h dimension, Pw is the size of the padding in w dimension, Ho is the height of the output tensor, Wo is the width of the output tensor, co is the channel of the output tensor, Hi is the height of the input tensor, dh is the dilation in h dimension, Kh is the height of the kernel Sh is height of the stride, Wi is the width of the input tensor, dw is the dilation in w dimension, Kw is the width of the kernel, Sw is the stride in w dimension, and k is the number of channels for output.

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

bias_type : Data type of the bias tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

bias_oc_type : Data type of the bias tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_type is int8, then input_type can be int8 or int16, and output_type can be float16, float32, or int16.

If filter_type is int16, then input_type can be int16, and output_type can be float16, float32, or int16.

Notes: The data type you set in bias_oc_type, bias_type, and out_oc_type must be the same.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filter_type - filter_oc_type - out_oc_type - out_type

Supported combinations are:

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

kh <= hi，kw <= wi

where `kh` is the height of the kernel, `hi` is the height of the input tensor, `kw` is width of the kernel, and `wi` is the width of the input tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

For best practice, we suggest you call the cnmlComputeReshapeOpForward_V4() API to reshape the tensor, if you set the input tensor with the following:

- The size of ci dimension is less than 64.
- The size of h dimension of the input tensor is greater than 200.
- The size of w dimension is greater than 200.

Also, when you set the output tensor of the cnmlComputeReshapeOpForward_V4() call, the size of h and w dimensions should be less than or equal to 200 and the value of ci dimension should be greater or equal to 64.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.31.12 cnmlComputeNdConvOpForward_V2

cnmlStatus_t `cnmlComputeNdConvOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified nd-convolution operators on MLU.

After the nd-convolution operator, input, output, and computational queue are created, they are introduced into the function to compute the nd-convolution operator.

Notes:

- conv2d, conv_depthwise, conv_group support MLU220,MLU270,1M20 and 1M70.
- conv3d, convfirst3d only support MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.

- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.32 Conv Depthwise Operation

4.32.1 cnmlCreateConvDepthwiseOpParam

cnmlStatus_t **cnmlCreateConvDepthwiseOpParam**(cnmlConvDepthwiseOpParam_t *param, int stride_height, int stride_width)

A function.

According to the pointer given by the user, the function creates a struct of the deep convolution operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of the deep convolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.32.2 cnmlCreateConvDepthwiseOpParam_V2

cnmlStatus_t **cnmlCreateConvDepthwiseOpParam_V2**(cnmlConvDepthwiseOpParam_t *param, int stride_height, int stride_width, int pad_height, int pad_width)

A function.

According to the pointer given by the user, the function creates a struct of the deep convolution operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of the deep convolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] pad_height: Input. Pad length.
- [in] pad_width: Input. Pad width.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.32.3 cnmlCreateConvDepthwiseOpParam_V3

cnmlStatus_t **cnmlCreateConvDepthwiseOpParam_V3**(cnmlConvDepthwiseOpParam_t *param, int stride_height, int stride_width, int dilation_height, int dilation_width, int pad_height, int pad_width)

A function.

According to the pointer given by the user, the function creates a struct of the deep convolution operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of the deep convolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] dilation_height: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in height dimension.
- [in] dilation_width: Input. A value greater than or equal to 1, the dilation factor of kernel (that is, weight tensor) in weight dimension.
- [in] pad_height: Input. Pad length.

- `[in] pad_width`: Input. Pad width.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.32.4 cnmlDestroyConvDepthwiseOpParam

cnmlStatus_t `cnmlDestroyConvDepthwiseOpParam`(cnmlConvDepthwiseOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the operation parameter of the deep convolution operator is freed.

After the operation of deep convolution operator is finished, the created struct pointer of deep convolution operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the operation parameter of the deep convolution operator.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.32.5 cnmlCreateConvDepthwiseOp

cnmlStatus_t `cnmlCreateConvDepthwiseOp`(cnmlBaseOp_t *op, cnmlConvDepthwiseOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor)

A function.

According to the basic operator pointer given by the user, a deep convolution operator is created.

Deprecated. This interface will be deleted in next version and cnmlCreateConvDepthwiseOpForward is recommended to use.

If pad_height and pad_weight is not 0, the input data edges need to be filled, that is, pad_height/2 0 needs to be filled in height direction, pad_weight/2 0 needs to be filled in weight direction. The arrangement order of input, output, and weight during computation is NCHW. The summation is performed on the subscripts h and w of the filter, and the summation range is [i * stride_height, min {i * stride_height + hf, hi}], the sum range of w is [j * stride_weight, min {j * stride_weight + jf, ji}]. If the bias tensor is not null, the final output should add bias. Quantization for filter tensor and input tensor, and channel quantization for filter tensor are supported. Computing performance could be better after setting quantization parameters, but precision could be lower. Setting channel quantization could improve precision. About setting quantization parameters, for more information, see Cambricon CNML User Guide.

Summary

For input[ni, ci, hi, wi], output[no, co, ho, wo], filter[nf, cf, hf, wf] and

bias[nb, cb, hb, wb], multiplier = co / ci

output[n, k * multiplier + q, i, j] = sum_{di, dj} input[n, stride_h * i + di, stride_w * j + dj, k] * filter[1, k * multiplier + q, di, dj] + bias[1, k * multiplier + q, 1, 1]

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale limitation:

MLU270:

if data_type = float16 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 26 * 26

if data_type = float32 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 18 * 18

Performance Optimization

a. The number of bytes in the C dimension is a multiple of 128.
b. multiplier = 1

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A deep convolution operation struct pointer.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi],supporting data of float16 and float32 type

- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](co = ci * multiplier),supporting data of float16 and float32 type.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, cf, hf, wf] (nf = 1),supporting data of float16 and float32 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, cb, hb, wb] (nb = 1, hb = 1, wb = 1, cb = co),supporting data of float16 and float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.32.6 cnmlCreateConvDepthwiseOpForward

This API has the same function as cnmlCreateConvDepthwiseOp. For detailed information, see cnmlCreateConvDepthwiseOp.

4.32.7 cnmlComputeConvDepthwiseOpForward_V3

cnmlStatus_t `cnmlComputeConvDepthwiseOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computing user-specified deep convolution operator on MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeConvDepthwiseOpForward_V4 is recommended to use.

After the deep convolution operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the deep convolution operator. Quantization for filter tensor and input tensor, and channel quantization for filter tensor are supported. Computing performance could be better after setting quantization parameters, but precision could be lower. Setting channel quantization could improve precision. About setting quantization parameters, for more information, see Cambricon CNML User Guide.

Summary

For input[ni, ci, hi, wi], output[no, co, ho, wo], filter[nf, cf, hf, wf] and bias[nb, cb, hb, wb], multiplier = co / ci

output[n, k * multiplier + q, i, j] = sum_{di, dj} input[n, stride_h * i + di, stride_w * j + dj, k] * filter[1, k * multiplier + q, di, dj] + bias[1, k * multiplier + q, 1, 1]

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale Limitation

MLU270:

if data_type = float16 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 26 * 26

if data_type = float32 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 18 * 18

Performance Optimization

a. The number of bytes in the C dimension is a multiple of 128.
b. multiplier = 1

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.32.8 cnmlComputeConvDepthwiseOpForward_V4

cnmlStatus_t `cnmlComputeConvDepthwiseOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified deep convolution operator on MLU.

After the deep convolution operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the deep convolution operator.

Summary

For input[ni, ci, hi, wi], output[no, co, ho, wo], filter[nf, cf, hf, wf] and

bias[nb, cb, hb, wb], multiplier = co / ci

output[n, k * multiplier + q, i, j] = sum_{di, dj} input[n, stride_h * i + di, stride_w * j + dj, k] * filter[1, k * multiplier + q, di, dj] + bias[1, k * multiplier + q, 1, 1]

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale limitation:

MLU270:

if data_type = float16 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 26 * 26

if data_type = float32 :

multiplier = 1, kh * kw * (ci > 256 ? 4 : ci / 64) <= 18 * 18

Performance Optimization

a. The number of bytes in the C dimension is a multiple of 128.
b. multiplier = 1

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.33 Conv First Operation

4.33.1 cnmlCreateConvFirstOpParam

cnmlStatus_t `cnmlCreateConvFirstOpParam`(cnmlConvFirstOpParam_t *param, int stride_height, int stride_width, int pad_l, int pad_r, int pad_t, int pad_b)

A function.

According to the pointer given by the user, the function creates a parameter struct of ConvFirst operator, and fills in the struct with parameters by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the parameter struct of the ConvFirst operator.
- `[in] stride_height`: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- `[in] stride_width`: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- `[in] pad_l`: Input. Size of padding on the left.

- [in] pad_r: Input. Size of padding on the right.
- [in] pad_t: Input. Size of padding on the top.
- [in] pad_b: Input. Size of padding on the bottom.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.33.2 cnmlCreateConvFirstOpParam_V2

cnmlStatus_t **cnmlCreateConvFirstOpParam_V2**(cnmlConvFirstOpParam_t *param, int stride_height, int stride_width, int dilation_height, int dilation_width, int pad_l, int pad_r, int pad_t, int pad_b)

A function.

According to the pointer given by the user, the function will create the parameter struct of ConvFirst operator, which should be filled by the user later. Comparing with cnmlCreateConvFirstOpParam() API, this API supports expanding the first convolution kernel in height and width direction.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the parameter of the ConvFirst operation.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] dilation_height: Input. A value greater than or equal to 1, to expand first convolution kernel in Height direction.
- [in] dilation_width: Input. A value greater than or equal to 1, to expand first convolution kernel in Width direction.
- [in] pad_l: Input. Size of padding on the left.
- [in] pad_r: Input. Size of padding on the right.
- [in] pad_t: Input. Size of padding on the top.
- [in] pad_b: Input. Size of padding on the bottom.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.33.3 cnmlDestroyConvFirstOpParam

cnmlStatus_t **cnmlDestroyConvFirstOpParam**(cnmlConvFirstOpParam_t *param)

A function.

Frees the memory that stores the parameter that is allocated with cnmlCreateConvFirstOpParam or cnmlCreateConvFirstOpParam_V2.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer pointing to parameter of the ConvFirst operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.33.4 cnmlCreateConvFirstOp

cnmlStatus_t **cnmlCreateConvFirstOp**(cnmlBaseOp_t *op, cnmlConvFirstOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t mean_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor, cnmlTensor_t std_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateConvFirstOpForward is recommended to use.

According to the base operator pointer given by the user, a ConvFirst operator is created.

After a pointer pointing to the address of the base operator, the operation parameter of the ConvFirst operator and the input-output tensor are created, they are introduced into the function to create the ConvFirst operator.

Before the creation of the ConvFirst operator, a pointer pointing to the parameter struct of the ConvFirst operator is declared and the required operator parameter is introduced into the function to set the operator parameter.

It includes the process of normalized (before computation, the mean value is subtracted from the input, the variance is divided, and the division by the variance is replaced with the multiplication by the reciprocal of variance in the actual computation) and the special convolution layer of performance optimization.

A simple 2-dimensional convolution can be seen as a process that a 2-dimensional convolution kernel (weight matrix) slides on 2-dimensional input data, matrix multiplication is performed on some of the elements currently input, and then the results are summed into a single input pixel. The convolution kernel repeats this process until it traverses the entire picture and converts a 2-dimensional matrix into another. The special operation padding is equivalent to filling the edge of the input data with 0 (filling padding_height/2 0 in the height direction, padding_weight/2 0 in the weight direction), and stride refers to the sliding step of the convolution kernel.

The n-dimension and c-dimension generally refer to batch and channel. The general convolution operation (4-dimensional convolution) can be seen as a process that the input and weight of a batch is taken, and convolution operation is performed on the 2-dimensional inputs and convolution kernels of different channels, the computation results of different channels are added to get the output of the batch, after all the batches are completed, the bias is added to get the output.

hf <= hi,wf <= wi

ci <= 4, co <= 224, wo > 1

The length of the weight in the Height direction must be less than or equal to the length of the input in the Height direction. The length of the weight in the Width direction must be less than or equal to the length of the input in the Width direction.

The dimension of input Feature is less than or equal to 4, the dimension of output Feature is less than or equal to 224, and the dimension of output Width is greater than 1.

Range of input values is [0,255].

Supports MLU220,MLU270,1M20,and 1M70.

Perfromance Optimization

For better performance, it is recommended that you set the size of C dimension to 4 in the framework layer.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer struct of convolution operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supporting only uint8-type data.
- `[in] mean_tensor`: Input. A 4-dimensional MLU mean tensor, the shape is [nm, cm, hm, wm] (nm = 1, hm = 1, wm = 1, cm = ci or nm = 1, hm = hi, wm = wi, cm = ci), supporting the data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni),supporting data of int8, int16, float16,float32 type. In channel quant, output' s tensor data type should not be int8, int16.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, cf, hf, wf] (nf = co, cf = ci),supporting data of int8, int16, float16 or float32 type
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, cb, hb, wb] (nb = 1, hb = 1, wb = 1, cb = co), supporting the data of float16 or float32 type.
- `[in] std_tensor`: Input. A 4-dimensional MLU variance tensor, the shape is [ns, cs, hs, ws] (ns = 1, hs = 1, ws = 1, cs = ci), supporting the data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op is a null pointer.
  - param is a null pointer.
  - input_tensor is a null pointer.
  - mean_tensor is a null pointer.
  - output_tensor is a null pointer.
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - CPU tensor bound by mean tensor is null.
  - when variance tensor is not null, the CPU tensor bound by variance tensor is null.
  - when bias tensor is not null, the CPU tensor bound by the bias tensor is null.

4.33.5 cnmlCreateConvFirstOpForward

This API has the same function as cnmlCreateConvFirstOp. For detailed information, see cnmlCreateConvFirstOp.

4.33.6 cnmlComputeConvFirstOpForward_V3

cnmlStatus_t `cnmlComputeConvFirstOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeConvFirstOpForward_V4 is recommended to use.

Computing user-specified ConvFirst operator on MLU.

After the ConvFirst operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the ConvFirst operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.33.7 cnmlComputeConvFirstOpForward_V4

cnmlStatus_t **cnmlComputeConvFirstOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified ConvFirst operator on MLU.

After the ConvFirst operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the ConvFirst operator.

Supports MLU220,MLU270,1M20,and 1M70.

Perfromance Optimization

For better performance, it is recommended that you set the size of C dimension to 4 in the framework layer.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.33.8 cnmlEnableConvFirstOpBgraMode

cnmlStatus_t **cnmlEnableConvFirstOpBgraMode**(cnmlBaseOp_t op)

A function.

This interface is a discarded interface and is not currently enabled.

4.33.9 cnmlEnableConvFirstOpFusionPadMode

cnmlStatus_t **cnmlEnableConvFirstOpFusionPadMode**(cnmlBaseOp_t op)

A function.

This interface is a discarded interface and is not currently enabled.

4.34 Conv Group Operation

4.34.1 cnmlCreateConvGroupOp

cnmlStatus_t **cnmlCreateConvGroupOp**(cnmlBaseOp_t *op, cnmlConvOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor, int group)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateConvGroupOpForward is recommended to use.

According to the base operator pointer given by the user, a grouping convolution operator is created.

Group convolution: firstly, the input and weight tensor are divided into several groups (the input is divided in the C dimension, the weight is divided in the N dimension), and the conv operation is performed on each group of input and weight correspondingly, and the computation result is sequentially spliced and output.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to the address of the base operator.
- [in] input_tensor: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi],supporting data of int8, int16, float16,float32 type.
- [in] output_tensor: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni), supporting data of float16, float32 type. In channel quant, output’ s tensor data type should not be int8, int16.

- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, cf, hf, wf] (nf = co, cf = ci),supporting data of int8, int16, float16,float32 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, cb, hb, wb] (nb = 1, hb = 1, wb = 1, cb = co),supporting data of float16, float32 type.
- `[in] group`: Input. Group number.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.34.2 cnmlCreateConvGroupOpForward

This API has the same function as cnmlCreateConvGroupOp. For detailed information, see cnmlCreateConvGroupOp.

4.34.3 cnmlComputeConvGroupOpForward_V3

cnmlStatus_t `cnmlComputeConvGroupOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeConvGroupOpForward_V4 is recommended to use.

Computing user-specified grouping convolution operator on MLU.

After the grouping convolution operator, input, output, parameter at runtime and computational queue are created, they are introduced into the function to compute grouping convolution operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.34.4 cnmlComputeConvGroupOpForward_V4

cnmlStatus_t `cnmlComputeConvGroupOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified grouping convolution operator on MLU.

After the grouping convolution operator, input, output, parameter at runtime and computational queue are created, they are introduced into the function to compute grouping convolution operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.35 Cos Operation

4.35.1 cnmlCreateCosOp

cnmlStatus_t `cnmlCreateCosOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateCosOp.

Create a cosine operator based on the base operator pointer given by the user. After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Cos operator.

Formula

out[n,c,h,w]=cos(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor. The shapes of input and output should be exactly the same.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op is empty.
  - input_tensor is empty.
  - output_tensor is empty.

4.35.2 cnmlComputeCosOpForward_V3

cnmlStatus_t `cnmlComputeCosOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeCosOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeCosOpForward_V4 is recommended to use. It is used to compute the user-specified cosine operator on the MLU.

Formula

out[n,c,h,w]=cos(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input parameter operator pointer op is empty.
  - Input parameter tensor pointer input is empty.
  - Output parameter tensor pointer output is empty.

4.35.3 cnmlComputeCosOpForward_V4

cnmlStatus_t `cnmlComputeCosOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeCosOpForward_V4.

It is used to compute the user-specified cosine operator on the MLU. Parameter explanation:

Formula

out[n,c,h,w]=cos(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.36 Cos Similarity Opreation

4.36.1 cnmlCreateCosSimilarityOp

cnmlStatus_t `cnmlCreateCosSimilarityOp`(cnmlBaseOp_t *op, `const` cnmlTensor_t inputTensorA, `const` cnmlTensor_t inputTensorB, `const` cnmlTensor_t outputTensor)

cnmlCreateCosSimilarityOp.

It creates a cos similarity operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, two inputs tensors and an output tensor, introduce them into function to creat a cos similarity operation.

Formula

Refer to: y = A.B/(||A||.||B||) = (Σ(Ai * Bi))/(sqrt(Σ(Ai^2)) * sqrt(Σ(Bi^2)))

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] inputA`: inputA. An MLU input tensor. Input data range should in [-1, 1]. if the input count exceeds 512, the error rate will increase.
- `[in] inputB`: inputB. An MLU input tensor. Input data range should in [-1, 1]. if the input count exceeds 512, the error rate will increase.
- `[in] output`: Output. An MLU address pointing to output tensor. The output shape is [1, 1, 1, 1].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - The input tensor type is either CNML_TENSOR or CNML_CONST.
  - The CPU tensor bound to the bias tensor is empty.

4.36.2 cnmlComputeCosSimilarityOpForward

cnmlStatus_t `cnmlComputeCosSimilarityOpForward`(cnmlBaseOp_t op, void *inputTensor1, void *inputTensor2, void *outputTensor, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeCosSimilarityOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeCosSimilarityOpForward_V2 is recommended to use.

It computes the user-specified cos similariy operation on the MLU.

After creating cos similarity operation, related parameters and computation stream, introduce them into the function to cumpute the cos similarity operation.

Formula

Refer to: y = A.B/(||A||.||B||) = (Σ(Ai * Bi))/(sqrt(Σ(Ai^2)) * sqrt(Σ(Bi^2)))

Supports MLU220 and MLU270.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input1`: Input1. An MLU address pointing to the input1 data, input range should in [-1, 1]. if the input count exceeds 512, the error rate will increase.
- `[in] input2`: Input2. An MLU address pointing to the input2 data, input range should in [-1, 1]. if the input count exceeds 512, the error rate will increase.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.36.3 cnmlComputeCosSimilarityOpForward_V2

cnmlStatus_t `cnmlComputeCosSimilarityOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input1, cnmlTensor_t input_tensor2, void *input2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeCosSimilarityOpForward_V2.

It computes the user-specified cos similariy operation on the MLU.

After creating cos similarity operation, related parameters and computation stream, introduce them into the function to cumpute the cos similarity operation.

Formula

Refer to: y = A.B/(||A||.||B||) = (Σ(Ai* Bi))/(sqrt(Σ(Ai^2)) * sqrt(Σ(Bi^2)))

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.37 Crop Operation

4.37.1 cnmlCreateCropOpParam

cnmlStatus_t `cnmlCreateCropOpParam`(cnmlCropOpParam_t *param, int start_index_n, int start_index_c, int start_index_h, int start_index_w, float space_number)

A function.

Constructing the param of the crop operator requires offset of the starting address in the four directions n, c, h, w.

Space_number is the number that needs to be complemented in four directions (currently, it is not supported to make a complement in each direction).

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A param pointer.
- `[in] start_index_n`: Input. The four int data are the start positions where the four dimensions are intercepted.
- `[in] start_index_c`: Input. The four int data are the start positions where the four dimensions are intercepted.
- `[in] start_index_h`: Input. The four int data are the start positions where the four dimensions are intercepted.
- `[in] start_index_w`: Input. The four int data are the start positions where the four dimensions are intercepted.
- `[in] space_number`: Input. A float type to fill up.

4.37.2 cnmlCreateNdCropOpParam

cnmlStatus_t `cnmlCreateNdCropOpParam`(cnmlNdCropOpParam_t *param, int dimNum, int *start)

A function.

Constructing the param of the crop operator requires offset of the starting address.

Space_number is the number that needs to be complemented in four directions (currently, it is not supported to make a complement in each direction).

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A param pointer.
- `[in] dimNum`: Input. Total dimNum of start.
- `[in] start`: Input. The start positions.

4.37.3 cnmlDestroyCropOpParam

cnmlStatus_t **cnmlDestroyCropOpParam**(cnmlCropOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of Grep operator operation parameter is freed.

At the end of Grep operator operation, the created struct pointer of Grep operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer pointing to the address of struct of Grep operator operation parameter.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.37.4 cnmlDestroyNdCropOpParam

cnmlStatus_t **cnmlDestroyNdCropOpParam**(cnmlNdCropOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of Grep operator operation parameter is freed.

At the end of Grep operator operation, the created struct pointer of Grep operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer pointing to the address of struct of nd Crop operator operation parameter.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.37.5 cnmlCreateCropOp

cnmlStatus_t **cnmlCreateCropOp**(cnmlBaseOp_t *op, cnmlCropOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the param input by the user, intercept the data from the param.start_index_n by the length of the output_Tensor_shape.n on the input_Tensor_shape.n, and perform the same operation in the C, H, W directions.

If only intercept in direction C(no = ni, ho = hi, wo = wi), it will call split operation actually.

If don’ t intercept in direction C(co = ci), it will call grep operation actually.

For other strategies, it will call stride_slice actually.

Formula

Intercept output tensor from each dimension of input tensor.

output[n, c, h, w] = input[start_index_n + n, start_index_c + c, start_index_h + h, start_index_w + w]

start_index_n is the start positions where the n dimensions are intercepted.

DataType

MLU270:

float16, float32, int32, uint8, int8

Scale Limitation

no <= (ni - start_index_n);

co <= (ci - start_index_c);

ho <= (hi - start_index_h);

wo <= (wi - start_index_w);

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] param: Input. Param of crop.
- [in] input_tensor: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.

- [in] output_tensor: Input. A 4-dimensional MLU input tensor, of which the shape is [no, co, ho, wo], supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: (At least one of) the following conditions is not met:
  - Op is empty.
  - Input_tensor is empty.
  - Output_tensor is empty.

4.37.6 cnmlCreateNdCropOp

cnmlStatus_t cnmlCreateNdCropOp(cnmlBaseOp_t *op, cnmlNdCropOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the param input by the user, intercept the data from the param.start_index_n by the length of the output_Tensor_shape.n on the input_Tensor_shape.n, and perform the same operation in the C, H, W directions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] param: Input. Param of nd crop.
- [in] input_tensor: Input. A n-dimensional MLU input tensor, of which the shape is array, supporting data of float16 type.
- [in] output_tensor: Input. A n-dimensional MLU input tensor, of which the shape is arry, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: (At least one of) the following conditions is not met:
  - Op is empty.
  - Input_tensor is empty.
  - Output_tensor is empty.

4.37.7 cnmlComputeCropOpForward_V3

cnmlStatus_t cnmlComputeCropOpForward_V3(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCropOpForward_V4 is recommended to use.

It is used to compute the user-specified threshold operator on the MLU.

Formula

Intercept output tensor from each dimension of input tensor.

output[n, c, h, w] = input[start_index_n + n, start_index_c + c, start_index_h + h, start_index_w + w]

start_index_n is the start positions where the n dimensions are intercepted.

DataType

MLU270:

float16, float32, int32, uint8, int8

Scale Limitation

no <= (ni - start_index_n);

co <= (ci - start_index_c);

ho <= (hi - start_index_h);

wo <= (wi - start_index_w);

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address that points to the output position.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.
- [in] compute_forw_param: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is not met:
  - Input parameter operator pointer op is empty.
  - Input parameter tensor pointer input is empty.

- Output parameter tensor pointer output is empty.

4.37.8 cnmlComputeCropOpForward_V4

cnmlStatus_t `cnmlComputeCropOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

It is used to compute the user-specified crop operator on the MLU.

Formula

Intercept output tensor from each dimension of input tensor.

output[n, c, h, w] = input[start_index_n + n, start_index_c + c, start_index_h + h, start_index_w + w]

start_index_n is the start positions where the n dimensions are intercepted.

DataType

MLU270:

float16, float32, int32, uint8, int8

Scale Limitation

no <= (ni - start_index_n);

co <= (ci - start_index_c);

ho <= (hi - start_index_h);

wo <= (wi - start_index_w);

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.37.9 cnmlComputeNdCropOpForward

cnmlStatus_t `cnmlComputeNdCropOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

It is used to compute the user-specified threshold operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Reason1 The task type of runtime is invalid.

4.38 Customized Active Operation

4.38.1 cnmlCreateCustomizedActiveOpParam

cnmlStatus_t `cnmlCreateCustomizedActiveOpParam`(cnmlCustomizedActiveOpParam_t *param, float x_start, float x_end, float y_min, int segment_num)

A function.

This function fills in the cnmlCustomizedActiveOpParam_t struct with the operation parameters input by the user and returns it to the user.

This function allocates param memory. After the usage is completed, the user needs to call cnmlDestroycnmlCustomizedActiveOpParam to destroy the param parameter at the appropriate time.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] param`: Output. A pointer pointing to the address of the struct of the convolution operator operation parameter.
- `[in] x_start`: Input. The range of independent variables of activation function.
- `[in] x_end`: Input. The range of independent variables of activation function.
- `[in] y_min`: Input. The infimum of activation function within the effective range (x_start, y_start).
- `[in] segment_num`: Input. The number of segment of the activation function. When implemented, piecewise linear interpolation is used to approximate the activation function.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.38.2 cnmlDestroyCustomizedActiveOpParam

cnmlStatus_t `cnmlDestroyCustomizedActiveOpParam`(cnmlCustomizedActiveOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the Active operator operation parameter is freed.

After the operation of the customized active operator is completed, the struct pointer of the Active operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the Active operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.38.3 cnmlCreateCustomizedActiveOp

cnmlStatus_t `cnmlCreateCustomizedActiveOp`(cnmlBaseOp_t *op, void *active_func, cnmlCustomizedActiveOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

This function creates a CustomizedActive operator according to the basic Operator pointer given by the user.

After the pointer pointing to the base operator, the CustomizedActive operator operation parameter and the input-output tensor are created, they are introduced into the function to create the CustomizedActive operator.

The shapes of input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] input`: Input. A 1 to n-dimensional tensor, supporting data of float16 type.
- `[in] output`: Input. A 1 to n-dimensional tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.38.4 cnmlComputeCustomizedActiveForward_V3

cnmlStatus_t `cnmlComputeCustomizedActiveForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCustomizedActiveForward_V4 is recommended to use.

Computing the CustomizedActive operator specified by the user on MLU.

After the CustomizedActive operators, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute CustomizedActive operators.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.38.5 cnmlComputeCustomizedActiveForward_V4

cnmlStatus_t `cnmlComputeCustomizedActiveForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing the CustomizedActive operator specified by the user on MLU.

After the CustomizedActive operators, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute CustomizedActive operators.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.39 Cycle Add Operation

4.39.1 cnmlCreateCycleAddOp

cnmlStatus_t `cnmlCreateCycleAddOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleAdd operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleAdd operator.

Formula

c[n c h w] = a[n c h w] + b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w] .

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.39.2 cnmlCreateNdCycleAddOp

cnmlStatus_t `cnmlCreateNdCycleAddOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleAdd operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleAdd operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.39.3 cnmlComputeCycleAddOpForward_V3

cnmlStatus_t `cnmlComputeCycleAddOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleAddOpForward_V4 is recommended to use.

Compute the CycleAdd operator on the CPU.

Transmit the created CycleAdd operator, input tensor, input address, output tensor, and output address to the function to compute the CycleAdd operator.

Formula

c[n c h w] = a[n c h w] + b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.39.4 cnmlComputeCycleAddOpForward_V4

cnmlStatus_t `cnmlComputeCycleAddOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleAdd operator on the MLU.

Transmit the created CycleAdd operator, input tensor, input address, output tensor, and output address to the function to compute the CycleAdd operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.39.5 cnmlComputeNdCycleAddOpForward

cnmlStatus_t `cnmlComputeNdCycleAddOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleAdd operator on the MLU.

Transmit the created CycleAdd operator, input tensor, input address, output tensor, and output address to the function to compute the CycleAdd operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.40 Cycle And Operation

4.40.1 cnmlCreateCycleAndOp

cnmlStatus_t `cnmlCreateCycleAndOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1 and then perform element-wise And operation.

Formula

c[n c h w] = (a[n c h w] !=0 && (b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [1, c, 1, 1], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [n, c, h, w], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.40.2 cnmlCreateNdCycleAndOp

cnmlStatus_t `cnmlCreateNdCycleAndOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1 and then perform element-wise And operation.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.40.3 cnmlComputeCycleAndOpForward_V3

cnmlStatus_t `cnmlComputeCycleAndOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleAndOpForward_V4 is recommended to use.

Compute the CycleAnd operator specified by users on the MLU.

After creating a CycleAnd operator, input, output, and computation stream, pass them into the function to compute the CycleAnd operator.

Formula

c[n c h w] = (a[n c h w] !=0 && (b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.

- [in] op: Input. A pointer which points to base operators.
- [in] input_1: Input. An MLU address pointing to input data 1.
- [in] input_2: Input. An MLU address pointing to input data 2.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.40.4 cnmlComputeCycleAndOpForward_V4

cnmlStatus_t **cnmlComputeCycleAndOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleAnd operator specified by users on the MLU.

After creating a CycleAnd operator, input, output, and computation queue, pass them into the function to compute the CycleAnd operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.40.5 cnmlComputeNdCycleAndOpForward

cnmlStatus_t **cnmlComputeNdCycleAndOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleAnd operator specified by users on the MLU.

After creating a NdCycleAnd operator, input, output, and computation queue, pass them into the function to compute the NdCycleAnd operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.41 Cycle Equal Operation

4.41.1 cnmlCreateCycleEqualOp

cnmlStatus_t `cnmlCreateCycleEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleEqual operator.

Formula

c[n c h w] = a[n c h w] == b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.41.2 cnmlCreateNdCycleEqualOp

cnmlStatus_t `cnmlCreateNdCycleEqualOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.41.3 cnmlComputeCycleEqualOpForward_V3

cnmlStatus_t `cnmlComputeCycleEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleEqualOpForward_V4 is recommended to use.

Compute the CycleEqual operator on the CPU.

Transmit the created CycleEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleEqual operator.

Formula

c[n c h w] = a[n c h w] == b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.41.4 cnmlComputeCycleEqualOpForward_V4

cnmlStatus_t `cnmlComputeCycleEqualOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleEqual operator on the MLU.

Transmit the created CycleEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleEqual operator.

Formula

c[n c h w] = a[n c h w] == b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.41.5 cnmlComputeNdCycleEqualOpForward

cnmlStatus_t `cnmlComputeNdCycleEqualOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleEqual operator on the MLU.

Transmit the created NdCycleEqual operator, input tensor, input address, output tensor, and output address to the function to compute the NdCycleEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.

- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.42 Cycle Greater Operation

4.42.1 cnmlCreateCycleGreaterOp

cnmlStatus_t cnmlCreateCycleGreaterOp(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is greater than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] > B[1][1][1][ci])? 1: 0.

The shape of the four-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Formula

c[n c h w] = a[n c h w] > b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A four-dimensional MLU input tensor, the shape is [n, c, h, w], supports data of float16 type.
- [in] input_tensor_2: Input. A four-dimensional MLU input tensor, the shape is [1, 1, 1, c], That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1, supports data of float16 type.
- [in] output_tensor: Input. A four-dimensional tensor, the shape is the same as that of the input tensor A, the data type is float16.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally. Throw exceptions when the function fails to execute.
- CNML_STATUS_INVALIDPARAM:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.42.2 cnmlCreateNdCycleGreaterOp

cnmlStatus_t cnmlCreateNdCycleGreaterOp(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is greater than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] > B[1][1][1][ci])? 1: 0.

The shape of the n-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type. The channel dimension of B is the same as that of A, and the remaining dimensions are 1.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.42.3 cnmlComputeCycleGreaterOpForward_V3

cnmlStatus_t `cnmlComputeCycleGreaterOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleGreaterOpForward_V4 is recommended to use.

Perform a comparison on the user-specified four-dimensional tensor A and one-dimensional tensor B to find whether one is greater than the other.

Formula

c[n c h w] = a[n c h w] > b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.42.4 cnmlComputeCycleGreaterOpForward_V4

cnmlStatus_t `cnmlComputeCycleGreaterOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and one-dimensional tensor B to find whether one is greater than the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.42.5 cnmlComputeNdCycleGreaterOpForward

cnmlStatus_t **cnmlComputeNdCycleGreaterOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified multi-dimensional tensor A and one-dimensional tensor B to find whether one is greater than the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.43 Cycle Great Equal Operation

4.43.1 cnmlCreateCycleGreaterEqualOp

cnmlStatus_t **cnmlCreateCycleGreaterEqualOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is greater than or equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] >= B[1][1][1][ci])? 1: 0.

The shape of the four-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Formula

c[n c h w] = a[n c h w] >= b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A four-dimensional MLU input tensor, the shape is [n, c, h, w], supports data of float16 type.
- [in] input_tensor_2: Input. A four-dimensional MLU input tensor, the shape is [1, 1, 1, c], That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1, supports data of float16 type.
- [in] output_tensor: Input. A four-dimensional tensor, the shape is the same as that of the input tensor A, the data type is float16.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally. Throw exceptions when the function fails to execute.
- CNML_STATUS_INVALIDPARAM:
  - Reason1 The operator pointer is null.

- Reason2 The input and output pointer is null.

4.43.2 cnmlCreateNdCycleGreaterEqualOp

cnmlStatus_t `cnmlCreateNdCycleGreaterEqualOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is nd greater than or equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] >= B[1][1][1][ci])? 1: 0.

The shape of the four-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type. The channel dimension of B is the same as that of A, and the remaining dimensions are 1.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.43.3 cnmlComputeCycleGreaterEqualOpForward_V3

cnmlStatus_t `cnmlComputeCycleGreaterEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleGreaterEqualOpForward_V4 is recommended to use.

Perform a comparison on the user-specified four-dimensional tensor A and one-dimensional tensor B to find whether one is greater than or equal to the other.

Formula

c[n c h w] = a[n c h w] >= b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.43.4 cnmlComputeCycleGreaterEqualOpForward_V4

cnmlStatus_t `cnmlComputeCycleGreaterEqualOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and one-dimensional tensor B to find whether one is greater than or equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.43.5 cnmlComputeNdCycleGreaterEqualOpForward

cnmlStatus_t `cnmlComputeNdCycleGreaterEqualOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and one-dimensional tensor B to find whether one is nd greater than or equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.44 Cycle Less Operation

4.44.1 cnmlCreateCycleLessOp

cnmlStatus_t `cnmlCreateCycleLessOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is less than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] < B[1][1][1][ci])? 1:0

The shape of the four-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Formula

c[n c h w] = a[n c h w] < b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape is [n, h, w, c], supports data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape is [1, 1, 1, c], That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1, supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional tensor, the shape is the same as that of the input tensor A, the data type is float16.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.44.2 cnmlCreateNdCycleLessOp

cnmlStatus_t `cnmlCreateNdCycleLessOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the vector B and the tensor A in each dimension to find whether one is less than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] < B[1][1][1][ci])? 1:0

The shape of the four-dimensional tensor B must be [1, 1, 1, c].

That is, the channel dimension of B is the same as that of A, and the remaining dimensions are 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type. The channel dimension of input_2 is the same as that of input_1, and the remaining dimensions are 1.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type. Input. A four-dimensional tensor, the shape is the same as that of the input tensor A, the data type is float16.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.44.3 cnmlComputeCycleLessOpForward_V3

cnmlStatus_t `cnmlComputeCycleLessOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleLessOpForward_V4 is recommended to use.

Compare the size of the user-specified vector B and tensor A in each dimension.

Formula

c[n c h w] = a[n c h w] < b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.44.4 cnmlComputeCycleLessOpForward_V4

cnmlStatus_t `cnmlComputeCycleLessOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compare the size of the user-specified vector B and tensor A in each dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.44.5 cnmlComputeNdCycleLessOpForward

cnmlStatus_t `cnmlComputeNdCycleLessOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compare the size of the user-specified vector B and tensor A in each dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.45 Cycle Less Equal Operation

4.45.1 cnmlCreateCycleLessEqualOp

cnmlStatus_t `cnmlCreateCycleLessEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleLessEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleLessEqual operator.

Formula

c[n c h w] = a[n c h w] <= b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.45.2 cnmlCreateNdCycleLessEqualOp

cnmlStatus_t `cnmlCreateNdCycleLessEqualOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleLessEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleLessEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.45.3 cnmlComputeCycleLessEqualOpForward_V3

cnmlStatus_t `cnmlComputeCycleLessEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleLessEqualOpForward_V4 is recommended to use.

Compute the CycleLessEqual operator on the MLU.

Transmit the created CycleLessEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleLessEqual operator.

Formula

c[n c h w] = a[n c h w] <= b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.45.4 cnmlComputeCycleLessEqualOpForward_V4

cnmlStatus_t `cnmlComputeCycleLessEqualOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleLessEqual operator on the MLU.

Transmit the created CycleLessEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleLessEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.

- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.45.5 cnmlComputeNdCycleLessEqualOpForward

cnmlStatus_t cnmlComputeNdCycleLessEqualOpForward(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleLessEqual operator on the MLU.

Transmit the created NdCycleLessEqual operator, input tensor, input address, output tensor, and output address to the function to compute the NdCycleLessEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.46 Cycle Max Equal Operation

4.46.1 cnmlCreateCycleMaxEqualOp

cnmlStatus_t cnmlCreateCycleMaxEqualOp(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an cycle max_equal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the cycle max_equal operator, pass them into the function to create the cycle max_equal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] >= input2[1, c, 1, 1] ? input1 : input2

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16/float32 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [1, c, 1, 1], supporting data of float16/float32 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [n, c, h, w], supporting data of float16/float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.46.2 cnmlCreateNdCycleMaxEqualOp

cnmlStatus_t **cnmlCreateNdCycleMaxEqualOp**(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an cycle max_equal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the cycle max_equal operator, pass them into the function to create the cycle max_equal operator.

output[n, c, h, w] = input1[n, c, h, w] >= input2[1, c, 1, 1] ? input1 : input2

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input1 tensor, supporting data of float16/float32 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input2 tensor, supporting data of float16/float32 type.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16/float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.46.3 cnmlComputeCycleMaxEqualOpForward

cnmlStatus_t **cnmlComputeCycleMaxEqualOpForward**(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the cycle max_equal operator specified by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleMaxEqualOpForward_V2 is recommended to use.

After creating cycle max_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle max_equal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] >= input2[1, c, 1, 1] ? input1 : input2

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.46.4 cnmlComputeCycleMaxEqualOpForward_V2

cnmlStatus_t `cnmlComputeCycleMaxEqualOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the cycle max_equal operator specified by users on the MLU.

After creating cycle max_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle max_equal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.46.5 cnmlComputeNdCycleMaxEqualOpForward

cnmlStatus_t `cnmlComputeNdCycleMaxEqualOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the cycle max_equal operator specified by users on the MLU.

After creating cycle max_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle max_equal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.47 Cycle Min Equal Operation

4.47.1 cnmlCreateCycleMinEqualOp

cnmlStatus_t `cnmlCreateCycleMinEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an cycle min_equal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the cycle min_equal operator, pass them into the function to create the cycle min_equal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] <= input2[1, c, 1, 1] ? input1 : input2

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16/float32 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [1, c, h, w], supporting data of float16/float32 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [n, c, h, w], supporting data of float16/float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.47.2 cnmlCreateNdCycleMinEqualOp

cnmlStatus_t `cnmlCreateNdCycleMinEqualOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an nd cycle min_equal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the cycle min_equal operator, pass them into the function to create the cycle min_equal operator.

output[n, c, h, w] = input1[n, c, h, w] <= input2[1, c, 1, 1] ? input1 : input2

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] dim`: Input. A dim mark.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16/float32 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16/float32 type.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16/float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.47.3 cnmlComputeCycleMinEqualOpForward

cnmlStatus_t `cnmlComputeCycleMinEqualOpForward`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the cycle min_equal operator specified by users on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleMinEqualOpForward_V2 is recommended to use.

After creating cycle min_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle min_equal operator.

Formula

c[n c h w] = [a < b ? a : b for (a,b) in zip(a[n c_i h w], b[1 c_i 1 1]) for c_i in range(c)]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.47.4 cnmlComputeCycleMinEqualOpForward_V2

cnmlStatus_t `cnmlComputeCycleMinEqualOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the cycle min_equal operator specified by users on the MLU.

After creating cycle min_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle min_equal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.47.5 cnmlComputeNdCycleMinEqualOpForward

cnmlStatus_t `cnmlComputeNdCycleMinEqualOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the nd cycle min_equal operator specified by users on the MLU.

After creating cycle min_equal operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the cycle min_equal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.48 Cycle Mult Operation

4.48.1 cnmlCreateCycleMultOp

cnmlStatus_t `cnmlCreateCycleMultOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleMult operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleMult operator.

Formula

c[n c h w] = a[n c h w] * b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.48.2 cnmlCreateNdCycleMultOp

cnmlStatus_t `cnmlCreateNdCycleMultOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleMult operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleMult operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.48.3 cnmlComputeCycleMultOpForward_V3

cnmlStatus_t `cnmlComputeCycleMultOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleMultOpForward_V4 is recommended to use.

Compute the CycleMult operator on the MLU.

Transmit the created CycleMult operator, input tensor, input address, output tensor, and output address to the function to compute the CycleMult operator.

Formula

c[n c h w] = a[n c h w] * b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.48.4 cnmlComputeCycleMultOpForward_V4

cnmlStatus_t `cnmlComputeCycleMultOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleMult operator on the MLU.

Transmit the created CycleMult operator, input tensor, input address, output tensor, and output address to the function to compute the CycleMult operator.

Formula

c[n c h w] = a[n c h w] * b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.48.5 cnmlComputeNdCycleMultOpForward

cnmlStatus_t `cnmlComputeNdCycleMultOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleMult operator on the MLU.

Transmit the created NdCycleMult operator, input tensor, input address, output tensor, and output address to the function to compute the CycleMult operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.49 Cycle N Equal Operation

4.49.1 cnmlCreateCycleNEqualOp

cnmlStatus_t `cnmlCreateCycleNEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleNEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleNEqual operator.

Formula

c[n c h w] = a[n c h w] != b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.49.2 cnmlCreateNdCycleNEqualOp

cnmlStatus_t `cnmlCreateNdCycleNEqualOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleNEqual operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleNEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.

4.49.3 cnmlComputeCycleNEqualOpForward_V3

cnmlStatus_t `cnmlComputeCycleNEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleNEqualOpForward_V4 is recommended to use.

Compute the CycleNEqual operator on the MLU.

Transmit the created CycleNEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleNEqual operator.

Formula

c[n c h w] = a[n c h w] != b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.49.4 cnmlComputeCycleNEqualOpForward_V4

cnmlStatus_t `cnmlComputeCycleNEqualOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleNEqual operator on the MLU.

Transmit the created CycleNEqual operator, input tensor, input address, output tensor, and output address to the function to compute the CycleNEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.49.5 cnmlComputeNdCycleNEqualOpForward

cnmlStatus_t `cnmlComputeNdCycleNEqualOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleNEqual operator on the MLU.

Transmit the created NdCycleNEqual operator, input tensor, input address, output tensor, and output address to the function to compute the NdCycleNEqual operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.

- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.50 Cycle Or Operation

4.50.1 cnmlCreateCycleOrOp

cnmlStatus_t `cnmlCreateCycleOrOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1, and then perform element-wise or operation.

Formula

c[n c h w] = (a[n c h w] !=0 || (b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape of which is [1, c, 1, 1], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [n, c, h, w], supporting data of float16 type

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.50.2 cnmlCreateNdCycleOrOp

cnmlStatus_t `cnmlCreateNdCycleOrOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1, and then perform element-wise or operation.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.50.3 cnmlComputeCycleOrOpForward_V3

cnmlStatus_t `cnmlComputeCycleOrOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleOrOpForward_V4 is recommended to use.

Compute the CycleOr operator specified by users on the MLU.

After creating a CycleOr operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the CycleOr operator.

Formula

c[n c h w] = (a[n c h w] !=0 || (b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.50.4 cnmlComputeCycleOrOpForward_V4

cnmlStatus_t `cnmlComputeCycleOrOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleOr operator specified by users on the MLU.

After creating a CycleOr operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the CycleOr operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.50.5 cnmlComputeNdCycleOrOpForward

cnmlStatus_t `cnmlComputeNdCycleOrOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleOr operator specified by users on the MLU.

After creating a NdCycleOr operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the NdCycleOr operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.51 Cycle Sub Operation

4.51.1 cnmlCreateCycleSubOp

cnmlStatus_t `cnmlCreateCycleSubOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a CycleSub operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a CycleSub operator.

Formula

c[n c h w] = a[n c h w] - b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. The input_tensor_1 is four-dimensional tensors, in which the shape of input_tensor_1 is [n, c, h, w].
- `[in] input_tensor_2`: Input. The input_tensor_2 is four-dimensional tensors, in which the shape of the shape of input_tensor_2 is [1, c, 1, 1].
- `[in] output_tensor`: Input. The output_tensor is a four-dimensional tensor, the shape of which is [n, c, h, w].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.51.2 cnmlCreateNdCycleSubOp

cnmlStatus_t `cnmlCreateNdCycleSubOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a NdCycleSub operator according to base operator pointers given by users.

Transmit two input tensors and one output tensor into the function to create a NdCycleSub operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A multi-dimensional MLU input_1 tensor.
- `[in] input_tensor_2`: Input. A multi-dimensional MLU input_2 tensor.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.51.3 cnmlComputeCycleSubOpForward_V3

cnmlStatus_t `cnmlComputeCycleSubOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleSubOpForward_V4 is recommended to use.

Compute the CycleSub operator on the MLU.

Transmit the created CycleSub operator, input tensor, input address, output tensor, and output address to the function to compute the CycleSub operator.

Formula

c[n c h w] = a[n c h w] - b[1 c 1 1]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.51.4 cnmlComputeCycleSubOpForward_V4

cnmlStatus_t `cnmlComputeCycleSubOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleSub operator on the MLU.

Transmit the created CycleSub operator, input tensor, input address, output tensor, and output address to the function to compute the CycleSub operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.

- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.51.5 cnmlComputeNdCycleSubOpForward

cnmlStatus_t cnmlComputeNdCycleSubOpForward(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the NdCycleSub operator on the MLU.

Transmit the created NdCycleSub operator, input tensor, input address, output tensor, and output address to the function to compute the NdCycleSub operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.52 Cycle Xor Operation

4.52.1 cnmlCreateCycleXorOp

cnmlStatus_t cnmlCreateCycleXorOp(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1 and then perform element-wise Xor operation.

Formula

c[n c h w] = (a[n c h w] != 0 && b[1 c 1 1] == 0) || (a[n c h w] == 0 && b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.

- [in] input_tensor_1: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16 type.
- [in] input_tensor_2: Input. A four-dimensional MLU input tensor, the shape of which is [1, c, 1, 1], supporting data of float16 type.
- [in] output_tensor: Input. A four-dimensional MLU output tensor, the shape of which is [n, c, h, w], supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.52.2 cnmlCreateNdCycleXorOp

cnmlStatus_t cnmlCreateNdCycleXorOp(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Extend the shape of input_tensor_2 to the same shape as input_tensor_1 and then perform element-wise Xor operation.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A multi-dimensional MLU input_1 tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A multi-dimensional MLU input_2 tensor, supporting data of float16 type.
- [in] output_tensor: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.52.3 cnmlComputeCycleXorOpForward_V3

cnmlStatus_t cnmlComputeCycleXorOpForward_V3(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeCycleXorOpForward_V4 is recommended to use.

Compute the CycleXor operator specified by users on the MLU.

After creating a CycleXor operator, input, output, and computation stream, pass them into the function to compute the CycleXor operator.

Formula

c[n c h w] = (a[n c h w] != 0 && b[1 c 1 1] == 0) || (a[n c h w] == 0 && b[1 c 1 1] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c <= 131072

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input_1: Input. An MLU address pointing to input data 1.
- [in] input_2: Input. An MLU address pointing to input data 2.
- [in] computue_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.52.4 cnmlComputeCycleXorOpForward_V4

cnmlStatus_t `cnmlComputeCycleXorOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleXor operator specified by users on the MLU.

After creating a CycleXor operator, input, output, and computation stream, pass them into the function to compute the CycleXor operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.52.5 cnmlComputeNdCycleXorOpForward

cnmlStatus_t `cnmlComputeNdCycleXorOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the CycleXor operator specified by users on the MLU.

After creating a CycleXor operator, input, output, and computation stream, pass them into the function to compute the CycleXor operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.53 Deconv Operation

4.53.1 cnmlCreateDeconvOpParam

cnmlStatus_t **cnmlCreateDeconvOpParam**(cnmlDeconvOpParam_t *param, int stride_height, int stride_width, int hu_crop, int hd_crop, int wl_crop, int wr_crop)

A function.

According to the pointer given by the user, the function creates a struct of deconvolution operator operation parameters, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of the struct of deconvolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] hu_crop: Input. A value greater than or equal to zero, number of rows to be removed on the upside of the hw data block.
- [in] hd_crop: Input. A value greater than or equal to zero, number of rows to be removed on the down side of the hw data block.
- [in] wl_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the left side of the hw block.
- [in] wr_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the right side of the hw block.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.53.2 cnmlCreateDeconvOpParam_V2

cnmlStatus_t **cnmlCreateDeconvOpParam_V2**(cnmlDeconvOpParam_t *param, int stride_height, int stride_width, int hu_crop, int hd_crop, int wl_crop, int wr_crop, int adj_w, int adj_h)

A function.

According to the pointer given by the user, the function creates a struct of deconvolution operator operation parameters, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of the struct of deconvolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] hu_crop: Input. A value greater than or equal to zero, number of rows to be removed on the upside of the hw data block.
- [in] hd_crop: Input. A value greater than or equal to zero, number of rows to be removed on the down side of the hw data block.
- [in] wl_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the left side of the hw block.
- [in] wr_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the right side of the hw block.
- [in] adj_h: Input. A value greater than or equal to zero, number of data columns supplemented in the row direction of the hw data block.
- [in] adj_w: Input. A value greater than or equal to zero, number of data rows supplemented in the column direction of the hw data block.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.53.3 cnmlCreateDeconvOpParam_V3

cnmlStatus_t **cnmlCreateDeconvOpParam_V3**(cnmlDeconvOpParam_t *param, int stride_height, int stride_width, int hu_crop, int hd_crop, int wl_crop, int wr_crop, int adj_w, int adj_h, int dila_h, int dila_w)

A function.

According to the pointer given by the user, the function creates a struct of deconvolution operator operation parameters, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- [out] param: Output. A pointer pointing to the address of the struct of deconvolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] hu_crop: Input. A value greater than or equal to zero, number of rows to be removed on the upside of the hw data block.
- [in] hd_crop: Input. A value greater than or equal to zero, number of rows to be removed on the down side of the hw data block.
- [in] wl_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the left side of the hw block.
- [in] wr_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the right side of the hw block.
- [in] adj_h: Input. A value greater than or equal to zero, number of data columns supplemented in the row direction of the hw data block.
- [in] dila_h: Input. A value greater than or equal to 1, the dilation factor of kernel in height dimension.
- [in] dila_w: Input. A value greater than or equal to 1, the dilation factor of kernel in width dimension
- [in] adj_w: Input. A value greater than or equal to zero, number of data rows supplemented in the column direction of the hw data block.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.53.4 cnmlCreateDeconvOpParam_V4

cnmlStatus_t **cnmlCreateDeconvOpParam_V4**(cnmlDeconvOpParam_t *param, int stride_height, int stride_width, int hu_crop, int hd_crop, int wl_crop, int wr_crop, int output_padding_h, int output_padding_w, float output_padding_value, int dila_h, int dila_w)

A function.

According to the pointer given by the user, the function creates a struct of deconvolution operator operation parameters, and fills in the struct with parameters input by the user. You can set output_padding_h and output_padding_w parameters to avoid deconvolution maps one input shapes into multiple shapes of the output when stride is greater than 1. The two parameters can be used to infer the output shape.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- [out] param: Output. A pointer pointing to the address of the struct of deconvolution operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] hu_crop: Input. A value greater than or equal to zero, number of rows to be removed on the upside of the hw data block.
- [in] hd_crop: Input. A value greater than or equal to zero, number of rows to be removed on the down side of the hw data block.
- [in] wl_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the left side of the hw block.
- [in] wr_crop: Input. A value greater than or equal to 0, and the number of rows to be removed on the right side of the hw block.
- [in] output_padding_h: Input. A value greater than or equal to zero, number of data columns supplemented in the row direction of the hw data block.
- [in] output_padding_w: Input. A value greater than or equal to zero, number of data rows supplemented in the column direction of the hw data block.
- [in] dila_h: Input. A value greater than or equal to 1, the dilation factor of kernel in height dimension.
- [in] dila_w: Input. A value greater than or equal to 1, the dilation factor of kernel in width dimension.
- [in] output_padding_value: Input. A float32 value, data will be filled into output_padding area.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.53.5 cnmlDestroyDeconvOpParam

cnmlStatus_t **cnmlDestroyDeconvOpParam**(cnmlDeconvOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the deconvolution operator operation parameter is freed.

At the end of the convolution operator operation, the created struct pointer of the deconvolution operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer pointing to the address of the struct of deconvolution operator operation parameter.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.53.6 cnmlCreateDeconvOp

cnmlStatus_t **cnmlCreateDeconvOp**(cnmlBaseOp_t *op, cnmlDeconvOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateDeconvOpForward is recommended to use.

According to the basic operator pointer given by the user, a deconvolution operator is created.

After the pointer pointing to the address of the base operator, the deconvolution operator parameters and the input-output tensor are created, they are introduced into the function to create the deconvolution operator.

Before the deconvolution operator is created, a pointer pointing to the address of the deconvolution operator parameter struct is declared and the required operator parameters are introduced into the function to set the operator parameters.

Deconvolution operator.deconv operator can be transformed into conv operator. Firstly, for the h-w plane, 0 is inserted between two adjacent valid input values (stride_width - 1 0 are filled between two adjacent values in the w direction, and stride_height - 1 0 are filled between adjacent values in the h direction.) Meanwhile, the input kernal is also changed: firstly, the data of n and c dimensions are exchanged; secondly, the data of each plane in the h-w plane is rotated 180 degrees by the center of the plane; finally, the processed input and weight are used for convolution

operation (the stride in both the h and w directions of the conv operation is 1, h_top_padding and h_bottom_padding are hf-1,w_left_padding and w_right_padding are wf-1, hf and wf are the size of dimensions of weight h and w respectively).

The crop operation is performed on the result after convolution, and if the bias is not null, add the bias to get the output of deconv.

If the weight is less than step, that is, hf < stride_height, wf < stride_wight, then add 0 to weight so that weight=step.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer pointing to the struct of a deconvolution operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, hi, wi, ci],supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, ho, wo, co],supporting data of float16 type.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, hf, wf, cf] (nf = ci, cf = co),supporting data of float16 type.
- `[in] bias_filter`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, hb, wb, cb](nb = hb = wb = 1, cb = co),supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - param is null.
  - input_tensor is null.
  - output_tensor is null.
  - filter_tensor is null.
  - when the bias tensor is not null, the CPU tensor bound by the bias tensor is null.

4.53.7 cnmlCreateDeconvOpForward

This API has the same function as cnmlCreateDeconvOp. For detailed information, see cnmlCreateDeconvOp.

4.53.8 cnmlComputeDeconvOpForward_V3

cnmlStatus_t `cnmlComputeDeconvOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeDeConvOpForward_V4 is recommended to use.

Computing user-specified Deconv operator on MLU.

After the Deconv operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the Deconv operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.53.9 cnmlComputeDeconvOpForward_V4

cnmlStatus_t `cnmlComputeDeconvOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified Deconv operator on MLU.

After the Deconv operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the Deconv operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.

- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.54 Deconv Depthwise Operation

4.54.1 cnmlCreateDeconvDepthwiseOpParam

cnmlStatus_t **cnmlCreateDeconvDepthwiseOpParam**(cnmlDeconvDepthwiseOpParam_t *param, int stride_height, int stride_width, int dilation_h, int dilatin_w, int crop_uh, int crop_dh, int crop_lw, int crop_rw)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateDeconvDepthwiseOpParam_V2 is recommended to use.

According to the pointer given by the user, the function creates a struct of the deconv depthwise operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of the deconv depthwise operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] dilation_h: Input. A value greater than or equal to 1, and the dilation in Height direction.
- [in] dilation_w: Input. A value greater than or equal to 1, and the dilation in Width direction.
- [in] crop_uh: Input. A value greater than or equal to 0, and the crop in Up Height direction.
- [in] crop_dh: Input. A value greater than or equal to 0, and the crop in Down Height direction.
- [in] crop_lw: Input. A value greater than or equal to 0, and the crop in Left Width direction.
- [in] crop_rw: Input. A value greater than or equal to 0, and the crop in Right Width direction.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.54.2 cnmlCreateDeconvDepthwiseOpParam_V2

cnmlStatus_t **cnmlCreateDeconvDepthwiseOpParam_V2**(cnmlDeconvDepthwiseOpParam_t *param, int stride_height, int stride_width, int dilation_h, int dilatin_w, int output_padding_h, int output_padding_w, float output_padding_value, int crop_uh, int crop_dh, int crop_lw, int crop_rw)

A function.

According to the pointer given by the user, the function creates a struct of the deconv depthwise operator operation parameter, and fills in the struct with parameter input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of the deconv depthwise operator operation parameter.
- [in] stride_height: Input. A value greater than or equal to 1, and the sliding step in Height direction.
- [in] stride_width: Input. A value greater than or equal to 1, and the sliding step in Width direction.
- [in] dilation_h: Input. A value greater than or equal to 1, and the dilation in Height direction.
- [in] dilation_w: Input. A value greater than or equal to 1, and the dilation in Width direction.
- [in] output_padding_h: Input. Output_padding length.
- [in] output_padding_w: Input. Output_padding width.
- [in] output_padding_value: Input. The data filled into output_padding field.
- [in] crop_uh: Input. A value greater than or equal to 0, and the crop in Up Height direction.
- [in] crop_dh: Input. A value greater than or equal to 0, and the crop in Down Height direction.
- [in] crop_lw: Input. A value greater than or equal to 0, and the crop in Left Width direction.
- [in] crop_rw: Input. A value greater than or equal to 0, and the crop in Right Width direction.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.54.3 cnmlDestroyDeconvDepthwiseOpParam

cnmlStatus_t `cnmlDestroyDeconvDepthwiseOpParam`(cnmlDeconvDepthwiseOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the operation parameter of the deconv depthwise operator is freed.

After the operation of deconv depthwise operator is finished, the created struct pointer of deconv depthwise operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the operation parameter of the deconv depthwise operator.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.54.4 cnmlCreateDeconvDepthwiseOpForward

cnmlStatus_t `cnmlCreateDeconvDepthwiseOpForward`(cnmlBaseOp_t *op, cnmlDeconvDepthwiseOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateDeconvDepthwiseOpForward is recommended to use.

Computes the deconv depthwise of a 4-Dimensional input tensor.

Creates a Deconv Depthwise operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input, output, filter and bias tensor, pass them to the function to create a Deconv Depthwise operator.

Note

Deconv depthwise: in the current version this op only support the case of multipiler = 1. So the input, filter and output tensors in C dimension should be equal.

Supports MLU220,MLU270,1M20,and 1M70.

DataType

MLU270 and MLU220:

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order where input represents the MLU input, oc_input represents the on-chip input, output represents the MLU output, and oc_output represents the on-chip output.

- input-oc_input-output-oc_output
  Supported combinations are:
  float16-float16-float16-float16
  float32-float32-float32-float32

Scale Limitation

Unlmited.

Performance Optimization

It will reach its optimal performance, when the number of bytes in the C dimension is a multiple of 64.

Life Cycle

Release the Deconv depthwise Op after calling the cnmlComputeDeconvDepthwiseOpForward function and getting the result.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supporting data of float16 and float32 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo]. supporting data of float16 and float32 type.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, cf, hf, wf]. supporting data of float16 and float32 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, cb, hb, wb] (nb = 1, hb = 1, wb = 1) supporting data of float16 and flaot 32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `At`: least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST
  - The CPU tensor bound by bias tensor is null.

4.54.5 cnmlComputeDeconvDepthwiseOpForward

cnmlStatus_t `cnmlComputeDeconvDepthwiseOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified deconv depthwise operator on MLU.

After the deconv depthwise operator, input, output and computational queue are created, they are introduced into the function to compute the deconv depthwise operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.55 Deconv Group Operation

4.55.1 cnmlCreateDeconvGroupOp

cnmlStatus_t `cnmlCreateDeconvGroupOp`(cnmlBaseOp_t *op, cnmlDeconvOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor, int group)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateDeconvGroupOpForward is recommended to use.

According to the base operator pointer given by the user, a grouping deconvolution operator is created.

Group deconvolution: the input and the weight tensor are divided into groups, the groups numbers is the size of C dimension, and the deconv opertion is not performed on each group of input and weight correspondingly, and the computation result is not sequentially spliced and output.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni = 1), supporting data of float16 type.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is [nf, cf, hf, wf] (nf = 1, cf = ci = co), supporting data of float16 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, cb, hb, wb] (nv = 1, hb = 1, wb = 1, cb co), supporting data of float16 type.
- `[in] group`: Input. The group number of how the input and weight tensors are divided. The value of group should be greater than 0, it should follow the rules below:
  a. cf * group = ci
  b. nf % group = 0

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `At`: least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST
  - The CPU tensor bound by bias tensor is null.

4.55.2 cnmlCreateDeconvGroupOpForward

This API has the same function as cnmlCreateDeconvGroupOp. For detailed information, see cnmlCreateDeconvGroupOp.

4.55.3 cnmlComputeDeconvGroupOpForward_V2

cnmlStatus_t `cnmlComputeDeconvGroupOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified deconvolution group operator on MLU.

After the deconvolution group operator, input, output and computational queue are created, they are introduced into the function to compute the deconvolution group operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.56 Deformable Convolution Operation

4.56.1 cnmlCreateDeformConvOpParam

cnmlStatus_t `cnmlCreateDeformConvOpParam`(cnmlDeformConvOpParam_t *param, int stride_height, int stride_width, int dilation_height, int dilation_width, int pad_top, int pad_bottom, int pad_left, int pad_right, int conv_group, int offset_group)

A function.

This function fills cnmlDeformConvOpParam_t struct that holds the description of the deformable convolution operation parameters.

This function allocates parameters memory. You need to call cnmlDestroyConvOpParam to free memory and destroy struct after usage.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of the struct of the deformable convolution operator parameter.
- `[in] stride_height`: Input. The sliding step in Height direction. The value should be greater than or equal to 1.
- `[in] stride_width`: Input. The sliding step in Width direction. The value should be greater than or equal to 1.
- `[in] dilation_height`: Input. The dilation factor of kernel (that is, weight tensor) in height dimension. The value should be greater than or equal to 1.
- `[in] dilation_width`: Input. The dilation factor of kernel (that is, weight tensor) in width dimension.The value should be greater than or equal to 1.
- `[in] pad_top`: Input. Size of padding on the top. The value should be greater than or equal to 1.
- `[in] pad_bottom`: Input. Size of padding on the bottom. The value should be greater than or equal to 1.
- `[in] pad_left`: Input. Size of padding on the left. The value should be greater than or equal to 1.
- `[in] pad_right`: Input. Size of padding on the right. The value should be greater than or equal to 1.
- `[in] conv_group`: Input. The group number of convolution. The value should be greater than or equal to 1. Currently only support 1.
- `[in] offset_group`: Input. The group number of offset. The value should be greater than or equal to 1. Currently only support 1.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer. For more information, see “Error Codes” section in this guide.

4.56.2 cnmlDestroyDeformConvOpParam

cnmlStatus_t `cnmlDestroyDeformConvOpParam`(cnmlDeformConvOpParam_t *param)

A function.

Free the struct pointer of deformable convolution operation parameters according to the pointer given by the user.

At the end of the deformable convolution operation, the struct pointer of the deformable convolution operation parameters is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of the struct of the deformable convolution operation parameters.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed. For more information, see “Error Codes” section in this guide.

4.56.3 cnmlCreateDeformConvOp

cnmlStatus_t `cnmlCreateDeformConvOp`(cnmlBaseOp_t *op, cnmlDeformConvOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t offset_tensor, cnmlTensor_t mask_tensor, cnmlTensor_t bias_tensor, cnmlTensor_t output_tensor)

A function.

Deformable convolution learns an offset for each point on the output feature map to achieve the effect of geometric deformation, and learns a mask for each point on the output feature map to achieve the effect of weighted sampling. Deformalbe convolution can achieve more accurate feature extraction.

According to the base operator pointer given by the user, a deformable convolution operator is created.

After a pointer pointing to the address of base operator, the operation parameter of deformable convolution operator and input-output tensor are created, they are introduced into the function to create the deformable convolution operator.

Before the deformable convolution operator is created, a pointer pointing to the address of the deformable convolution operator parameters struct is declared, and the pointer and required operator parameters are introduced to the cnmlCreateDeformOpParam function to set the parameter struct.

2-D Deformable Convolution

A simple 2-dimensional convolution can be seen as a process that a 2-dimensional convolution kernel (weight matrix) slides on 2-dimensional input data, matrix multiplication is performed on the elements of the current input and kernel, and then the results are summed into a single output pixel. The convolution kernel repeats this process until it traverses the entire picture and converts to another 2-dimensional matrix kernel.

Deformable convolution means the matrix multiplication input elements position is not fixed, input elements should perform a bilinear interpolation according to the offset input, if deformable convolution has a mask input, the interpolated inputs should also mult the mask pixel by pixel, then the interpolated inputs perform a matrix multiplication to get the output pixel.

The special operation padding is equivalent to filling the edge of the input data with 0 (filling pad_top in the top direction, filling pad_bottom in the bottom direction, filling pad_left in the left direction, filling pad_right in the right direction),

The stride refers to the sliding step of the convolution kernel.

The dilation refers to the dilation factor of the convolution kernel.

The conv_group refers to the number which input channel and output channel is seperated to, the matrix multiplication is performed in conv_group times, then the output pixels is concated.

The offset_group refers to the number which input channel and offset is seperated to, the bilinear interpolation is performed in offset_group times.

filter_height <= input_height, filter_width <= input_width

The length of the weight in the Height direction must be less than or equal to the length of the input in the Height direction. The length of the weight in the Width direction must be less than or equal to the length of the input in the Width direction.

Summary

If the shape of an input tensor is [batch, in_height, in_width, in_channels] and the shape of a filter tensor or a kernel tensor is [out_channels, filter_height, filter_width, in_channels], with the default NHWC format.

output[b, i, j, k] = sum_{di, dj, q} input[b, strides[1] * i + di, strides[2] * j + dj, q] * filter[di, dj, q, k]

Ph = p_top + p_botton

Pw = p_left + p_right

Ho = (Hi + Ph - dh(Kh - 1) -1) / Sh + 1

Wo = (Wi + Pw - dw(Kw - 1) -1)/ Sw + 1

co = K

where `di` is the dialation in h dimension, `dj` is the dialation in w dimension, `q` is control variable in ci dimension, `k` is the variable in co dimension, `Ph` is the size of the padding in h dimension, `Pw` is the size of the padding in w dimension, `Ho` is the height of the output tensor, `Wo` is the width of the output tensor, `co` is the channel of the output tensor, `Hi` is the height of the input tensor, `dh` is the dilation in h dimension, `Kh` is the height of the

kernel `Sh` is height of the stride, `Wi` is the width of the input tensor, `dw` is the dilation in w dimension, `Kw` is the width of the kernel, `Sw` is the stride in w dimension, and `k` is the number of channels for output.

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

offset_type : Data type of the offset tensor.

mask_type : Data type of the mask tensor.

bias_type : Data type of the bias tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_oc_type is int8, then input_oc_type can be int8 or int16, and output_oc_type can be float16, float32, or int16.

If filter_oc_type is int16, then input_oc_type can be int16, and output_oc_type can be float16, float32, or int16.

Offset_type and mask_type can be float16 and float32.

Notes:

The data type you set in bias_type, and out_oc_type must be the same.

The data type you set in offset_type, and mask_type must be the same.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filter_type - filter_oc_type - out_oc_type - out_type

Supported combinations are:

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

kh <= hi，kw <= wi

where `kh` is the height of the kernel, `hi` is the height of the input tensor, `kw` is width of the kernel, and `wi` is the width of the input tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

For best practice, we suggest you call the cnmlComputeReshapeOpForward_V4() API to reshape the tensor, if you set the input tensor with the following:

- The size of ci dimension is less than 64.
- The size of h dimension of the input tensor is greater than 200.
- The size of w dimension is greater than 200.

Also, when you set the output tensor of the cnmlComputeReshapeOpForward_V4() call, the size of h and w dimensions should be less than or equal to 200 and the value of ci dimension should be greater or equal to 64.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer struct of deformable convolution operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, hi, wi, ci],supporting data of float16 type.
- `[in] filter_tensor`: Input. A 4-dimensional MLU weight tensor, the shape is nf, hf, wf, cf,supporting data of float16 type.
- `[in] offset_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [nof, hof, wof, cof] (nof = offset_group),supporting data of float16 type.
- `[in] mask_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nm, hm, wm, cm] (nb = offset_group), supporting data of float16 type.
- `[in] bias_tensor`: Input. A 4-dimensional MLU bias tensor, the shape is [nb, hb, wb, cb] (nb = 1, hb = 1, wb = 1, cb = co),supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, ho, wo, co](no = ni), supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by the bias tensor is null. For more information, see “Error Codes” section in this guide.

4.56.4 cnmlComputeDeformConvOpForward

cnmlStatus_t `cnmlComputeDeformConvOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t offset_tensor, void *offset, cnmlTensor_t mask_tensor, void *mask, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Computing user-specified deformable convolution operator on MLU.

After deformable convolution operator, input, output, parameters at runtime, and computational queue are created, they are introduced into the function to compute deformable convolution operator.

Summary

If the shape of an input tensor is [batch, in_height, in_width, in_channels] and the shape of a filter tensor or a kernel tensor is [output_channels, filter_height, filter_width, in_channels], with the default NHWC format.

output[b, i, j, k] = sum_{di, dj, q} input[b, strides[1] * i + di, strides[2] * j + dj, q] * filter[di, dj, q, k]

Ph = p_top + p_botton

Pw = p_left + p_right

Ho = (Hi + Ph - dh(Kh - 1) -1) / Sh + 1

Wo = (Wi + Pw - dw(Kw - 1) -1)/ Sw + 1

co = K

where `di` is the dialation in h dimension, `dj` is the dialation in w dimension, `q` is control variable in ci dimension, `k` is the variable in co dimension, `Ph` is the size of the padding in h dimension, `Pw` is the size of the padding in w dimension, `Ho` is the height of the output tensor, `Wo` is the width of the output tensor, `co` is the channel of the output tensor, `Hi` is the height of the input tensor, `dh` is the dilation in h dimension, `Kh` is the height of the kernel `Sh` is height of the stride, `Wi` is the width of the input tensor, `dw` is the dilation in w dimension, `Kw` is the width of the kernel, `Sw` is the stride in w dimension, and `k` is the number of channels for output.

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

offset_type : Data type of the offset tensor.

mask_type : Data type of the mask tensor.

bias_type : Data type of the bias tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_oc_type is int8, then input_oc_type can be int8 or int16, and output_oc_type can be float16, float32, or int16.

If filter_oc_type is int16, then input_oc_type can be int16, and output_oc_type can be float16, float32, or int16.

Offset_type and mask_type can be float16 and float32.

Notes:

The data type you set in bias_type, and out_oc_type must be the same.

The data type you set in offset_type, and mask_type must be the same.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filter_type - filter_oc_type - out_oc_type - out_type

Supported combinations are:

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

kh <= hi，kw <= wi

where `kh` is the height of the kernel, `hi` is the height of the input tensor, `kw` is width of the kernel, and `wi` is the width of the input tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

For best practice, we suggest you call the cnmlComputeReshapeOpForward_V4() API to reshape the tensor, if you set the input tensor with the following:

- The size of ci dimension is less than 64.
- The size of h dimension of the input tensor is greater than 200.
- The size of w dimension is greater than 200.

Also, when you set the output tensor of the cnmlComputeReshapeOpForward_V4() call, the size of h and w dimensions should be less than or equal to 200 and the value of ci dimension should be greater or equal to 64.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.

- [in] input: Input. An MLU address pointing to input data.
- [in] offset_tensor: Input. Offset MLU tensor pointer. Pass NULL if not used.
- [in] offset: Input. An MLU address pointing to offset data.
- [in] mask_tensor: Input. Mask MLU tensor pointer. Pass NULL if not used.
- [in] mask: Input. An MLU address pointing to mask data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.57 Device Memcpy Operation

4.57.1 cnmlCreateDeviceMemcpyOp

cnmlStatus_t cnmlCreateDeviceMemcpyOp(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the basic Operator pointer given by the user, a DeviceMemcpy operator is created.

After a pointer pointing to the address of the base operator and an input-output tensor are created, they are introduced into the function to create the DeviceMemcpy operator.

Summary

Copy a tensor to another tensor.

DataType:

MLU270:

input output data and their onchip datatype should be the same, the data type is not limited

Scale limitation:

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- [out] op: Output. A pointer pointing to the address of the base operator.
- [in] input: Input. A 4-dimensional MLU input tensor, the shape is [n, h, w, c],supporting data of float16 type.
- [in] output: Input. A 4-dimensional MLU weight tensor, the shape is [n, h, w, c],supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.57.2 cnmlComputeDeviceMemcpyOpForward_V3

cnmlStatus_t cnmlComputeDeviceMemcpyOpForward_V3(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computing DeviceMemcpy operators specified by users on MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeDeviceMemcpyOpForward_V4 is recommended to use.

After the Mult operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute DeviceMemcpy operators.

Datatype

MLU270:

Unlimited

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.57.3 cnmlComputeDeviceMemcpyOpForward_V4

cnmlStatus_t `cnmlComputeDeviceMemcpyOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing DeviceMemcpy operators specified by users on MLU.

After the Mult operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute DeviceMemcpy operators.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.58 Div Sqrt Dim Operation

4.58.1 cnmlCreateDivSqrtDimOp

cnmlStatus_t `cnmlCreateDivSqrtDimOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Creates a DivSqrtDim operator that rescales the input tensor by taking the square root of the C dimension size of the input tensor.

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer to the DivSqrtDim operator you have created.
- `[in] input`: Input. A 4-D MLU input tensor. The shape of the tensor is [n, h, w, c]. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor() API.
- `[in] output`: Input. The descriptor of the 4-D output tensor. The shape of the tensor is [n, h, w, c]. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor() API.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The size of C dimension should be greater than 0.

4.58.2 cnmlComputeDivSqrtDimOpForward

cnmlStatus_t `cnmlComputeDivSqrtDimOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeDivSqrtDimOpForward_V2 is recommended to use.

Rescales the input tensor by taking the square root of the C dimension size of the input tensor on MLU.

Supports MLU220 and MLU270.

Formula

output[n c h w] = input[n c h w] / sqrt(C) (C is channel dimension size)

DataType

MLU270:

inputDataType float16, float32

outputDataType float16, float32, int16, int8

Scale Limitation

MLU270:

c < 130400

Parameters

- `[out] output`: Output. A pointer to output data after the DivSqrtDim operator is applied.
- `[in] op`: Input. A pointer to the DivSqrtDim operator you have created.
- `[in] input`: Input. A pointer to the input data you want to rescale.
- `[in] compute_forw_param`: Input. A pointer to the struct address that records the data parallelism and device affinity for runtime.
- `[in] queue`: Input. A pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDARG`: One of the following conditions are met:
  - Operator pointer is NULL.
  - Output pointer is NULL.

4.58.3 cnmlComputeDivSqrtDimOpForward_V2

cnmlStatus_t `cnmlComputeDivSqrtDimOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Rescales the input tensor by taking the square root of the C dimension size of the input tensor on MLU.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.59 Dyadic Select Operation

4.59.1 cnmlCreateDyadicSelectOp

cnmlStatus_t `cnmlCreateDyadicSelectOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t input_tensor_3, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a binary select operator.

Then creates a pointer to the base operator address and input output tensor, and introduce them into the function to create a binary select operator.

Three inputs and one output should have the same shape, input1 and input2 are two portions of data to be selected, input3 is an option, the value of Input3 can only be filled with 1 or 0.

Formula

batch_index = false:

for (int i = 0;i < n * h * w * c;i++) {

if (in3[i] == 0) out[i] = in1[i];

if (in3[i] != 0) out[i] = in2[i];

}

batch_index = true:

for (int i = 0;i < n;i++) {

if (in3[i]) {

for (int j = 0;j < h * w * c;j++) {

out[i * h * w * c + j] = in1[i * h * w * c + j];

}

}

if (!in3[i]) {

for (int j = 0;j < h * w * c;j++) {

out[i * h * w * c + j] = in2[i * h * w * c + j];

} } }

DataType

MLU270:

input1,input2: float16, float32, int32

input3: uint32, bool

output: float16, float32, int32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], only supports data of float16 type.
- `[in] output_tensor_2`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], only supports data of float16 type..
- `[in] input_tensor_3`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], only supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [ni, ci, hi, wi], only supports data of float16 type..

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The input pointer is null.
  - Reason3 The output pointer is null.

4.59.2 cnmlDyadicSelectOpSetParam

cnmlStatus_t `cnmlDyadicSelectOpSetParam`(cnmlBaseOp_t op, bool bool_index, bool batch_index)

A function.

According to the pointer given by the user, set the attribut of cnmlBaseOp_t;

Formula

batch_index = false:

for (int i = 0;i < n * h * w * c;i++) {

if (in3[i] == 0) out[i] = in1[i];

if (in3[i] != 0) out[i] = in2[i];

}

batch_index = true:

for (int i = 0;i < n;i++) {

if (in3[i]) {

for (int j = 0;j < h * w * c;j++) {

out[i * h * w * c + j] = in1[i * h * w * c + j];

}

}

if (!in3[i]) {

for (int j = 0;j < h * w * c;j++) {

out[i * h * w * c + j] = in2[i * h * w * c + j];

} } }

DataType

MLU270:

input1,input2: float16, float32, int32

input3: uint32, bool

output: float16, float32, int32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of struct of cnmlBaseOp_t.
- `[in] bool_index`: Input. If bool_index is flase, the value of tensor_c can be arbitrary number, if bool_index is true, the value of tensor_c is only 0 or 1;
- `[in] batch_index`: Input. If batch_index is false, the shape of tensor_c is [n, c, h, w]; if batch_index is true, the shape of tensor_c is [n, 1, 1, 1]

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.59.3 cnmlComputeDyadicSelectOpForward_V3

cnmlStatus_t `cnmlComputeDyadicSelectOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *input_3, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeDyadicSelectOpForward_V4 is recommended to use.

For computing the user-specified binary select operator on the MLU.

After creating binary select operator, input, output and computation stream, introduce them to the function to compute the binary select operator.

Formula

batch_index = false:

for (int i = 0;i < n * h * w * c;i++) {

if (in3[i] == 0) out[i] = in1[i];

if (in3[i] != 0) out[i] = in2[i];

}

batch_index = true:


```
for (int i = 0;i < n;i++) {
if (in3[i]) {
for (int j = 0;j < h * w * c;j++) {
out[i * h * w * c + j] = in1[i * h * w * c + j];
}
}
if (!in3[i]) {
for (int j = 0;j < h * w * c;j++) {
out[i * h * w * c + j] = in2[i * h * w * c + j];
} } }
```

DataType

MLU270:

input1,input2: float16, float32, int32

input3: uint32, bool

output: float16, float32, int32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data.
- `[in] input_2`: Input. An MLU address pointing to input data.
- `[in] input_3`: Input. An MLU address pointing to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.59.4 cnmlComputeDyadicSelectOpForward_V4

cnmlStatus_t `cnmlComputeDyadicSelectOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t input_tensor3, void *input_3, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified binary select operator on the MLU.

After creating binary select operator, input, output and computation queue, introduce them to the function to compute the binary select operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] input_tensor3`: Input. Input MLU tensor pointer3. Pass NULL if not used.
- `[in] input_3`: Input. MLU address pointing to input3 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.60 Dynamic Read & Write Operation

4.60.1 cnmlCreateDynamicRWParam

cnmlStatus_t **cnmlCreateDynamicRWParam**(cnmlDynamicRWParam_t *param, cnmlDimension_t dim, int length)

A function.

According to the pointer given by the user, the function creates a DynamicRW operation parameter struct and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of DynamicRW operator operation parameter.
- `[in] dim;`: Input. The dimension that is used for dynamic reading or writing. For detailed supported values, see cnmlDimension_t data type.
- `[in] length;`: Input. the length for dynamic reading or writing. retval CNML_STATUS_SUCESS The function ends normally.

Return Value

- `CNML_STATUS_INVALIDPARAM`: At least one of the following condition is met:
  - param is a null pointer

4.60.2 cnmlDestroyDynamicRWParam

cnmlStatus_t **cnmlDestroyDynamicRWParam**(cnmlDynamicRWParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of DynamicRW operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70

After the operation of the DynamicRW operator is finished, the created struct pointer of DynamicRW operator operation parameter is freed.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of DynamicRW operator operation parameter.

Return Value

- `CNML_STATUS_SUCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is null pointer.
  - The content of the pointer pointed to by param has been freed.

4.60.3 cnmlCreateDynamicReadOp

cnmlStatus_t **cnmlCreateDynamicReadOp**(cnmlBaseOp_t *op, cnmlDynamicRWParam_t param, cnmlTensor_t input_tensor_X, cnmlTensor_t input_tensor_Y, cnmlTensor_t output_tensor)

cnmlCreateDynamicReadOp.

Create an DynamicRead operator based on the base operator pointer given by the user.

Reads data from input_X [N1, C1, H1, W1] to output [N2, C2, H2, W2]. The offset address of the input_X is specified by input_Y [1 1 1 1].

You can specify the dimension and length that is used for dynamic reading in the param .

Formula

The formula varies depending on the values of the `dim` and `length` parameters you set in the `cnmlDynamicRWParam_t` struct.

If the `dim` parameter is set to DIM_N:

C1 = C2, H1 = H2, W1 = W2, and N2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++)

output[idx C2 H2 W2] = input_X[(idx + input_Y) C1 H1 W1]

Note: Make sure input_Y you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < N1

If the `dim` parameter is set to DIM_C:

N1 = N2, H1 = H2, W1 = W2, and C2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++)

output[N2 idx H2 W2] = input_X[N1 (idx + input_Y) H1 W1]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < C1

If the `dim` parameter is set to DIM_H:

N1 = N2, C1 = C2, W1 = W2, and H2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

output[N2 C2 idx W2] = input_X[N1 C1 (idx + input_Y) W1]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < H1

If the `dim` parameter is set to DIM_W:

N1 = N2, C1 = C2, H1 = H2, and W2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

output[N2 C2 H2 idx] = input_X[N1 C1 H1 (idx + input_Y)]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < W1

DataType

Unlimited

Scale Limitation

(N1 * C1 * H1 * W1) >= (N2 * C2 * H2 * W2)

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] param`: Input. A cnmlDynamicRWParam_t.
- `[in] input_tensor_X`: Input. X 4-dimensional MLU input tensor.
- `[in] input_tensor_Y`: Input. Y 4-dimensional MLU input tensor.
- `[in] output`: Input. A 4-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.60.4 cnmlComputeDynamicReadOpForward

cnmlStatus_t **`cnmlComputeDynamicReadOpForward`**(cnmlBaseOp_t op, cnmlTensor_t input_tensor_X, void *input_X, cnmlTensor_t input_tensor_Y, void *input_Y, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeDynamicReadOpForward.

It is used to compute the user-specified DynamicRead operator on the MLU.

After creating the DynamicRead operator, Input, Output, and computation stream, pass them to the function to It is used to compute the DynamicRead operator.

Formula

The formula varies depending on the values of the `dim` and `length` parameters you set in the `cnmlDynamicRWParam_t` struct.

If the `dim` parameter is set to DIM_N:

C1 = C2, H1 = H2, W1 = W2, and N2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++)

output[idx C2 H2 W2] = input_X[(idx + input_Y) C1 H1 W1]

Note: Make sure input_Y you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < N1

If the `dim` parameter is set to DIM_C:

N1 = N2, H1 = H2, W1 = W2, and C2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++)

output[N2 idx H2 W2] = input_X[N1 (idx + input_Y) H1 W1]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < C1

If the `dim` parameter is set to DIM_H:

N1 = N2, C1 = C2, W1 = W2, and H2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

output[N2 C2 idx W2] = input_X[N1 C1 (idx + input_Y) W1]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < H1

If the `dim` parameter is set to DIM_W:

N1 = N2, C1 = C2, H1 = H2, and W2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

output[N2 C2 H2 idx] = input_X[N1 C1 H1 (idx + input_Y)]

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Y) < W1

DataType

Unlimited

Scale Limitation

(N1 * C1 * H1 * W1) >= (N2 * C2 * H2 * W2)

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_X`: Input. X 4-dimensional MLU input tensor.
- `[in] input_tensor_Y`: Input. Y 4-dimensional MLU input tensor.
- `[in] output`: Input. A 4-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The runtime task type is invalid

4.60.5 cnmlCreateDynamicWriteOp

cnmlStatus_t **`cnmlCreateDynamicWriteOp`**(cnmlBaseOp_t *op, cnmlDynamicRWParam_t param, cnmlTensor_t input_tensor_X, cnmlTensor_t input_tensor_Y, cnmlTensor_t input_tensor_Z, cnmlTensor_t output_tensor)

cnmlCreateDynamicWriteOp.

Create an DynamicWrite operator based on the base operator pointer given by the user.

Writes data from input_Y [N2 C2 H2 W2] to input_X [N1, C1, H1, W1] and then copies the input_X to output [N1, C1, H1, W1]. The offset address of the input_X is specified by input_Z [1 1 1 1].

You can specify the dimension and length that is used for dynamic writing in the `cnmlDynamicRWParam_t` struct.

Formula

The formula varies depending on the values of the `dim` and `length` parameters you set in the `cnmlDynamicRWParam_t` struct.

If the `dim` parameter is set to DIM_N:

C1 = C2, H1 = H2, W1 = W2, and N2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[(idx + input_Z) C1 H1 W1] = input_Y[idx C2 H2 W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < N1

If the `dim` parameter is set to DIM_C:

N1 = N2, H1 = H2, W1 = W2, and C2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 (idx + input_Z) H1 W1] = input_Y[N2 idx H2 W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < C1

If the `dim` parameter is set to DIM_H:

N1 = N2, C1 = C2, W1 = W2, and H2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 C1 (idx + input_Z) W1] = input_Y[N2 C2 idx W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < H1

If the `dim` parameter is set to DIM_W:

N1 = N2, C1 = C2, H1 = H2, and W2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 C1 H1 (idx + input_Z)] = input_Y[N2 C2 H2 idx]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < W1

DataType

Unlimited

Scale Limitation

(N1 * C1 * H1 * W1) >= (N2 * C2 * H2 * W2)

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] param`: Input. A cnmlDynamicRWParam_t.
- `[in] input_tensor_X`: Input. X 4-dimensional MLU input tensor.
- `[in] input_tensor_Y`: Input. Y 4-dimensional MLU input tensor.
- `[in] input_tensor_Z`: Input. Z 4-dimensional MLU input tensor.
- `[in] output`: Input. A 4-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.60.6 cnmlComputeDynamicWriteOpForward

cnmlStatus_t `cnmlComputeDynamicWriteOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor_X, void *input_X, cnmlTensor_t input_tensor_Y, void *input_Y, cnmlTensor_t input_tensor_Z, void *input_Z, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeDynamicWriteOpForward.

It is used to compute the user-specified DynamicWrite operator on the MLU.

After creating the DynamicWrite operator, Input, Output, and computation stream, pass them to the function to It is used to compute the DynamicWrite operator.

Formula

The formula varies depending on the values of the `dim` and `length` parameters you set in the `cnmlDynamicRWParam_t` struct.

If the `dim` parameter is set to DIM_N:

C1 = C2, H1 = H2, W1 = W2, and N2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[(idx + input_Z) C1 H1 W1] = input_Y[idx C2 H2 W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < N1

If the `dim` parameter is set to DIM_C:

N1 = N2, H1 = H2, W1 = W2, and C2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 (idx + input_Z) H1 W1] = input_Y[N2 idx H2 W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < C1

If the `dim` parameter is set to DIM_H:

N1 = N2, C1 = C2, W1 = W2, and H2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 C1 (idx + input_Z) W1] = input_Y[N2 C2 idx W2]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < H1

If the `dim` parameter is set to DIM_W:

N1 = N2, C1 = C2, H1 = H2, and W2 equals to the value of `length`

for (idx=0; idx < the value of the `length` parameter; idx++) {

input_X[N1 C1 H1 (idx + input_Z)] = input_Y[N2 C2 H2 idx]

output = input_X }

Note: Make sure input_Z you specified meet the following limitation, otherwise an error occurred:

(idx + input_Z) < W1

DataType

Unlimited

Scale Limitation

(N1 * C1 * H1 * W1) >= (N2 * C2 * H2 * W2)

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_X`: Input. X 4-dimensional MLU input tensor.
- `[in] input_tensor_Y`: Input. Y 4-dimensional MLU input tensor.
- `[in] input_tensor_Z`: Input. Z 4-dimensional MLU input tensor.
- `[in] output`: Input. A 4-dimensional MLU input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The runtime task type is invalid

4.61 ELU Operation

4.61.1 cnmlCreateEluOp

cnmlStatus_t `cnmlCreateEluOp`(cnmlBaseOp_t *op, cnmlTensor_t input, cnmlTensor_t output)

cnmlCreateEluOp.

Create an elu activation operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, elu activation operator operation parameters, input and output Tensors, pass them to the function to create a elu activation operator.

Before creating the elu activation operator, declare a pointer to the address of the elu activation operator operation structure structure and pass it to the function together with the required operator parameters to set the operator parameters.

output[i, j, k, l] = (input[i, j, k, l] < 0) ? alpha * (exp(input[i, j, k, l]) â€ “1) : input[i, j, k, l]

alpha: const float type, the value being 1

Shapes of the input and output should be consistent.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty

- The input pointer is empty.
- The output pointer is empty.

4.61.2 cnmlComputeEluOpForward_V3

cnmlStatus_t **cnmlComputeEluOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeEluOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeEluOpForward_V4 is recommended to use.

It is used to compute the user-specified elu activation function operator on the MLU.

After creating the elu activation function operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the elu activation function operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.61.3 cnmlComputeEluOpForward_V4

cnmlStatus_t **cnmlComputeEluOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeEluOpForward_V4.

It is used to compute the user-specified elu activation function operator on the MLU.

After creating the elu activation function operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the elu activation function operator.

Supports MLU220,MLU270,1M20,and 1M70.

Input. A pointer which points to base operators.

Parameters

- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.62 Embedding Operation

4.62.1 cnmlCreateEmbeddingOp

cnmlStatus_t **cnmlCreateEmbeddingOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t weight_tensor)

cnmlCreateEmbeddingOp.

Create a embedding operator based on the base operator pointer given by the user.

After creating a pointer pointing to the base operator address, input, output and weight tensor of the embedding operator, pass them to the function to create the embedding operator.

Formula

This operator maps integer indices to vector representations

a. weight is a two dimension array, can be regarded as a dict in python.
b. input is a 1 to n-dimension array which stores a series of keys, this means the range of value in input must be [0, weight_shape[0]).
c. output is also a 1 to n-dimension array store those series of values which input corrspond in that dict(weight).
d. output_shape = input_shape + weight_shape[1:]
   for example, input_shape = (1, 2), weight_shape = (4, 5), output_shape = (1, 2, 5)
e. for example, input = [1, 0, 0], weight = [[0, 1], [2, 3]], then output = [[2, 3], [0, 1], [0, 1]]

DataType

The datatype of input must be int32.

The datatype of weight and output should be exactly the same, such as fp32 or fp16.

Scale limitation

a. if using multi-dimension tensor api, then input' s shape and output' s shape can be multi-dimension, but weight' s shape must be two dimension. if not using multi-dimension tensor api, then input' s shape and weight' s shape must be two dimension and output' s shape must be three dimension.
b. input_shape[0] * input_shape[1] * ⋯input_shape[input_shape.size() - 1] + weight_shape[1] <= 63000

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU input tensor, supporting data of int32 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU output tensor, supporting data of float16 or float32 type.
- `[in] weight_tensor`: Input. A 1 to n-dimensional MLU weight tensor, only support two dimension, supporting data of float16 or float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null
  - The weight tensor is null

4.62.2 cnmlComputeEmbeddingOpForward

cnmlStatus_t **cnmlComputeEmbeddingOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeEmbeddingOpForward.

Compute the embedding operator on the MLU.

After creating the embedding operator, input, output, runtime parameters, computation queue, pass them to the function to It is used to compute the embedding operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.62.3 cnmlComputeEmbeddingOpForward_V2

cnmlStatus_t **cnmlComputeEmbeddingOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeEmbeddingOpForward_V2.

Compute the embedding operator on the MLU.

After creating the embedding operator, input_tensor, input,output_tensor, output, and computation queue, pass them to the function to compute the embedding operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.

- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.62.4 cnmlComputeEmbeddingOpTrainingForward

cnmlStatus_t **cnmlComputeEmbeddingOpTrainingForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t weight_tensor, void *weight, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeEmbeddingOpTrainingForward.

After creating the embedding operator, input_tensor, input, output_tensor, output, weight_tensor, weight and computation queue, pass them to the function to compute the embedding operator on the MLU.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] weight_tensor: Input. Weight MLU tensor pointer. Pass NULL if not used.
- [in] weight: Input. An MLU address pointing to weight data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.63 Equal Operation

4.63.1 cnmlCreateEqualOp

cnmlStatus_t **cnmlCreateEqualOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the tensor B and the tensor A in each position to find whether one is equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] == B[ni][hi][wi][ci])? 1: 0. The shapes of two inputs and one output should be exactly the same.

Formula

c[n c h w] = a[n c h w] == b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type. supports data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.63.2 cnmlComputeEqualOpForward_V3

cnmlStatus_t cnmlComputeEqualOpForward_V3(cnmlBaseOp_t op, void *inputA, void *inputB, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeEqualOpForward_V4 is recommended to use.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is equal to the other.

Formula

c[n c h w] = a[n c h w] == b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] inputA: Input. An MLU address pointing to input data 1.
- [in] inputB: Input. An MLU address pointing to input data 2.
- [in] computue_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.63.3 cnmlComputeEqualOpForward_V4

cnmlStatus_t cnmlComputeEqualOpForward_V4(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *inputA, cnmlTensor_t input_tensor_B, void *inputB, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor_A: Input. Input MLU tensor pointerA. Pass NULL if not used.
- [in] inputA: Input. MLU address pointing to input1 data.
- [in] input_tensor_B: Input. Input MLU tensor pointerB. Pass NULL if not used.
- [in] inputB: Input. MLU address pointing to inputB data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.64 Erf Operation

4.64.1 cnmlCreateErfOp

cnmlStatus_t **cnmlCreateErfOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a Gaussian error operator according to the basic operator pointer given by the user. The output dimension is same as the input dimension, and the output element is the Gaussian error function value corresponding to the input element.

The numerical range of input data is [-2Ï€, 2Ï€]. Use the radian measure instead of the degree measure.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [no, co, ho, wo], supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - Op is empty.
  - Input_tensor is empty.
  - Output_tensor is empty.

4.64.2 cnmlComputeErfOpForward_V3

cnmlStatus_t **cnmlComputeErfOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the user-specified Gaussian error operator on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeErfOpForward_V4 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input parameter operator pointer op is empty.
  - Input parameter tensor pointer input is empty.
  - Output parameter tensor pointer output is empty.

4.64.3 cnmlComputeErfOpForward_V4

cnmlStatus_t **cnmlComputeErfOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified Gaussian error operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Reason1 The task type of runtime is invalid.

4.65 Exp Operation

4.65.1 cnmlCreateExpOp

cnmlStatus_t `cnmlCreateExpOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an exponent operator with the base being e.

The shapes of input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The shape of output tensor is different from that of input tensor.

4.65.2 cnmlComputeExpOpForward_V3

cnmlStatus_t `cnmlComputeExpOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeExpOpForward_V4 is recommended to use.

Compute the user-specified exponent operator with the base being e.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.65.3 cnmlComputeExpOpForward_V4

cnmlStatus_t `cnmlComputeExpOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified exponent operator with the base being e.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Reason1 The task type of runtime is invalid.

4.66 Floor Operation

4.66.1 cnmlCreateFloorOp

cnmlStatus_t `cnmlCreateFloorOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a floor operator.

The shapes of input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The shape of output tensor is different from that of input tensor.

4.66.2 cnmlComputeFloorOpForward_V3

cnmlStatus_t `cnmlComputeFloorOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeFloorOpForward_V4 is recommended to use.

Compute the user-specified rounding down operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.66.3 cnmlComputeFloorOpForward_V4

cnmlStatus_t `cnmlComputeFloorOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified rounding down operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Reason1 The task type of runtime is invalid.

4.67 Frozen Batch Norm Operation

4.67.1 cnmlCreateFrozenBatchNormOp

cnmlStatus_t `cnmlCreateFrozenBatchNormOp`(cnmlBaseOp_t *op, cnmlTensor_t input, cnmlTensor_t filter, cnmlTensor_t bias, cnmlTensor_t mean, cnmlTensor_t var, cnmlTensor_t output)

A function.

Creates a FrozenBatchNorm operator that scales parameters after batch normalization. This function is the same as cnmlCreateBatchNormOp and cnmlCreateScaleOp together. If you do not want to scale results, use cnmlCreateBatchNormOp instead.

Supports MLU220,MLU270,1M20,and 1M70.

DataType

```
inputDataType, filterDataType, biasDataType, meanDataType, varDataType: float16, float32

outputDataType: float16, float32
```

Formula

output[n c h w] = (input[n c h w] - mean[1 c 1 1]) / srqt(var[1 c 1 1]) * filter[1 c 1 1] + bias[1 c 1 1] The mean is mean value of input tensor. The var is variance value of input tensor. The filter is a parameter to scale results. The bias is an offset added to the normalized tensor.

Note The filter, bias, mean, var are constant data in frozen batchnorm operation. The type of these tensors need to be CNML_CONST, and it need to bind data infomration by cnmlBindConstData_V2.

Parameters

- `[out] op`: Output. A pointer to the FrozenBatchNorm operator you have created.
- `[in] input`: Input. A 4-D MLU input tensor. The shape of the tensor is [n, c, h, w].
- `[in] filter`: Input. A 4-D filter tensor. The shape of the tensor is [1, c, 1, 1]. Supporting data type same as input tensor.
- `[in] bias`: Input. A 4-D bias tensor. The shape of the tensor is [1, c, 1, 1]. Supporting data type same as input tensor.
- `[in] mean`: Input. A 4-D mean tensor of MLU. The shape of the tensor is [1, c, 1, 1]. Supporting data type same as input tensor.
- `[in] var`: Input. A 4-D variance tensor of MLU. The shape of the tensor is [1, c, 1, 1]. Supporting data type same as input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function was success.
- `CNML_STATUS_INVAILDPA`: One of the following conditions are met:
  - The operator pointer is NULL.
  - The input pointer is NULL.

4.67.2 cnmlComputeFrozenBatchNormOpForward

cnmlStatus_t `cnmlComputeFrozenBatchNormOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the Fused FrozenBatchNorm operator specified by users on the CPU.

Before calling this API, you need to create the CPU input tensor, CPU input address, CPU output tensor, and CPU output address.

Formula

output[n c h w] = (input[n c h w] - mean[1 c 1 1]) / srqt(var[1 c 1 1]) * filter[1 c 1 1] + bias[1 c 1 1]

Parameters

- `[out] output`: Output. Pointer to the data after the FrozenBatchNorm operator is applied.
- `[in] compute_forw_param`: Input. Pointer to the struct that records the data parallelism and device affinity for runtime.
- `[in] queue`: Input. Pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The input pointer is NULL.

4.67.3 cnmlComputeFrozenBatchNormOpForward_V2

cnmlStatus_t `cnmlComputeFrozenBatchNormOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Scales parameters after batch normalization on MLU.

Only supports MLU270.

Before call this API, you need to create a FrozenBatchNorm operator, input, output, and computation queue.

Note: the inputs and output must meet the options set in cnmlCreateFrozenBatchNormOp.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. Pointer to the data of the tensor you want to compute.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. Pointer to the data after the FrozenBatchNorm operator is applied.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.68 Gatherv2 Operation

4.68.1 cnmlCreateGatherV2Op

cnmlStatus_t `cnmlCreateGatherV2Op`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t index_tensor, cnmlTensor_t output_tensor, cnmlDimension_t dim)

A function.

The function creates a GatherV2 operator according to the base operator pointer given by users.

After creating a pointer pointing to base operators, the GatherV2 operator input tensor, index tensor, and output Tensor, pass them into the function to create the GatherV2 operator.

Obtain slices of input on a specified dimension according to the index,.Taking axis = CNML_DIM_N as an example, output[n, h, w, c] = input[index[n], h, w, c].

On the dimensions specified by axis, the shape of index is the same as that of output. On other dimensions, all the shape of index is 1, and the shape of output is the same as that of input. Taking axis = CNML_DIM_N as an example, the input shape is [ni, hi, wi, ci], the valid shape of index is [no, 1, 1, 1], and the valid shape of output is [no, hi, wi, ci].

Formula

if index[i] != -1:

dim==N:out[i][c][h][w]=input[index[i]][c][h][w]

dim==H:out[n][c][i][w]=input[n][c][index[i]][w]

dim==W:out[n][c][h][i]=input[n][c][h][index[i]]

if index[i] == -1:

dim==N:out[i][c][h][w]=0

dim==H:out[n][c][i][w]=0

dim==W:out[n][c][h][i]=0

DataType

MLU270:

input_dt == output_dt

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.

- [in] input_tensor: Input. A four-dimensional input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- [in] indexTensor: Input. A four-dimensional index tensor, [n_index, h_index, w_index, c_index], only supporting data of uint32 type.
- [in] output_tensor: Input. A four-dimensional output tensor, the shape of which is [no, co, ho, wo] (no = nl, co = nr, ho = 1, wo = 1), supporting data of float16 type.
- [in] axies: Input. An enumeration variable of cnmlDimension_t, type can be one of CNML_DIM_N, CNML_DIM_C, CNML_DIM_H, and CNML_DIM_W.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.68.2 cnmlComputeGatherV2OpForward_V3

cnmlStatus_t **cnmlComputeGatherV2OpForward_V3**(cnmlBaseOp_t op, void *input, void *index, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeGatherV2OpForward_V4 is recommended to use.

Compute the GatherV2 operator specified by users on the MLU.

After creating the GatherV2 operators, input, output, and computation stream, pass them into the function to compute the GatherV2 operator.

Formula

if index[i] != -1:

dim==N:out[i][c][h][w]=input[index[i]][c][h][w]

dim==H:out[n][c][i][w]=input[n][c][index[i]][w]

dim==W:out[n][c][h][i]=input[n][c][h][index[i]]

if index[i] == -1:

dim==N:out[i][c][h][w]=0

dim==H:out[n][c][i][w]=0

dim==W:out[n][c][h][i]=0

DataType

MLU270:

input_dt == output_dt

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address specifying the output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address which points to input data.
- [in] index: Input. An MLU address pointing to index input data.
- [in] param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.68.3 cnmlComputeGatherV2OpForward_V4

cnmlStatus_t **cnmlComputeGatherV2OpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t index_tensor, void *index, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the GatherV2 operator specified by users on the MLU.

After creating the GatherV2 operators, input, output, and computation queue, pass them into the function to compute the GatherV2 operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. MLU address pointing to input data.
- [in] index_tensor: Input. Index MLU tensor pointer. Pass NULL if not used.
- [in] index: Input. MLU address pointing to index data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.

- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.68.4 cnmlCreateNdGatherV2Op

cnmlStatus_t **cnmlCreateNdGatherV2Op**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t index_tensor, cnmlTensor_t output_tensor, int axies)

A function.

The function creates a multidimensional GatherV2 operator according to the base operator pointer given by users.

After creating a pointer pointing to base operators, the multidimensional GatherV2 operator input tensor, index tensor, and output Tensor, pass them into the function to create the Nd GatherV2 operator.

Obtain slices of input on a specified dimension according to the index.Taking axis = 0 as an example, output[n, h, w] = input[index[n], h, w].

On the dimensions specified by axis, the product of multidimensional index is the same as that of output. The shape of output is the same as that of input except for dimension specified by axis. Taking axis = 0 as an example, the input shape is [ni, hi, wi], the shape of index is [x, y], and the valid shape of output is [x, y, hi, wi].

Formula

if index[i] != -1:

dim==0:out[i][c][h]=input[index[i]][c][h]

dim==1:out[n][i][h]=input[n][index[i]][h]

dim==2:out[n][c][i]=input[n][c][index[i]]

if index[x, y] != -1:

dim==0:out[x][y][c][h]=input[index[x, y]][c][h]

dim==1:out[n][x][y][h]=input[n][index[x, y]][h]

dim==2:out[n][c][x][y]=input[n][c][index[x, y]]

if index[i] == -1:

dim==0:out[i][c][h]=0

dim==1:out[n][i][h]=0

dim==2:out[n][c][i]=0

DataType

MLU270:

input_dt == output_dt

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor: Input. A multidimensional input tensor, supporting data of float16/float32 type.
- [in] indexTensor: Input. A multidimensional index tensor, only supporting data of uint32 type. It records shape value of input_tensor at dimensions pointed by axies.
- [in] output_tensor: Input. A multidimensional output tensor, supporting data of float16/float32 type.
- [in] axies: Input. An int type variable.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output_tensor pointer is null.
  - Reason3 The input_tensor pointer is null.
  - Reason4 The index_tensor pointer is null.

4.68.5 cnmlComputeNdGatherV2OpForward

cnmlStatus_t `cnmlComputeNdGatherV2OpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t index_tensor, void *index, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computes the multidimensional GatherV2 operator specified by users on the MLU.

After creating the multidimensional GatherV2 operators, input, output, and computation queue, pass them into the function to compute the multidimensional GatherV2 operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. MLU address pointing to input data.
- `[in] index_tensor`: Input. Index MLU tensor pointer. Pass NULL if not used. It records shape value of input_tensor at dimensions pointed by axies.
- `[in] index`: Input. MLU address pointing to index data. These data are shape value of input_tensor at dimensions pointed by axies.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.69 Greater Operation

4.69.1 cnmlCreateGreaterOp

cnmlStatus_t `cnmlCreateGreaterOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Call after the tensor is created, and according to the base operator pointer given by the user, create an operator for performing an element-wise comparison on two tensors to find whether one is greater than the other.

The shapes of two inputs and one output must be exactly the same.

Algorithm explanation: perform an element-wise comparison on the A and B to find whether one is greater than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] > B[ni][hi][wi][ci])? 1:0

Formula

c[n c h w] = a[n c h w] > b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] input_tensor_2`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The input tensor type is not CNML_TENSOR or CNML_CONST.
  - Reason2 The input and output pointer is null.

4.69.2 cnmlComputeGreaterOpForward_V3

cnmlStatus_t `cnmlComputeGreaterOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeGreaterOpForward_V4 is recommended to use.

Call after the operation is created.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is greater than or equal to the other.

Formula

c[n c h w] = a[n c h w] > b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.69.3 cnmlComputeGreaterOpForward_V4

cnmlStatus_t `cnmlComputeGreaterOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Call after the operation is created.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is greater than or equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.70 Greater Equal Operation

4.70.1 cnmlCreateGreaterEqualOp

cnmlStatus_t `cnmlCreateGreaterEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Call after the tensor is created, and according to the base operator pointer given by the user, create an operator for performing an element-wise comparison on two four-dimensional tensors to find whether one is greater than or equal to the other.

The shapes of two inputs and one output should be exactly the same.

Algorithm explanation: perform an element-wise comparison on the four-dimensional tensors A and B to find whether one is greater than or equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] >= B[ni][hi][wi][ci])? 1:0

Formula

c[n c h w] = a[n c h w] >= b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] input_tensor_2`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type..

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Throw exceptions when the function fails to execute.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The CPU tensor bound by the bias tensor is null.
  - Reason2 The input tensor type is not CNML_TENSOR or CNML_CONST.

4.70.2 cnmlComputeGreaterEqualOpForward_V3

cnmlStatus_t `cnmlComputeGreaterEqualOpForward_V3`(cnmlBaseOp_t op, void *inputA, void *inputB, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeGreaterEqualOpForward_V4 is recommended to use.

Call after the operation is created.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is greater than or equal to the other.

Formula

c[n c h w] = a[n c h w] >= b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.

- [in] computue_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.70.3 cnmlComputeGreaterEqualOpForward_V4

cnmlStatus_t **cnmlComputeGreaterEqualOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *inputA, cnmlTensor_t input_tensor_B, void *inputB, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Call after the operation is created.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is greater than or equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor_A: Input. Input MLU tensor pointerA. Pass NULL if not used.
- [in] input_A: Input. MLU address pointing to inputA data.
- [in] input_tensor_B: Input. Input MLU tensor pointerB. Pass NULL if not used.
- [in] input_B: Input. MLU address pointing to inputB data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.71 Grep Operation

4.71.1 cnmlCreateGrepOpParam

cnmlStatus_t **cnmlCreateGrepOpParam**(cnmlGrepOpParam_t *param, int start_index_N, int start_index_H, int start_index_W, float space_number)

cnmlCreateGrepOpParam.

This function creates a Grep operation parameter struct based on the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer to the address of the Grep operator operation parameter struct.
- [in] start_index_N: Input. A starting point of interception in the direction of N, int data type.
- [in] start_index_H: Input. A starting point of interception in the direction of H, int data type.
- [in] start_index_W: Input. A starting point of interception in the direction of W, int data type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met: -param is a null pointer.

4.71.2 cnmlDestroyGrepOpParam

cnmlStatus_t **cnmlDestroyGrepOpParam**(cnmlGrepOpParam_t *param)

cnmlDestroyGrepOpParam.

This function destroys a Grep operator operation parameter struct based on the pointer given by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to the address of the Grep operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: -param is a null pointer.

4.71.3 cnmlCreateGrepOp

cnmlStatus_t **cnmlCreateGrepOp**(cnmlBaseOp_t *op, cnmlGrepOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateGrepOp.

It creates a Grep operator based on the base operator pointer given by the user.

After a pointer to the base operator address is created and the Grep operator inputs and outputs the Tensor, they are passed to the function to create a Grep operator.

Before creating the Grep operator, declare a pointer to the address of the Grep operator operation parameter struct and pass it to the function with the desired operation parameters to set the operator parameters.

Formula

Output [no, ho, wo, co]= input[(start_n : start_n + no) ,(start_h : start_h + ho), (start_w : start_w + wo), c]

DataType

MLU270:

input: int8, int16, float16, float32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] param`: Input. An ImageDetect operation struct pointer.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, hi, wi, c], supporting data of type float16.
- `[in] output`: Input. A 4-dimensional MLU output tensor, of which the shape is [no, ho, wo, c], supporting data of type float16.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - The input tensor type is either CNML_TENSOR or CNML_CONST.
  - The CPU tensor bound to the bias tensor is empty.

4.71.4 cnmlComputeGrepOpForward_V3

cnmlStatus_t **cnmlComputeGrepOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeGrepOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeGrepOpForward_V4 recommended to use.

It is used to compute the user-specified Grep operator on the MLU.

After creating the Grep operator, related parameters and computation stream, pass them to the function to It is used to compute the Grep operator.

Formula

Output [no, ho, wo, co]= input[(start_n : start_n + no) ,(start_h : start_h + ho), (start_w : start_w + wo), c]

DataType

MLU270:

input: int8, int16, float16, float32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The runtime task type is invalid.

4.71.5 cnmlComputeGrepOpForward_V4

cnmlStatus_t `cnmlComputeGrepOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeGrepOpForward_V4.

It is used to compute the user-specified Grep operator on the MLU.

After creating the Grep operator, related parameters and computation queue, pass them to the function to It is used to compute the Grep operator.

Formula

Output [no, ho, wo, co]= input[(start_n : start_n + no) ,(start_h : start_h + ho), (start_w : start_w + wo), c]

DataType

MLU270:

input: int8, int16, float16, float32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.72 Grep Channel Operation

4.72.1 cnmlCreateGrepChannelOpParam

cnmlStatus_t **cnmlCreateGrepChannelOpParam**(cnmlGrepChannelOpParam_t *param, int channel_)

A function.

According to the pointer given by the user, the function creates a GrepChannel operator operation parameter struct, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of GrepChannel operator operation parameter.
- `[in] channel`: Input. The discarded length in c direction.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.72.2 cnmlCreateGrepChannelOpParam_V2

cnmlStatus_t **cnmlCreateGrepChannelOpParam_V2**(cnmlGrepChannelOpParam_t *param, int c_front_, int c_back_)

A function.

According to the pointer given by the user, the function creates a struct of GrepChannel operator operation parameter, and fills in the struct with parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of GrepChannel operator operation parameter.
- `[in] c_front_`: Input. The discarded length above the c direction.
- `[in] c_back_`: Input. The discarded length down in the c direction.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.72.3 cnmlDestroyGrepChannelOpParam

cnmlStatus_t **cnmlDestroyGrepChannelOpParam**(cnmlGrepChannelOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of GrepChannel operator operation parameter is freed.

At the end of GrepChannel operator operation, the created struct pointer of GrepChannel operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of GrepChannel operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.72.4 cnmlCreateGrepChannelOp

cnmlStatus_t **cnmlCreateGrepChannelOp**(cnmlBaseOp_t *op, cnmlGrepChannelOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the basic Operator pointer given by the user, a GrepChannel operator is created.

Similarly to grep, a small tensor is intercepted from the input tensor. Note: can only be intercepted from c direction.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A struct pointer of normalized operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi],supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo],supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.72.5 cnmlComputeGrepChannelOpForward_V3

cnmlStatus_t **cnmlComputeGrepChannelOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeGrepChannelOpForward_V4 is recommended to use.

Computing user-specified GrepChannel operators on MLU.

After the GrepChannel operators, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute GrepChannel operators.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to the output position.
- [in] op: Input. A pointer pointing to the base operator.
- [in] input: Input. An MLU address pointing to the input data.
- [in] compute_forw_param: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.72.6 cnmlComputeGrepChannelOpForward_V4

cnmlStatus_t **cnmlComputeGrepChannelOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified GrepChannel operators on MLU.

After the GrepChannel operators, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute GrepChannel operators.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.73 Group Norm Operation

4.73.1 cnmlCreateGroupNormOp

cnmlStatus_t `cnmlCreateGroupNormOp`(cnmlBaseOp_t *op, int groupnum, cnmlTensor_t input, cnmlTensor_t gamma, cnmlTensor_t beta, cnmlTensor_t output)

A function.

Create a GroupNorm operator according to base operator pointers given by users.

Formula

input transfer groupnum inputtemps. And the shape of inputtemps is[ni, hi, wi, ci / groupnum].

output_temps[n, h1, w, c] = 1 / sigma * (inputtemps[n, h1, w, c] - mean)

output_temps[n, h2, w, c] = 1 / sigma * (inputtemps[n, h2, w, c] - mean)

…

Output[n, sum(h1, h2, ⋯, hh), w, c] = [output_temps([n, h1, w, c], [n, h2, w, c], ⋯, [n, hn, w,c])]*gamma + beta

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float32.
- `[in] groupnum`: Input. The ci must be divisible by groupnum.
- `[in] gamma`: Input. A four-dimensional MLU input tensor, the shape is [1, 1, 1, ci], The data type of this tensor descriptor must be float32.
- `[in] beta`: Input. A four-dimensional MLU input tensor, the shape is [1, 1, 1, ci], The data type of this tensor descriptor must be float32.
- `[in] output`: Input. A four-dimensional MLU output tensor, the shape is [no, ho, wo, co], the shape of output is the same as that of input. The data type of this tensor descriptor must be float32.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVAILDPA`:
  - The operator pointer is null.
  - The input pointer is null.

4.73.2 cnmlComputeGroupNormOpForward

cnmlStatus_t `cnmlComputeGroupNormOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Parameters

- `[out] output`: Output. Pointer to the data after the GroupNorm operator is applied. the shape is [no, ho, wo, co], the shape of output is the same as that of input. The data type of this tensor descriptor must be float32.
- `[in] op`: Input. Pointer to the GroupNorm operator you have created.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni,hi,wi,ci].
- `[in] input`: Input. The shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float32.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [no,ho,wo,co].
- `[in] queue`: Input. Pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The input pointer is NULL.

4.74 GRU Operation

4.74.1 cnmlCreateGRUOp

cnmlStatus_t `cnmlCreateGRUOp`(cnmlBaseOp_t *op, cnmlRNNOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t state_input_tensor[], cnmlTensor_t output_tensor, cnmlTensor_t state_ouput_tensor[], cnmlTensor_t rx_weight_tensor[], cnmlTensor_t rh_weight_tensor[], cnmlTensor_t rx_bias_tensor[], cnmlTensor_t rh_bias_tensor[], cnmlTensor_t zx_weight_tensor[], cnmlTensor_t zh_weight_tensor[], cnmlTensor_t zx_bias_tensor[], cnmlTensor_t zh_bias_tensor[], cnmlTensor_t nx_weight_tensor[], cnmlTensor_t nh_weight_tensor[], cnmlTensor_t nx_bias_tensor[], cnmlTensor_t nh_bias_tensor[])

A function.

According to the basic operator pointer given by the user, a GRU operator is created.

gru(gated recurrent unit) operator is a variant of LSTM. gru has no cell, but only reset gate and update gate, which is simpler than LSTM struct.

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_type is int8, then filter_oc_type must be int8 and input_type can be float16 and float32.

If filter_type is int16, then filter_oc_type can be int16, and input_type can be float16 and float32.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - out_oc_type - out_type- filter_type - filter_oc_type

Supported combinations are:

float32-int16-int16-float32-int16-int16

float32-int16-int16-float32-float16-int16

float32-int16-int16-float32-float32-int16

float16-int16-int16-float16-int16-int16

float16-int16-int16-float16-float16-int16

float16-int16-int16-float16-float32-int16

float32-int8-int8-float32-int8-int8

float32-int8-int8-float32-float16-int8

float32-int8-int8-float32-float32-int8

float16-int8-int8-float16-int8-int8

float16-int8-int8-float16-float16-int8

float16-int8-int8-float16-float32-int8

Scale Limitation

MLU270:

filter_oc_type, in_oc_type and output_oc_type mast be same, and filter_oc_type, in_oc_type and output_oc_type can be int8 or int16

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer pointing to the struct describing the parameter of RNN operator.
- `[in] input_tensor`: Input. A 3-dimensional input tensor of MLU, the shape is [t, n, ci], supporting the data of float16 type.
- `[in] state_input_tensor`: Input. A state input tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [n, 1, 1,co],supporting the data of float16 type.
- `[in] rx_weight_tensor`: Input. A reset gate input weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,ci],supporting the data of float16 type.
- `[in] rh_weight_tensor`: Input. A reset gate state weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] rx_bias_tensor`: Input. A reset gate input bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] rh_bias_tensor`: Input. A reset gate state bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] zx_weight_tensor`: Input. An update gate input weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,ci],supporting the data of float16 type.
- `[in] zh_weight_tensor`: Input. An update gate state weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] zx_bias_tensor`: Input. An update gate input bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] zh_bias_tensor`: Input. An update gate state bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] nx_weight_tensor`: Input. An input weight tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1,1, ci],supporting the data of float16 type.
- `[in] nh_weigh_tensor`: Input. A state weight tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] nx_bias_tensor`: Input. An input bias tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] nh_bias_tenosr`: Input. A state bias tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] output_tensor`: Input. An output tensor of MLU, the shape is [n, 1, 1,co],supporting the data of float16 type.
- `[in] state_output_tensor`: Input. A state output tensor array of MLU, and the length of array is the number of gru layers, the shape of each tensor is [n, 1, 1,co],supporting the data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Any input is a null pointer.

4.74.2 cnmlComputeGRUOpForward

cnmlStatus_t **cnmlComputeGRUOpForward**(cnmlBaseOp_t op, void *input, void *state_input[], void *output, void *state_output[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

The basic recurrent neural network operator, input, output, parameter at runtime, and computational queue are created and called, and GRU operation is performed on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeGRUOpForward_V2 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to the output position.
- [in] op: Input. A pointer pointing to the base operator.
- [in] input: Input. An MLU address pointing to the input data.
- [in] state_input: Input. An MLU address points to state_input data array.
- [in] state_output: Input. An MLU address points to state_output data array.
- [in] compute_forw_param: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - op is a null pointer.
  - output is a null pointer.
  - state_output is a null pointer.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - The task type is invalid at runtime.

4.74.3 cnmlComputeGRUOpForward_V2

cnmlStatus_t **cnmlComputeGRUOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t state_input_tensors[], void *state_input[], cnmlTensor_t output_tensor, void *output, cnmlTensor_t state_output_tensors[], void *state_output[], cnrtQueue_t queue, void *extra)

A function.

The basic recurrent neural network operator, input, output, parameter at runtime, and computational queue are created and called, and GRU operation is performed on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensors: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. MLU address pointing to input data.
- [in] state_input_tensors: Input. State input MLU tensor array pointer. Pass NULL if not used.
- [in] state_input: Input. MLU address pointing to state_input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] state_output_tensors: Input. State_output MLU tensor array pointer. Pass NULL if not used.
- [out] state_ouput: Output. MLU address pointing to output array position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.75 GRU Pro Operation

4.75.1 cnmlCreateGRUProOp

cnmlStatus_t `cnmlCreateGRUProOp`(cnmlBaseOp_t *op, cnmlRNNOpParam_t param, cnmlGRUMode_t gru_mode, cnmlTensor_t input_tensor, cnmlTensor_t state_input_tensor[], cnmlTensor_t output_tensor, cnmlTensor_t state_ouput_tensor[], cnmlTensor_t rx_weight_tensor[], cnmlTensor_t rh_weight_tensor[], cnmlTensor_t rx_bias_tensor[], cnmlTensor_t rh_bias_tensor[], cnmlTensor_t zx_weight_tensor[], cnmlTensor_t zh_weight_tensor[], cnmlTensor_t zx_bias_tensor[], cnmlTensor_t zh_bias_tensor[], cnmlTensor_t nx_weight_tensor[], cnmlTensor_t nh_weight_tensor[], cnmlTensor_t nx_bias_tensor[], cnmlTensor_t nh_bias_tensor[])

A function.

According to the basic operator pointer given by the user, a GRU operator is created.

gru(gated recurrent unit) operator is a variant of LSTM. gru has no cell, but only reset gate and update gate, which is simpler than LSTM struct.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer pointing to the struct describing the parameter of RNN operator.
- `[in] gru_mode`: Input. An enumeration value representing the mode of gru operation.
- `[in] input_tensor`: Input. A 3-dimensional input tensor of MLU, the shape is [t, n, ci], supporting the data of float16 type.
- `[in] state_input_tensor`: Input. A state input tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [n, 1, 1,co],supporting the data of float16 type.
- `[in] rx_weight_tensor`: Input. A reset gate input weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,ci],supporting the data of float16 type.
- `[in] rh_weight_tensor`: Input. A reset gate state weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] rx_bias_tensor`: Input. A reset gate input bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] rh_bias_tensor`: Input. A reset gate state bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] zx_weight_tensor`: Input. An update gate input weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,ci],supporting the data of float16 type.
- `[in] zh_weight_tensor`: Input. An update gate state weight tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] zx_bias_tensor`: Input. An update gate input bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] zh_bias_tensor`: Input. An update gate state bias tensor array of MLU with the length of the number of layers of gru, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] nx_weight_tensor`: Input. An input weight tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1,1, ci],supporting the data of float16 type.
- `[in] nh_weigh_tensor`: Input. A state weight tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1,co],supporting the data of float16 type.
- `[in] nx_bias_tensor`: Input. An input bias tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] nh_bias_tenosr`: Input. A state bias tensor array of MLU in candidate state, and the length of array is the number of gru layers, the shape of each tensor is [co, 1, 1, 1], supporting the data of float16 type.
- `[in] output_tensor`: Input. An output tensor of MLU, the shape is [n, 1, 1,co],supporting the data of float16 type.
- `[in] state_output_tensor`: Input. A state output tensor array of MLU, and the length of array is the number of gru layers, the shape of each tensor is [n, 1, 1,co],supporting the data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Any input is a null pointer.

4.75.2 cnmlComputeGRUProOpForward

cnmlStatus_t `cnmlComputeGRUProOpForward`(cnmlBaseOp_t op, void *input, void *state_input[], void *output, void *state_output[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

The basic recurrent neural network operator, input, output, parameter at runtime, and computational queue are created and called, and GRU operation is performed on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeGRUProOpForward_V2 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.

- [in] state_input: Input. An MLU address points to state_input data array.
- [in] state_output: Input. An MLU address points to state_output data array.
- [in] compute_forw_param: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - op is a null pointer.
  - output is a null pointer.
  - state_output is a null pointer.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - The task type is invalid at runtime.

4.75.3 cnmlComputeGRUProOpForward_V2

cnmlStatus_t **cnmlComputeGRUProOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t state_input_tensors[], void *state_input[], cnmlTensor_t output_tensor, void *output, cnmlTensor_t state_output_tensors[], void *state_output[], cnrtQueue_t queue, void *extra)

A function.

The basic recurrent neural network operator, input, output, parameter at runtime, and computational queue are created and called, and GRU operation is performed on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. MLU address pointing to input data.
- [in] state_input_tensors: Input. State input MLU tensor array pointer. Pass NULL if not used.
- [in] state_input: Input. MLU address pointing to state_input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] state_output_tensors: Input. State output MLU tensor array pointer. Pass NULL if not used.
- [out] state_output: Output. An MLU address pointing to state_output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.76 Interp Operation

4.76.1 cnmlCreateInterpOpParam

cnmlStatus_t **cnmlCreateInterpOpParam**(cnmlInterpOpParam_t *param, int output_width, int output_height, bool align_corners)

A function.

Create a struct of parameters required by the interpolation operator.

Summary

For input[ni, ci, hi, wi] and output[no, co, ho, wo], if ni = no and ci = co:

The hi / ho, wi / wo are used to find interp location.

index_h = floor(hi / ho * h_iter)

index_w = floor(wi / wo *w_iter)

ho[h_iter] = (hi / ho * h_iter - index_h) * (hi[index_h] + 1) + (index_h + 1 - hi / ho * h_iter) * hi[index_h]

wo[w_iter] = (wi / wo * w_iter - index_w) * (wi[index_w] + 1) + (index_w + 1 - wi / wo * w_iter) * wi[index_w]

where inde_h represents the y-coordinates of data points, index_w represents the x-coordinates of data points, h_iter and w_iter represent the number of iteration, ho[h_iter] represents the ouput grid in y direction, wo[h_iter] represents the ouput grid in x direction

The value of wo is the same as the value of ho. The value of h_iter should be less than the value of ho. The value of w_iter should be less than the value of wo.

DataType

MLU270 and MLU220:

input_type = output_type

float16, float32

Scale Limitation

MLU270 and MLU220:

It is suggested that wo size not more than 1920. If exceedind this num, memory allocating errors may occury, and throw launch kernel failed error.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The smaller the size of hi, wi, ho, and wo, the better performance.

Parameters

- `[out] param`: Output. A pointer executing the operator struct.
- `[in] output_width`: Input. Specifying the width of output.
- `[in] output_height`: Input. Specifying the height of output.
- `[in] align_corner`: Input. When set to True, the values on the four corners of the output are the same as those on the four corners of the input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.2 cnmlCreateInterpOpParamByRatio

cnmlStatus_t `cnmlCreateInterpOpParamByRatio`(cnmlInterpOpParam_t *param, float zoom, bool align_corners)

A function.

Use the zoom coefficients to create a struct of interpolation operator parameters.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the operator struct.
- `[in] zoom`: Input. Coefficients.
- `[in] align_corner`: Input. When set to True, the values on the four corners of the output are the same as those on the four corners of the input.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.3 cnmlDestroyInterpOpParam

cnmlStatus_t `cnmlDestroyInterpOpParam`(cnmlInterpOpParam_t *param)

A function.

Release the struct of parameters required by the interpolation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer of the operator parameter struct to be released.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.4 cnmlCreateInterpOpParam_V2

cnmlStatus_t `cnmlCreateInterpOpParam_V2`(cnmlInterpOpParam_t *param, int output_width, int output_height, bool align_corners, bool align_center)

A function.

Create a struct of parameters required by the interpolation operator.

Summary

For input[ni, ci, hi, wi] and output[no, co, ho, wo], if ni = no and ci = co:

The hi / ho, wi / wo are used to find interp location.

offset_h = 0.5 * (hi / ho - 0.5)

offset_w = 0.5 * (wi / wo - 0.5)

index_h = floor(hi / ho * h_iter + offset_h)

index_w = floor(wi / wo *w_iter + offset_w)

ho[h_iter] = (hi / ho * h_iter + offset_h - index_h) * (hi[index_h] + 1) + (index_h + 1 - hi / ho * h_iter - offset_h) * hi[index_h]

wo[w_iter] = (wi / wo * w_iter + offset_w - index_w) * (wi[index_w] + 1) + (index_w + 1 - wi / wo * w_iter - offset_w) * wi[index_w]

where inde_h represents the y-coordinates of data points, index_w represents the x-coordinates of data points, h_iter and w_iter represent the number of iteration, ho[h_iter] represents the ouput grid in y direction, wo[h_iter] represents the ouput grid in x direction

The value of wo is the same as the value of ho. The value of h_iter should be less than the value of ho. The value of w_iter should be less than the value of wo.

DataType

MLU220 and MLU270:

input_type = output_type

float16, float32

Scale Limitation

MLU220 and MLU270:

It is suggested that wo size not more than 1920. If exceedind this num, memory allocating errors may occury, and throw launch kernel failed error.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The smaller the size of hi, wi, ho, and wo, the better performance.

Parameters

- `[out] param`: Output. A pointer executing the operator struct.
- `[in] output_width`: Input. Specifying the width of output.
- `[in] output_height`: Input. Specifying the height of output.
- `[in] align_corner`: Input. When set to True, the values on the four corners of the output are the same as those on the four corners of the input.
- `[in] align_center`: Input. When set to True, the values coordinate have 0.5 offset

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.5 cnmlCreateInterpOp

cnmlStatus_t **cnmlCreateInterpOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlInterpOpParam_t param)

A function.

Create an interpolation operator according to input and parameters of interpolation operator.

Summary

For input[ni, ci, hi, wi] and output[no, co, ho, wo], if ni = no and ci = co:

The hi / ho, wi / wo are used to find interp location.

index_h = floor(hi / ho * h_iter)

index_w = floor(wi / wo *w_iter)

ho[h_iter] = (hi / ho * h_iter - index_h) * (hi[index_h] + 1) + (index_h + 1 - hi / ho * h_iter) * hi[index_h]

wo[w_iter] = (wi / wo * w_iter - index_w) * (wi[index_w] + 1) + (index_w + 1 - wi / wo * w_iter) * wi[index_w]

where inde_h represents the y-coordinates of data points, index_w represents the x-coordinates of data points, h_iter and w_iter represent the number of iteration, ho[h_iter] represents the ouput grid in y direction, wo[h_iter] represents the ouput grid in x direction

The value of wo is the same as the value of ho. The value of h_iter should be less than the value of ho. The value of w_iter should be less than the value of wo.

DataType

MLU220 and MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

MLU220 and MLU270:

It is suggested that wo size not more than 1920. If exceedind this num, memory allocating errors may occury, and throw launch kernel failed error.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The smaller the size of hi, wi, ho, and wo, the better performance.

Parameters

- `[in] op`: Input. A pointer pointing to base operators address.
- `[in] input`: Input. A four-dimensional input tensor([n, h, w, c]).
- `[in] output`: Input. A four-dimensional output tensor([n, output_height, output_width, c]).
- `[in] param`: Input. A pointer of the operator parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.6 cnmlComputeInterpOpForward_V3

cnmlStatus_t `cnmlComputeInterpOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeInterpOpForward_V4 is recommended to use.

Compute the interpolation operator.

Summary

For input[ni, ci, hi, wi] and output[no, co, ho, wo], if ni = no and ci = co:

The hi / ho, wi / wo are used to find interp location.

index_h = floor(hi / ho * h_iter)

index_w = floor(wi / wo *w_iter)

ho[h_iter] = (hi / ho * h_iter - index_h) * (hi[index_h] + 1) + (index_h + 1 - hi / ho * h_iter) * hi[index_h]

wo[w_iter] = (wi / wo * w_iter - index_w) * (wi[index_w] + 1) + (index_w + 1 - wi / wo * w_iter) * wi[index_w]

where inde_h represents the y-coordinates of data points, index_w represents the x-coordinates of data points, h_iter and w_iter represent the number of iteration, ho[h_iter] represents the ouput grid in y direction, wo[h_iter] represents the ouput grid in x direction

The value of wo is the same as the value of ho. The value of h_iter should be less than the value of ho. The value of w_iter should be less than the value of wo.

DataType

MLU220 and MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

MLU220 and MLU270:

It is suggested that wo size not more than 1920. If exceedind this num, memory allocating errors may occury, and throw launch kernel failed error.

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The smaller the size of hi, wi, ho, and wo, the better performance.

Parameters

- `[in] op`: Input. A pointer pointing to the operator.
- `[in] input`: Input. Pointing to input data address.
- `[in] output`: Input. Pointing to output data address.
- `[in] compute_forw_param`: Input. Pointing to the address of parameter structs such as runtime data and device affinity.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.76.7 cnmlComputeInterpOpForward_V4

cnmlStatus_t `cnmlComputeInterpOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the interpolation operator.

Summary

For input[ni, ci, hi, wi] and output[no, co, ho, wo], if ni = no and ci = co:

The hi / ho, wi / wo are used to find interp location.

index_h = floor(hi / ho * h_iter)

index_w = floor(wi / wo *w_iter)

ho[h_iter] = (hi / ho * h_iter - index_h) * (hi[index_h] + 1) + (index_h + 1 - hi / ho * h_iter) * hi[index_h]

wo[w_iter] = (wi / wo * w_iter - index_w) * (wi[index_w] + 1) + (index_w + 1 - wi / wo * w_iter) * wi[index_w]

where inde_h represents the y-coordinates of data points, index_w represents the x-coordinates of data points, h_iter and w_iter represent the number of iteration, ho[h_iter] represents the ouput grid in y direction, wo[h_iter] represents the ouput grid in x direction

The value of wo is the same as the value of ho. The value of h_iter should be less than the value of ho. The value of w_iter should be less than the value of wo.

DataType

MLU220 and MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

MLU220 and MLU270:

It is suggested that wo size not more than 1920. If exceedind this num, memory allocating errors may occury, and throw launch kernel failed error.

Performance Optimization

The smaller the size of hi, wi, ho, and wo, the better performance.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.77 Less Operation

4.77.1 cnmlCreateLessOp

cnmlStatus_t `cnmlCreateLessOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Call after the tensor is created, and according to the base operator pointer given by the user, create an operator for comparing whether tensor 1 is smaller than tensor 2. Perform an element-wise comparison on the two tensors to find whether one is less than the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] < B[ni][hi][wi][ci])? 1:0

The shapes of two inputs and one output should be exactly the same.

Formula

c[n c h w] = a[n c h w] < b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU tensor, supports data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU tensor, supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU tensor, supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The input tensor type is not CNML_TENSOR or CNML_CONST.
  - Reason2 The CPU tensor bound by the bias tensor is null.

4.77.2 cnmlComputeLessOpForward_V3

cnmlStatus_t `cnmlComputeLessOpForward_V3`(cnmlBaseOp_t op, void *inputA, void *inputB, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLessOpForward_V4 is recommended to use.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is less than the other.

Formula

c[n c h w] = a[n c h w] < b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.77.3 cnmlComputeLessOpForward_V4

cnmlStatus_t `cnmlComputeLessOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor_A, void *inputA, cnmlTensor_t input_tensor_B, void *inputB, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform an element-wise comparison on the two four-dimensional tensors on the MLU to find whether one is less than the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor_A`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_A`: Input. MLU address pointing to inputA data.
- `[in] input_tensor_B`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_B`: Input. MLU address pointing to inputB data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.78 Less Equal Operation

4.78.1 cnmlCreateLessEqualOp

cnmlStatus_t `cnmlCreateLessEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the tensor B and the tensor A in each position to find whether one is less than or equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] <= B[ni][hi][wi][ci])? 1: 0.

The shapes of two inputs and one output should be exactly the same.

Formula

c[n c h w] = a[n c h w] <= b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] input_tensor_2`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.78.2 cnmlComputeLessEqualOpForward_V3

cnmlStatus_t `cnmlComputeLessEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLessEqualOpForward_V4 is recommended to use.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is less than or equal to the other.

Formula

c[n c h w] = a[n c h w] <= b[n c h w] ? 1 : 0

DataType

MLU270:

input : float16, float32, int32

output : the same as input or bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.78.3 cnmlComputeLessEqualOpForward_V4

cnmlStatus_t **cnmlComputeLessEqualOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is less than or equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.79 Log Operation

4.79.1 cnmlCreateLogOp

cnmlStatus_t **cnmlCreateLogOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a logarithm operator with the base being e.

The shapes of input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] input_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor. supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: The shape of output tensor is different from that of input tensor.

4.79.2 cnmlComputeLogOpForward_V3

cnmlStatus_t **cnmlComputeLogOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLogOpForward_V4 is recommended to use.

Compute the user-specified logarithmic operator with the base being e.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address pointing to input data.
- [in] computue_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.79.3 cnmlComputeLogOpForward_V4

cnmlStatus_t **cnmlComputeLogOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified exponent operator with the base being e.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.80 Log2 Operation

4.80.1 cnmlCreateLog2Op

cnmlStatus_t **cnmlCreateLog2Op**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Creates a logarithmic operator with base 2.

The shapes of input and output tensor should be the same.

Formula

output[n c h w] = log2(input[n c h w])

DataType

MLU270: float16, float32

Scale Limitation

MLU270: unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the logarithm operator you have created.
- [in] input_tensor: Input. A 1 to N dimension input tensor. The data type of this tensor descriptor must be float16 or float32. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.
- [in] output_tensor: Input. The descriptor of the 1 to N dimensional output tensor. The data type of this tensor descriptor must be float16 or float32. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.

Return Value

- CNML_STATUS_SUCCESS: This function run successfully.
- CNML_STATUS_INVALIDPARAM: The shape of the output tensor is different from the shape of the input tensor.

4.80.2 cnmlComputeLog2OpForward

cnmlStatus_t `cnmlComputeLog2OpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLog2OpForward_V2 is recommended to use.

Computes the logarithmic operator with base 2 on MLU.

Formula

output[n c h w] = log2(input[n c h w])

DataType

MLU270: float16, float32

Scale Limitation

MLU270: unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. A pointer to output data after the logarithmic operator is applied.
- `[in] op`: Input. A pointer to the logarithmic operator you have created.
- `[in] input`: Input. A pointer to the input data you want to compute.
- `[in] computue_forw_param`: Input. A pointer to the struct address that records the data parallelism and device affinity for runtime.
- `[in] queue`: Input.A pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions are met:
  - The operator pointer is NULL.
  - The output pointer is NULL.
- `CNML_STATUS_INVALIDARG`: The runtime task type is invalid.

4.80.3 cnmlComputeLog2OpForward_V2

cnmlStatus_t `cnmlComputeLog2OpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified exponent operator with the base being 2.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.81 Log Softmax Operation

4.81.1 cnmlCreateLogSoftmaxOp

cnmlStatus_t `cnmlCreateLogSoftmaxOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlDimension_t dim)

cnmlCreateLogSoftmaxOp.

This function creates a logarithmic exponential normalization operator based on the basic operator pointer given by the user. In creation, it is necessary to indicate in which dimension the logarithmic index normalization operation is performed.

Formula

Such as when input[ni, ci, hi, wi], output[no, co, ho, wo] and `d` direction is `c`, then output[n, c, h, w] = logsoftmax(n, c, h, w) = log(exp(input[n, c, h, w]) / sum_i(exp(input[n, i, h, w])))

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c < 1024 because of at least process one full line at a time

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. 4-dimensional input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. 4-dimensional output tensor, of which the shape is [no, co, ho, wo] (no = ni, co = ci, ho = ci, wo = wi), supporting data of float16 type.
- `[in] dim`: Input. enumeration variables of cnmlDimension_t type that specifies which dimension to operate in. The options include CNML_DIM_N, CNML_DIM_C, CNML_DIM_H, and CNML_DIM_W.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDARG.`: At least one of the following conditions are met:
  - Input is not empty.

4.81.2 cnmlCreateNdLogSoftmaxOp

cnmlStatus_t **`cnmlCreateNdLogSoftmaxOp`**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, int dim)

cnmlCreateNdLogSoftmaxOp.

A function that creates an operator for the Log Softmax operation (logarithmic index normalization) based on the base operator pointer given by the user. In the cpu version of the operator, OUT = IN - log(reduce_sum(exp(IN)), dim)); in the mlu version, IN = IN - reduce_max(IN, dim), OUT = IN - log(reduce_sum(exp(IN), dim)).

Notes:

- Support tensor of arbitrary dimensional. Require to have the same input and output shape.
- dim belongs to [0, dim_num-1], and dim_num is the number of dimensions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] output`: Input. A MLU tensor of arbitrary dimension, supporting data of float16 type.
- `[in] input`: Input. A MLU tensor of arbitrary dimension, supporting data of float16 type.
- `[in] dim`: Input. The selected dimension, counting from 0

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - The operator pointer is empty.
  - The input and output is empty.

4.81.3 cnmlComputeLogSoftmaxOpForward_V3

cnmlStatus_t **`cnmlComputeLogSoftmaxOpForward_V3`**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeLogSoftmaxOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeLogSoftmaxOpForward_V4 is recommended to use.

It is used to compute the user-specified MLP operator on the MLU.

After creating the MLP operator, Input, Output, and computation stream, pass them to the function to It is used to compute the MLP operator.

Formula

Such as when input[ni, ci, hi, wi], output[no, co, ho, wo] and `d` direction is `c`, then output[n, c, h, w] = logsoftmax(n, c, h, w) = log(exp(input[n, c, h, w]) / sum_i(exp(input[n, i, h, w])))

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

c < 1024 because of at least process one full line at a time

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address specifying the output position.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.
- [in] compute_forw_param: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.81.4 cnmlComputeLogSoftmaxOpForward_V4

cnmlStatus_t **cnmlComputeLogSoftmaxOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeLogSoftmaxOpForward_V4.

It is used to compute the user-specified MLP operator on the MLU.

After creating the MLP operator, Input, Output, and computation queue, pass them to the function to It is used to compute the MLP operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.81.5 cnmlComputeNdLogSoftmaxOpForward

cnmlStatus_t **cnmlComputeNdLogSoftmaxOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNdLogSoftmaxOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeNdLogSoftmaxOpForward_V2 is recommended to use.

Compute a Log Softmax operation that supports arbitrary dimensions (logarithmic index normalization).

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address that points to the output data.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address pointing to the input data tensor.
- [in] compute_forw_param: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.81.6 cnmlComputeNdLogSoftmaxOpForward_V2

cnmlStatus_t **cnmlComputeNdLogSoftmaxOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNdLogSoftmaxOpForward_V2.

Compute a Log Softmax operation that supports arbitrary dimensions (logarithmic index normalization).

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.82 Lrn Operation

4.82.1 cnmlCreateLrnOpParam

cnmlStatus_t **cnmlCreateLrnOpParam**(cnmlLrnOpParam_t *param, cnmlLrnType_t lrn_type, int local_size, double alpha, double beta, double k)

A function.

This function creates a struct of Lrn operator operation parameters according to pointers given by users, and fills the struct with the parameters input by users.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the struct address of Lrn operator operaion parameters.
- `[in] type`: Input. An enumeration type, representing the mode of lrn parameters.
- `[in] local_size`: Input. Extract local_size numbers in the direction c.
- `[in] alpha`: Input. A variance scaling parameter.
- `[in] beta`: Input. An exponential item.
- `[in] k`: Input. A hyper parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. param limitation: Highly recommend to use hyper parameters as follows: local_size: 5 alpha: 0.0001 beta: 0.75 k: 2

4.82.2 cnmlDestroyLrnOpParam

cnmlStatus_t **cnmlDestroyLrnOpParam**(cnmlLrnOpParam_t *param)

A function.

Release the struct pointer of Lrn operator operation parameters according to the pointer given by users.

After the operation of the Lrn operator is finished, release the struct pointer of the Lrn operator operation parameters.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the struct address of the Lrn operator operation parameters.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.82.3 cnmlCreateLrnOp

cnmlStatus_t **cnmlCreateLrnOp**(cnmlBaseOp_t *op, cnmlLrnOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create an Lrn operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create Lrn operator.

Formula

a. CNML_LRN_V1, Yi = Xi / [(alpha * sum(Xj^2) / m + k) ^ beta], m = min(local_size, 2*ci-1)
b. CNML_LRN_V2, Yi = Xi / [(alpha * sum(Xj^2) + k) ^ beta]
c. CNML_LRN_V3, Yi = Xi / [(alpha * sum(Xj^2) / local_size + k) ^ beta]
   j = i - local_size / 2 ~~ i + local_size / 2, (i and j are in C dimention)

DataType

MLU270:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Conditions & Limitations

The onchip position and scale of input tensor should be calculated by alhpa*(input_data^2).

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] param`: Input. A pointer pointing to the struct of the Lrn operator operation parameters.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape of which is [n, h, w, c], supporting data of float16 type.
- `[in] output`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, ho, wo, co], supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.82.4 cnmlComputeLrnOpForward_V3

cnmlStatus_t `cnmlComputeLrnOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLrnOpForward_V4 is recommended to use.

Compute the Lrn operator specified by users on the MLU.

After creating a Lrn operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Lrn operator.

Formula

a. CNML_LRN_V1, Yi = Xi / [(alpha * sum(Xj^2) / m + k) ^ beta], m = min(local_size, 2*ci-1)
b. CNML_LRN_V2, Yi = Xi / [(alpha * sum(Xj^2) + k) ^ beta]
c. CNML_LRN_V3, Yi = Xi / [(alpha * sum(Xj^2) / local_size + k) ^ beta]
   j = i - local_size / 2 ~~ i + local_size / 2, (i and j are in C dimention)

DataType

MLU270:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.82.5 cnmlComputeLrnOpForward_V4

cnmlStatus_t `cnmlComputeLrnOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Lrn operator specified by users on the MLU.

After creating a Lrn operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Lrn operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.83 Lstm Pro Opreation

4.83.1 cnmlCreateLSTMClipParam

cnmlStatus_t `cnmlCreateLSTMClipParam`(cnmlLSTMClipParam_t *param, double min, double max)

A function.

Create a parameter for describing LSTM for clip.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to pointer, the pointer that is pointed points to a struct. The struct is used for describing how to intercept cell_state, the interception range is [min, max]. Namely: if cell_state<min, then cell_state = min; if cell_state > max, cell_sate = max.
- `[in] min`: Input. The minimum value in the interception range.
- `[in] max`: Input. The maximum value in the interception range.

4.83.2 cnmlDestroyLSTMClipParam

cnmlStatus_t `cnmlDestroyLSTMClipParam`(cnmlLSTMClipParam_t *param)

A function.

Destroy the parameter for describing LSTM intercepts cellState.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: A pointer to pointer, the pointer that is pointed points to a parameter struct.

4.83.3 cnmlCreateLSTMProjectionParam

cnmlStatus_t `cnmlCreateLSTMProjectionParam`(cnmlLSTMProjectionParam_t *param, bool is_rec_proj, int rec_proj_size, cnmlActiveFunction_t rec_active_func, bool is_out_proj, int out_proj_size, cnmlActiveFunction_t out_active_func)

A function.

Create a parameter for describing LSTM for recurrent projection and output projection.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct.
- `[in] is_rec_proj`: Input. Whether to enable recurrent projection.
- `[in] rec_proj_size`: Input. Size after projection.
- `[in] rec_act_fun`: Input. An activation function used on the projection result after recurrent projection.
- `[in] is_out_proj`: Input. Not supported yet.
- `[in] out_proj_size`: Input. Not supported yet.
- `[in] out_act_fun`: Input. Not supported yet.

4.83.4 cnmlDestroyLSTMProjectionParam

cnmlStatus_t `cnmlDestroyLSTMProjectionParam`(cnmlLSTMProjectionParam_t *param)

A function.

Destroy the parameter for describing LSTM for recurrent projection and output projection.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: Successfully created a clipping operation. Return the corresponding error code when execution is failed.

4.83.5 cnmlCreateLSTMPeepholeParam

cnmlStatus_t `cnmlCreateLSTMPeepholeParam`(cnmlLSTMPeepholeParam_t *param, bool enable_peephole)

A function.

Create a parameter for describing LSTM for peephole.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct.
- `[in] peephole`: Input. Whether to enable peephole.

4.83.6 cnmlDestroyLSTMPeepholeParam

cnmlStatus_t `cnmlDestroyLSTMPeepholeParam`(cnmlLSTMPeepholeParam_t *param)

A function.

Destroy the parameter of describing LSTM for peephole.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct.

4.83.7 cnmlCreateLSTMProParam

cnmlStatus_t `cnmlCreateLSTMProParam`(cnmlLSTMProParam_t *param, cnmlRNNOpParam_t rnn_param, cnmlLSTMClipParam_t clip_param, cnmlLSTMProjectionParam_t proj_param, cnmlLSTMPeepholeParam_t peephole_param)

A function.

Create parameters for describing LSTMPro operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct that describes LSTMPro operator.
- `[in] rnn_param`: Input. A previously created pointer to the recurrent neural network parameter struct.
- `[in] clip_param`: Input. Parameter for describing LSTM intercepts cellState.
- `[in] proj_param`: Input. Parameter for describing LSTM recurrent projection and output projection.
- `[in] peephole_param`: Input. Parameter for describing whether LSTM enables peephol.

4.83.8 cnmlDestroyLSTMProParam

cnmlStatus_t `cnmlDestroyLSTMProParam`(cnmlLSTMProParam_t *param)

A function.

Destroy the parameters of LSTMPro operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to pointer, the pointer that is pointed points to a parameter struct.

4.83.9 cnmlCreateLSTMProOp

cnmlStatus_t `cnmlCreateLSTMProOp`(cnmlBaseOp_t *op, cnmlLSTMProParam_t lstm_pro_param, cnmlTensor_t x, cnmlTensor_t hx[], cnmlTensor_t cx[], cnmlTensor_t y, cnmlTensor_t hy[], cnmlTensor_t cy[], cnmlTensor_t filter_forget_x[], cnmlTensor_t filter_forget_h[], cnmlTensor_t filter_forget_c[], cnmlTensor_t bias_forget_x[], cnmlTensor_t bias_forget_h[], cnmlTensor_t filter_input_x[], cnmlTensor_t filter_input_h[], cnmlTensor_t filter_input_c[], cnmlTensor_t bias_input_x[], cnmlTensor_t bias_input_h[], cnmlTensor_t filter_update_x[], cnmlTensor_t filter_update_h[], cnmlTensor_t bias_update_x[], cnmlTensor_t bias_update_h[], cnmlTensor_t filter_output_x[], cnmlTensor_t filter_output_h[], cnmlTensor_t filter_output_c[], cnmlTensor_t bias_output_x[], cnmlTensor_t bias_output_h[], cnmlTensor_t filter_rec_proj[], cnmlTensor_t bias_rec_proj[], cnmlTensor_t filter_out_proj[], cnmlTensor_t bias_out_proj[])

A function.

Create a parameter for describing LSTM intercepts cellState.

input [N, C] , [N, T, C] or [T, N, C]

output [N, C] , [N, T, C] or [T, N, C]

Formula

design formulas of Standard

f = sigmoid(w1*x + w2*h + b1 + b2)

i = sigmoid(w4*x + w5*h + b4 + b5)

c = f (*) ct-1 + i (*) tanh(w7*x + w8*h + b7 + b8)

o = sigmoid(w9*x + w10*h + b9 + b10)

design formulas of Clip

f = sigmoid(w1*x + w2*h + b1 + b2)

i = sigmoid(w4*x + w5*h + b4 + b5)

c = f (*) ct-1 + i (*) tanh(w7*x + w8*h + b7 + b8)

c = c.clip(min, max)

o= sigmoid(w9*x + w10*h + w11(*)c + b9 + b10)

design formulas of Projection

f = sigmoid(w1*x + w2*h + b1 + b2)

i = sigmoid(w4*x + w5*h + b4 + b5)

c = f (*) ct-1 + i (*) tanh(w7*x + w8*h + b7 + b8)

o = sigmoid(w9*x + w10*h + b9 + b10)

h = o * tanh(ct)

h = tanh(w12*h + b12)

design formulas of Peephole

f = sigmoid(w1*x + w2*h + w3(*)c + b1 + b2)

i = sigmoid(w4*x + w5*h + w6(*)c + b4 + b5)

c = f (*) ct-1 + i (*) tanh(w7*x + w8*h + b7 + b8)

o = sigmoid(w9*x + w10*h + w11(*)c + b9 + b10)

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_type is int8, then filter_oc_type must be int8 and input_type can be float16 and float32.

If filter_type is int16, then filter_oc_type can be int16, and input_type can be float16 and float32.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - out_oc_type - out_type- filter_type - filter_oc_type

Supported combinations are:

float32-int16-int16-float32-int16-int16

float32-int16-int16-float32-float16-int16

float32-int16-int16-float32-float32-int16

float16-int16-int16-float16-int16-int16

float16-int16-int16-float16-float16-int16

float16-int16-int16-float16-float32-int16

float32-int8-int8-float32-int8-int8

float32-int8-int8-float32-float16-int8

float32-int8-int8-float32-float32-int8

float16-int8-int8-float16-int8-int8

float16-int8-int8-float16-float16-int8

float16-int8-int8-float16-float32-int8

Scale Limitation

MLU270:

filter_oc_type, in_oc_type and output_oc_type mast be same, and filter_oc_type, in_oc_type and output_oc_type can be int8 or int16

The interface is used for creating a LSTMPro operator.

If all the parameters in the interface are arrays, the length of the arrays is num_layers, and the array element with the subscript being 0 indicates the initial value of the hidden state of the first layer, the subscript being 1 indicates the second layer, and so on.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] lstm_pro_param`: Input. Describes the struct pointer of all parameters of LSTM.
- `[in] x`: Input. A three-dimensional tensor, the shape is [seqLength,batchSize,vectorSize].
- `[in] hx`: Input. A four-dimensional tensor array, the hiddenState input of LSTM. The shape is [batchSize,hiddenSize,1,1]. If there is recurrent projection, the shape is [batchSize,recProjSize,1,1].
- `[in] cx`: Input. A four-dimensional tensor array, the cellState input of LSTM. The shape is [batchSize,hiddenSize,1,1].
- `[in] y`: Input. A three-dimensional tensor, the shape is [seqLength,batchSize,hiddenSize]. If there is recurrent projection, the shape is [seqLength,batchSize,recProjSize].
- `[in] hy`: Input. A four-dimensional tensor array, the hiddenState output of LSTM. The shape is [batchSize,hiddenSize,1,1]. If there is recurrent projection, the shape is [batchSize,recProjSize,1,1].
- `[in] cy`: Input. A four-dimensional tensor array, the cellState output of LSTM. The shape is [batchSize,hiddenSize,1,1].
- `[in] filter_forget_x`: Input. A four-dimensional tensor array, the weight of x in the forget gate.
- `[in] filter_forget_h`: Input. A four-dimensional tensor array, the weight of hiddenState in the forget gate.
- `[in] filter_forget_c`: Input. A four-dimensional tensor array, the weight of cellState in the forget gate.
- `[in] bias_forget_x`: Input. A four-dimensional tensor array, the bias of x in the forget gate.
- `[in] bias_forget_h`: Input. A four-dimensional tensor array, the bias of hiddenState in the forget gate.
- `[in] filter_input_x`: Input. A four-dimensional tensor array, the weight of x in the input gate.
- `[in] filter_input_h`: Input. A four-dimensional tensor array, the weight of hiddenState in the input gate .
- `[in] filter_input_c`: Input. A four-dimensional tensor array, the weight of cellState in the input gate.
- `[in] bias_input_x`: Input. A four-dimensional tensor array, the bias of x in the input gate..
- `[in] bias_input_h`: Input. A four-dimensional tensor array, the bias of hiddenState in the input gate
- `[in] filter_update_x`: Input. A four-dimensional tensor array, the weight of x in the update gate.
- `[in] filter_update_h`: Input. A four-dimensional tensor array, the weight of hiddenState in the update gate.
- `[in] bias_update_x`: Input. A four-dimensional tensor array, the bias of x in the update gate. .
- `[in] bias_update_h`: Input. A four-dimensional tensor array, the bias of hiddenState in the update gate.
- `[in] filter_output_x`: Input. A four-dimensional tensor array, the weight of x in the output gate.
- `[in] filter_output_h`: Input. A four-dimensional tensor array, the weight of hiddenState in the output gate.
- `[in] filter_output_c`: Input. A four-dimensional tensor array, the weight of cellState in the output gate.
- `[in] bias_output_x`: Input. A four-dimensional tensor array, the bias of x in the output gate.
- `[in] bias_output_h`: Input. A four-dimensional tensor array, the bias of hiddenState in the output gate.
- `[in] filter_rec_proj`: Input. A four-dimensional tensor array, the weight of recurrent projection.
- `[in] bias_rec_proj`: Input. A four-dimensional tensor array, the bias of recurrent projection.
- `[in] filter_out_proj`: Input. Retain, not supported yet.
- `[in] bias_out_proj`: Input. Retain, not supported yet.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.83.10 cnmlAddLSTMMask

cnmlStatus_t **cnmlAddLSTMMask**(cnmlBaseOp_t op, cnmlTensor_t mask[])

A function.

According to the base operator pointer given by the user, lstm’ s mask tensor array has been add into.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to the address of the base operator.
- [in] mask: Input. A 2-dimensional MLU tensor, the shape is [t, n], supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by the bias tensor is null.

4.83.11 cnmlComputeLSTMProOpForward

cnmlStatus_t **cnmlComputeLSTMProOpForward**(cnmlBaseOp_t op, void *x, void *hx[], void *cx[], void *y, void *hy[], void *cy[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeLSTMProOpForward_V2 is recommended to use.

Compute the previously created LSTM operator on the MLU.

If all the parameters in the interface are arrays, the length of the arrays is num_layers, and the array element with the subscript being 0 indicates the initial value of the hidden state of the first layer, the subscript being 1 indicates the second layer, and so on.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] y: Output. Points to the output data initial address of LSTM.
- [out] hy: Output. A pointer array, points to the hiddenState output data initial address of LSTM.
- [out] cy: Output. A pointer array, points to the cellState output data initial address of LSTM.
- [in] x: Input. Points to the input data initial address of LSTM.
- [in] hx: Input. A pointer array, points to the hiddenState input data initial address of LSTM.
- [in] cx: Input. A pointer array, points to the cellState input data initial address of LSTM.
- [in] compute_forw_param: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- [in] queue: Input. A computation queue pointer..

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.83.12 cnmlComputeLSTMProOpForward_V2

cnmlStatus_t **cnmlComputeLSTMProOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t x_tensor, void *x, cnmlTensor_t hx_tensors[], void *hx[], cnmlTensor_t cx_tensors[], void *cx[], cnmlTensor_t y_tensor, void *y, cnmlTensor_t hy_tensors[], void *hy[], cnmlTensor_t cy_tensors[], void *cy[], cnrtQueue_t queue, void *extra)

A function.

Compute the previously created LSTM operator on the MLU.

If all the parameters in the interface are arrays, the length of the arrays is num_layers, and the array element with the subscript being 0 indicates the initial value of the hidden state of the first layer, the subscript being 1 indicates the second layer, and so on.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] x_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] x: Input. MLU address pointing to x data.
- [in] hx_tensor: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] hx: Input. MLU address pointing to hx data.
- [in] cx_tensor: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] cx: Input. MLU address pointing to cx data.
- [in] y_tensor: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] y: Input. MLU address pointing to y data.
- [in] hy_tensor: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] hy: Input. MLU address pointing to hy data.
- [in] cy_tensor: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] cy: Input. MLU address pointing to cy data.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.83.13 cnmlComputeLSTMMaskProOpForward

cnmlStatus_t **cnmlComputeLSTMMaskProOpForward**(cnmlBaseOp_t op, void *x, void *hx[], void *cx[], void *mask[], void *y, void *hy[], void *cy[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the previously created LSTM operator on the MLU.

If all the parameters in the interface are arrays, the length of the arrays is num_layers, and the array element with the subscript being 0 indicates the initial value of the hidden state of the first layer, the subscript being 1 indicates the second layer, and so on.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] y: Output. Points to the output data initial address of LSTM.
- [out] hy: Output. A pointer array, points to the hiddenState output data initial address of LSTM.
- [out] cy: Output. A pointer array, points to the cellState output data initial address of LSTM.
- [in] x: Input. Points to the input data initial address of LSTM.
- [in] hx: Input. A pointer array, points to the hiddenState input data initial address of LSTM.
- [in] cx: Input. A pointer array, points to the cellState input data initial address of LSTM.
- [in] mask: Input. A pointer array, points to the mask input data initial address of LSTM.
- [in] compute_forw_param: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- [in] queue: Input. A computation queue pointer..

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.84 MaskZero Operation

4.84.1 cnmlCreateMaskZeroOp

cnmlStatus_t **cnmlCreateMaskZeroOp**(cnmlBaseOp_t *op, cnmlMaskZeroLayerParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t label_tensor, cnmlTensor_t output_tensor)

A function.

Description

Creates a Maskzero operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, a pointer to maskzero operation param, input, label and output tensors, pass them to the function to create a Maskzero operator.

The maskzero operation is to mask some data in input tensor according to label tensor. The C dimension of label tensor is equal to the W dimension of input tensor, in order to mask input tensor C and H dimension data according to N and W dimension index.

Related APIs

Before calling this API, you need to call the following APIs: cnmlCreateMaskZeroOpParam, cnmlCreateTensor

After calling this API, you may need to call the following APIs: cnmlDestroyBaseOp, cnmlFuseOp(optional), cnmlComputeMaskZeroOp(optional)

Lifetime

The op can be released after calling cnmlFuseOp or cnmlComputeMaskZeroOp. The param, input_tensor, label_tensor, and output_tensor can be released intermediatelly after this function.

Formula

fuse_relu == 0:

if label[n, w, 1, 1] == pad_label:

output[n, c, h, w] = 0

else:

output[n, c, h, w] = input[n, c, h, w]

fuse_relu == 1:

if label[n, w, 1, 1] == pad_label:

output[n, c, h, w] = 0

else if input[n, c, h, w] < 0:

output[n, c, h, w] = slope * input[n, c, h, w]

else:

output[n, c, h, w] = input[n, c, h, w]

DataType

MLU220:

- input: float16, float32
- output: float16, float32

MLU270:

- input: float16, float32
- output: float16, float32

Scale Limitation

MLU220:

Unlimited.

MLU270:

Unlimited.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A struct pointer of maskzero operation parameter.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi].
- `[in] label_tensor`: Input. A 4-dimensional MLU label tensor, the shape is [ni, wi, 1, 1].
- `[in] output_tensor`: Output. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni, co = ci, ho = hi, wo = wi).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is neither CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.84.2 cnmlCreateMaskZeroOpParam

cnmlStatus_t **`cnmlCreateMaskZeroOpParam`**(cnmlMaskZeroLayerParam_t *param, float pad_label, float slope, bool fuse_relu)

cnmlCreateMaskZeroOpParam.

According to the pointer given by the user, the function creates the parameters required by MaskZero operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Output. A pointer pointing to the address of the parameter of the MaskZero operator.
- `[in] pad_label`: Input. It should be an float32 value
- `[in] slope`: Input. It should be an float32 value.
- `[in] fuse_relu`: Input. It should be an bool value.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.84.3 cnmlDestroyMaskZeroOpParam

cnmlStatus_t **`cnmlDestroyMaskZeroOpParam`**(cnmlMaskZeroLayerParam_t *param)

A function.

This function is used to destroy an instance of cnmlMaskZeroLayerParam_t.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer instance pointing to the cnmlMaskZeroParam.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The pointer instance of cnmlQuantizedParam is null.
  - The content of the pointer pointed to by param has been freed.

4.84.4 cnmlComputeMaskZeroOpForward

cnmlStatus_t `cnmlComputeMaskZeroOpForward`(cnmlBaseOp_t op, void *input, void *label, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeMaskZeroOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeMaskZeroOpForward_V4 is recommended to use.

Compute the MaskZero operator given by users on the MLU.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. A 4-D tensor, supporting data of float16/float32 type.
- `[in] label`: Input. A 4-D tensor, supporting data of float16/float32 type.
- `[out] output`: output. A 4-D tensor, supporting data pf float16/float32 type
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.84.5 cnmlComputeMaskZeroOpForward_V4

cnmlStatus_t `cnmlComputeMaskZeroOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t label_tensor, void *label, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeMaskZeroOpForward_V4.

Compute the MaskZero operator given by users on the MLU.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. A 4-D tensor, supporting data of float16/float32 type.
- `[in] label_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] label`: Input. A 4-D tensor, supporting data of float16/float32 type.
- `[out] output_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[out] output`: output. A 4-D tensor, supporting data pf float16/float32 type
- `[in] queue`: Input. A computational queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.85 Matrix Mult Operation

4.85.1 cnmlCreateMatrixMultOp

cnmlStatus_t `cnmlCreateMatrixMultOp`(cnmlBaseOp_t *op, cnmlTensor_t lhs_tensor, cnmlTensor_t rhs_tensor, cnmlTensor_t output_tensor, cnmlTensor_t bias_tensor)

A function.

The function creates a matrix multiplication operator according to base operator pointers given by users.

After creating base operator pointers, matrix multiplication operator input tensor, weight tensor, output tensor, and bias Tensor, pass them into the function to create matrix multiplication operators.

If the order of input, weight, and output is NCHW, output [n, coo, 1, 1] = Σinput[n, cii, h, w]

- filter[coo, cii, h, w] and sum the subscripts cii, h, and W. The sum range is [0, cl-1], [0, hl-1], and [0, wl-1] respectively. If the bias tensor is not null, bias is added to the final output.

Notice: matrix_mult is not supported in single core situation. If running single op, the core_limit in cnmlCompileBaseOp and core_num in cnmlSetBaseOpCoreNum should not equal to 1. And if running a network including this op, the core_limit in cnmlCompileFusionOp also should not equanl to 1.

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer which points to the base operator address.
- `[in] lhs_tensor`: Input. A four-dimensional input tensor, the shape of which is [nl, cl, hl, wl], supporting data of float16 type.

- [in] rhs_tensor: Input. A four-dimensional weight tensor, the shape of which is [nr, cr, hr, wr] (cr = cl, hr = hl, wr = wl), supporting data of float16 type.
- [in] output_tensor: Input. A four-dimensional output tensor, the shape of which is [no, co, ho, wo] (no = nl, co = nr, ho = 1, wo = 1), supporting data of float16 type.
- [in] bias_tensor: Input. A four-dimensional bias tensor, the shape of which is [nb, cb, hb, wb] (nb = 1, cb = co, hb = 1, wb = 1), supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.85.2 cnmlComputeMatrixMultOpForward_V3

cnmlStatus_t **cnmlComputeMatrixMultOpForward_V3**(cnmlBaseOp_t op, void *lhs, void *rhs, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMatrixMultOpForward_V4 is recommended to use.

Compute the matrix multiplication operator specified by users on the MLU.

After creating matrix multiplication operators, input, output, and computation stream, pass them into the function to compute the matrix multiplication operator.

Supports MLU220 and MLU270.

Parameters

- [out] output: Output. An MLU address specifying the output position.
- [in] op: Input. A pointer which points to base operators.
- [in] lhs: Input. An MLU address which points to input data.
- [in] rhs: Input. An MLU address which points to input weight data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.85.3 cnmlComputeMatrixMultOpForward_V4

cnmlStatus_t **cnmlComputeMatrixMultOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t lhs_tensor, void *lhs, cnmlTensor_t rhs_tensor, void *rhs, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the matrix multiplication operator specified by users on the MLU.

After creating matrix multiplication operators, input, output, and computation queue, pass them into the function to compute the matrix multiplication operator.

Supports MLU220 and MLU270.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] lhs_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] lhs: Input. MLU address pointing to lhs data.
- [in] rhs_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] rhs: Input. MLU address pointing to rhs data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.86 Max Operation

4.86.1 cnmlCreateMaxOp

cnmlStatus_t `cnmlCreateMaxOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t index_tensor)

A function.

Create a max operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, pass them into the fucntion to create a max operator.

This operator compares all the input data, outputs the maximum value, and gives the index of the maximum value. Assuming the position of the maximum value is input[n, c, h, w], then index = n * H * W * C + h * W * C + w * C + c.

Formula

max = MAX(a[n c h w])

DataType

MLU270:

The output_data_type and input_data_type must be the same. The supported data types are float16 and float32.

index_data_type:

if input_data_type = float16 then index_data_type = int16 or int32

if input_data_type = float32 then index_data_type = int32

recommeded to be int32, almost has no performance difference with int16.

Scale Limitation

MLU270:

if index_data_type = int16, input_size = N * H * W * C <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16, float32 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [1, 1, 1, 1], supporting data of float16, float32 type.
- `[in] index_tensor`: Input. A four-dimensional MLU index vector the shape of which is [1, 1, 1, 1], supporting data of int16 and int32 types.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is neither CNML_TENSOR nor CNML_CONST.

4.86.2 cnmlComputeMaxOpForward_V3

cnmlStatus_t `cnmlComputeMaxOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, void *index, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMaxOpForward_V4 is recommended to use.

Compute the max operator specified by users on the MLU.

After creating a max operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the max operator.

Formula

max = MAX(a[n c h w])

DataType

MLU270:

The output_data_type and input_data_type must be the same. The supported data types are float16 and float32.

index_data_type:

if input_data_type = float16 then index_data_type = int16 or int32

if input_data_type = float32 then index_data_type = int32

recommeded to be int32, almost has no performance difference with int16.

Scale Limitation

MLU270:

if index_data_type = int16, input_size = N * H * W * C <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[out] index`: Output. An MLU address pointing to the index position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.86.3 cnmlComputeMaxOpForward_V4

cnmlStatus_t `cnmlComputeMaxOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t index_tensor, void *index, cnrtQueue_t queue, void *extra)

A function.

Compute the max operator specified by users on the MLU.

After creating a max operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the max operator.

Formula

max = MAX(a[n c h w])

DataType

MLU270:

The output_data_type and input_data_type must be the same. The supported data types are float16 and float32.

index_data_type:

if input_data_type = float16 then index_data_type = int16 or int32

if input_data_type = float32 then index_data_type = int32

recommeded to be int32, almost has no performance difference with int16.

Scale Limitation

MLU270:

if index_data_type = int16, input_size = N * H * W * C <= 32768

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input`: Input. MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[in] output`: Input. MLU address pointing to output data.
- `[in] index_tensor`: Input. Index MLU tensor pointer. Pass NULL if not used.
- `[out] index`: Output. An MLU address pointing to index position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.87 Maximum Operation

4.87.1 cnmlCreateMaximumOp

cnmlStatus_t `cnmlCreateMaximumOp`(cnmlBaseOp_t *op, `const` cnmlTensor_t inputTensorA, `const` cnmlTensor_t inputTensorB, `const` cnmlTensor_t outputTensor)

A function.

According to the base operator pointer given by the user, create a maximum operator.

A maximum operator is selet the bigger one from the two input tensors at same position and write it at the same position in output tensors.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] inputTensorA`: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float16 or float32.
- `[in] inputTensorB`: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float16 or float32.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [no, ho, wo, co], the shape of output is the same as that of input. The data type of this tensor descriptor must be float16 or float32.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The shape of output tensor is different from that of input tensor.

4.87.2 cnmlComputeMaximumOpForward

cnmlStatus_t `cnmlComputeMaximumOpForward`(cnmlBaseOp_t op, void *inputTensor1, void *inputTensor2, void *outputTensor, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMaximumOpForward_V2 is recommended to use.

Compute the maximum operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. An pointer which points to base operators.
- `[in] inputTensor1[in] inputTensor2`: Input. An MLU address pointing to input data.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.87.3 cnmlComputeMaximumOpForward_V2

cnmlStatus_t `cnmlComputeMaximumOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the maximum operator given by users on the MLU.

After creating a maixmum operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the division operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. An MLU address pointing to input data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.88 Max Equal Operation

4.88.1 cnmlCreateMaxEqualOp

cnmlStatus_t **cnmlCreateMaxEqualOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an maxequal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the maxequal operator, pass them into the function to create the maxequal operator.

Before creating the maxequal operator, declare a pointer pointing to the struct address of operation parameters of the maxequal operator, and pass the pointer and operator parameters required into the function to set operator parameters.

output will be the max value of two input with same index, if two vaule eqaul both if the value of input1 >= input2 then output will be input1 ,else input2.

output[n, c, h, w] = input1[n, c, h, w] >= input2[n, c, h, w] ? input1 : input2

The shapes of two inputs and one output should be exactly the same.

Formula

output[n, c, h, w] = input1[n, c, h, w] >= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A four-dimensional MLU input tensor, the shape of which is [ni, hi, wi, ci], supporting data of float16 type.
- [in] input_tensor_2: Input. A four-dimensional MLU output tensor, the shape of which is [ni, hi, wi, ci], supporting data of float16 type.
- [in] output_tensor: Input. A four-dimensional MLU weight tensor, the shape of which is [no, ho, wo, co] (no = ni, co = ci, ho = hi, wi = wo), supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.88.2 cnmlComputeMaxEqualOpForward_V3

cnmlStatus_t **cnmlComputeMaxEqualOpForward_V3**(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMaxEqualOpForward_V4 is recommended to use.

Compute the maxequal operator specified by users on the MLU.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the maxequal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] >= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.88.3 cnmlComputeMaxEqualOpForward_V4

cnmlStatus_t `cnmlComputeMaxEqualOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the maxequal operator specified by users on the MLU.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the maxequal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] >= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.89 Min Operation

4.89.1 cnmlCreateMinOp

cnmlStatus_t `cnmlCreateMinOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t index_tensor)

A function.

Create a min operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, input and output tensor, and index tensor, pass them into the fucntion to create min operator.

This operator compares all the input data, outputs the minimum value, and gives the index of the minimum value. Assuming that the position of the minimum value is input[n, c, h, w], then index = n * H * W * C + h * W * C + w * C + c.

Formula

min = MIN(a[n c h w])

DataType

MLU270:

The output_data_type and input_data_type must be the same. The supported data types are float16 and float32.

index_data_type:

if input_data_type = float16 then index_data_type = int16 or int32

if input_data_type = float32 then index_data_type = int32

recommeded to be int32, almost has no performance difference with int16.

Scale Limitation

MLU270:

if index_data_type = int16, input_size = N * H * W * C <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [1, 1, 1, 1], supporting data of float16 type.
- `[in] index_tensor`: Input. A four-dimensional MLU index tensor, the shape of which is [1, 1, 1, 1], supporting data of unit32 and float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is neither CNML_TENSOR nor CNML_CONST.

4.89.2 cnmlComputeMinOpForward_V3

cnmlStatus_t `cnmlComputeMinOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, void *index, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMinOpForward_V4 is recommended to use.

Compute the min operator specified by users on the MLU.

After creating a min operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the min operator.

Formula

min = MIN(a[n c h w])

DataType

MLU270:

The output_data_type and input_data_type must be the same. The supported data types are float16 and float32.

index_data_type:

if input_data_type = float16 then index_data_type = int16 or int32

if input_data_type = float32 then index_data_type = int32

recommeded to be int32, almost has no performance difference with int16.

Scale Limitation

MLU270:

if index_data_type = int16, input_size = N * H * W * C <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [out] index: Output. An MLU address pointing to the index position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address which points to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output pointer is null.

4.89.3 cnmlComputeMinOpForward_V4

cnmlStatus_t **cnmlComputeMinOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t index_tensor, void *index, cnrtQueue_t queue, void *extra)

A function.

Compute the min operator specified by users on the MLU.

After creating a min operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the min operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. MLU address pointing to input data.
- [in] output_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] output: Input. MLU address pointing to output data.
- [in] index_tensor: Input. Index MLU tensor pointer. Pass NULL if not used.
- [out] index: Output. An MLU address pointing to index position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.90 Minimum Operation

4.90.1 cnmlCreateMinimumOp

cnmlStatus_t **cnmlCreateMinimumOp**(cnmlBaseOp_t *op, **const** cnmlTensor_t inputTensorA, **const** cnmlTensor_t inputTensorB, **const** cnmlTensor_t outputTensor)

A function.

According to the base operator pointer given by the user, create a minimum operator.

A minimum operator is selet the bigger one from the two input tensors at same position and write it at the same position in output tensors.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] inputTensorA: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float16 or float32.
- [in] inputTensorB: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], The data type of this tensor descriptor must be float16 or float32.
- [in] output_tensor: Input. A four-dimensional MLU output tensor, the shape is [no, ho, wo, co], the shape of output is the same as that of input. The data type of this tensor descriptor must be float16 or float32.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: The shape of output tensor is different from that of input tensor.

4.90.2 cnmlComputeMinimumOpForward

cnmlStatus_t `cnmlComputeMinimumOpForward`(cnmlBaseOp_t op, void *inputTensor1, void *inputTensor2, void *outputTensor, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the minimum operator.

Deprecated. This interface will be deleted in next version and cnmlComputeMinimumOpForward_V2 is recommended to use.

Supports MLU220,MLU270,1M20,and 1M70.

Compute the minimum operator.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. An pointer which points to base operators.
- `[in] inputTensor1``[in] inputTensor2`: Input. An MLU address pointing to input data.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. An pointer which points to base operators.
- `[in] inputTensor1``[in] inputTensor2`: Input. An MLU address pointing to input data.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.90.3 cnmlComputeMinimumOpForward_V2

cnmlStatus_t `cnmlComputeMinimumOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the minimum operator given by users on the MLU.

After creating a minimum operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the division operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. An MLU address pointing to input data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.91 Min Equal Operation

4.91.1 cnmlCreateMinEqualOp

cnmlStatus_t `cnmlCreateMinEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create an minequal operator according to base operator pointers given by users.

After creating a pointer pointing to the base operator address, operation parameters, input and output tensor of the minequal operator, pass them into the function to create the minequal operator.

Before creating the minequal operator, declare a pointer pointing to the struct address of operation parameters of the minequal operator, and pass the pointer and operator parameters required into the function to set operator parameters.

output will be the min value of two input with same index, if two vaule equal will be the equal value if the value of input1 <= input2 then output will be input1 ,else input2.

output[n, c, h, w] = input1[n, c, h, w] <= input2[n, c, h, w] ? input1 : input2

The shapes of two inputs and one output should be exactly the same.

Formula

output[n, c, h, w] = input1[n, c, h, w] <= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, hi, wi, ci], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU output tensor, the shape of which is [ni, hi, wi, ci], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, ho, wo, co] (no = ni, co = ci, ho = hi, wi = wo), supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null
  - The input pointer is null
  - The output tensor is null

4.91.2 cnmlComputeMinEqualOpForward_V3

cnmlStatus_t `cnmlComputeMinEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMinEqualOpForward_V4 is recommended to use.

Compute the minequal operator specified by users on the MLU.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the minequal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] <= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input_1: Input. An MLU address which points to input data.
- [in] input_2: Input. An MLU address which points to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.91.3 cnmlComputeMinEqualOpForward_V4

cnmlStatus_t cnmlComputeMinEqualOpForward_V4(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the minequal operator specified by users on the MLU.

After creating adivision operator, input, output, runtime parameters, and computation queues, pass them into the function to compute the minequal operator.

Formula

output[n, c, h, w] = input1[n, c, h, w] <= input2[n, c, h, w] ? input1 : input2

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.92 Minus Operation

4.92.1 cnmlCreateMinusOp

cnmlStatus_t `cnmlCreateMinusOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user,create a Minus operator.

Then creates a pointer to the base operator address and Minus operator input output tensor, introduce them into the function to create a Minus operator. The operator can realize negation on each dimension, output [n, c, h, w] = 0.0 - input [n, c, h, w].

The shapes of input and output should be exactly the same.

Deprecated.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input tensor type is not CNML_TENSOR.

4.92.2 cnmlComputeMinusOpForward_V3

cnmlStatus_t `cnmlComputeMinusOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the user-specified Minus operator.

After creating Minus operator, input, output and computation stream, introduce them to the function to compute the Minus operator.

Deprecated.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.

4.92.3 cnmlComputeMinusOpForward_V4

cnmlStatus_t `cnmlComputeMinusOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified Minus operator.

After creating Minus operator, input, output and computation queue, introduce them to the function to compute the Minus operator.

Deprecated.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.93 Mlp Operation

4.93.1 cnmlCreateMlpOp

cnmlStatus_t `cnmlCreateMlpOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t filter_tensor, cnmlTensor_t bias_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateMlpOpForward is recommended to use.

According to the basic operator pointer given by the use, this function creates a Multiple-layer perceptron operator.

After the pointer pointing to the base operator, the sparse mode of MLP operator and the input-output tensor are created, they are introduced into the function to create the MLP operator.

Summary

Assuming input, weight, and output are arranged in NCHW order, output[n, coo, 1, 1] = Σinput[n,cii, h, w] * filter[coo, cii, h, w]

the sum of subscripts cii, h and W is carried out, and the summation range is [0, ci - 1],[0, hi - 1],[0, wi - 1], if the bias tensor is not null, the final output should add bias.

DataType

MLU270: input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

bias_type : Data type of the bias tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

bias_oc_type : Data type of the bias tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_type is int8, then input_type can be int8 or int16, and output_type can be float16, float32, or int16.

If filter_type is int16, then input_type can be int16, and output_type can be float16, float32, or int16.

Notes: The data type you set in bias_oc_type, bias_type, and out_oc_type must be the same.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filte_type - filte_oc_type - output_oc__type - output_type

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For best practice, we suggest you call the cnmlComputeReshapeOpForward_V4() API to reshape the tensor, if you set the input tensor with the following:

- The size of ci dimension is less than 64.
- The size of h dimension of the input tensor is greater than 200.
- The size of w dimension is greater than 200.

Also, when you set the output tensor of the cnmlComputeReshapeOpForward_V4() call, the size of h and w dimensions should be less than or equal to 200 and the value of ci dimension should be greater or equal to 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator
- `[in] input_tensor`: Input. A 4-dimensional input tensor with the shape of [ni, ci, hi, wi],supporting data of int8, int16, float16, float32 type.
- `[in] output_tensor`: Input. A 4-dimensional output tensor,the shape is [no, co, ho, wo] (no = ni, co = nf, ho = 1, wo = 1),supporting data of int8, int16, float16, float32 type. In channel quant, output’ s tensor data type should not be int8, int16.
- `[in] filter_tensor`: Input. A 4-dimensional weight tensor with the shape of [nf, cf, hf, wf] (nf = co, cf = ci, hf = hi, wf = wi),supporting data of int8, int16, float16, float32 type.
- `[in] bias_tensor`: Input. A 4-dimensional bias tensor with the shape of [nb, cb, hb, wb] (nb = 1, cb = co, hb = 1, wb = 1),supporting data of float16, float32 type. selected.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.93.2 cnmlComputeMlpOpForward_V3

cnmlStatus_t **`cnmlComputeMlpOpForward_V3`**(cnmlBaseOp_t op, void *inputs, void *outputs, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMlpOpForward_V4 is recommended to use.

Computing the MLP operator specified by the user on MLU.

After the MLP operator, input, output and computational stream are created, they are introduced into the function to compute the MLP operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that specifies the position of the output.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.93.3 cnmlComputeMlpOpForward_V4

cnmlStatus_t **`cnmlComputeMlpOpForward_V4`**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing the MLP operator specified by the user on MLU.

After the MLP operator, input, output and computational stream are created, they are introduced into the function to compute the MLP operator.

Summary

Assuming input, weight, and output are arranged in NCHW order, output[n, coo, 1, 1] = Σinput[n,cii, h, w] * filter[coo, cii, h, w]

the sum of subscripts cii, h and W is carried out, and the summation range is [0, ci - 1],[0, hi - 1],[0, wi - 1], if the bias tensor is not null, the final output should add bias.

DataType

MLU270:

input_type : Data type of the input tensor.

filter_type : Data type of the filter tensor.

bias_type : Data type of the bias tensor.

output_type : Data type of the output tensor.

in_oc_type : Data type of the input tensor used for computing.

filter_oc_type : Data type of the filter tensor used for computing.

bias_oc_type : Data type of the bias tensor used for computing.

output_oc_type : Data type of the output tensor used for computing.

If filter_type is int8, then input_type can be int8 or int16, and output_type can be float16, float32, or int16.

If filter_type is int16, then input_type can be int16, and output_type can be float16, float32, or int16.

Notes: The data type you set in bias_oc_type, bias_type, and out_oc_type must be the same.

The supported combinations of the data type of the tensors are as follows. The data type are shown in the following order:

input_type - input_oc_type - filte_type - filte_oc_type - output_oc__type - output_type

int8-int8-int8-int8-float16-float16;

int8-int8-int8-int8-float16-int8;

int8-int8-int8-int8-float32-float32;

int8-int8-int8-int8-float32-int8;

float16-int8-int8-int8-float16-float16;

float16-int8-int8-int8-float16-int8;

float32-int8-int8-int8-float32-float32;

float32-int8-int8-int8-float32-int8;

int16-int16-int8-int8-float16-float16;

int16-int16-int8-int8-float16-int16;

int16-int16-int8-int8-float32-float32;

int16-int16-int8-int8-float32-int16;

float16-int16-int8-int8-float16-float16;

float16-int16-int8-int8-float16-int16;

float32-int16-int8-int8-float32-float32;

float32-int16-int8-int8-float32-int16;

int16-int16-int16-int16-float16-float16;

int16-int16-int16-int16-float16-int16;

int16-int16-int16-int16-float32-float32;

int16-int16-int16-int16-float32-int16;

float16-int16-int16-int16-float16-float16;

float16-int16-int16-int16-float16-int16;

float32-int16-int16-int16-float32-float32;

float32-int16-int16-int16-float32-int16;

Scale Limitation

MLU270:

kh <= hi，kw <= wi

where `kh` is the height of the kernel, `hi` is the height of the input tensor, `kw` is width of the kernel, and `wi` is the width of the input tensor.

Performance Optimization

For best practice, we suggest you call the cnmlComputeReshapeOpForward_V4() API to reshape the tensor, if you set the input tensor with the following:

- The size of ci dimension is less than 64.
- The size of h dimension of the input tensor is greater than 200.
- The size of w dimension is greater than 200.

Also, when you set the output tensor of the cnmlComputeReshapeOpForward_V4() call, the size of h and w dimensions should be less than or equal to 200 and the value of ci dimension should be greater or equal to 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.

- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.93.4 cnmlEnableMlpOpBinaryMode

cnmlStatus_t **cnmlEnableMlpOpBinaryMode**(cnmlBaseOp_t op)

A function.

According to the MLP operator given by the user, the 8-bit quantization mode is turned on.

After the MLP operator is created, this function is called to set the operation mode to binary mode.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to the base operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.94 Mult Operation

4.94.1 cnmlCreateMultOp

cnmlStatus_t **cnmlCreateMultOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a Mult operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a Mult operator.

The shapes of the two inputs and one output should be exactly the same.

Formula

output[n c h w] = intput1[n c h w] * input2[n c h w]

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.94.2 cnmlComputeMultOpForward_V3

cnmlStatus_t `cnmlComputeMultOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeMultOpForward_V4 is recommended to use.

Compute the Mult operator given by users on the MLU.

After creating a Mult operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Mult operator.

Formula

c[n c h w] = a[n c h w] * b[n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.94.3 cnmlComputeMultOpForward_V4

cnmlStatus_t `cnmlComputeMultOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Mult operator given by users on the MLU.

After creating a Mult operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Mult operator.

Formula

output[n c h w] = intput1[n c h w] * input2[n c h w]

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. First input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. First MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Second input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. Second MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.

- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.95 N-Dimensional Dyadic Operation

4.95.1 cnmlCreateNdDyadicOp

cnmlStatus_t **cnmlCreateNdDyadicOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor, cnmlDyadicType_t d_type)

A function.

Unrestricts the order of input tensors. Expands each dimension of the input respectively to the minimum common multiple of the dimension, and then perform element-wise computing on the input to obtain the output. Currently supports element-wise add, sub, and mult operations.

Transmit two N-dimensional input tensors, one output tensor, and the dyadic type into the function to create a NdDyadic operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A multi-dimensional MLU input_1 tensor.
- [in] input_tensor_2: Input. A multi-dimensional MLU input_2 tensor.
- [in] d_type: Input. The operation type, including CNML_DYADIC_ADD, CNML_DYADIC_SUB, CNML_DYADIC_MULT.
- [in] output_tensor: Input. A multi-dimensional MLU output tensor.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.95.2 cnmlComputeNdDyadicOpForward

cnmlStatus_t **cnmlComputeNdDyadicOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the nddyadic operator given by users on the MLU.

After creating a nddyadic operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the nddyadic operator. The nddyadic operator is determined by cnmlDyadicType_t.

Formula

c[n c h] = a[c h] op b[n c h]

c[n c h] = a[n c h] op b[c h]

c[n c h w] = a[1 c 1 1] op b[n c h w]

c[n c h w] = a[n c h w] op b[1 c 1 1]

c[n c h w d] = a[n c h w d] op b[1 1 1 1 1]

c[n c h w d] = a[1 1 1 1 1] op b[n c h w d]

c[n c h w d] = a[n c h w d] op b[n c h w d]

c[n c h w d] = a[n c 1 w d] op b[n 1 h 1 d]

The op type is determined by cnmlDyadicType_t.

DataType

MLU270/MLU220:

- input: float16, float32
- output: float16, float32

Scale Limitation

MLU270/MLU220:

DataType = float16 : c <= 65536

DataType = float32 : c <= 32768

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. First input MLU tensor pointer. Pass NULL if not used.
- [in] input_1: Input. First MLU address pointing to input1 data.
- [in] input_tensor2: Input. Second input MLU tensor pointer. Pass NULL if not used.
- [in] input_2: Input. Second MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.96 Nearest Neighbor Operation

4.96.1 cnmlCreateNearestNeighborOpParam

cnmlStatus_t **cnmlCreateNearestNeighborOpParam**(cnmlNearestNeighborOpParam_t *param, int output_width, int output_height)

A function.

According to the pointer given by the user, the function creates a struct for the computation parameters of pixel approximation point operator, and sets the operator parameter struct through the width and height of the output image.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer to the address of the struct for the computation parameters of pixel approximation point operator.
- [in] output_width: Input. The width of the output image.
- [in] output_height: Input. The height of the output image.

Return Value

- CNML_STATUS_SUCCESS: Successfully created a clipping operation. Return the corresponding error code when execution is failed.

4.96.2 cnmlCreateNearestNeighborOpParamByRatio

cnmlStatus_t **cnmlCreateNearestNeighborOpParamByRatio**(cnmlNearestNeighborOpParam_t *param, int zoom)

A function.

According to the pointer given by the user, the function creates a struct for the computation parameters of pixel approximation point operator, and sets the operator parameter struct through the zoom multiple.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer to the address of the struct for the computation parameters of pixel approximation point operator.
- [in] zoom: Input. Zoom multiple.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: Param is a null pointer.

4.96.3 cnmlDestroyNearestNeighborOpParam

cnmlStatus_t **cnmlDestroyNearestNeighborOpParam**(cnmlNearestNeighborOpParam_t *param)

A function.

According to the pointer given by the user, free the pointer of the struct for the computation parameters of pixel approximation point operator.

After the convolution operator finishes operation, free the previously created pointer of the struct for the computation parameters of pixel approximation point operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer to the address of the struct for the computation parameters of pixel approximation point operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is not met:
  - Param is a null pointer.
  - The pointer content pointed by param is already freed.

4.96.4 cnmlCreateNearestNeighborOp

cnmlStatus_t `cnmlCreateNearestNeighborOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlNearestNeighborOpParam_t param)

A function.

According to the base operator pointer given by the user, create a pixel approximation point operator.

When zooming the image, select the pixel that is the most approximate to the input image position as the projection output.

First, determine the respective zoom factors of the width and height:

alpha_height = (hi- 1) / (ho- 1);

alpha_width = (wi- 1) / (wo- 1);

let the output pixel be on (w,h), then the source image pixel obtained by approximation projection is on:

src_h = round(h * alpha_height);

src_w = round(w * alpha_width);

copy the value of the coordinate of input(n,c, src_w,src_h) to output (n,c,w,h).

When zooming an image, ni = no, ci = co;

i.e., the input and output must have the same n dimension and the c dimension, but may have different h and w (namely, the size of the image), the computation mode of output width and height can be obtained according to the above-mentioned parameter type either-or principle:

if (_output_h> 0 && _output_w> 0)

ho = _output_h, wo =_output_w; //after ho and wo are specified, there is no need to specify zoomelas if (zoom > 0)

ho = zoom * hi; //ho and wo can be inferred after zoom is specified

wo = zoom * wi;

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supports data of float16 type.
- `[in] output`: Input. A four-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni), supports data of float16 type.
- `[in] param`: Input. A pixel approximation point computation struct pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - The input tensor type is not CNML_TENSOR or CNML_CONST.
  - The CPU tensor bound by the bias tensor is null.

4.96.5 cnmlComputeNearestNeighborOpForward_V3

cnmlStatus_t `cnmlComputeNearestNeighborOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNearestNeighborOpForward_V4 is recommended to use.

For computing the user-specified pixel approximation point operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.96.6 cnmlComputeNearestNeighborOpForward_V4

cnmlStatus_t `cnmlComputeNearestNeighborOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNearestNeighborOpForward_V4 is recommended to use.

For computing the user-specified pixel approximation point operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.96.7 cnmlSetNearestNeighborAlignCorner

cnmlStatus_t `cnmlSetNearestNeighborAlignCorner`(cnmlNearestNeighborOpParam_t *param, bool align_corners)

A function.

According to the pointer given by the user, the function set align_corner for the operator parameter struct.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of the struct for the computation parameters of pixel approximation point operator.
- `[in] align_corner`: Input. align_corner.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: Param is a null pointer.

4.97 Nms Operation

4.97.1 cnmlCreateNmsOpParam

cnmlStatus_t `cnmlCreateNmsOpParam`(cnmlNmsOpParam_t *param, int box_size, int out_size, float nms_thresh, float nms_scale, float score_thresh, bool filter_scores, bool normalized_bbox)

cnmlCreateNmsOpParam.

According to the pointer given by the user, the function creates a struct of computation parameters of Nms, and fills the parameters input by the user to the struct.

Parameters

- `[out] param`: Output. A pointer to the address of the struct of computation parameters for Nms operator.
- `[in] box_size`: Input. An integer tensor, the number of candidate boxes.
- `[in] out_size`: Input. An integer tensor, representing the maximum number of boxes selected through NMS.
- `[in] nms_thresh`: Input. A threshold representing the determination of whether a box has too much overlap with IoU (Intersection over Union, a standard for gauging the degree of accuracy of an object in a specific dataset).
- `[in] nms_scale`: Input. A single score corresponding to each box.
- `[in] score_thresh`: Input. score threshold.
- `[in] filter_scores`: Input. bool value, whether to zoom.
- `[in] normalized_bbox`: Input. A candidate box after normalize.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: -param is a null pointer.

4.97.2 cnmlDestroyNmsOpParam

cnmlStatus_t **cnmlDestroyNmsOpParam**(cnmlNmsOpParam_t *param)

cnmlDestroyNmsOpParam.

The function destroys an Nms computation parameters struct according to the pointer given by the user.

Parameters

- [in] param: Input. A pointer to the address of the struct of computation parameters for Nms operator.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met: -param is a null pointer.

4.97.3 cnmlCreateNmsOp

cnmlStatus_t **cnmlCreateNmsOp**(cnmlBaseOp_t *op, cnmlNmsOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateNmsOp.

According to the base operator pointer given by the user, create an Nms operator.

After creating a pointer to the base operator address, Nms operator computation parameters and input output tensor, introduce them into the function to create an Nms operator.

Before creating the Nms operator, declare a pointer to the address of the struct of computation parameters for the Nms operator as well as the required operator parameters to the function to set the operator parameters.

The C dimension of output should be exactly divisible by 16;

N, H, W dimensions of input must be 1, 1, 5.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] param: Input. An Nms computation struct pointer.
- [in] input_tensor: Input. A four-dimensional MLU input tensor, the shape is [1, box_size, 1, 5], supports data of float16 type.
- [in] output_tensor: Input. A four-dimensional MLU output tensor, the shape is [1, output_size, 1, 5], supports data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is not met:
  - The input tensor type is not CNML_TENSOR or CNML_CONST.
  - The CPU tensor bound by the bias tensor is null.

4.97.4 cnmlComputeNmsOpForward_V3

cnmlStatus_t **cnmlComputeNmsOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNmsOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeNmsOpForward_V4 is recommended to use.

For computing the user-specified Nms operator on the MLU.

After creating Nms operator, related parameters and computation stream, introduce them into the function to compute the Nms operator.

Parameters

- [out] output: Output. An MLU address that points to the output position.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.
- [in] compute_forw_param: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - The runtime task type is invalid

4.97.5 cnmlComputeNmsOpForward_V4

cnmlStatus_t `cnmlComputeNmsOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNmsOpForward_V4.

For computing the user-specified Nms operator on the MLU.

After creating Nms operator, related parameters and computation stream, introduce them into the function to compute the Nms operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.98 Normalize Operation

4.98.1 cnmlCreateNormalizeOpParam

cnmlStatus_t `cnmlCreateNormalizeOpParam`(cnmlNormalizeOpParam_t *param, int p, int use_scale, int mode, float weight)

A function.

According to the pointer given by the user, the function creates a normalized operator operation parameter struct, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of the normalized operator operation parameter.
- `[in] p`: Input. When the p is 0, L2 norm is adopted, otherwise, L1 norm is adopted.
- `[in] use_scale`: Input. When the use_scale is 1, scale_tensor is used to control the value range of output; otherwise, the weight in the parameter is used to control the value range of output.
- `[in] mode`: Input. The range of value is 0-5, mode=(0, 1, 2, 3, 4, 5) perform normalization on (C, HW, HWC, N, H, W) respectively.
- `[in] weight`: Input. If use_scale == 0, when normalization is complete, multiply the result by weight.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.98.2 cnmlCreateNormalizeOpParamV2

cnmlStatus_t `cnmlCreateNormalizeOpParamV2`(cnmlNormalizeOpParam_t *param, int p, int use_scale, int mode, float weight, float eps)

A function.

According to the pointer given by the user, the function creates a normalized operator operation parameter struct, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of the normalized operator operation parameter.
- `[in] p`: Input. When the input is 1, L1 norm is adopted, otherwise L2 norm is adopted.
- `[in] use_scale`: Input. When the input is 1, scale_tensor is used to control the value range of output; otherwise, the weight in the parameter is used to control the value range of output.
- `[in] mode`: Input. The range of value is 0-5, mode=(0, 1, 2, 3, 4, 5) perform normalization on (C, HW, HWC, N, H, W) respectively.
- `[in] weight`: Input. If use_scale == 0, when normalization is complete, multiply the result by weight.*
- `[in] eps`: Input. Add eps param at denominator to avoid denominator value is zero .

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.98.3 cnmlDestroyNormalizeOpParam

cnmlStatus_t `cnmlDestroyNormalizeOpParam`(cnmlNormalizeOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of normalized operator operation parameter is freed.

At the end of normalized operator operation, the created struct pointer of normalized operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of normalized operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.98.4 cnmlCreateNormalizeOp

cnmlStatus_t `cnmlCreateNormalizeOp`(cnmlBaseOp_t *op, cnmlNormalizeOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t scale_tensor, bool is_fix8_mode)

A function.

Creates a Normalization operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, a pointer to normalize operation param, input, scale(optional) and output tensors, pass them to the function to create a Normalization operator.

The normalization operation performs L1 or L2 normalization on varied dimensions of input tensor. Additionally, a zooming factor can be used to control the range of normalized data values.

Related APIs APIs Before: createCnmlNormalizeOpParamV2(or createCnmlNormalizeOpParam), cnmlCreateTensor APIs After: cnmlDestroyBaseOp, cnmlFuseOp(optional), cnmlComputeNormalizeOp(optional)

Lifetime The op can be released after calling cnmlFuseOp or cnmlComputeNormalizeOp. The param, input_tensor, scale_tensor, and output_tensor can be released intermediatelly after this function.

Formula L1 normalization: for all n in [0, N), c in [O, C), h in [0, H), w in [0, W): out[n, c, h, w] = in[n, c, h, w] / (Sum(Abs(in[n, c, h, w]), specified_dims) + eps) if use_scale == 0: out[n, c, h, w] = weight * out[n, c, h, w] else: out[n, c, h, w] = scale[n, c, h, w] * out[n, c, h, w] L2 normalization: for all n in [0, N), c in [O, C), h in [0, H), w in [0, W): out[n, c, h, w] = in[n, c, h, w] / Sqrt((Sum(Square(in[n, c, h, w]), specified_dims) + eps) if use_scale == 0: out[n, c, h, w] = weight * out[n, c, h, w] else: out[n, c, h, w] = scale[n, c, h, w] * out[n, c, h, w]

The `specified_dims` are dimensions to calculate the L1 or L2 norm, which is specified by the variable mode in param. The eps is a value specified in param. `in[]` specifies the input_tensor, `scale[]` specifies the scale_tensor and `out[]` specifies the output_tensor.

DataType

MLU270:

- input: float16, float32
- output: float16, float32

Scale Limitation

MLU270:

Unlimited.

Performance Optimization

The normalization across Channel(mode = c) or Instance(mode = hwc) is faster than other modes, since other modes require inner data transpose.

The number of bytes of elements within specified_dims is a multiple of 128.

Deep Fusion

Not involved

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A struct pointer of normalized operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi],supporting data of float16 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni).
- `[in] scale_tensor`: Input. Optional. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi]. Required when the variable use_scale of param is not equal to 0.
- `[in] is_fix8_mode`: Input. A bool type to set fix8 quant mode.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.

- The CPU tensor bound by bias tensor is null.

4.98.5 cnmlComputeNormalizeOpForward_V3

cnmlStatus_t `cnmlComputeNormalizeOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, void *scale, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNormalizeOpForward_V4 is recommended to use.

Computing user-specified normalized operators on MLU.

After the normalized operator, input, output, parameter at runtime and computational queue are created, they are introduced into the function to compute normalized operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.98.6 cnmlComputeNormalizeOpForward_V4

cnmlStatus_t `cnmlComputeNormalizeOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t scale_tensor, void *scale, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified normalized operators on MLU.

After the normalized operator, input, output, parameter at runtime and computational queue are created, they are introduced into the function to compute normalized operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] scale_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] scale`: Input. An MLU address pointing to input data.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.99 Normalize Lite Operation

4.99.1 cnmlCreateNormalizeLiteOpParam

cnmlStatus_t **cnmlCreateNormalizeLiteOpParam**(cnmlNormalizeLiteOpParam_t *param, int axis, float epsilon)

A function. According to the pointer given by the user, the function createss a sample normalized operator operation parameter struct, and fills in the struct with the parameters input by the user.

Description

Description: normalize lite operation use L2-norm

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer pointing to the address of struct of the L2-Normalized operator parameter.
- `[in] axis`: Input. The range of value is [-4 ,3] , axis={0,1,2,3} or axis={-4,-3,-2,-1} perform normalization on (N,C,H,W) respectively.
- `[in] epsilon`: Input. Add epsilon param at denonminator to avoid denominator value is zero.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.

4.99.2 cnmlDestroyNormalizeLiteOpParam

cnmlStatus_t **cnmlDestroyNormalizeLiteOpParam**(cnmlNormalizeLiteOpParam_t *param)

A function.

Description

According to the pointer given by the user, the struct pointer of normalized operator operation parameter is freed.

At the end of normalized operator operation, the created struct pointer of normalized operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of normalized operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.99.3 cnmlCreateNormalizeLiteOp

cnmlStatus_t **cnmlCreateNormalizeLiteOp**(cnmlBaseOp_t *op, cnmlNormalizeLiteOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t norm_tensor, cnmlTensor_t output_tensor)

A function.

Description

According to the basic Operator pointer given by the user, a normalized operator is created. Typically the normalization is performed by calculating the mean and standard deviation of a subgroup in your input tensor. The normalization operation is performed on the input N, C, H, W,direction (Only L2 norm can be used). The axis can be used to control the range of normalized data values.

Reference: This function is based on the https://arxiv.org/pdf/1803.08494/pdf

- **Formula**

L2 noramalization:

such as `specified_dims` is set to 3, Tensor Layout is NHWC

for all n in [0,N), h in [0,H), w in [0,W);

sum = param.epsilon;

for c in [0,C);

sum =+ Square(in[n,c,h,w]);

norm[n,1,w,h] = Sqrt(sum);

for c in [0,C];

out[n,c,h,w] = in[n,c,h,w]/norm[n,1,w,h];

The `spacified_dim`` are dimensions to calculate the L2 norm, which is specified by the

variable model param: axis, it can be a negative number;

`in[]` specifies the input_tensor;

`norm[]` spacifies the norm_tensor;

`out[]` spacifies the output_tensor;

DataType

MLU270:

-input: float16, float32

-output:float16, float32

Scale Limitation

MLU270:

Unlimited in FP32.

limited in FP16: Due to the range of fp16 numbers, output may be some deviation , when norm is less than 1E-3.

Support only MLU270.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A struct pointer of normalized operation param.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, hi, wi].
- `[out] norm_tensor`: Output. A 4-dimensional MLU output tensor, if param.axis=0 or -4 , the shape of norm tensor is [1, ci, hi, wi]. if param.axis=1 or -3 , the shape of norm tnesor is [ni, 1, hi, wi]. if param.axis=2 or -2 , the shape of norm tnesor is [ni, ci, 1, wi]. if param.axis=3 or -1 , the shape of norm tnesor is [ni, ci, hi, 1].
- `[out] output_tensor`: Output. A 4-dimensional MLU output tensor, the shape is [ni, ci, hi, wi].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bound by bias tensor is null.

4.99.4 cnmlComputeNormalizeLiteOpForward

cnmlStatus_t **cnmlComputeNormalizeLiteOpForward**(cnmlBaseOp_t op, cnmlTensor_t *input_tensor, void *input, cnmlTensor_t *norm_tensor, void *norm, cnmlTensor_t *output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function. Computing user-specified normalized operators on MLU.

Description After the normalized operator, input, output and computational stream are created, they are introduced into the function to compute normalized operator.

Supports MLU220,MLU270,1M20,and 1M70.

DataType MLU270: -input:float16, float32 -output:float16,float32

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[out] norm`: Output. An MLU address pointing to the denominator of Normalization.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The task type is invalid at runtime.

4.100 Not Operation

4.100.1 cnmlCreateNotOp

cnmlStatus_t **cnmlCreateNotOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Perform element-wise not operation on input tensor.

The shapes of input and output should be exactly the same.

Formula

output[n c h w] = ! input[n c h w]

DataType

MLU270:

float16, float32, bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the operator.
- `[in] input_Tensor`: Input. Input tensor.
- `[in] output_tensor`: Input. Output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.100.2 cnmlComputeNotOpForward_V3

cnmlStatus_t `cnmlComputeNotOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNotOpForward_V4 is recommended to use.

Compute the not operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to the operator.
- `[in] input`: Input. Pointing to a pointer of input data.
- `[in] output`: Input. A pointer pointing to output data.
- `[in] compute_forw_param`: Input. A pointer pointing to a struct address of runtime parameters.
- `[in] stream`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.100.3 cnmlComputeNotOpForward_V4

cnmlStatus_t `cnmlComputeNotOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the not operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.101 Not Equal Operation

4.101.1 cnmlCreateNEqualOp

cnmlStatus_t `cnmlCreateNEqualOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an operator, and perform a comparison on the tensor B and the tensor A in each position to find whether one is not equal to the other.

I.e., C[ni][hi][wi][ci] = (A[ni][hi][wi][ci] != B[ni][hi][wi][ci])? 1: 0.

The shapes of two inputs and output should be exactly the same.

Formula

c[n c h w] = a[n c h w] != b[n c h w] ? 1 : 0

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] input_tensor_2`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`:
  - Reason1 The operator pointer is null.
  - Reason2 The input and output pointer is null.

4.101.2 cnmlComputeNEqualOpForward_V3

cnmlStatus_t `cnmlComputeNEqualOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNEqualOpForward_V4 is recommended to use.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is not equal to the other.

Formula

c[n c h w] = a[n c h w] != b[n c h w] ? 1 : 0

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.101.3 cnmlComputeNEqualOpForward_V4

cnmlStatus_t **cnmlComputeNEqualOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Perform a comparison on the user-specified four-dimensional tensor A and B to find whether one is not equal to the other.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.102 Or Operation

4.102.1 cnmlCreateOrOp

cnmlStatus_t **cnmlCreateOrOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Perform element-wise Or operation on two Tensors.

The shapes of two input tensor should be the same;

Formula

c[n c h w] = (a[n c h w] !=0 || (b[n c h w] != 0)

DataType

MLU270:

float16, float32, bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supports data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.102.2 cnmlComputeOrOpForward_V3

cnmlStatus_t `cnmlComputeOrOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeOrOpForward_V4 is recommended to use.

Compute the Or operator specified by users on the MLU.

After creating an Or operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Or operator.

Formula

c[n c h w] = (a[n c h w] !=0 || (b[n c h w] != 0)

DataType

MLU270:

float16, float32, bool

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.102.3 cnmlComputeOrOpForward_V4

cnmlStatus_t `cnmlComputeOrOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Or operator specified by users on the MLU.

After creating an Or operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Or operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.103 Plugin Operation

4.103.1 cnmlCreatePluginOp

cnmlStatus_t `cnmlCreatePluginOp`(cnmlBaseOp_t *op, `const` char *name, void *kernel, cnrtKernelParamsBuffer_t params, cnmlTensor_t *input_tensors, int input_num, cnmlTensor_t *output_tensors, int output_num, cnmlTensor_t *statics_tensor, int static_num)

cnmlCreatePluginOp.

Create a Plugin operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, the operator name, the kernel function pointer, the kernel parameter pointer, and the input and output tensors, pass them to the function to create a Plugin operator.

PluginOp is firstly a CNML Op, and Plugin means that the specific behavior of this instance of Op is determined by the Kernel of its Plugin compiler. The core operation is to provide the kernel of the compiler with the same environment provided by cnrtInvokeKernel, so that the compiler’s operators can be run using CNML.

The size of the Plugin operator is related to the specific kernel being written.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] name`: Input. operator name for distinguishing several Plugin Ops.
- `[in] kernel`: Input. kernel function pointer.
- `[in] params`: Input. parameter pointer needed in kernel
- `[in] inputs_tensor`: Input. int output_num.
- `[in] input_num`: Input. number of input tensors.
- `[in] outputs_tensor`: Input. Array of output tensors.
- `[in] output_num`: Input. number of output tensors.
- `[in] statics_tensor`: Input. Array of constant tensors.
- `[in] static_num`: Input. number of constant tensors.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.103.2 cnmlComputePluginOpForward_V3

cnmlStatus_t `cnmlComputePluginOpForward_V3`(cnmlBaseOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputePluginOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputePluginOpForward_V4 is recommended to use.

It is used to compute the user-specified Plugin operator on the MLU.

After creating the Plugin operator, Input, Output, and computation queue, pass them to the function to It is used to compute the Plugin operator.

Parameters

- `[out] outputs`: Output. An array of MLU addresses pointing to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] inputs`: Input. An array of MLU addresses pointing to the input data.
- `[in] input_num`: Input. number of the input tensors.
- `[in] output_num`: Input. number of the output tensors.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The runtime task type is invalid

4.103.3 cnmlComputePluginOpForward_V4

cnmlStatus_t `cnmlComputePluginOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

cnmlComputePluginOpForward_V4.

It is used to compute the user-specified Plugin operator on the MLU.

After creating the Plugin operator, Input, Output, and computation queue, pass them to the function to It is used to compute the Plugin operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[in] inputs`: Input. A pointer array, each element pointing to the MLU address of input data.
- `[in] intput_num`: Input. The number of elements in the inputs array.
- `[in] output_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[out] outputs`: Output. A pointer array, each element pointing to the MLU address of output position.
- `[in] output_num`: Input. The number of elements in the outputs array.
- `[in] queue`: Input. A computation stream pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.103.4 cnmlPluginOpParamsBufferMarkTensorDimension

cnmlStatus_t `cnmlPluginOpParamsBufferMarkTensorDimension`(cnrtKernelParamsBuffer_t params, cnmlTensor_t *mark_tensors, cnmlDimension_t *dim, int mark_num)

cnmlPluginOpParamsBufferMarkTensorDimension.

In some situations, one pluginOp’ s param relates to its Input/Outputs’ dimensions and can not be determined before compilation. Use this API to mark that dimension and param, a particular value will be filled back during compilation.

Parameters

- `[in] params`: Input. parameter pointer needed in kernel
- `[in] mark_tensors`: Input. Array of tensors.
- `[in] dim`: Input. Array of dims.
- `[in] mark_num`: Input. number of tensors and dims.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: -The input tensor is nullptr. -The param pointer is nullptr.

4.104 Pool Operation

4.104.1 cnmlCreatePoolOpParam

cnmlStatus_t `cnmlCreatePoolOpParam`(cnmlPoolOpParam_t *param, int window_height, int window_width, int stride_height, int stride_width, int pad_height, int pad_width, int dilation_height, int dilation_width, cnmlPoolMode_t pool_mode, cnmlPoolStrategyMode_t strategy_mode, bool real)

A function.

This function creates a pooling param object by allocating the memory needed to hold its opaque structure.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww].

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. The returning param descriptor.
- `[in] window_height`: Input. Height of the pooling window.
- `[in] window_width`: Input. Width of the pooling window.
- `[in] stride_height`: Input. Pooling vertical stride.
- `[in] stride_width`: Input. Pooling horizontal stride.
- `[in] pad_height`: Input. Size of vertical padding.
- `[in] pad_width`: Input. Size of horizontal padding.
- `[in] dilation_height`: Input. Size of vertical dilation.
- `[in] dilation_width`: Input. Size of horizontal dilation.
- `[in] pool_mode`: Input. Enumerant to specify the pooling mode.
- `[in] strategy_mode`: Input. Enumerant to specify the pooling strategy mode.
- `[in] real`: Input. Boolean specifies if the paddings are used to compute in the N-dimensional pooling computing in CNML_POOL_AVG mode. If you set this parameter to false, the padding is commputed in the N-dimensional pooling computation. If you set this parameter to true, the padding will not be computed in the N-dimensional pooling computation.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param pointer is null.

4.104.2 cnmlCreatePoolOpParam_V2

cnmlStatus_t `cnmlCreatePoolOpParam_V2`(cnmlPoolOpParam_t *param, int window_height, int window_width, int stride_height, int stride_width, int pad_height, int pad_width, int dilation_height, int dilation_width, cnmlPoolMode_t pool_mode, cnmlPoolStrategyMode_t strategy_mode, bool real, float blend_factor)

A function.

This function creates a pooling param object by allocating the memory needed to hold its opaque structure.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww]

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. The returning param descriptor.
- [in] window_height: Input. Height of the pooling window.
- [in] window_width: Input. Width of the pooling window.
- [in] stride_height: Input. Pooling vertical stride.
- [in] stride_width: Input. Pooling horizontal stride.
- [in] pad_height: Input. Size of vertical padding.
- [in] pad_width: Input. Size of horizontal padding.
- [in] dilation_height: Input. Size of vertical dilation.
- [in] dilation_width: Input. Size of horizontal dilation.
- [in] pool_mode: Input. Enumerant to specify the pooling mode.
- [in] strategy_mode: Input. Enumerant to specify the pooling strategy mode.
- [in] real: Input. Boolean specifies if the paddings are used to compute in the N-dimensional pooling computing in CNML_POOL_AVG mode. If you set this parameter to false, the padding is commputed in the N-dimensional pooling computation. If you set this parameter to true, the padding will not be computed in the N-dimensional pooling computation.
- [in] blend_factor: Input. BlendFactor used in blend pool mode.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param pointer is null.

4.104.3 cnmlCreatePoolOpParam_V3

cnmlStatus_t cnmlCreatePoolOpParam_V3(cnmlPoolOpParam_t *param, int window_height, int window_width, int stride_height, int stride_width, int pad_top, int pad_bottom, int pad_left, int pad_right, int dilation_height, int dilation_width, cnmlPoolMode_t pool_mode, cnmlPoolStrategyMode_t strategy_mode, bool real, float blend_factor)

A function.

This function creates a pooling param object by allocating the memory needed to hold its opaque structure.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww]

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. The returning param descriptor.
- [in] window_height: Input. Height of the pooling window.
- [in] window_width: Input. Width of the pooling window.
- [in] stride_height: Input. Pooling vertical stride.
- [in] stride_width: Input. Pooling horizontal stride.
- [in] pad_top: Input. Size of vertical top padding.
- [in] pad_bottom: Input. Size of vertical bottom padding.
- [in] pad_left: Input. Size of horizontal left padding.
- [in] pad_bottom: Input. Size of horizontal right padding.
- [in] dilation_height: Input. Size of vertical dilation.
- [in] dilation_width: Input. Size of horizontal dilation.
- [in] pool_mode: Input. Enumerant to specify the pooling mode.
- [in] strategy_mode: Input. Enumerant to specify the pooling strategy mode.

- `[in] real`: Input. Boolean specifies if the paddings are used to compute in the N-dimensional pooling computing in CNML_POOL_AVG mode. If you set this parameter to false, the padding is commputed in the N-dimensional pooling computation. If you set this parameter to true, the padding will not be computed in the N-dimensional pooling computation.
- `[in] blend_factor`: Input. BlendFactor used in blend pool mode.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - param pointer is null.

4.104.4 cnmlCreateNdPoolOpParam

cnmlStatus_t `cnmlCreateNdPoolOpParam`(cnmlNdPoolOpParam_t *param, cnmlPoolMode_t pool_mode, cnmlPoolStrategyMode_t strategy_mode, bool real, int array_length, int kernel_size[], int dilations[], int strides[], int paddings[][2])

A function.

Creates a pooling ND param object by allocating the memory used for holding the opaque structure of the pooling ND op.

Formula

maxpool:

output[n, c, d, h, w] = max(input[n, c, d * sd + dd, h * sh + dh, w * sw + dw]), {0 <= dd <= wd, 0 <= dh < wh, 0 <= dw < ww}

avgpool:

output[n, c, d, h, w] = sum(input[n, c, d* sd + dd, h * sh + dh, w * sw + dw]) / (wd * wh * ww), {0 <= dd <= wd, 0 <= di < wh, 0 <= dj < ww}

OutputDim:

if strategy_mode is KVALID:

outputDim = 1 + (inputDim + 2 * paddings - kernel_size) / strides

else if strategy_mode if KFULL:

if poolingStride > poolingKernel:

outputDim = (inputDim - 1) / strides + 1

else:

outputDim = (inputDim + 2 * paddings - 1 + strides - kernel_size) / strides + 1

Datatype

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Only supports MLU270.

Parameters

- `[out] param`: Output. A pointer to a param descriptor.
- `[in] pool_mode`: Input. An enumerated type specifies the mode of how the pool is used to compute in the pooling ND operation. When used as pool 3d, supported values are: CNML_POOL_AVG and CNML_POOL_MAX, used as pool 2d, support value same as pool 2d. You need to use the cnmlPoolMode_t datatype to declare the pool_mode value first. For more information about the supported values, see the cnmlPoolMode_t datatype.
- `[in] strategy_mode`: Input. An enumerated type specifies whether the edge is used to compute in the pooling ND operation if the edge is shorter than a stride length. Supported values are: CNML_POOL_KFULL and CNML_POOL_KVALID. You need to use the nmlActiveFunction_t datatype to declare the strategy_mode value first. For more information about the supported values, see the cnmlPoolStrategyMode_t datatype.
- `[in] real`: Input. Boolean specifies if the paddings are used to compute in the N-dimensional pooling computing in CNML_POOL_AVG mode. If you set this parameter to false, the padding is commputed in the N-dimensional pooling computation. If you set this parameter to true, the padding will not be computed in the N-dimensional pooling computation.
- `[in] array_length`: Input. An integer specifies the length of the data arrays for kenel_size, dilations, strides, and paddings. You need to set the array length in the order of kenel_size, dilations, strides, and paddings.
- `[in] kernel_size`: Input. An integer array specifies all kernel sizes of the pooling ND operation. When using this function for pool3d, the values need to be set in the d, h, w order, and for pool2d, order should be h, w.
- `[in] dilations`: Input. An integer array specifies all dilation of the pooling ND operation. The array values must be greater than 0. This parameter is used to control the size of the kernel extension. The kernel is extended to the size based on the forma: (kernel_size â€ “1)*dilations + 1. When using this function for pool3d, the values need to be set in the d, h, w order, and for pool2d, order should be h, w.
- `[in] strides`: Input. An integer array specifies all the strides of the pooling ND operation. When using this function for pool3d, the values need to be set in d, h, w order, and for pool2d, order should be h, w.
- `[in] paddings`: Input. An integer array specifies all padding of the pooling ND operation. When using this function for pool3d, the values need to be set in the d, h, w and front, behind, top, bottom, left, right order, and for pool2d, order should be h, w and top, bottom, left, right.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.

- `CNML_STATUS_INVALIDPARAM`: The param pointer is NULL.

4.104.5 cnmlDestroyPoolOpParam

cnmlStatus_t `cnmlDestroyPoolOpParam`(cnmlPoolOpParam_t *param)

A function.

This function destroys a previously created pooling param descriptor object

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww]

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. Pointer to the structure holding the description of the pooling param to be deleted.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.104.6 cnmlDestroyNdPoolOpParam

cnmlStatus_t `cnmlDestroyNdPoolOpParam`(cnmlNdPoolOpParam_t *param)

A function.

Destroys a previously created pooling ND param, ndpool support pool3d and pool2d now.

Only supports MLU270.

Parameters

- `[in] param`: Input. Pointer to the pooling ND param you want to destroy.

Return Value

- `CNML_STATUS_SUCCESS`: The descriptor destroyed successfully.

4.104.7 cnmlCreatePoolOp

cnmlStatus_t `cnmlCreatePoolOp`(cnmlBaseOp_t *op, cnmlPoolOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreatePoolOpForward is recommended to use.

This function creates a pooling op object by allocating the memory needed to hold its opaque structure.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww].

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. The returning op descriptor.
- `[in] param`: Input. Param of this pooling op.
- `[in] input`: Input. Input cnml tensor of this pooling op.
- `[in] output`: Input. Input cnml tensor of this pooling op.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.104.8 cnmlCreateNdPoolOp

cnmlStatus_t `cnmlCreateNdPoolOp`(cnmlBaseOp_t *op, cnmlNdPoolOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdPoolOpForward_V2 is recommended to use.

Creates a pooling ND op object by allocating the memory needed to hold the opaque structure of the op.

Formula

maxpool:

output[n, c, d, h, w] = max(input[n, c, d * sd + dd, h * sh + dh, w * sw + dw]), {0 <= dd <= wd, 0 <= dh < wh, 0 <= dw < ww}

avgpool:

output[n, c, d, h, w] = sum(input[n, c, d* sd + dd, h * sh + dh, w * sw + dw]) / (wd * wh * ww), {0 <= dd <= wd, 0 <= di < wh, 0 <= dj < ww}

OutputDim:

if strategy_mode is KVALID:

outputDim = 1 + (inputDim + 2 * paddings - kernel_size) / strides

else if strategy_mode if KFULL:

if poolingStride > poolingKernel:

outputDim = (inputDim - 1) / strides + 1

else:

outputDim = (inputDim + 2 * paddings - 1 + strides - kernel_size) / strides + 1

Datatype

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Only supports MLU270.

Parameters

- `[out] op`: Output. A pointer to the pooling ND operator you created.
- `[in] param`: Input. A pointer to the params of this pooling ND op.
- `[in] input`: Input. A pointer to tensor descriptor to do pooling ND op.

- `[in] output`: Input. The descriptor of the output tensor.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions occurred:
  - The operator pointer is NULL.
  - The output pointer is NULL.

4.104.9 cnmlCreatePoolOpForward

This API has the same function as cnmlCreatePoolOp. For detailed information, see cnmlCreatePoolOp.

4.104.10 cnmlCreateNdPoolOpForward

This API has the same function as cnmlCreateNdPoolOp. For detailed information, see cnmlCreateNdPoolOp.

4.104.11 cnmlComputePoolOpForward_V3

cnmlStatus_t `cnmlComputePoolOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputePoolOpForward_V4 is recommended to use.

Computing pool operator run on MLU.

After the pool operator, input, output, and computational stream are created, they are introduced into the function for operation.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], window[wh, ww]

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < wh, 0 <= dj < ww}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (wh * ww), {0 <= di < wh, 0 <= dj < ww}

Datatype

MLU270:

maxpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] type`: Input. An enumeration value specifying how to compute on MLU.
- `[in] stream`: Input. A computational stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.104.12 cnmlComputePoolOpForward_V4

cnmlStatus_t `cnmlComputePoolOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing pool operator run on MLU.

After the pool operator, input, output, and computational stream are created, they are introduced into the function for operation.

Formula

For input[ni, ci, hi, wi], output[no. co. ho. wo], stride[sh, sw], kernel[kh, kw]

maxpool:

output[n, c, i, j] = max(input[n, c, i * sh + di, j * sw + dj]), {0 <= di < kh, 0 <= dj < kw}

avgpool:

output[n, c, i, j] = sum(input[n, c, i * sh + di, j * sw + dj]) / (kh * kw), {0 <= di < kh, 0 <= dj < kw}

Datatype

MLU270:

maxpool:

input_typetype: int8, int16, float16, float32

compute_type: float16, float32

output_type: int8, int16, float16, float32

compute_type.dataSize >= input_type.dataSize

compute_type.dataSize >= output_type.dataSize

avgpool:

input: int8, int16, float16, float32

compute: float16, float32

output: int8, int16, float16, float32

compute_type.dataSize >= input_type.dataSize

compute_type.dataSize >= output_type.dataSize

Scale Limitation

MLU270:

unlimited

Performance Optimization

maxpool and avgpool :

The number of bytes in the C dimension is a multiple of 128.

avgpool:

When kh * kw <= 36, the computing speed is higher with the loss of precision up to 1%

When kh * kw > 36, the precision is better with lower computing speed.

For best practice and higher performance, the size of the data in C dimension should be less than 131,072B, and the data in C dimension should be a multiple of 128. Otherwise the performance is getting worse.

Where `kh` is the height of the kernel tensor and `kw` is the width of the kernel tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.104.13 cnmlComputeNdPoolOpForward

cnmlStatus_t `cnmlComputeNdPoolOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdPoolOpForward_V2 is recommended to use.

Computes the pool operation on MLU.

Formula

maxpool:

output[n, c, d, h, w] = max(input[n, c, d * sd + dd, h * sh + dh, w * sw + dw]), {0 <= dd <= wd, 0 <= dh < wh, 0 <= dw < ww}

avgpool:

output[n, c, d, h, w] = sum(input[n, c, d* sd + dd, h * sh + dh, w * sw + dw]) / (wd * wh * ww), {0 <= dd <= wd, 0 <= di < wh, 0 <= dj < ww}

OutputDim:

if strategy_mode is KVALID:

outputDim = 1 + (inputDim + 2 * paddings - kernel_size) / strides

else if strategy_mode if KFULL:

if poolingStride > poolingKernel:

outputDim = (inputDim - 1) / strides + 1

else:

outputDim = (inputDim + 2 * paddings - 1 + strides - kernel_size) / strides + 1

Datatype

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Only supports MLU270.

Parameters

- `[out] output`: Output. A pointer to the output data after the pooling ND operator is applied.
- `[in] op`: Input. A pointer to the pooling ND operator you have created.
- `[in] input`: Input. A pointer to the input data you want to compute.
- `[in] compute_forw_param`: Input. A pointer to the struct address that records the data parallelism.
- `[in] queue`: Input. A pointer to the queue that is used to implement the computation.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions occurred:
  - The operator pointer is NULL.
  - The output pointer is NULL.

4.104.14 cnmlComputeNdPoolOpForward_V2

cnmlStatus_t `cnmlComputeNdPoolOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing pool operator run on MLU.

After the pool operator, input, output, and computational queue are created, they are introduced into the function for operation.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.105 Power Operation

4.105.1 cnmlCreatePowerOp

cnmlStatus_t `cnmlCreatePowerOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, float power_c)

A function.

According to the base operator pointer given by the user, create a power operator.

Then creates a pointer to the base operator address, input output tensor, and introduce them into the function to create a function operator.

Input and output should have the same shape.

The maximum value of power_c: it must be ensured that output is the representable range of float16, which is [-65504,65504], e.g., when input is 3, the maximum value of power_c is 10.09.

The minimum value of inputs: because 2^-24 is the representable minimum value of float16, the minimum of value of input should be greater than pow(2^-24, 1/power_c). If the input is lower than that, the output will be approximated to 0.

The value of inputs can be negative if the power_c is a integer. If the power_c is not a integer, for example 2.5, the value of input should be positive.

Formula

output[n c h w] = in[n c h w] ^ power_c;

Datatype

MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

-65504 < power_c < 65504

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.
- `[in] power_c`: Input. A float type data, the maximum value of power_c must ensure that output is the representable range of float16, which is [-65504,65504].

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - Reason1 The operator pointer is null.
  - Reason2 The input pointer is null.
  - Reason3 The output pointer is null.

4.105.2 cnmlSetPowerHighPrecision

cnmlStatus_t `cnmlSetPowerHighPrecision`(cnmlBaseOp_t *op, bool high_precision_flag)

A function.

According to the power operator given by the user, enable high precision mode.

If the high precision mode is set, the precision will be improved when input interval is [0,1].

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer to the base operator address.
- `[in] high_precision_flag`: Input. The flag for setting high precision mode. True flag set the high precision mode.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: Op is a null pointer.

4.105.3 cnmlComputePowerOpForward_V3

cnmlStatus_t `cnmlComputePowerOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputePowerOpForward_V4 is recommended to use.

For computing the user-specified power operator on the MLU.

After creating power operator, input, output and computation stream, introduce them to the function to compute the power operator.

Formula

output[n c h w] = in[n c h w] ^ power_c;

Datatype

MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

-65504 < power_c < 65504

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.105.4 cnmlComputePowerOpForward_V4

cnmlStatus_t `cnmlComputePowerOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified power operator on the MLU.

After creating power operator, input, output and computation stream, introduce them to the function to compute the power operator.

Formula

output[n c h w] = in[n c h w] ^ power_c;

Datatype

MLU270:

input_dt = output_dt

float16, float32

Scale Limitation

-65504 < power_c < 65504

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.106 PreLU Operation

4.106.1 cnmlCreatePreluOp

cnmlStatus_t **cnmlCreatePreluOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t prelu_param)

A function.

According to the base operator pointer given by the user,create a Prelu operator. Prelu is a variation of relu operator.

The operator multiplies the negative part by a coefficient to create a rectification effect which is similar to breaking. By contrast, relu operator directly zeroes the negative part directly. The parameter of prelu controls the slope of rectification. When p=0.1, the operator is leaky relu operator.

The shapes of input and output must be the same.

Formula

if prelu_param.shape == [1,1,1,1]

out[n c h w] = in[n c h w] > 0 ? in[n c h w] : prelu_param[1,1,1,1] * in[n,c,h,w];

else

out[n c h w] = in[n c h w] > 0 ? in[n c h w] : prelu_param[1,c,1,1] * in[n,c,h,w];

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape is [ni, ci, hi, wi], supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [no, co, ho, wo](no = ni), supports data of float16 type.
- `[in] prelu_param`: Input. A four-dimensional MLU tensor, the shape is [1, 1, 1, 1] or [1,c,1,1], supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Otherwise, return the corresponding error.

4.106.2 cnmlCreateNdPreluOp

cnmlStatus_t **cnmlCreateNdPreluOp**(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t prelu_param)

A function.

According to the base operator pointer given by the user,create a NdPrelu operator. NdPrelu is a variation of relu operator.

The operator multiplies the negative part by a coefficient to create a rectification effect which is similar to breaking. By contrast, relu operator directly zeroes the negative part directly. The parameter of Ndprelu controls the slope of rectification. When p=0.1, the operator is leaky relu operator.

The shapes of input and output must be the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A multi-dimensional MLU input tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.

- `[in] nd_prelu_param`: Input. A multi-dimensional MLU prelu tensor, supporting data of float16 type. The channel dim is same to input or set value 1.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. Otherwise, return the corresponding error.

4.106.3 cnmlComputePreluOpForward_V3

cnmlStatus_t **cnmlComputePreluOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Compute the user-specified Prelu operator on the MLU.

Deprecated. This interface will be deleted in next version and cnmlComputePreluOpForward_V4 is recommended to use.

After creating Prelu operator, input, output and computation queue, introduce them to the function to compute the Prelu operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.106.4 cnmlComputePreluOpForward_V4

cnmlStatus_t **cnmlComputePreluOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified Prelu operator on the MLU.

After creating Prelu operator, input, output and computation queue, introduce them to the function to compute the Prelu operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.106.5 cnmlComputeNdPreluOpForward

cnmlStatus_t **cnmlComputeNdPreluOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified Nd Prelu operator on the MLU.

After creating Nd Prelu operator, input, output and computation queue, introduce them to the function to compute the Nd Prelu operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.107 Random Uniform Operation

4.107.1 cnmlCreateRandomUniformOpParam

cnmlStatus_t **cnmlCreateRandomUniformOpParam**(cnmlRandomUniformOpParam_t *param, cnmlRngType_t type, float min_val, float max_val)

A function.

This function fills cnmlRandomUniformOpParam_t struct with random_uinform operation parameters giver by user.

This function allocates param memory, and after usage is done, user needs to call cnmlDestroyRandomUniformOpParam to destroy this param instance.

Supports MLU220 and MLU270.

Parameters

- [out] param: Output. A pointer pointing to the address of the struct of random_uinform operation.
- [in] type: Input. An enumerate type to describe underlying algorithm used to generate pseudorandom number.
- [in] seed: Input. Seed used to initialize seed value in algorithm.
- [in] min_val: Input. A value represents the left bound of distribution range.
- [in] max_val: Input. A value represents the right bound of distribution range. max_val should be greater than min val, and the distribution range is [min_val, max_val).

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.

4.107.2 cnmlDestroyRandomUniformOpParam

cnmlStatus_t **cnmlDestroyRandomUniformOpParam**(cnmlRandomUniformOpParam_t *param)

A function.

Free the struct pointer of random_uniform operator operation parameter according to the pointer given by the user.

At the end of the random_uniform operator operation, the struct pointer of the random_uniform operator operation parameter is freed.

Supports MLU220 and MLU270.

Parameters

- [in] param: Input. A pointer pointing to the address of the struct of the random_uniform operation parameter.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.107.3 cnmlCreateRandomUniformOp

cnmlStatus_t `cnmlCreateRandomUniformOp`(cnmlBaseOp_t *op, cnmlRandomUniformOpParam_t param, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create an random_uniform number generator operator.

After creating a pointer to the base operator address, the random_uinform param and output tensor are created, they are introduced into the function to create the random_uinform operator.

Formla

for each float value z in output_tensor:

a. x = rng.get_rand_uint32()
b. man = x & 0x7FFFFF
c. exp = static_cast<unsigned int>(127)
d. val = (exp << 23) | man;
e. memcpy(&z, &val, sizeof(float32))

DataType

output_tensor: float32

Scale Limitation

output_tensor[n c h w]: n > 0, c > 0, h > 0, w > 0

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] param`: Input. A random_uniform param structure to describe op’ s attribution.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, only support float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason3 The output pointer is null.

4.107.4 cnmlSetRandomSeed

cnmlStatus_t `cnmlSetRandomSeed`(cnmlBaseOp_t op, unsigned int seed)

A function.

For setting random seed to random number generator operation. Default seed value is 1 if no cnmlSetRandSeed is called.

Support MLU270.

Parameters

- `[out] output`: Output. A pointer pointing to base operator address
- `[in] seed`: Input. random seed

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 Seed function is not compatible with underlying prng type.

4.107.5 cnmlComputeRandomUniformOpForward

cnmlStatus_t `cnmlComputeRandomUniformOpForward`(cnmlBaseOp_t op, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified random_uniform operator on the MLU.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. An pointer which points to base operator.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:

- Reason1 The task type of runtime is invalid.

4.108 Real Div Operation

4.108.1 cnmlComputeRealDivOpForward_V3

cnmlStatus_t `cnmlComputeRealDivOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeRealDivOpForward_V4 is recommended to use.

Compute the division operator given by users on the MLU.

After creating a division operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the division operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The operator pointer is null.

4.108.2 cnmlComputeRealDivOpForward_V4

cnmlStatus_t `cnmlComputeRealDivOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the division operator given by users on the MLU.

After creating a division operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the division operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_1`: Input. An MLU address pointing to input data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input_2`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.108.3 cnmlCreateRealDivOp

cnmlStatus_t `cnmlCreateRealDivOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a division operator according to base operator pointers given by users. After creating a pointer pointing to base operator address, and input and output Tesor, pass them into the fucntion to create the division operator.

Before creating a division operator, declare a pointer pointing to the struct address of operation parameters of the division operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Perform element-wise division on the two inputs to obtain output.

The shape of the second input can be the same shape of first input or (1, 1, 1, 1). If the shape of second input is (1, 1, 1, 1), operation will expand it to a vector to excute div operation. If the input tensor shape is (1, 1, 1, 1), the high precision interface should not be called.

Formula

c[n c h w] = a[n c h w] / b[n c h w]

or

c[n c h w] = a[n c h w] / b[1 1 1 1]

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU output tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU weight tensor, the shape of which is [no, co, ho, wo] (no = ni, co = ci, ho = hi, wi = wo), supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.108.4 cnmlSetRealDivHighPrecision

cnmlStatus_t `cnmlSetRealDivHighPrecision`(cnmlBaseOp_t *op, bool high_precision_flag)

A function.

Set a high-precision mode according to the division operator given by users.

After creating a division operator, calling this interface will start the high-precision mode.

If the high precision mode is set, the divisor can be negative. The precision will be improved when divisor interval is [0,1].

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] high_precision_flag`: Input. Set the operator as a high-precision identifier. The” true” represents the mode is set to the high-precision mode.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM:`: At least one of the following conditions are met:
  - op is a null pointer.

4.109 Reduce And Operation

4.109.1 cnmlCreateReduceAndOp

cnmlStatus_t `cnmlCreateReduceAndOp`(cnmlBaseOp_t *op, cnmlReduce_andDim_t dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReduceAndOp.

Deprecated. This interface will be deleted in next version and cnmlCreateReduceAndOpForward is recommended to use.

Create a Reduce And operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Reduce And operator.

Performs the logic AND operation on the specified dimension of a tensor.

Formula:

DIM_N: out[1 c 1 1] = logic and(in[n c 1 1], DIM_N)

DIM_C: out[n 1 1 1] = logic and(in[n c 1 1], DIM_C)

DataType:

MLU270:

input: float16, float32, bool

output: float16, float32, bool

Scale limitation:

MLU270:

in FP16 and bool:

a. the max supported C is 65472 (8192 * 32 / 4 -64)

in FP32:

a. the max supported C is 32736 (8192 * 16 / 4 -32)

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] dim`: Input. A number, the dimension to be reduced by the user, supporting N, C.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional tensor, the size of dimensions of which the shape is reduced must be 1; the size of other dimenisions is consistent with that of input, h and w must be 1. supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.109.2 cnmlComputeReduceAndOpForward

cnmlStatus_t `cnmlComputeReduceAndOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReduceAndOpForward.

It is used to compute the user-specified Reduce And operator on the MLU.

After creating the Reduce And operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Reduce And operator.

Formula:

DIM_N: out[1 c 1 1] = logic and(in[n c 1 1], DIM_N)

DIM_C: out[n 1 1 1] = logic and(in[n c 1 1], DIM_C)

DataType:

MLU270:

input: float16, float32, bool

output: float16, float32, bool

Scale limitation:

MLU270:

in FP16 and bool:

a. the max supported C is 65472 (8192 * 32 / 4 -64)

in FP32:

a. the max supported C is 32736 (8192 * 16 / 4 -32)

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.110 Reduce Max Operation

4.110.1 cnmlCreateReduceMaxOp

cnmlStatus_t `cnmlCreateReduceMaxOp`(cnmlBaseOp_t *op, cnmlDimension_t reduce_max_mode, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a Reduce Max operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output tensor, pass them into the fucntion to create a Reduce Max operator.

The Reduce Max loads data continuously, therefore, if the direction of reduce is n and the size of seg*H*W is too large to load two Ns, the maximum value cannot be obtained.

Summary

input[n, c, h, w],and compute the max value of given direction.

such as direction is n, output[1, c, h, w]

such as direction is c, output[n, 1, h, w]

...

DataType

MLU270:

value_data_type = input_data_type:float16, float32, int32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] mode`: Input. An enumeration variable, a dimension that users will reduce, supporting N, C, H, and W.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, and the size of dimensions of which the shape is reduced must be 1. The size of other dimenisions is consistent with that of input Tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is neither CNML_TENSOR nor CNML_CONST.

4.110.2 cnmlCreateNdReduceMaxOp

cnmlStatus_t `cnmlCreateNdReduceMaxOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a Reduce Max operator supporting arbitrary dimension tensor according to base operator pointers given by users.

This operator extens to support arbitrary dimension tensor based on the Reduce Max operator.

After creating a pointer pointing to base operator address, dimension of reduce operation operation, and input and output Tensor, pass them into the fucntion to create a Reduce Max operator supporting multi-dimension.

Before creating a reduce max operator, declare a pointer pointing to the struct address of operation parameters of the operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Compute the maximum value along the dimension that users will reduce.

The reduce_max loads data continuously, therefore, if the direction of reduce is n and the size of seg*H*W is too large to load two Ns, the maximum value cannot be obtained.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] dim`: Input. A variable of int type, specifying dimensions of the reduce operation.
- `[in] input`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.
- `[in] output`: Input. A multi-dimensional MLU weight tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.110.3 cnmlComputeReduceMaxOpForward_V3

cnmlStatus_t `cnmlComputeReduceMaxOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeReduceMaxOpForward_V4 is recommended to use.

Compute the Reduce Max operator given by users on the MLU.

After creating Reduce Max operator, input, output, runtime parameters, and computation queue, pass them into the function to compute Reduce Max operator.

Summary

input[n, c, h, w],and compute the max value of given direction.

such as direction is n, output[1, c, h, w]

such as direction is c, output[n, 1, h, w]

…

DataType

MLU270:

value_data_type = input_data_type:float16, float32, int32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

at least process one full line c at a time

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.110.4 cnmlComputeReduceMaxOpForward_V4

cnmlStatus_t `cnmlComputeReduceMaxOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Reduce Max operator given by users on the MLU.

After creating Reduce Max operator, input, output, runtime parameters, and computation queue, pass them into the function to compute Reduce Max operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.110.5 cnmlComputeNdReduceMaxOpForward

cnmlStatus_t `cnmlComputeNdReduceMaxOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdReduceMaxOpForward_V2 is recommended to use.

Compute the Reduce Max operator supporting multi-dimension on the MLU.

After creating a Reduce Max operator supporting multi-dimension, input, output, and computation stream, pass them into the function to compute the Reduce Max operator supporting multi-dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.110.6 cnmlComputeNdReduceMaxOpForward_V2

cnmlStatus_t `cnmlComputeNdReduceMaxOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Reduce Max operator supporting multi-dimension on the MLU.

After creating a Reduce Max operator supporting multi-dimension, input, output, and computation stream, pass them into the function to compute the Reduce Max operator supporting multi-dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.

- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.111 Reduce Mean Operation

4.111.1 cnmlCreateReduceMeanOp

cnmlStatus_t **cnmlCreateReduceMeanOp**(cnmlBaseOp_t *op, cnmlDimension_t dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateNdReduceSumOp.

Create a reduce mean operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, reduce dimension, input and output tensors, pass them to the function to create a reduce mean operator.

Before creating the operator, declare a pointer to the address of the operator parameter struct and pass it to the function together with the desired operator parameters to set the operator parameters.

Perform mean operation in the selected dimension.

if the mode is CNML_DIM_N, the shape is [1, c, h, w];

if the mode is CNML_DIM_C, the shape is [n, 1, h, w];

if the mode is CNML_DIM_H, the shape is [n, c, 1, w];

if the mode is CNML_DIM_W, the shape is [n, c, h, 1];

Formula

DIM_N: out[1 c h w] = mean(in[n c h w], DIM_N)

DIM_C: out[n 1 h w] = mean(in[n c h w], DIM_C)

DIM_H: out[n c 1 w] = mean(in[n c h w], DIM_H)

DIM_W: out[n c h 1] = mean(in[n c h w], DIM_W)

DataType

MLU270:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

We recommend you use FP32 rather than FP16 to prevent precision problem. If you want to use FP16, all below conditions should be met:

a. # of reduce dimension under 10k.
b. Input data is normal distribution with mean value [-1, 1], variation[-10, 10]

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] mode: Input. An enumeration constant of the reduce dimension specified on the MLU, where the options include CNML_DIM_N, CNML_DIM_C, CNML_DIM_H and CNML_DIM_W.
- [in] input: Input. A 4-dimensional MLU input tensor, of which the shape is [n, c, h, w], supporting data of float16 type.
- [in] output: Input. A 4-dimensional MLU input tensor, of which the shape is [n, c, h, w], supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is empty
  - The input pointer is empty.
  - The output pointer is empty.

4.111.2 cnmlCreateNdReduceMeanOp

cnmlStatus_t `cnmlCreateNdReduceMeanOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateNdReduceMeanOp.

Create a reduce mean operator that supports arbitrary dimension tensor based on the base operator pointer given by the user.

This operator is extended to support arbitrary dimension tensor based on the reduce mean operator.

After creating a pointer to the base operator address, reduce dimension, input and output tensors, pass them to the function to create a reduce mean operator that supports multi dimensions.

Before creating the operator, declare a pointer to the address of the operator parameter struct and pass it to the function together with the desired operator parameters to set the operator parameters. Perform mean operation in the selected dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] dim`: Input. A variable of int type for specifying the dimension where the reduce operation is performed.
- `[in] input`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.
- `[in] output`: Input. A multi-dimensional MLU weight tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The input pointer is empty.
  - The output pointer is empty.

4.111.3 cnmlComputeReduceMeanOpForward_V3

cnmlStatus_t `cnmlComputeReduceMeanOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeReduceMeanOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeReduceMeanOpForward_V4 is recommended to use.

It is used to compute the reduce mean operator that supports multi dimensions on the MLU. After creating the reduce mean operator, Input, Output, computation type, and computation stream, pass them to the function to It is used to compute the reduce mean operator.

Formula

DIM_N: out[1 c h w] = mean(in[n c h w], DIM_N)

DIM_C: out[n 1 h w] = mean(in[n c h w], DIM_C)

DIM_H: out[n c 1 w] = mean(in[n c h w], DIM_H)

DIM_W: out[n c h 1] = mean(in[n c h w], DIM_W)

DataType

MLU270:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

We recommend you use FP32 rather than FP16 to prevent precision problem. If you want to use FP16, all below conditions should be met:

a. # of reduce dimension under 10k.
b. Input data is normal distribution with mean value [-1, 1], variation[-10, 10]

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] queue`: Input. A computation queue pointer.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.111.4 cnmlComputeReduceMeanOpForward_V4

cnmlStatus_t `cnmlComputeReduceMeanOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReduceMeanOpForward_V4.

It is used to compute the reduce mean operator that supports multi dimensions on the MLU. After creating the reduce mean operator, Input, Output, computation type, and computation stream, pass them to the function to It is used to compute the reduce mean operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.111.5 cnmlComputeNdReduceMeanOpForward

cnmlStatus_t `cnmlComputeNdReduceMeanOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNdReduceMeanOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeNdReduceMeanOpForward_V2 is recommended to use.

It is used to compute the reduce mean operator that supports multi dimensions on the MLU.

After creating a reduce mean operator that supports multi dimensions, Input, Output, and computation stream, pass them to the function to It is used to compute the reduce mean operator that supports multi dimensions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.111.6 cnmlComputeNdReduceMeanOpForward_V2

cnmlStatus_t `cnmlComputeNdReduceMeanOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNdReduceMeanOpForward.

It is used to compute the reduce mean operator that supports multi dimensions on the MLU.

After creating a reduce mean operator that supports multi dimensions, Input, Output, and computation stream, pass them to the function to It is used to compute the reduce mean operator that supports multi dimensions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.112 Reduce Product Operation

4.112.1 cnmlCreateReduceProductOp

cnmlStatus_t `cnmlCreateReduceProductOp`(cnmlBaseOp_t *op, cnmlDimension_t mode, cnmlTensor_t input, cnmlTensor_t output)

A function.

Create a Reduce Product operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output tensor, pass them into the fucntion to create a Reduce Product operator.

The Reduce Product loads data continuously, therefore, if the direction of reduce is n and the size of seg*H*W is too large to load two Ns, the maximum value cannot be obtained.

Supports MLU220,MLU270,1M20,and 1M70.

Formula

Such as when input[ni, ci, hi, wi], output[no, co, ho, wo] and `d` direction is `c`, then output[n, c, h, w] = reduceproduct(n, c, h, w) = product_i(input[n, i, h, w])

Datatype

MLU270:

float16, float32, int16, int32

Scale Limitation

MLU270:

c < 65280 because of at least process one full line c at a time

Parameters

- `[out] op`: Output. A pointer to the Reduce Product operator you have created.
- `[in] mode`: Input. An enumerated type specifies the dimension to reduce. Supported values are: N, C, H, and W.
- `[in] input_tensor`: Input. A 4-D MLU tensor to reduce. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.
- `[in] output_tensor`: Input. The descriptor of the 4-D MLU output tensor. The size of other dimensions must be consistent with the input tensor. Supported data type of this tensor descriptor is FP32. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.

Return Value

- `CNML_STATUS_SUCCESS`: This function run successfully.
- `CNML_STATUS_INVALIDPARAM`: The type of input tensor is neither CNML_TENSOR nor CNML_CONST.

4.112.2 cnmlComputeReduceProductOpForward

cnmlStatus_t `cnmlComputeReduceProductOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Computes the product elements for a specified dimension of a tensor on MLU.

Deprecated. This interface will be deleted in next version and cnmlComputeReduceProductOpForward_V2 is recommended to use.

After creating Reduce Product operator, input, output, runtime parameters, and computation queue, pass them into the function to compute Reduce Product operator.

Supports MLU220,MLU270,1M20,and 1M70.

Formula

Such as when input[ni, ci, hi, wi], output[no, co, ho, wo] and `d` direction is `c`, then output[n, c, h, w] = reduceproduct(n, c, h, w) = product_i(input[n, i, h, w])

Datatype

MLU270:

float16, float32, int16, int32

Scale Limitation

MLU270:

c < 65280 because of at least process one full line c at a time

Parameters

- [in] op: Input. A pointer to the Reduce Product operator you have created.
- [in] input: Input. A pointer to the data of the tensor you want to reduce.
- [out] output: Output. A pointer to the data of the reduced tensor.
- [in] compute_forw_param: Input. A pointer to the struct address that records the data parallelism and device affinity for runtime.
- [in] queue: Input. A pointer to the queue that is used to implement the computation.

Return Value

- CNML_STATUS_SUCCESS: This function run successfully.
- CNML_STATUS_INVALIDPARAM: One of the following conditions occurred:
  - The pointer to the Reduce Product operator is null.
  - The pointer to the data of the reduced tensor is null.

4.112.3 cnmlComputeReduceProductOpForward_V2

cnmlStatus_t **cnmlComputeReduceProductOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Reduce Product operator given by users on the MLU.

After creating Reduce Product operator, input, output, runtime parameters, and computation queue, pass them into the function to compute Reduce Product operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.113 Reduce Sum Operation

4.113.1 cnmlCreateReduceSumOp

cnmlStatus_t **cnmlCreateReduceSumOp**(cnmlBaseOp_t *op, cnmlDimension_t dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReduceSumOp.

Create a Reduce Sum operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Reduce Sum operator.

The functionality of this operator is to sum the input Tensor in the selected dimensions to be reduced.

Formula

DIM_N: out[1 c h w] = sum(in[n c h w], DIM_N)

DIM_C: out[n 1 h w] = sum(in[n c h w], DIM_C)

DIM_H: out[n c 1 w] = sum(in[n c h w], DIM_H)

DIM_W: out[n c h 1] = sum(in[n c h w], DIM_W)

DataType

MLU270:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

We recommend you use FP32 rather than FP16 to prevent precision problem. If you want to use FP16, all below conditions should be met:

a. # of reduce dimension under 10k.
b. Input data is normal distribution with mean value [-1, 1], variation[-10, 10]

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] mode`: Input. An enumeration variable, the dimension to be reduced by the user, supporting N, C, H, W.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional tensor, the size of dimensions of which the shape is reduced must be 1; the size of other dimenisions is consistent with that of input, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.113.2 cnmlCreateNdReduceSumOp

cnmlStatus_t **cnmlCreateNdReduceSumOp**(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateNdReduceSumOp.

Create a reduce sum operator to support arbitrary dimension tensor based on the base operator pointer given by the user.

This operator is extended to support arbitrary dimension based on the reduce sum operator.

After creating a pointer to the base operator address, dimension of reduce operation, input and output tensors, pass them to the function to create a reduce sum operator that supports multi dimensions.

Before creating the operator, declare a pointer to the address of the operator parameter struct and pass it to the function together with the desired operator parameters to set the operator parameters.

Perform summation in the selected dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] dim`: Input. A variable of int type for specifying the dimension where the reduce operation is performed.
- `[in] input`: Input. A multi-dimensional MLU tensor, supporting data of float16 type.
- `[in] output`: Input. A multi-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - The operator pointer is empty.
  - The input pointer is empty.
  - The output pointer is empty.

4.113.3 cnmlComputeReduceSumOpForward_V3

cnmlStatus_t **cnmlComputeReduceSumOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeReduceSumOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeReduceSumOpForward_V4 is recommended to use.

It is used to compute the user-specified Reduce Sum operator on the MLU.

After creating the Reduce Sum operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Reduce Sum operator.

Formula

DIM_N: out[1 c h w] = sum(in[n c h w], DIM_N)

DIM_C: out[n 1 h w] = sum(in[n c h w], DIM_C)

DIM_H: out[n c 1 w] = sum(in[n c h w], DIM_H)

DIM_W: out[n c h 1] = sum(in[n c h w], DIM_W)

DataType

MLU270:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

We recommend you use FP32 rather than FP16 to prevent precision problem. If you want to use FP16, all below conditions should be met:

a. # of reduce dimension under 10k.
b. Input data is normal distribution with mean value [-1, 1], variation[-10, 10]

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty * - The output pointer is empty

4.113.4 cnmlComputeReduceSumOpForward_V4

cnmlStatus_t **cnmlComputeReduceSumOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReduceSumOpForward_V4.

It is used to compute the user-specified Reduce Sum operator on the MLU.

After creating the Reduce Sum operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Reduce Sum operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.113.5 cnmlComputeNdReduceSumOpForward

cnmlStatus_t **cnmlComputeNdReduceSumOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNdReduceSumOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeNdReduceSumOpForward_V2 is recommended to use.

It is used to compute the reduce sum operator that supports multi dimensions on the MLU.

After creating a multidimensional reduce sum operator, Input, Output, and computation stream, pass them to the function to It is used to compute the reduce sum operator that supports multi dimensions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.113.6 cnmlComputeNdReduceSumOpForward_V2

cnmlStatus_t `cnmlComputeNdReduceSumOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNdReduceSumOpForward_V2.

It is used to compute the reduce sum operator that supports multi dimensions on the MLU.

After creating a multidimensional reduce sum operator, Input, Output, and computation stream, pass them to the function to It is used to compute the reduce sum operator that supports multi dimensions.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.114 Reduce Or Operation

4.114.1 cnmlCreateReduceOrOp

cnmlStatus_t `cnmlCreateReduceOrOp`(cnmlBaseOp_t *op, cnmlReduce_orDim_t dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReduceOrOp.

Create a Reduce Or operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Reduce Or operator.

Performs the logic OR operation on the specified dimension of a tensor.

Formula:

DIM_N: out[1 c 1 1] = logic or(in[n c 1 1], DIM_N)

DIM_C: out[n 1 1 1] = logic or(in[n c 1 1], DIM_C)

DataType:

MLU270:

input: float16, float32, bool

output: float16, float32, bool

Scale limitation:

MLU270:

in FP16 and bool:

a. the max supported C is 65472 (8192 * 32 / 4 -64)

in FP32:

a. the max supported C is 32736 (8192 * 16 / 4 -32)

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] dim`: Input. A number, the dimension to be reduced by the user, supporting N, C.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, 1, 1], supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional tensor, the size of dimensions of which the shape is reduced must be 1; the size of other dimenisions is consistent with that of input, h and w must be 1. supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.114.2 cnmlComputeReduceOrOpForward

cnmlStatus_t `cnmlComputeReduceOrOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReduceOrOpForward.

It is used to compute the user-specified Reduce Or operator on the MLU.

After creating the Reduce Or operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Reduce Or operator.

Formula:

DIM_N: out[1 c 1 1] = logic or(in[n c 1 1], DIM_N)

DIM_C: out[n 1 1 1] = logic or(in[n c 1 1], DIM_C)

DataType:

MLU270:

input: float16, float32, bool

output: float16, float32, bool

Scale limitation:

MLU270:

in FP16 and bool:

a. the max supported C is 65472 (8192 * 32 / 4 -64)

in FP32:

a. the max supported C is 32736 (8192 * 16 / 4 -32)

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.115 Reorg Operation

4.115.1 cnmlCreateReorgOpParam

cnmlStatus_t `cnmlCreateReorgOpParam`(cnmlReorgOpParam_t *param, int reorg_h, int reorg_w, bool reverse)

cnmlCreateReorgOpParam.

The function creates a reorg operator operation parameter struct based on the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of the reorg operator operation parameter struct.
- `[in] reorg_h`: Input. The remodeling coefficient in the direction of h.If reverse is false, ho = hi / reorg_h, and in this case, hi is required to be divisible by reorg_h; if reverse is true, ho = hi * reorg_h.
- `[in] reorg_w`: Input. The remodeling coefficient in the direction of w. If reverse is false, wo = wi / reorg_w, which requires wi to be divisible by reorg_w; if reverse is true, wo = wi * reorg _w.
- `[in] reverse`: Input. bool value, false is forward and true is reverse (splitting or merging). If forward, co = ci * reorg_h * reorg_w. If reverse, co = ci / (reorg_h * reorg_w).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.115.2 cnmlDestroyReorgOpParam

cnmlStatus_t `cnmlDestroyReorgOpParam`(cnmlReorgOpParam_t *param)

cnmlDestroyReorgOpParam.

The reorg operator operation parameter struct pointer is released according to the pointer given by the user.

The created reorg operator operation parameter struct pointer is released after the convolution operator operation ends.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to the address of the reorg operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.115.3 cnmlCreateReorgOp

cnmlStatus_t `cnmlCreateReorgOp`(cnmlBaseOp_t *op, cnmlReorgOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReorgOp.

Create a reorg operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, reorg operator parameters, and input and output tensors, pass them to the function to create a reorg operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] param`: Input. A reorg operation struct pointer.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi] and given by the user.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, of which the shape is [no, co, ho, wo] (no = ni) and computed by cnmlGetReorgOpOutputDim based on the input dimension information. If reverse = true, co = ci / (reorg_h * reorg_w), ho = hi * reorg_h, wo = wi * reorg_w. If reverse = false, co = ci * reorg_h * reorg_w, ho = hi / reorg_h, wo = wi / reorg_w.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The input param is empty.
  - The input input_tensor and output_tensor are empty.

4.115.4 cnmlComputeReorgOpForward_V3

cnmlStatus_t `cnmlComputeReorgOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeReorgOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeReorgOpForward_V4 is recommended to use.

It is used to compute the reorg operator specified by the user on the MLU.

After creating the reorg operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the reorg operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.115.5 cnmlComputeReorgOpForward_V4

cnmlStatus_t `cnmlComputeReorgOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReorgOpForward_V4.

It is used to compute the reorg operator specified by the user on the MLU.

After creating the reorg operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the reorg operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.116 Reshape Operation

4.116.1 cnmlCreateReshapeOpParam

cnmlStatus_t `cnmlCreateReshapeOpParam`(cnmlReshapeOpParam_t *param, int no, int co, int ho, int wo, cnmlDataOrder_t data_order)

cnmlCreateReshapeOpParam.

The function creates a reshape operator operation parameter struct according to the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of the reshape operator operation parameter struct.
- `[in] no`: Input. The length of the output n dimension.
- `[in] co`: Input. The length of the output c dimension.
- `[in] ho`: Input. The length of the output h dimension.
- `[in] wo`: Input. The length of the output w dimension.
- `[in] df`: Input. output data dimension order, which is same as the created cpu tensor dimension order.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.

4.116.2 cnmlCreateNdReshapeOpParam

cnmlStatus_t `cnmlCreateNdReshapeOpParam`(cnmlReshapeOpParam_t *param, int dims[], int dim_num)

cnmlCreateNdReshapeOpParam.

The function creates a multi-dimension reshape operator operation parameter struct according to the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of the reshape operator operation parameter struct.
- `[in] dims[]`: Input. The array of the output dimensions.
- `[in] dim_num`: Input. The number of the output dimensions.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.

4.116.3 cnmlDestroyReshapeOpParam

cnmlStatus_t **cnmlDestroyReshapeOpParam**(cnmlReshapeOpParam_t *param)

cnmlDestroyReshapeOpParam.

Release the reshape operator operation parameter struct pointer according to the pointer given by the user.

Release the created reshape operator operation parameter struct pointer after the reshape operator operation ends.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to the address of the reshape operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.
  - The pointer content pointed to by param has been released.

4.116.4 cnmlCreateReshapeOp

cnmlStatus_t **cnmlCreateReshapeOp**(cnmlBaseOp_t *op, cnmlReshapeOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReshapeOp.

Create a Reshape operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, reshape operator operation parameters, input and output tensors, pass them to the function to create a reshape operator.

Formula

Change input shape[ni ci hi wi] to output shape [no co ho wo]

Change input DataType to output DataType

DataType

MLU270:

if input and output DataType are same:

float16, float32, int8, int16, int32, uint8, uint16, uint32, bool

else

input to output:

uint8 to float16, int8 to float16, float16 to int8, float16 to float32, float32 to float16, int16 to float16, float16 to int16, int8 to float32, float32 to int8, int16 to float32, float32 to int16

Scale Limitation

MLU270:

no * co * ho * wo = ni * ci * hi * wi

Performance Optimization

The value of Ci and logic_c_ are the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tesor`: Output. A pointer to the mlu end Tensor
- `[in] op`: Input. A pointer to the base operator
- `[in] input_tensor`: Input. A pointer to the mlu end
- `[in] param`: Input. A pointer to the address of the reshape operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: Op, input, output, and param are not empty.

4.116.5 cnmlCreateReshapeOp_V2

cnmlStatus_t `cnmlCreateReshapeOp_V2`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateReshapeOp_V2.

Create a YUVtoRGB operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a reshape operator.

cnmlBindCpuDataInfo must be called before this interface is called.

Formula

Change input shape[ni ci hi wi] to output shape [no co ho wo]

Change input DataType to output DataType

DataType

MLU270:

if input and output DataType are same:

float16, float32, int8, int16, int32, uint8, uint16, uint32, bool

else

input to output:

uint8 to float16, int8 to float16, float16 to int8, float16 to float32, float32 to float16, int16 to float16, float16 to int16, int8 to float32, float32 to int8, int16 to float32, float32 to int16

Scale Limitation

MLU270:

no * co * ho * wo = ni * ci * hi * wi

Performance Optimization

The value of Ci and logic_c_ are the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tesor`: Output. A pointer to the mlu end Tensor
- `[in] op`: Input. A pointer to the base operator
- `[in] input_tensor`: Input. A pointer to the mlu end

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: Op, input, and output are not empty.

4.116.6 cnmlComputeReshapeOpForward_V3

cnmlStatus_t `cnmlComputeReshapeOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeReshapeOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeReshapeOpForward_V4 is recommended to use.

It is used to compute the user-specified reshape operator on the MLU.

After creating the reshape operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the reshape operator.

Formula

Change input shape[ni ci hi wi] to output shape [no co ho wo]

Change input DataType to output DataType

DataType

MLU270:

if input and output DataType are same:

float16, float32, int8, int16, int32, uint8, uint16, uint32, bool

else

input to output:

uint8 to float16, int8 to float16, float16 to int8, float16 to float32, float32 to float16, int16 to float16, float16 to int16, int8 to float32, float32 to int8, int16 to float32, float32 to int16

Scale Limitation

MLU270:

no * co * ho * wo = ni * ci * hi * wi

Performance Optimization

The value of Ci and logic_c_ are the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.116.7 cnmlComputeReshapeOpForward_V4

cnmlStatus_t `cnmlComputeReshapeOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeReshapeOpForward_V3.

It is used to compute the user-specified reshape operator on the MLU.

After creating the reshape operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the reshape operator.

Formula

Change input shape[ni ci hi wi] to output shape [no co ho wo]

Change input DataType to output DataType

DataType

MLU270:

if input and output DataType are same:

float16, float32, int8, int16, int32, uint8, uint16, uint32, bool

else

input to output:

uint8 to float16, int8 to float16, float16 to int8, float16 to float32, float32 to float16, int16 to float16, float16 to int16, int8 to float32, float32 to int8, int16 to float32, float32 to int16

Scale Limitation

MLU270:

no * co * ho * wo = ni * ci * hi * wi

Performance Optimization

The value of Ci and logic_c_ are the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.117 Reverse Operation

4.117.1 cnmlCreateReverseOp

cnmlStatus_t `cnmlCreateReverseOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlDimension_t reverse_axis)

A function.

Create a reverse operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output tensor, pass them into the fucntion to create a reverse operator.

The shape of input tensor and output tensor should be the same.

Formula

if input data shape: NHWC are 2,2,2,2

input data: [[[0, 1], [2, 3]], [[4, 5], [6, 7]]], [[[8, 9], [10, 11]], [[12, 13], [14, 15]]]

Reverse in dim N

output data: [[[8, 9], [10, 11]], [[12, 13], [14, 15]]], [[[0, 1], [2, 3]], [[4, 5], [6, 7]]]

Reverse in dim H

output data: [[[4, 5], [6, 7]], [[0, 1], [2, 3]]], [[[12, 13], [14, 15]], [[8, 9], [10, 11]]]

Reverse in dim W:

output data: [[[2, 3], [0, 1]], [[6, 7], [4, 5]]], [[[10, 11], [8, 9]], [[14, 15], [12, 13]]]

Reverse in dim C:

output data: [[[1, 0], [3, 2]], [[5, 4], [7, 6]]], [[[9, 8], [11, 10]], [[13, 12], [15, 14]]]

DataType

MLU270:

inputDataType: float16, float32, int16, int8

outputDataType: float16, float32, int16, int8

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape of which is [n, c, h, w], supporting data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape of which is [n, c, h, w], supporting data of float16 type.
- `[in] reverse_axis`: Input. An enumeration variable, supporting reversing along the dimension N, C, H, W, and HW

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The type of input tensor is neither CNML_TENSOR nor CNML_CONST.

4.117.2 cnmlComputeReverseOpForward_V3

cnmlStatus_t `cnmlComputeReverseOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeReverseOpForward_V4 is recommended to use.

Compute the reverse operator specified by users on the MLU.

After creating a reverse operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the reverse operator.

Formula

if input data shape: NHWC are 2,2,2,2

input data: [[[0, 1], [2, 3]], [[4, 5], [6, 7]]], [[[8, 9], [10, 11]], [[12, 13], [14, 15]]]

Reverse in dim N

output data: [[[8, 9], [10, 11]], [[12, 13], [14, 15]]], [[[0, 1], [2, 3]], [[4, 5], [6, 7]]]

Reverse in dim H

output data: [[[4, 5], [6, 7]], [[0, 1], [2, 3]]], [[[12, 13], [14, 15]], [[8, 9], [10, 11]]]

Reverse in dim W:

output data: [[[2, 3], [0, 1]], [[6, 7], [4, 5]]], [[[10, 11], [8, 9]], [[14, 15], [12, 13]]]

Reverse in dim C:

output data: [[[1, 0], [3, 2]], [[5, 4], [7, 6]]], [[[9, 8], [11, 10]], [[13, 12], [15, 14]]]

DataType

MLU270:

inputDataType: float16, float32, int16, int8

outputDataType: float16, float32, int16, int8

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.117.3 cnmlComputeReverseOpForward_V4

cnmlStatus_t `cnmlComputeReverseOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the reverse operator specified by users on the MLU.

After creating a reverse operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the reverse operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.118 Rsqrt Operation

4.118.1 cnmlCreateRsqrtOp

cnmlStatus_t `cnmlCreateRsqrtOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a square root reciprocal operator.

The shapes of input and output shoul be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The shape of output tensor is different from that of input tensor.

4.118.2 cnmlComputeRsqrtOpForward_V3

cnmlStatus_t `cnmlComputeRsqrtOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeRsqrtOpForward_V4 is recommended to use.

Compute the user-specified square root reciprocal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.

4.118.3 cnmlComputeRsqrtOpForward_V4

cnmlStatus_t `cnmlComputeRsqrtOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified square root reciprocal operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.119 Scale Operation

4.119.1 cnmlCreateScaleOp

cnmlStatus_t `cnmlCreateScaleOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t alpha_tensor, cnmlTensor_t beta_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateScaleOpForward is recommended to use.

Create a linear transformation operator, which can perform linear transformation on the input.(out = alpha * in + beta).

Formula

alpha can be (1 c 1 1) or (1 1 1 1)

beta can be (1 c 1 1) or (1 1 1 1)

output[n c h w] = alpha[1 c 1 1] * input[n c h w] + beta[1 c 1 1]

DataType

```
inputDataType, alphaDataType, betaDataType: float16, float32

outputDataType: float16, float32, int16, int8
```

Scale Limitation

MLU270:

if (alpha(1 c 1 1) && beta (1 1 1 1)) or (alpha(1 1 1 1) && beta(1 c 1 1))

then c < 65000

else if (alpha(1 c 1 1)&& beta(1 c 1 1)

then c < 43000

else if (alpha(1 1 1 1)&& beta(1 1 1 1)

then c < 130400

Supports MLU220,MLU270,1M20,and 1M70.

Note The alpha, beta are constant data in scale operation. The type of these tensors need to be CNML_CONST, and it need to bind data infomration by cnmlBindConstData_V2.

Parameters

- `[out] op`: Output. A pointer pointing to the operator.
- `[in] input`: Input. A four-dimensional input tensor([n, h, w, c]).
- `[in] output`: Input. A four-dimensional output tensor([n, h, w, c]).
- `[in] alpha`: Input. Coefficients of linear transformation(out = alpha * in + beta). Supporting data type same as input tensor.
- `[in] beta`: Input. Bias of linear transformation(out = alpha * in + beta). Supporting data type same as input tensor.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.119.2 cnmlCreateNdScaleOp

cnmlStatus_t `cnmlCreateNdScaleOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t alpha_tensor, cnmlTensor_t beta_tensor)

A function.

Description

Deprecated. This interface will be deleted in next version.

Create a linear transformation operator, which can perform linear transformation on the input.(out = alpha * in + beta).

Formula

alpha can be (1 c 1 1) or (1 1 1 1)

beta can be (1 c 1 1) or (1 1 1 1)

output[n c h w] = alpha[1 c 1 1] * input[n c h w] + beta[1 c 1 1]

DataType

```
- inputDataType, alphaDataType, betaDataType: float16, float32

- computeDataType: computeDataType = inputDataType

- outputDataType: float16, float32, int16, int8
```

Scale Limitation:

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The value of C dimension is a multiple of 128.

Note The alpha, beta are constant data in NdScale operation. The type of these tensors need to be CNML_CONST, and it need to bind data infomration by cnmlBindConstData_V2.

Parameters

- `[out] op`: Output. A pointer pointing to the operator.
- `[in] dim`: Input. Int num which marks operating dim.
- `[in] input`: Input. A multi-dimensional MLU input tensor.
- `[in] output`: Input. A multi-dimensional MLU output tensor.
- `[in] alpha`: Input. A multi-dimensional MLU tensor. Supporting data type same as input tensor. Coefficients of linear transformation(out = alpha * in + beta).
- `[in] beta`: Input. A multi-dimensional MLU tensor, supporting data of float16 type. Supporting data type same as input tensor. Bias of linear transformation(out = alpha * in + beta).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. For more information, see "Error Codes" section in this guide.

4.119.3 cnmlCreateScaleOpForward

This API has the same function as cnmlCreateScaleOp. For detailed information, see cnmlCreateScaleOp.

4.119.4 cnmlComputeScaleOpForward_V3

cnmlStatus_t `cnmlComputeScaleOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Description

Compute the linear transformation operator.

Formula

alpha can be (1 c 1 1) or (1 1 1 1)

beta can be (1 c 1 1) or (1 1 1 1)

output[n c h w] = alpha[1 c 1 1] * input[n c h w] + beta[1 c 1 1]

DataType

MLU270:

- inputDataType: float16, float32
- computeDataType: computeDataType = inputDataType
- outputDataType: float16, float32, int16, int8

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Performance Optimization

The value of C dimension is a multiple of 128.

Parameters

- `[in] op`: Input. An operator pointer.
- `[in] input`: Input. Pointing to input tensor.
- `[in] output`: Input. Pointing to output tensor.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] stream`: Input. Pointing to computation stream.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally. For more information, see "Error Codes" section in this guide.

4.119.5 cnmlComputeScaleOpForward_V4

cnmlStatus_t `cnmlComputeScaleOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Deprecated. This interface will be deleted in next version and cnmlComputeScaleOpForward_V4 is recommended to use.

Compute the linear transformation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.119.6 cnmlComputeNdScaleOpForward

cnmlStatus_t `cnmlComputeNdScaleOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Deprecated. This interface will be deleted in next version and cnmlComputeNdScaleOpForward is recommended to use.

Compute the linear transformation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.119.7 cnmlComputeNdScaleOpForward_V2

cnmlStatus_t `cnmlComputeNdScaleOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t alpha_tensor, void *alpha, cnmlTensor_t beta_tensor, void *beta, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Compute the linear transformation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.

- [in] input: Input. An MLU address pointing to input data.
- [in] alpha_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] alpha: Input. An MLU address pointing to input data.
- [in] beta_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] beta: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.119.8 cnmlComputeScaleOpForwardUltra_V3

cnmlStatus_t **cnmlComputeScaleOpForwardUltra_V3**(cnmlBaseOp_t op, void *input, void *alpha, void *beta, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Description

Deprecated. This interface will be deleted in next version and cnmlComputeScaleOpForwardUltra_V4 is recommended to use.

Compute the linear transformation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. An operator pointer.
- [in] input: Input. Pointing to input tensor.
- [in] alpha: Input. Coefficients of linear transformation.
- [in] beta: Input. Bias of linear transformation.
- [in] output: Input. Pointing to output tensor.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] stream: Input. Pointing to computation stream.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.119.9 cnmlComputeScaleOpForwardUltra_V4

cnmlStatus_t **cnmlComputeScaleOpForwardUltra_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t alpha_tensor, void *alpha, cnmlTensor_t beta_tensor, void *beta, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Description

Compute the linear transformation operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] alpha_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] alpha: Input. An MLU address pointing to input data.
- [in] beta_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] beta: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.

- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.120 Sequence Mask Operation

4.120.1 cnmlCreateSequenceMaskOpParam

cnmlStatus_t **cnmlCreateSequenceMaskOpParam**(cnmlSequenceMaskOpParam_t *param, bool use_sequence_length, int sequence_length[], int axis, float value)

A function.

According to the pointer given by the user, this function creates a struct of RNN operator operation parameters, and fills in the struct with parameters input by the user.

Parameters

- [in] param: Input. A pointer pointing to another pointer, and the pointed pointer points to a struct describing the operation parameters of SequenceMask operators.
- [in] use_sequence_length: Input. A bool variable determins whethter to use variable sequence_length or not. If it is false, this operation works as identity operator.
- [in] sequence_length: Input. A pointer pointing to a int array of dimension of batch size, which specifys the variable sequence length.
- [in] axis: Input. A int variable to specify the sequence axis in input.
- [in] value: Input. A float variable. The elements out of the sequence length would be replaced by this value

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.120.2 cnmlDestroySequenceMaskOpParam

cnmlStatus_t **cnmlDestroySequenceMaskOpParam**(cnmlSequenceMaskOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the SequenceMask operator operation parameter is freed.

Parameters

- [in] param: Input. A pointer pointing to another pointer, and the pointed pointer points to a struct describing the operation parameters of SequenceMask operators.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is a null pointer.
  - The content of the pointer pointed to by param has been freed.

4.120.3 cnmlCreateSequenceMaskOp

cnmlStatus_t **cnmlCreateSequenceMaskOp**(cnmlBaseOp_t *op, cnmlSequenceMaskOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the basic operator pointer given by the user, a SequenceMask operator is created.

After a pointer pointing to the address of base operator, the operation parameter of sequence mask operator and input-output tensor are created, they are introduced into the function to create the sequence mask operator.

Before the sequence mask operator is created, a pointer pointing to the address of the sequence mask operator parameter struct is declared, and the pointer and required operator parameter are introduced to the function to set the operator parameter.

The sequence mask operator set all elements outside the sequence to a constant value.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to the address of the base operator.
- [in] param: Input. A pointer pointing to the struct describing the parameter of SequenceMask operator.
- [in] input_tensor: Input. A 3-dimensional input tensor of MLU, the shape is [max_sequence_length, batch_size, other_feature_dims] or [batch_size, max_sequence_length, other_feature_dims], supporting the data of float16 and float32 type.
- [in] output_tensor: Input. An output tensor of MLU, and its shape is the same as that of input_tensor.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Any input is a null pointer.

4.120.4 cnmlComputeSequenceMaskOpForward_V4

cnmlStatus_t `cnmlComputeSequenceMaskOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

The SequenceMask operator, input, output, parameter at runtime, and computational queue are created and called, and SequenceMask operation is performed on the MLU.

Summary

For input[ti, ni, ci] in input[t, n, c],

if use_sequence_length,

output[ti, ni, ci] = ti < sequence_length[ni]? input[ti, ni, ci] : value

if not use_sequence_lenth,

output[ti, ni, ci] = input[ti, ni, ci]

Datatype

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale limitation:

MLU270:

Unlimited

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] queue`: Input. A computational queue pointer.
- `[in] extra`: Input. A pointer reserved for future use

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op is a null pointer.
  - output is a null pointer.
  - state_output is a null pointer.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - The task type is invalid at runtime.

4.121 SELU Operation

4.121.1 cnmlCreateSeluOp

cnmlStatus_t `cnmlCreateSeluOp`(cnmlBaseOp_t *op, cnmlTensor_t input, cnmlTensor_t output)

cnmlCreateSeluOp.

Create a selu activation operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, selu activation operator parameters, input and output Tensors, and pass them to the function to create a selu activation operator.

Before creating the selu activation operator, declare a pointer to the address of the elu activation operator operation struct and pass it to the function together with the required operator parameters to set the operator parameters.

output[i, j, k, l] = (input[i, j, k, l] < 0) ? lambda * alpha * (exp(input[i, j, k, l]) â€ “1) : lambda * input[i, j, k, l]

alpha: const float type, the value being 1.67326324235437728481704299167 17

lambda: const float type, the value being 1.05070098735548049341933498529 46

Shapes of the input and output should be consistent.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is empty
  - The input pointer is empty.
  - The output pointer is empty.

4.121.2 cnmlComputeSeluOpForward_V3

cnmlStatus_t **cnmlComputeSeluOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeSeluOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeSeluOpForward_V4 is recommended to use.

It is used to compute the user-specified selu activation function operator on the MLU.

After creating the selu activation function operator, Input, Output, runtime parameters, computation queue, pass them to the function to It is used to compute the selu activation function operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address that points to the output location.
- [in] op: Input. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.
- [in] compute_forw_param: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.121.3 cnmlComputeSeluOpForward_V4

cnmlStatus_t **cnmlComputeSeluOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeSeluOpForward_V4.

It is used to compute the user-specified selu activation function operator on the MLU.

After creating the selu activation function operator, Input, Output, runtime parameters, computation queue, pass them to the function to It is used to compute the selu activation function operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.122 Shuffle Channel Operation

4.122.1 cnmlCreateShuffleChannelOp

cnmlStatus_t `cnmlCreateShuffleChannelOp`(cnmlBaseOp_t *op, cnmlTensor_t *input_tensors, cnmlTensor_t *output_tensors, int group)

A function.

Create a shuffle operator to adjust data on channel c to the data evenly distribute on channel c.

Formula

Shuffle the channels according to the input param Group:

N = C / Group

C_out[id] = C_in[id % N * Group + id / N], id in {0, 1, 2, ⋯, C-1}

DataType

MLU270:

input and output DataType are same

input: float16, float32

compute: float16, float32

output: float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. Pointing to the operator.
- `[in] inputs_ptr`: Input. Input data address.
- `[in] outputs_ptr`: Input. Output data address.
- `[in] group`: Input. A shuffle parameter, and the parameter should be able to be divided by the width of channel c (c_sizegroup == 0).

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.122.2 cnmlComputeShuffleChannelOpForward_V3

cnmlStatus_t `cnmlComputeShuffleChannelOpForward_V3`(cnmlBaseOp_t op, void *inputs[], void *outputs[], cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeShuffleChannelOpForward_V4 is recommended to use.

Compute the shuffle operator.

Formula

Shuffle the channels according to the input param Group:

N = C / Group

C_out[id] = C_in[id % N * Group + id / N], id in {0, 1, 2, ⋯, C-1}

DataType

MLU270:

input and output DataType are same

input: float16, float32

compute: float16, float32

output: float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. Pointing to an operator address.
- `[in] inputs`: Input. An address of input data
- `[in] outputs`: Input. An address of output data.

- [in] compute_forw_param: Input. A pointer pointing to a struct address of runtime parameters.
- [in] stream: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.122.3 cnmlComputeShuffleChannelOpForward_V4

cnmlStatus_t **cnmlComputeShuffleChannelOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the shuffle operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensors: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [in] inputs: Input. A pointer array, each element pointing to the MLU address of input data.
- [in] output_tensors: Input. Input MLU tensor array pointer. Pass NULL if not used.
- [out] outputs: Output. A pointer array, each element pointing to the MLU address of output position.
- [in] queue: Input. A computation stream pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.123 Sign Operation

4.123.1 cnmlCreateSignOp

cnmlStatus_t **cnmlCreateSignOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a sign operator. A sign operator is that, y=1 if x>0; y=0 if x=0; y=-1 if x<0.

Then creates a pointer to the base operator address, input output tensor, and introduce them into the function to create a sign operator.

The shapes of Input and output should be exactly the same.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] input_tensor: Input. A 1 to n-dimensional MLU tensor.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.123.2 cnmlComputeSignOpForward_V3

cnmlStatus_t **cnmlComputeSignOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

For computing the user-specified sign operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. An pointer which points to base operators.
- [in] input: Input. An MLU address pointing to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.123.3 cnmlComputeSignOpForward_V4

cnmlStatus_t `cnmlComputeSignOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified sign value operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.124 Sin Operation

4.124.1 cnmlCreateSinOp

cnmlStatus_t `cnmlCreateSinOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateSinOp.

Create a Sin operator based on the base operator pointer given by the user. After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Sin operator.

Formula

out[n,c,h,w]=sin(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor.

- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor. The shapes of input and output should be exactly the same.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op is empty.
  - input_tensor is empty.
  - output_tensor is empty.

4.124.2 cnmlComputeSinOpForward_V3

cnmlStatus_t `cnmlComputeSinOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeSinOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeSinOpForward_V4 is recommended to use.

It is used to compute the user-specified sine operator on the MLU.

Formula

out[n,c,h,w]=sin(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input parameter operator pointer op is empty.
  - Input parameter tensor pointer input is empty.
  - Output parameter tensor pointer output is empty.

4.124.3 cnmlComputeSinOpForward_V4

cnmlStatus_t `cnmlComputeSinOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeSinOpForward_V4.

It is used to compute the user-specified sine operator on the MLU.

Formula

out[n,c,h,w]=sin(in[n,c,h,w]);

DataType

MLU270:

input: float16, float32

output: float16, float32

MLU220:

input: float16, float32

output: float16, float32

Scale Limitation

MLU270:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

MLU220:

To avoid precision problem in FP16:

The numerical range of the input data is [-50, 50]. Use the radian measure instead of the degree.

Unlimited in FP32

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.125 Softmax Operation

4.125.1 cnmlCreateSoftmaxOp

cnmlStatus_t `cnmlCreateSoftmaxOp`(cnmlBaseOp_t *op, cnmlDimension_t dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateSoftmaxOpForward is recommended to use.

According to the base operator pointer given by the user, create a probabilistic activation function operator.

Summary

Such as when `input[ni, ci, hi, wi]`, `output[no, co, ho, wo]` and `d` direction is `c`, then `output[n, c, h, w] = softmax(n, c, h, w) = exp(input[n, c, h, w]) / sum_i(exp(input[n, i, h, w]))`

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For better performance, all of the following conditions are met:

- when ‘d’ direction is ‘c’
- The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] dim`: Input. An input tensor that specifies a different dimensional direction to compute softmax. All of the four dimensions N,C,H,W can be specified.
- `[in] input_tensor`: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], supports data of float16 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [no, ho, wo, co], the shape of output is the same as that of input. supports data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The shape of output tensor is different from that of input tensor.

4.125.2 cnmlCreateNdSoftmaxOp

cnmlStatus_t `cnmlCreateNdSoftmaxOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Deprecated. This interface will be deleted in next version and cnmlCreateNdSoftmaxOpForward is recommended to use.

According to the base operator pointer given by the user, create a multi-dimensional probabilistic activation function.

Input and output should have the same shape.

Support 1 to n-dimensional Tensor.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, only supports data of float16 type.
- `[in] dim`: Input. Specify a different dimensional direction to compute softmax. All of the four dimensions N,C,H,W can be specified.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.125.3 cnmlCreateSoftmaxOpForward

This API has the same function as cnmlCreateSoftmaxOp. For detailed information, see cnmlCreateSoftmaxOp.

4.125.4 cnmlCreateNdSoftmaxOpForward

This API has the same function as cnmlCreateNdSoftmaxOp. For detailed information, see cnmlCreateNdSoftmaxOp.

4.125.5 cnmlComputeSoftmaxOpForward_V3

cnmlStatus_t `cnmlComputeSoftmaxOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSoftmaxOpForward_V4 is recommended to use.

Compute the user-specified probabilistic activation function operator.

Summary

Such as when `input[ni, ci, hi, wi]`, `output[no, co, ho, wo]` and `d` direction is `c`, then `output[n, c, h, w] = softmax(n, c, h, w) = exp(input[n, c, h, w]) / sum_i(exp(input[n, i, h, w]))`

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.

4.125.6 cnmlComputeSoftmaxOpForward_V4

cnmlStatus_t `cnmlComputeSoftmaxOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified probabilistic activation function operator.

Summary

Such as when `input[ni, ci, hi, wi]`, `output[no, co, ho, wo]` and `d` direction is `c`, then `output[n, c, h, w] = softmax(n, c, h, w) = exp(input[n, c, h, w]) / sum_i(exp(input[n, i, h, w]))`

DataType

MLU270:

input_type = output_type : float16 or float32

Scale Limitation

MLU270:

Unlimited

For better performance, all of the following conditions are met:

- when ‘d’ direction is ‘c’
- The number of bytes in the C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.125.7 cnmlComputeNdSoftmaxOpForward

cnmlStatus_t `cnmlComputeNdSoftmaxOpForward`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdSoftmaxOpForward_V2 is recommended to use.

For computing a multi-dimensional probabilistic activation function.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] type`: Input. An enumeration constant, representing a task type of runtime.

- [in] stream: Input. A computation stream pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.125.8 cnmlComputeNdSoftmaxOpForward_V2

cnmlStatus_t cnmlComputeNdSoftmaxOpForward_V2(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

For computing a multi-dimensional probabilistic activation function.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.126 Softplus Operation

4.126.1 cnmlCreateSoftplusOp

cnmlStatus_t cnmlCreateSoftplusOp(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user,create a softplus operator, then create a pointer to the base operator address and softplus operator input output tensor, introduce them into the function to create a softplus operator.

SoftPlus can be seen as a smoothed ReLu, it is an analytic function form of a smooth approximation to ReLu.

log (exp (input[n, c, h, w]) + 1.0).

The shapes of input and output should be exactly the same.

Supports MLU220 and MLU270.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] input_tensor: input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Input tensor type is not CNML_TENSOR.

4.126.2 cnmlComputeSoftplusOpForward_V3

cnmlStatus_t `cnmlComputeSoftplusOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSoftplusOpForward_V4 is recommended to use.

Compute the user-specified softplus operator.

After creating softplus operator, input, output and computation stream, introduce them to the function to compute the softplus operator.

Supports MLU220 and MLU270.

Parameters

- `[out] output`: Output. An MLU address that points to the output position.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.

4.126.3 cnmlComputeSoftplusOpForward_V4

cnmlStatus_t `cnmlComputeSoftplusOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified softplus operator.

After creating softplus operator, input, output and computation stream, introduce them to the function to compute the softplus operator.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.127 Softsign Operation

4.127.1 cnmlCreateSoftsignOp

cnmlStatus_t `cnmlCreateSoftsignOp`(cnmlBaseOp_t *op, cnmlTensor_t input, cnmlTensor_t output)

cnmlCreateSoftsignOp.

Create an softsign activation operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output Tensors, and pass them to the function to create a softsign activation operator.

output[i, j, k, l] = input[i, j, k, l] / (1 + fabs(input[i, j, k, l])

Shapes of the input and output must be the same.

Deprecated.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.

- `[in] output`: Input. A 4-dimensional MLU input tensor, of which the shape is [batch, channel, height, width] and only supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input tensor type is not CNML_TENSOR.

4.127.2 cnmlComputeSoftsignOpForward_V3

cnmlStatus_t `cnmlComputeSoftsignOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeSoftsignOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeSoftsignOpForward_V4 is recommended to use.

Compute the user-specified Softsign operator on the MLU.

After creating Softsign operator, input, output and computation queue, introduce them to the function to compute the Softsign operator.

Deprecated.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.127.3 cnmlComputeSoftsignOpForward_V4

cnmlStatus_t `cnmlComputeSoftsignOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeSoftsignOpForward_V4.

Compute the user-specified Softsign operator on the MLU.

After creating Softsign operator, input, output and computation queue, introduce them to the function to compute the Softsign operator.

Deprecated.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.128 Space2batch Operation

4.128.1 cnmlCreateSpace2batchOp

cnmlStatus_t `cnmlCreateSpace2batchOp`(cnmlBaseOp_t *op, int w_block_size, int h_block_size, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateSpace2batchOp.

Create a Space2batch operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, the zoom ratio of the h direction and the w direction, input and output tensors, pass them to the function to create a Space2batch operator.

Formula

this is a IO function, the idea is to reshape the input to another from, in order to accelerate the conv operation with dilation bigger than 1.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] output`: Output. A pointer to the mlu end Tensor
- `[in] op`: Input. A pointer to the base operator
- `[in] input`: Input. A pointer to the mlu end
- `[in] w_block_size[in] Input.`: The zoom ratio representing the move from the w direction to the n direction, wo = wi / w_block_size.
- `[in] h_block_size[in] Input.`: The zoom ratio representing the move from the h direction to the n direction, ho = hi / h_block_size.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op, input, and output are not empty.

4.128.2 cnmlComputeSpace2batchOpForward_V3

cnmlStatus_t `cnmlComputeSpace2batchOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeSpace2batchOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeSpace2batchOpForward_V4 is recommended to use.

It is used to compute the user-specified Space2batch operator on the MLU.

After creating the Space2batch operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Space2batch operator.

Formula

this is a IO function, the idea is to reshape the input to another from, in order to accelerate the conv operation with dilation bigger than 1.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.128.3 cnmlComputeSpace2batchOpForward_V4

cnmlStatus_t **cnmlComputeSpace2batchOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeSpace2batchOpForward_V4.

It is used to compute the user-specified Space2batch operator on the MLU.

After creating the Space2batch operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Space2batch operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.129 Split Operation

4.129.1 cnmlCreateSplitOpParam

cnmlStatus_t **cnmlCreateSplitOpParam**(cnmlSplitOpParam_t *param, int input_num, int output_num, cnmlDimension_t split_mode)

A function.

Create a parameter struct of the split operator, and the number of input tensor in the parameter must be 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer pointing to the parameter struct.
- [in] intput_num: Input. The number of input tensor must be 1.
- [in] output_num: Input. The number of output tensor.
- [in] mode: Input. Specifying the splitting mode: CNML_SPLIT_FEAT, CNML_SPLIT_BATCH, CNML_SPLIT_HEIGHT, and CNML_SPLIT_WIDTH.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.129.2 cnmlDestroySplitOpParam

cnmlStatus_t **cnmlDestroySplitOpParam**(cnmlSplitOpParam_t *param)

A function.

Release the parameter struct of the split operators.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] param: Input. A pointer pointing to the parameter struct.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.129.3 cnmlCreateSplitOp

cnmlStatus_t `cnmlCreateSplitOp`(cnmlBaseOp_t *op, cnmlSplitOpParam_t param, cnmlTensor_t *input_tensors, int input_num, cnmlTensor_t *output_tensors, int output_num)

A function.

Create a split operator.

Formula

if dim =N:

Output split_num * [n / split_num, h, w, c] = input[n, h, w, c]

if dim = C:

Output split_num * [n, h, w, c / split_num] = input[n, h, w, c]

if dim = H:

Output split_num * [n, h / split_num, w, c] = input[n, h, w, c]

if dim = W:

Output split_num * [n, h, w / split_num, c] = input[n, h, w, c]

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For best practices and higher performance, it is recommended that you set either the N dimension or C dimension of the input and output tensors with the following conditions:

- All of the following conditions are met in N dimension:

a. The input and output data is in the middle layer of the network.
b. data_size_input = ni * hi*wI *ci
c. data_size_out = no * ho * wo *co
d. The value of `data_size_out` and `data_size_input` are multiple of 64.

- Set the size of the C-dimension is greater than 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. Pointing to the split operator.
- `[in] param`: Input. Parameters of the split operator.
- `[in] input_ptr`: Input. A 4-dimensional Tensor, supporting data type od float16.
- `[in] input_num`: Input. The number of input tensor must be 1.
- `[in] output_ptr`: Input. A 4-dimensional Tensor, supporting data type od float16.
- `[in] output_num`: Input. The number of output tensor must be 1.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.129.4 cnmlCreateNdSplitOp

cnmlStatus_t `cnmlCreateNdSplitOp`(cnmlBaseOp_t *op, int dim, cnmlTensor_t *input_tensors, int input_num, cnmlTensor_t *output_tensors, int output_num)

A function.

Create an NdSplit operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, NdSplit operator operation parameters, and input and output Tensor, pass them into the fucntion to create an NdSplit operator.

dim starts from 0.

input_num must be 1.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[out] outputs`: Output. A tensor array, and each element is a n-dimensional MLU tensor, supporting data of float16 type.
- `[in] dim`: Input. Specifying dimensions of Split operation.
- `[in] inputs`: Input. A tensor array, and each element is a n-dimensional MLU tensor, supporting data of float16 type.

- `[in] intput_num`: Input. The size of inputs array.
- `[in] output_num`: Input. The size of outputs array.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.129.5 cnmlComputeSplitOpForward_V3

cnmlStatus_t `cnmlComputeSplitOpForward_V3`(cnmlBaseOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSplitOpForward_V4 is recommended to use.

Compute the split operator.

Formula

if dim =N:

Output split_num * [n / split_num, h, w, c] = input[n, h, w, c]

if dim = C:

Output split_num * [n, h, w, c / split_num] = input[n, h, w, c]

if dim = H:

Output split_num * [n, h / split_num, w, c] = input[n, h, w, c]

if dim = W:

Output split_num * [n, h, w / split_num, c] = input[n, h, w, c]

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. Pointing to an operator address.
- `[in] inputs`: Input. The address of input tensor data.
- `[in] intput_num`: Input. The number of input tensor.
- `[in] outputs`: Input. The address of output tensor data.
- `[in] output_num`: Input. The number of output tensor.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] stream`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.129.6 cnmlComputeSplitOpForward_V4

cnmlStatus_t `cnmlComputeSplitOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

A function.

Compute the split operator.

Formula

if dim = N:

Output split_num * [n / split_num, h, w, c] = input[n, h, w, c]

if dim = C:

Output split_num * [n, h, w, c / split_num] = input[n, h, w, c]

if dim = H:

Output split_num * [n, h / split_num, w, c] = input[n, h, w, c]

if dim = W:

Output split_num * [n, h, w / split_num, c] = input[n, h, w, c]

DataType

MLU270:

input: int8, int16, float16, float32, int32

output: same as input

Scale Limitation

MLU270:

Unlimited

Performance Optimization

For best practices and higher performance, it is recommended that you set either the N dimension or C dimension of the input and output tensors with the following conditions:

- All of the following conditions are met in N dimension:

a. The input and output data is in the middle layer of the network.
b. data_size_input = ni * hi*wI *ci
c. data_size_out = no * ho * wo *co
d. The value of `data_size_out` and `data_size_input` are multiple of 64.

- Set the size of the C-dimension is greater than 64.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[in] inputs`: Input. MLU addresses pointing to inputs data.
- `[in] intput_num`: Input. The number of input tensor.
- `[in] output_tensors`: Input. Output MLU tensors pointer. Pass NULL if not used.
- `[out] outputs`: Output. MLU addresses pointing to output position.
- `[in] output_num`: Input. The number of output tensor.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.129.7 cnmlComputeNdSplitOpForward

cnmlStatus_t `cnmlComputeNdSplitOpForward`(cnmlBaseOp_t op, void *inputs[], int input_num, void *outputs[], int output_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdSplitOpForward_V2 is recommended to use.

Compute the operator specified by users on the MLU.

After creating an NdSplit operator, input, output, and computation stream, pass them into the function to compute the NdSplit operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] inputs`: Input. A pointer array, each element pointing to the MLU address of input data.
- `[in] intput_num`: Input. The number of elements in the inputs array.
- `[in] outputs`: Input. A pointer array, each element pointing to the MLU address of output position.
- `[in] output_num`: Input. The number of elements in the outputs array.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`:

4.129.8 cnmlComputeNdSplitOpForward_V2

cnmlStatus_t `cnmlComputeNdSplitOpForward_V2`(cnmlBaseOp_t op, cnmlTensor_t input_tensors[], void *inputs[], int input_num, cnmlTensor_t output_tensors[], void *outputs[], int output_num, cnrtQueue_t queue, void *extra)

A function.

Compute the operator specified by users on the MLU.

After creating an NdSplit operator, input, output, and computation stream, pass them into the function to compute the NdSplit operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[in] inputs`: Input. A pointer array, each element pointing to the MLU address of input data.
- `[in] intput_num`: Input. The number of elements in the inputs array.
- `[in] output_tensors`: Input. Input MLU tensor array pointer. Pass NULL if not used.
- `[out] outputs`: Output. A pointer array, each element pointing to the MLU address of output position.
- `[in] output_num`: Input. The number of elements in the outputs array.
- `[in] queue`: Input. A computation stream pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.130 Sqrt Operation

4.130.1 cnmlCreateSqrtOp

cnmlStatus_t `cnmlCreateSqrtOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a square root operation operator. Perform a square root operation on the input tensor.

The shapes of input and output should be exactly the same. The input data must be positive.

Formula

output[n c h w] = input[n c h w] ^ (1 / 2);

DataType

MLU270:

float16, float32

Scale Limitation

input_dt = output_dt

input[n c h w] > 0

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- `[in] output_tensor`: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The input tensor type is not CNML_TENSOR or CNML_CONST.
  - Reason2 The CPU tensor bound by the bias tensor is null.

4.130.2 cnmlComputeSqrtOpForward_V3

cnmlStatus_t `cnmlComputeSqrtOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSqrtOpForward_V4 is recommended to use.

Compute the user-specified square root operator on the MLU.

Formula

output[n c h w] = input[n c h w] ^ (1 / 2);

DataType

MLU270:

float16, float32

Scale Limitation

input_dt = output_dt

input[n c h w] > 0

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.130.3 cnmlComputeSqrtOpForward_V4

cnmlStatus_t `cnmlComputeSqrtOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the user-specified square root operator on the MLU.

Formula

output[n c h w] = input[n c h w] ^ (1 / 2);

DataType

MLU270:

float16, float32

Scale Limitation

input_dt = output_dt

input[n c h w] > 0

Performance Optimization

The value of C dimension is a multiple of 128.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.131 Square Operation

4.131.1 cnmlCreateSquareOp

cnmlStatus_t **cnmlCreateSquareOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateSquareOp.

Creates a square operator that performs element-wise square.

Perform element-wise square on one input to obtain output.

The formula is as follows:

output[n, c, h, w] = input[n, c, h, w] ^ 2

The shapes of the input and output should be the same. The datatype of the input and output should be the same. Before creating the square operator, declare a pointer pointing to the struct address of the operator.

Formula

output[n c h w] = (input[n c h w]) ^ 2

DataType

input_dt = output_dt

float16, float32

Scale Limitation

Unlimited

Only supports MLU270.

Parameters

- [out] op: Output. A pointer to the square operator you created.
- [in] input_tensor: Input. A 4-D MLU input tensor. The shape of the tensor is [ni, hi, wi, ci]. The data type of this tensor descriptor must be float16 or float32. You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.
- [in] output_tensor: Input. The descriptor of the 4-D MLU output tensor. The shape of the tensor is [no, ho, wo, co] (no = ni, co = ci, ho = hi, wi = wo). The data type of this tensor descriptor must be float16 or float32 type.You need to declare a tensor using the cnmlTensor_t datatype and create the tensor using the cnmlCreateTensor API.

Return Value

- CNML_STATUS_SUCCESS: The function run successfully.
- CNML_STATUS_INVALIDPARAM: One of the following conditions are met:
  - The operator pointer is NULL.
  - The input pointer is NULL.
  - The output tensor is NULL.

4.131.2 cnmlComputeSquareOpForward

cnmlStatus_t **cnmlComputeSquareOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeSquareOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeSquareOpForward_V2 is recommended to use.

Computes the square operation on the MLU.

Formula

output[n c h w] = (input[n c h w]) ^ 2

DataType

input_dt = output_dt

float16, float32

Scale Limitation

Unlimited

Only supports MLU270.

Parameters

- [in] op: Input. A pointer to the square operator you have created.

- [in] input: Input. A pointer to the input data you want to compute.
- [out] output: Output. A pointer to output data after the square operator is applied.
- [in] compute_forw_param: Input. A pointer to the struct address that records the data parallelism and device affinity for runtime.
- [in] queue: Input. A pointer to the queue that is used to implement the computation.

Return Value

- CNML_STATUS_SUCCESS: This function run successfully.
- CNML_STATUS_INVALIDPARAM: One of the following conditions are met:
  - The operator pointer is NULL.
  - The output pointer is NULL.

4.131.3 cnmlComputeSquareOpForward_V2

cnmlStatus_t **cnmlComputeSquareOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeSquareOpForward_V2.

Computes the square operation on the MLU.

Only supports MLU270.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.132 Squared Difference Operation

4.132.1 cnmlCreateSquaredDiffOp

cnmlStatus_t **cnmlCreateSquaredDiffOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a squared difference operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a squared difference operator.

output = (input1 - input2) * (input1 - input2)

The types of the two inputs and one output should be exactly the same.

Formula

c[n c1 h w] = a[n c1 h w] - b[n c2 h w]

c2 can be the same with c2 or be 1.

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 4-dimensional MLU tensor, supporting data of float16/float32 type.
- [in] input_tensor_2: Input. A 4-dimensional MLU tensor, supporting data of float16/float32 type.
- [in] output_tensor: Input. A 4-dimensional MLU tensor, supporting data of float16/float32 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.132.2 cnmlComputeSquaredDiffOpForward_V3

cnmlStatus_t **cnmlComputeSquaredDiffOpForward_V3**(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSquaredDiffOpForward_V4 is recommended to use.

Compute the squared difference operator specified by users on the MLU.

After creating a squared difference operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the squared difference operator.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input_1: Input. An MLU address which points to input data.
- [in] input_2: Input. An MLU address which points to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.132.3 cnmlComputeSquaredDiffOpForward_V4

cnmlStatus_t **cnmlComputeSquaredDiffOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the squared difference operator specified by users on the MLU.

After creating a squared difference operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the squared difference operator.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor1: Input. Input MLU tensor pointer1. Pass NULL if not used.
- [in] input_1: Input. MLU address pointing to input1 data.
- [in] input_tensor2: Input. Input MLU tensor pointer2. Pass NULL if not used.
- [in] input_2: Input. MLU address pointing to input2 data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.133 Std Dev Operation

4.133.1 cnmlCreateStdDevOp

cnmlStatus_t `cnmlCreateStdDevOp`(cnmlBaseOp_t *op, cnmlDimension_t mode, bool unbiased, cnmlTensor_t input, cnmlTensor_t output)

cnmlCreateStdDevOp. Create a Std Dev operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a Std Dev operator.

The functionality of this operator is to compute std of the input Tensor in the selected dimensions to be reduced.

Formula

unbiased is false:

out[1 c h w] = sqrt{mean{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}}, DIM_N

out[n 1 h w] = sqrt{mean{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}}, DIM_C

out[n c 1 w] = sqrt{mean{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}}, DIM_H

out[n c h 1] = sqrt{mean{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}}, DIM_W

unbiased is true:

out[1 c h w] = sqrt{{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}/(n-1)}, DIM_N

out[n 1 h w] = sqrt{{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}/(c-1)}, DIM_C

out[n c 1 w] = sqrt{{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}/(h-1)}, DIM_H

out[n c h 1] = sqrt{{(in[n, c, h, w]-mean(in[i, j, k, l]))^2}/(w-1)}, DIM_W

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] mode`: Input. An enumeration variable, the dimension to be reduced by the user, supporting N, C, H, W.
- `[in] unbiased`: Input. A bool variable, use Bessel correction or not supporting True and False
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 and float32 type.
- `[in] output`: Input. A 4-dimensional tensor, the size of dimensions of which the shape is reduced must be 1; the size of other dimenisions is consistent with that of input, supporting data of float16 and float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: The input tensor type is either CNML_TENSOR or CNML_CONST.

4.133.2 cnmlComputeStdDevOpForward

cnmlStatus_t `cnmlComputeStdDevOpForward`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeStdDevOpForward.

It is used to compute the user-specified Std Dev operator on the MLU.

After creating the Std Dev operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the Std Dev operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.134 Strided Slice Operation

4.134.1 cnmlCreateStridedSliceOpParam

cnmlStatus_t **cnmlCreateStridedSliceOpParam**(cnmlStridedSliceOpParam_t *param, int nb, int cb, int hb, int wb, int ne, int ce, int he, int we, int ns, int cs, int hs, int ws)

cnmlCreateStridedSliceOpParam.

This function is used to create parameters for partial area stride slicing operations. The parameter information includes the start index of each dimension, the end index of each dimension, and the stride of each dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the parameter struct of the initialized variable
- `[in] nb`: Input. n dimension start index
- `[in] cb`: Input. c dimension start index
- `[in] hb`: Input. h dimension start index
- `[in] wb`: Input. w dimension start index
- `[in] ne`: Input. n dimension end index
- `[in] ce`: Input. c dimension end index
- `[in] he`: Input. h dimension end index
- `[in] we`: Input. w dimension end index
- `[in] ns`: Input. n dimension stride index
- `[in] cs`: Input. c dimension stride index
- `[in] hs`: Input. h dimension stride index
- `[in] ws`: Input. w dimension stride index

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - Param is not empty

4.134.2 cnmlCreateNdStridedSliceOpParam

cnmlStatus_t **cnmlCreateNdStridedSliceOpParam**(cnmlNdStridedSliceOpParam_t *param, int dim_num, int begin[], int end[], int stride[])

cnmlCreateNdStridedSliceOpParam.

This function is used to create parameters for partial area stride slicing operations. The parameter information includes the start index of each dimension, the end index of each dimension, and the stride of each dimension. This API is the extension of cnmlCreateStridedSliceOpParam, which can support one to any dimension tensor

Parameters

- `[out] param`: Output. A pointer to the parameter struct of the initialized variable.
- `[in] dim_num`: Input. the length of input Data.
- `[in] degin[]`: Input. the start index of each dimension.
- `[in] end[]`: Input. the end index of each dimension.
- `[in] stride[]`: Input. the stride index of each dimension.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - Param is not empty

4.134.3 cnmlDestroyStridedSliceOpParam

cnmlStatus_t **cnmlDestroyStridedSliceOpParam**(cnmlStridedSliceOpParam_t *param)

cnmlDestroyStridedSliceOpParam.

This function is used to release a pointer to the previously created StridedSlice operation parameter struct.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to the StridedSlice parameter struct

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - Param is not empty
  - The space pointed by param is not released.

4.134.4 cnmlDestroyNdStridedSliceOpParam

cnmlStatus_t **cnmlDestroyNdStridedSliceOpParam**(cnmlNdStridedSliceOpParam_t *param)

cnmlDestroyNdStridedSliceOpParam.

This function is used to release a pointer to the previously created NdStridedSlice operation parameter struct.

Parameters

- [in] param: Input. A pointer to the NdStridedSlice parameter struct

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: One of the following conditions is not satisfied:
  - Param is not empty
  - The space pointed by param is not released.

4.134.5 cnmlCreateStridedSliceOp

cnmlStatus_t **cnmlCreateStridedSliceOp**(cnmlBaseOp_t *op, cnmlStridedSliceOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateStridedSliceOp.

Deprecated. This interface will be deleted in next version and cnmlCreateStridedSliceOpForward is recommended to use.

This function is used to create an operation of slicing by stride in partial area.

Formula

n_len = (ne - nb) > 0 ? ne - nb : nb - ne

h_len = (he - hb) > 0 ? he - hb : hb - he

w_len = (we - wb) > 0 ? we - wb : wb - we

c_len = (ce - cb) > 0 ? ce - cb : cb - ce

n_out = n_len / ns + (n_len % ns > 0 ? 1 : 0)

h_out = h_len / hs + (h_len % hs > 0 ? 1 : 0)

w_out = w_len / ws + (w_len % ws > 0 ? 1 : 0)

c_out = c_len / cs + (c_len % cs > 0 ? 1 : 0)

DataType

MLU270:

float16, float32, int8

Scale Limitation

According to the definition of StridedSlice, the input params should satisfy the following condition.

ns * cs * hs * ws != 0

when ns > 0, (nb >= 0 && ne > nb && n_input >= ne) should be true.

when ns < 0, (ne + 1 >= (-1) * n_input && nb > ne && nb <= -1) should be true.

when cs > 0, (cb >= 0 && ce > cb && c_input >= ce) should be true.

when cs < 0, (ce + 1 >= (-1) * c_input && cb > ce && cb <= -1) should be true.

when hs > 0, (hb >= 0 && he > hb && h_input >= he) should be true.

when hs < 0, (he + 1 >= (-1) * h_input && hb > he && hb <= -1) should be true.

when ws > 0, (wb >= 0 && we > wb && w_input >= we) should be true.

when ws < 0, (we + 1 >= (-1) * w_input && wb > we && wb <= -1) should be true.

Other Limit:

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. returns a pointer to the created base op
- [in] param: Input. A pointer to the StridedSlice parameter struct
- [in] input: Input. A 4-dimensional MLU tensor, having the dimension of [ni,hi,wi,ci], supporting data of float16 type.
- [in] output: Input. A 4-dimensional MLU tensor, having the dimension of [no,ho,wo,co], supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: One of the following conditions is not satisfied:
  - Op is not empty
  - param is not empty

- input is not empty
- output are not empty

4.134.6 cnmlCreateNdStridedSliceOp

cnmlStatus_t `cnmlCreateNdStridedSliceOp`(cnmlBaseOp_t *op, cnmlNdStridedSliceOpParam_t param, cnmlTensor_t input, cnmlTensor_t output)

cnmlCreateNdStridedSliceOp.

This API is the extension of cnmlCreateStridedSliceOp, using for create NdStridedSlice, which can support one to any diemnsion input tensor. We recommend you use the cnmlCreateNdStridedSliceOp interface to create the StridedSlice operator.

Parameters

- `[out] op`: Output. returns a pointer to the created base op.
- `[in] param`: Input. A pointer to the NdStridedSlice parameter struct.
- `[in] input`: Input. A one to any dimensional MLU tensor, supporting data of float16 type.
- `[in] output`: Input. A one to any dimensional MLU tensor, supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: One of the following conditions is not satisfied:
  - Op is not empty
  - param is not empty
  - input is not empty
  - output are not empty

4.134.7 cnmlCreateStridedSliceOpForward

This API has the same function as cnmlCreateStridedSliceOp. For detailed information, see cnmlCreateStridedSliceOp.

4.134.8 cnmlComputeStridedSliceOpForward_V3

cnmlStatus_t `cnmlComputeStridedSliceOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeStridedSliceOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeStridedSliceOpForward_V4 is recommended to use.

It is used to compute the user-specified StridedSlice operator.

After creating the Cast operator, Input, Output, and computation streams, pass them to the function to It is used to compute the Strided-Slice operator.

Formula

n_len = (ne - nb) > 0 ? ne - nb : nb - ne

h_len = (he - hb) > 0 ? he - hb : hb - he

w_len = (we - wb) > 0 ? we - wb : wb - we

c_len = (ce - cb) > 0 ? ce - cb : cb - ce

n_out = n_len / ns + (n_len % ns > 0 ? 1 : 0)

h_out = h_len / hs + (h_len % hs > 0 ? 1 : 0)

w_out = w_len / ws + (w_len % ws > 0 ? 1 : 0)

c_out = c_len / cs + (c_len % cs > 0 ? 1 : 0)

DataType

MLU270:

float16, float32, int8

Scale Limitation

According to the definition of StridedSlice, the input params should satisfy the following condition.

ns * cs * hs * ws != 0

when ns > 0, (nb >= 0 && ne > nb && n_input >= ne) should be true.

when ns < 0, (ne + 1 >= (-1) * n_input && nb > ne && nb <= -1) should be true.

when cs > 0, (cb >= 0 && ce > cb && c_input >= ce) should be true.

when cs < 0, (ce + 1 >= (-1) * c_input && cb > ce && cb <= -1) should be true.

when hs > 0, (hb >= 0 && he > hb && h_input >= he) should be true.

when hs < 0, (he + 1 >= (-1) * h_input && hb > he && hb <= -1) should be true.

when ws > 0, (wb >= 0 && we > wb && w_input >= we) should be true.

when ws < 0, (we + 1 >= (-1) * w_input && wb > we && wb <= -1) should be true.

Other Limit:

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty
  - The output pointer is empty

4.134.9 cnmlComputeStridedSliceOpForward_V4

cnmlStatus_t **cnmlComputeStridedSliceOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeStridedSliceOpForward_V4.

Deprecated. This interface will be deleted in next version and cnmlComputeStridedSliceOpForward_V4 is recommended to use.

It is used to compute the user-specified StridedSlice operator.

After creating the Cast operator, Input, Output, and computation streams, pass them to the function to It is used to compute the StridedSlice operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.134.10 cnmlComputeNdStridedSliceOpForward

cnmlStatus_t **cnmlComputeNdStridedSliceOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeNdStridedSliceOpForward.

Deprecated. This interface will be deleted in next version and cnmlComputeNdStridedSliceOpForward_V2 is recommended to use.

It is used to compute the user-specified NdStridedSlice operator.

After creating the NdStridedSlice operator, Input, Output, and computation stream, pass them to the function to calculate the NdStridedSlice operator.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] stream`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:

  - The operator pointer is empty
  - The output pointer is empty
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - The runtime task type is invalid

4.134.11 cnmlComputeNdStridedSliceOpForward_V2

cnmlStatus_t **cnmlComputeNdStridedSliceOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeNdStridedSliceOpForward.

It is used to compute the user-specified NdStridedSlice operator.

After creating the NdStridedSlice operator, Input, Output, and computation stream, pass them to the function to calculate the NdStridedSlice operator.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.135 Sub Operation

4.135.1 cnmlCreateSubOp

cnmlStatus_t **cnmlCreateSubOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Create a Sub operator according to base operator pointers given by users.

After creating a pointer pointing to base operator address, and input and output Tensor, pass them into the fucntion to create a Sub operator.

The shapes of the two inputs and one output should be exactly the same.

Formula

c[n c h w] = a[n c h w] - b[n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.135.2 cnmlComputeSubOpForward_V3

cnmlStatus_t `cnmlComputeSubOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeSubOpForward_V4 is recommended to use.

Compute the Sub operator specified by users on the MLU.

After creating a Sub operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Sub operator.

Formula

c[n c h w] = a[n c h w] - b[n c h w]

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address which points to input data.
- `[in] input_2`: Input. An MLU address which points to input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.135.3 cnmlComputeSubOpForward_V4

cnmlStatus_t `cnmlComputeSubOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Sub operator specified by users on the MLU.

After creating a Sub operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Sub operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.136 Threshold Operation

4.136.1 cnmlCreateThrsOp

cnmlStatus_t `cnmlCreateThrsOp`(cnmlBaseOp_t *op, float threshold, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateThrsOp.

Create a threshold operator according to the base operator pointer given by the user. The output dimension is the same as the input dimension. Each input element is compared with the threatshold. If the input element is greater than the threshold, the corresponding output element is 1; if the input element is less than the threshold, the corresponding output element is 0.

Formula

y = (x > threshold) ? 1 : 0

DataType

MLU270:

in_type-in_oc_type-out_oc_type-out_type

float16-float16 -float16 -float16

float32-float32 -float32 -float32

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] threshold`: Input. A constant of float16 or float32 type
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 and float32 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi], supporting data of float16 and float32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - op is empty.
  - input_tensor is empty.
  - output_tensor is empty.

4.136.2 cnmlComputeThrsOpForward_V3

cnmlStatus_t `cnmlComputeThrsOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeThrsOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeThrsOpForward_V4 is recommended to use.

It is used to compute the user-specified threshold operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation stream pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Input parameter operator pointer op is empty.
  - Input parameter tensor pointer input is empty.
  - Output parameter tensor pointer output is empty.

4.136.3 cnmlComputeThrsOpForward_V4

cnmlStatus_t `cnmlComputeThrsOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeThrsOpForward_V4.

It is used to compute the user-specified threshold operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.137 Top K Operation

4.137.1 cnmlCreateTopkOp

cnmlStatus_t `cnmlCreateTopkOp`(cnmlBaseOp_t *op, int k, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t index_tensor, cnmlDimension_t dim)

cnmlCreateTopkOp.

The interface creates a TopK operator based on the base operator pointer given by the user.

The operator finds the top K largest number from the user-specified dimension, and outputs the value and the position.

Summary

input[n, c, h, w],and compute max num of k by direction.

such as direction is n, output[k, c, h, w]

such as direction is c, output[n, k, h, w]

other direction is the same condition. k is the size of max num to calculate.

DataType

MLU270:

output_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Performance Optimization

direction c: The smaller the size of n, h, w, and k, the better performance.

direction n h w: The smaller the size of k, the better performance.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tensor[out] :`: Output. 4-dimensional NCHW tensor, supporting data of float type.
- `[out] index_tensor[out] :`: Output. 4-dimensional NCHW tensor, supporting float16, int, and unsigned int types.

- `[in] k[in]` :: Input. Integer, the first k largest numbers to find.
- `[in] input_tensor[in]` :: Input. 4-dimensional NCHW tensor, supporting data of float16 type.
- `[in] dim[in]` :: Input. Enumerated type, the dimension where it is found, including four values: CNML_DIM_C, CNML_DIM_H, CNML_DIM_W, and CNML_DIM_N.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.137.2 cnmlCreateTopkOp_V2

cnmlStatus_t **cnmlCreateTopkOp_V2**(cnmlBaseOp_t *op, int k, cnmlTensor_t input, cnmlTensor_t output, cnmlTensor_t index, cnmlDimension_t ch, cnmlTopkOpMode_t kmode)

cnmlCreateTopkOp_V2.

The interface creates a TopK operator based on the base operator pointer given by the user.

The operator finds the top K largest or the bottom K smallest number from the user-specified dimension, and outputs the value and the position.

Summary

input[n, c, h, w],and compute max num of k by direction.

such as direction is n, output[k, c, h, w]

such as direction is c, output[n, k, h, w]

other directions have the same condition. k is the size of max num to calculate.

DataType

MLU270:

output_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Performance Optimization

direction c: The smaller the size of n, h, w, and k, the better performance.

direction n h w: The smaller the size of k, the better performance.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tensor[out]` :: Output. 4-dimensional NCHW tensor, supporting data of float type.
- `[out] index_tensor[out]` :: Output. 4-dimensional NCHW tensor, supporting float16, int, and unsigned int types.
- `[in] k[in]` :: Input. Integer, the first k largest numbers to find.
- `[in] input_tensor[in]` :: Input. 4-dimensional NCHW tensor, supporting data of float16 type.
- `[in] dim[in]` :: Input. Enumerated type, the dimension where it is found, including four values: CNML_DIM_C, CNML_DIM_H, CNML_DIM_W, and CNML_DIM_N.
- `[in] kmode[in]` :: Input. Enumerated type, specifies whether the largest number or the smallest number to find, including 2 values: CNML_TOPK_OP_MODE_MAX, or CNML_TOPK_OP_MODE_MIN. Only support CNML_TOPK_OP_MODE_MAX temporarily.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.137.3 cnmlCreateNdTopkOp

cnmlStatus_t `cnmlCreateNdTopkOp`(cnmlBaseOp_t *op, int k, cnmlTensor_t input, cnmlTensor_t output, cnmlTensor_t index, int ch, cnmlTopkOpMode_t kmode)

cnmlCreateNdTopkOp.

The interface creates a TopK operator based on the base operator pointer given by the user.

The operator finds the top K largest or the bottom K smallest number from the user-specified dimension, and outputs the value and the position.

Summary

- Support tensor of arbitrary dimensional. Require to have the same input and output shape.
- dim belongs to [0, dim_num-1], and dim_num is the number of dimensions.

DataType

MLU270:

value_data_type = input_data_type:float16, float32

index_data_type:

if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

If find the topk in the direction of c, input_c can’ t exceed 2^23(8388608). When size of c is greater than 130368(fp16) or 65312(fp32), the k should be set less than 18000(fp16) or 16000(fp32).

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tensor[out]` :: Output. A MLU tensor of arbitrary dimension, supporting data of float type.
- `[out] index_tensor[out]` :: Output. A MLU tensor of arbitrary dimension, supporting float16, int, and unsigned int types.
- `[in] k[in]` :: Input. Integer, the first k largest numbers to find.
- `[in] input_tensor[in]` :: Input. A MLU tensor of arbitrary dimension, supporting data of float16 type.
- `[in] dim[in]` :: Input. Enumerated type, the dimension where it is found, counting from 0
- `[in] kmode[in]` :: Input. Enumerated type, specifies whether the largest number or the smallest number to find, including 2 values: CNML_TOPK_OP_MODE_MAX, or CNML_TOPK_OP_MODE_MIN. Only support CNML_TOPK_OP_MODE_MAX temporarily.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.137.4 cnmlComputeTopkOpForward_V3

cnmlStatus_t `cnmlComputeTopkOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, void *index, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeTopkOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeTopkOpForward_V4 is recommended to use.

The function is same as the cnmlComputeTopkOpForward interface.

In addition to the following parameters, the remaining parameters refer to the cnmlComputeTopkOpForward interface.

Summary

input[n, c, h, w],and compute max num of k by direction.

such as direction is n, output[k, c, h, w]

such as direction is c, output[n, k, h, w]

DataType

MLU270:

output_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

at least one c must be processed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output[out]` :: Output. An MLU address that points to the output data.
- `[out] index[out]` :: Output. An MLU address pointing to the index output.
- `[in] op[in]` :: Input. A pointer to the base operator.
- `[in] input[in]` :: Input. An MLU address that points to the input data.
- `[in] stream[in]` :: Input. A computation stream pointer.
- `[in] compute_forw_param[in]` :: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue[in]` :: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.137.5 cnmlComputeTopkOpForward_V4

cnmlStatus_t `cnmlComputeTopkOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t index_tensor, void *index, cnrtQueue_t queue, void *extra)

cnmlComputeTopkOpForward_V4.

The function is same as the cnmlComputeTopkOpForward interface.

In addition to the following parameters, the remaining parameters refer to the cnmlComputeTopkOpForward interface.

Summary

input[n, c, h, w],and compute max num of k by direction.

such as direction is n, output[k, c, h, w]

such as direction is c, output[n, k, h, w]

other direction is the same condition. k is the size of max num to calculate.

DataType

MLU270:

output_data_type = input_data_type:float16, float32

index_data_type:

if direction is c, index_data_type = int32

else if input_data_type = float16 then index_data_type = int16

else if input_data_type = float32 then index_data_type = int32

Scale Limitation

MLU270:

align2num(k, 2*ct_line_num)*2 + align2num(c, 2 * ct_line_num) < 8192 * ct_line_num,

align2num -> make k_pad % (2 *ct_line_num) == 0

input_dt == float16: ct_line_num = 32

input_dt == float32: ct_line_num = 64

Performance Optimization

direction c: The smaller the size of n, h, w, and k, the better performance.

direction n h w: The smaller the size of k, the better performance.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] index_tensor`: Input. Index MLU tensor pointer. Pass NULL if not used.
- `[out] index`: Output. An MLU address pointing to index position.

- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.137.6 cnmlComputeNdTopkOpForward

cnmlStatus_t **cnmlComputeNdTopkOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t index_tensor, void *index, cnrtQueue_t queue, void *extra)

cnmlComputeNdTopkOpForward.

Compute a Topk operation that supports arbitrary dimensions.

Supports MLU270.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] index_tensor: Input. Index MLU tensor pointer. Pass NULL if not used.
- [out] index: Output. An MLU address pointing to index position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.138 Transpose Pro Operation

4.138.1 cnmlCreateNdTransposeOpParam

cnmlStatus_t **cnmlCreateNdTransposeOpParam**(cnmlNdTransposeOpParam_t *param, int dim_order[], int dim_num)

A function.

This function fills in the cnmlNdTransposeOpParam_t struct with the Transpose multi-dimension operation parameters input by users and returns them to the user.

This function allocates param memory. After the usage is completed, the user needs to call cnmlDestroyNdTransposeOpParam to destroy the param parameter at the appropriate time.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] param: Output. A pointer to the address of struct of the NdTranspose operator operation parameter.
- [in] dim_order[]: Input. The dimensions of input_order corresponding to the user_output_order.
- [in] dim_num: Input. The number of dim_order, equal to the number of input tensor and output tensor dimension.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.138.2 cnmlCreateTransposeOpParam

cnmlStatus_t `cnmlCreateTransposeOpParam`(cnmlTransposeOpParam_t *param, cnmlDataOrder_t data_order, int dim_id_0, int dim_id_1, int dim_id_2, int dim_id_3)

A function.

This function fills in the cnmlTransposeOpParam_t struct with the Transpose operation parameters input by users and returns them to the user.

This function allocates param memory. After the usage is completed, the user needs to call cnmlDestroyTransposeOpParam to destroy the param parameter at the appropriate time.

The corresponding input_order of cpuDataOrder is 0123 regardless of the arrangement order. dimId0 - dimId3 is a rearrangement of input_order, for example, input_order is NCHW (0123) actually, dimId0 - dimId3 is 3210 in turn, the actual user_output_order is WHCN(3210). If input_order is actually NHWC (0123), then the user_output_order should be CWHN (3210).

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of struct of the Transpose operator operation parameter.
- `[in] cpu_data_order`: Input. The data of cnmlTransposeOpParam_t type. There are 24 possible arrangement orders of the four dimensions, such as NCHW, while currently only NCHW and NHWC are supported by transpose_pro operator.
- `[in] dim_id_0`: Input. The dimension of input_order corresponding to the zeroth dimension of user_output_order.
- `[in] dim_id_1`: Input. The dimension of input_order corresponding to the first dimension of user_output_order.
- `[in] dim_id_2`: Input. The dimension of input_order corresponding to the second dimension of user_output_order.
- `[in] dim_id_3`: Input. The dimension of input_order corresponding to the third dimension of user_output_order.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.138.3 cnmlDestroyNdTransposeOpParam

cnmlStatus_t `cnmlDestroyNdTransposeOpParam`(cnmlNdTransposeOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the Transpose multi-dimension operator operation parameter is freed.

After the operation of the nd transpose operator is completed, the created struct pointer of the nd transpose operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the NdTranspose operator operation

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.138.4 cnmlDestroyTransposeOpParam

cnmlStatus_t `cnmlDestroyTransposeOpParam`(cnmlTransposeOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of the Transpose operator operation parameter is freed.

After the operation of the transpose operator is completed, the created struct pointer of the Transpose operator operation parameter is freed.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer pointing to the address of struct of the Transpose operator operation parameter.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.138.5 cnmlCreateTransposeProOp

cnmlStatus_t `cnmlCreateTransposeProOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTransposeOpParam_t param)

A function.

This function creates a TransposePro operator based on the basic Operator pointer given by the user.

After a pointer pointing to the base operator, the TransposePro operator operation parameter and the input-output tensor are created, they are introduced into the function to create the TransposePro operator.

Formula

output[n,h,w,c] = input[n,h,w,c]

output[n,w,h,c] = input[n,h,w,c]

output[h,n,w,c] = input[n,h,w,c]

output[h,w,n,c] = input[n,h,w,c]

output[w,n,h,c] = input[n,h,w,c]

output[w,h,n,c] = input[n,h,w,c]

output[n,h,c,w] = input[n,h,w,c]

output[n,c,h,w] = input[n,h,w,c]

output[h,n,c,w] = input[n,h,w,c]

output[h,c,n,w] = input[n,h,w,c]

output[c,n,h,w] = input[n,h,w,c]

output[c,h,n,w] = input[n,h,w,c]

output[n,w,c,h] = input[n,h,w,c]

output[n,c,w,h] = input[n,h,w,c]

output[w,n,c,h] = input[n,h,w,c]

output[w,c,n,h] = input[n,h,w,c]

output[c,n,w,h] = input[n,h,w,c]

output[c,w,n,h] = input[n,h,w,c]

output[h,w,c,n] = input[n,h,w,c]

output[h,c,w,n] = input[n,h,w,c]

output[w,h,c,n] = input[n,h,w,c]

output[w,c,h,n] = input[n,h,w,c]

output[c,h,w,n] = input[n,h,w,c]

output[c,w,n,h] = input[n,h,w,c]

DataType

MLU270:

float16, float32, int8, int16, int32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] input`: Input. A 4-dimensional input tensor with the shape of [ni,ci,hi,wi],supporting data of float16 type.
- `[in] output`: Input. A 4-dimensional output tensor,the shape is [no,co,ho,wo](after transposition, one dimension of the output data should be the same size as the dimension of corresponding input data) ,supporting data of float16 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Operator pointer is null.
  - Input pointer is null.
  - Output tensor is null.

4.138.6 cnmlCreateNdTransposeProOp

cnmlStatus_t `cnmlCreateNdTransposeProOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlNdTransposeOpParam_t param)

A function.

Create a Transpose operator supporting multi-dimension according to base operator pointers given by users.

This operator extens to support multi-dimension based on the Transpose operator.

After creating a pointer pointing to base operator address, dimension of transpose operation, and input and output Tensor, pass them into the fucntion to create a transpose operator supporting multi-dimension.

Before creating a transpose operator, declare a pointer pointing to the struct address of operation parameters of the operator, and pass the pointer and operator parameters required into the function to set operator parameters.

Compute the tensor position along the dimension that users will trans.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer pointing to base operators address.
- `[in] input_tensor`: Input. A multi-dimensional MLU input tensor, supporting data of float16 type.

- `[in] output_tensor`: Input. A multi-dimensional MLU output tensor, supporting data of float16 type.
- `[in] param`: Input. A pointer pointing to the address of struct of the NdTranspose operator param.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

4.138.7 cnmlComputeTransposeProOpForward_V3

cnmlStatus_t `cnmlComputeTransposeProOpForward_V3`(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeTransposeProOpForward_V4 is recommended to use.

Computing user-specified TransposePro operators on MLU.

After the TransposePro operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the TransposePro operator.

Formula

output[n,h,w,c] = input[n,h,w,c]

output[n,w,h,c] = input[n,h,w,c]

output[h,n,w,c] = input[n,h,w,c]

output[h,w,n,c] = input[n,h,w,c]

output[w,n,h,c] = input[n,h,w,c]

output[w,h,n,c] = input[n,h,w,c]

output[n,h,c,w] = input[n,h,w,c]

output[n,c,h,w] = input[n,h,w,c]

output[h,n,c,w] = input[n,h,w,c]

output[h,c,n,w] = input[n,h,w,c]

output[c,n,h,w] = input[n,h,w,c]

output[c,h,n,w] = input[n,h,w,c]

output[n,w,c,h] = input[n,h,w,c]

output[n,c,w,h] = input[n,h,w,c]

output[w,n,c,h] = input[n,h,w,c]

output[w,c,n,h] = input[n,h,w,c]

output[c,n,w,h] = input[n,h,w,c]

output[c,w,n,h] = input[n,h,w,c]

output[h,w,c,n] = input[n,h,w,c]

output[h,c,w,n] = input[n,h,w,c]

output[w,h,c,n] = input[n,h,w,c]

output[w,c,h,n] = input[n,h,w,c]

output[c,h,w,n] = input[n,h,w,c]

output[c,w,n,h] = input[n,h,w,c]

DataType

MLU270:

float16, float32, int8, int16, int32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to the output position.
- `[in] op`: Input. A pointer pointing to the base operator.
- `[in] input`: Input. An MLU address pointing to the input data.
- `[in] compute_forw_param`: Input. A pointer pointing to the address of the struct, which records the degree of data parallelism and device affinity at runtime.
- `[in] queue`: Input. A computational queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.

- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Operator pointer is null.
  - Output pointer is null.

4.138.8 cnmlComputeTransposeProOpForward_V4

cnmlStatus_t **cnmlComputeTransposeProOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified TransposePro operators on MLU.

After the TransposePro operator, input, output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the TransposePro operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.138.9 cnmlComputeNdTransposeProOpForward

cnmlStatus_t **cnmlComputeNdTransposeProOpForward**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeNdTransposeProOpForward_V2 is recommended to use.

Compute the transpose operator supporting multi-dimension on the MLU.

After creating a transpose operator supporting multi-dimension, input, output, and computation queue, pass them into the function to compute the transpose operator supporting multi-dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input: Input. An MLU address which points to input data.
- [in] compute_forw_param: Input. A pointer pointing to the struct address, which records the degree of data parallelism
- [in] queue: Input. A computational queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.138.10 cnmlComputeNdTransposeProOpForward_V2

cnmlStatus_t **cnmlComputeNdTransposeProOpForward_V2**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the transpose operator supporting multi-dimension on the MLU.

After creating a transpose operator supporting multi-dimension, input, output, and computation queue, pass them into the function to compute the transpose operator supporting multi-dimension.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.

- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.139 Tile Operation

4.139.1 cnmlCreateTileOp

cnmlStatus_t **cnmlCreateTileOp**(cnmlBaseOp_t *op, const cnmlTensor_t input_tensor, const cnmlTensor_t output_tensor)

cnmlCreateTileOp.

Create a tile operator based on the base operator pointer given by the user. The output dimension is equal to input dimension multiplied by multiplier.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] input_tensor: Input. A 1-D or more MLU input tensor, of which the shape is [ni, hi, wi,ci], supporting data of float32 type.
- [in] output_tensor: Input. A 1-D or more MLU output tensor, of which the shape is [no, ho, wo, co], supporting data of float32 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is met:
  - op is empty.
  - input_tensor is empty.
  - output_tensor is empty.

4.139.2 cnmlComputeTileOpForward

cnmlStatus_t **cnmlComputeTileOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeTileOpForward.

Compute the Tile operator given by users on the MLU.

After creating Tile operator, input, output, runtime parameters, and computation queue, pass them into the function to compute Tile operator.

Parameters

- [out] output: Output. An MLU address pointing to output position.
- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. input tensor, Pass null if not used.
- [in] input: Input. An MLU address which points to input data.
- [in] output_tensor: Output. output tensor, Pass null if not used.
- [in] queue: Input. A computational queue pointer.
- [in] extra: Input. Reserved for funture use. Pass null if not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The output pointer is null.

4.139.3 cnmlCreateNdTileOp

cnmlStatus_t **cnmlCreateNdTileOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

Create a n-dimensional tile operator according to base operator pointers given by users, which is the extension of tile operator. It supports tensor from one to any dimension. We recommend you use this interface to create the broadcast operator.

a. The shape of input cannot be 0.
b. Each dimension of the output shape is an integral multiple of the corresponding dimension of the input shape.

Note

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address.
- [in] input_tensor: Input. A one to any dimensional MLU tensor, supporting data of float16 type.

- [in] output_tensor: Input. A one to any dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - The operator pointer is null.
  - The input pointer is null.
  - The output tensor is null.

4.139.4 cnmlComputeNdTileOpForward

cnmlStatus_t **cnmlComputeNdTileOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the extensional tile operator on the MLU.

After creating a NdTile operator, input, output, and computation stream, pass them into the function to compute the nd tile operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.140 Tril Operation

4.140.1 cnmlCreateTrilOpParam

cnmlStatus_t **cnmlCreateTrilOpParam**(cnmlTrilOpParam_t *param, int dia)

A function.

According to the pointer given by the user, the function creates a tril operation parameter struct and fills in the struct with the parameters input by the user.

Support only MLU270.

Parameters

- [out] param: Output. A pointer pointing to the address of struct of Tril operator operation parameter.
- [in] dia;: Input. Diagonal of the matrix.

Return Value

- CNML_STATUS_SUCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following condition is met:
  - param is a null pointer

4.140.2 cnmlDestroyTrilOpParam

cnmlStatus_t **cnmlDestroyTrilOpParam**(cnmlTrilOpParam_t *param)

A function.

According to the pointer given by the user, the struct pointer of Tril operator operation parameter is freed.

Support only MLU270.

After the operation of the Tril operator is finished, the created struct pointer of Tril operator operation parameter is freed.

Parameters

- [in] param: Input. A pointer pointing to the address of struct of Tril operator operation parameter.

Return Value

- CNML_STATUS_SUCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - param is null pointer.
  - The content of the pointer pointed to by param has been freed.

4.140.3 cnmlCreateTrilOp

cnmlStatus_t `cnmlCreateTrilOp`(cnmlBaseOp_t *op, cnmlTrilOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the basic Operator pointer given by the user, a Tril operator is created.

The tril operation in the n c direction can be set to get the low triangle matrix.

The input tenosr and output tensor must have the same shape.

Datatype

MLU270:

float16, float32

Formula:

output[no,co,1,1] = (ci - ni <= dia) ? input[ni,ci,1,1] : 0.

Scale Limitation

MLU270:

Unlimited

Support only MLU270.

Parameters

- `[out] op`: Output. A pointer pointing to the address of the base operator.
- `[in] param`: Input. A pointer pointing to the struct of Tril operation.
- `[in] input_tensor`: Input. A 4-dimensional MLU input tensor, the shape is [ni, ci, 1, 1], supporting data of float16 type and float32 type.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor, the shape is [no, co, 1, 1], supporting data of float16 type and float32 type.

Return Value

- `CNML_STATUS_SUCESS`: the function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The type of input tensor is not CNML_TENSOR nor CNML_CONST.
  - The CPU tensor bounded by bias tensor is null.

4.140.4 cnmlComputeTrilOpForward_V4

cnmlStatus_t `cnmlComputeTrilOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computing user-specified Tril operator on MLU.

After the Tril operator, Input. output, parameter at runtime, and computational queue are created, they are introduced into the function to compute the Tril operator.

Datatype

MLU270:

float16, float32

Formula

output[no,co,1,1] = (ni - ci >= dia) ? input[ni,ci,1,1] : 0.

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.141 Unary Select Operation

4.141.1 cnmlCreateUnarySelectOp

cnmlStatus_t `cnmlCreateUnarySelectOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor, cnmlTensor_t count_cnml)

A function.

According to the base operator pointer given by the user, create an unary select operator.

Then creates a pointer to the base operator address, input output tensor, and introduce them into the function to create an unary select operator.

2 input and output should have the same shape, input1 is data to be selected, input2 is an option, the value of input2 can only be filled with 1 or 0.

Formula

output[i] = select(input[k], index[k])

input = [1, 2, 3, 4], [5, 6, 7, 8]

index = [0, 1, 1, 0], [0, 1, 0, 0]

output = [2, 3, 6]

count_output = 3

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

input: 1 c 1 1

index: 1 c 1 1

output:1 c 1 1

Notice: when output count datatype is int16, make sure the input data size(here is c) must small then 32767(2^15-1)

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input_tensor_1`: Input. A four-dimensional MLU input tensor, the shape is [1, ci, 1, 1], supports data of float16 and float32 type.
- `[in] input_tensor_2`: Input. A four-dimensional MLU input tensor, the shape is [1, ci, 1, 1], supports data of float16 and float32 type.
- `[in] output_tensor`: Input. A four-dimensional MLU output tensor, the shape is [1, ci, 1, 1], supports data of float16 and float32 type.
- `[in] count_cnml`: Input. A four-dimensional MLU tensor, the shape is [1, 1, 1, 1], supports data of int16 and int32 type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The input pointer is null.
  - Reason3 The output pointer is null.

4.141.2 cnmlComputeUnarySelectOpForward_V3

cnmlStatus_t `cnmlComputeUnarySelectOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, void *count, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeUnarySelectOpForward_V4 is recommended to use.

For computing the user-specified unary select operator on the MLU.

After creating unary select operator, input, output and computation stream, introduce them to the function to compute the unary select operator.

Formula

output[i] = select(input[k], index[k])

input = [1, 2, 3, 4], [5, 6, 7, 8]

index = [0, 1, 1, 0], [0, 1, 0, 0]

output = [2, 3, 6]

count_output = 3

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

input: 1 c 1 1

index: 1 c 1 1

output:1 c 1 1

Notice: when output count datatype is int16, make sure the input data size(here is c) must small then 32767(2^15-1)

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data.
- `[in] input_2`: Input. An MLU address pointing to input data.
- `[in] count`: Input. An MLU address pointing to input data.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The runtime task type is invalid.

4.141.3 cnmlComputeUnarySelectOpForward_V4

cnmlStatus_t `cnmlComputeUnarySelectOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnmlTensor_t count_tensor, void *count, cnrtQueue_t queue, void *extra)

A function.

For computing the user-specified unary select operator on the MLU.

After creating unary select operator, input, output and computation queue, introduce them to the function to compute the unary select operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] count_tensor`: Input. Count MLU tensor pointer. Pass NULL if not used.
- `[out] count`: Output. An MLU address pointing to count position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.142 Unpool Operation

4.142.1 cnmlCreateUnpoolOpParam

cnmlStatus_t `cnmlCreateUnpoolOpParam`(cnmlUnpoolOpParam_t *param, int window_height, int window_width, int stride_height, int stride_width, cnmlUnpoolMode_t unpool_mode)

cnmlCreateUnpoolOpParam.

The function creates an unpool operator operation parameter struct according to the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] param`: Output. A pointer to the address of the unpool operator operation parameter struct.
- `[in] window_height`: Input. height of the sliding window.
- `[in] window_width`: Input. width of the sliding window.
- `[in] stride_height`: Input. step size in column direction.
- `[in] stride_width`: Input. step size in row direction.
- `[in] unpool_mode`: Input. The unpooling mode defined in cnmlUnpoolMode_t, supported values are CNML_MAXPOOLBP, CNML_AVGPOOLBP, CNML_DILATION_UNPOOL.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.

4.142.2 cnmlDestroyUnpoolOpParam

cnmlStatus_t `cnmlDestroyUnpoolOpParam`(cnmlUnpoolOpParam_t *param)

cnmlDestroyUnpoolOpParam.

Release the unpool operator operation parameter struct pointer according to the pointer given by the user.

After the unpool operator ends, release the created unpool operator operation parameter struct pointer.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] param`: Input. A pointer to the address of the unpool operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - param is a null pointer.
  - The pointer content pointed to by param has been released.

4.142.3 cnmlCreateUnpoolOpParam_V2

cnmlStatus_t `cnmlCreateUnpoolOpParam_V2`(cnmlUnpoolOpParam_t *param, int window_height, int window_width, int stride_height, int stride_width, int pad_left, int pad_right, int pad_up, int pad_down, cnmlUnpoolMode_t unpool_mode, cnmlUnPoolStrategyMode_t padding_mode, bool count_include_pad)

cnmlCreateUnpoolOpParam_V2.

The function creates an unpool operator operation parameter struct according to the pointer given by the user, and fills in the struct with the parameters input by the user.

Scale Limitation DILATION_UNPOOL set param : padding_mode = CNML_UNPOOL_KSAME count_include_pad = false

Parameters

- `[out] param`: Output. A pointer to the address of the unpool operator operation parameter struct.
- `[in] window_height`: Input. height of the sliding window.
- `[in] window_width`: Input. width of the sliding window.
- `[in] stride_height`: Input. step size in column direction.
- `[in] stride_width`: Input. step size in row direction.
- `[in] pad_left`: Input. padding size in left row direction.
- `[in] pad_right`: Input. padding size in right row direction.
- `[in] pad_up`: Input. padding size in up column direction.
- `[in] pad_down`: Input. padding size in down column direction.
- `[in] unpool_mode`: Input. The unpooling mode defined in cnmlUnpoolMode_t, supported values are CNML_MAXPOOLBP, CNML_AVGPOOLBP, CNML_DILATION_UNPOOL.
- `[in] padding_mode`: Input. including SAME, VALID
- `[in] count_include_pad`: Input. judge while pad part join calculate of CNML_AVGPOOLBP

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.

4.142.4 cnmlCreateUnpoolOp

cnmlStatus_t `cnmlCreateUnpoolOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t index_tensor, cnmlTensor_t output_tensor, cnmlUnpoolOpParam_t unpool_param)

cnmlCreateUnpoolOp.

Create an unpool operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, unpool operator parameters, input and output Tensors, pass them to the function to create an unpool operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output_tensor`: Output. A pointer to the mlu end Tensor
- `[in] op`: Input. A pointer to the base operator
- `[in] input_tensor`: Input. A pointer to the mlu end
- `[in] unpool_param`: Input. A pointer to the address of the unpool operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met: Op, input, output, and unpool_param are not empty.

4.142.5 cnmlComputeUnpoolOpForward_V3

cnmlStatus_t `cnmlComputeUnpoolOpForward_V3`(cnmlBaseOp_t op, void *input, void *index, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeUnpoolOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeUnpoolOpForward_V4 is recommended to use.

It is used to compute the user-specified unpool operator on the MLU.

After creating the unpool operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the unpool operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] index`: Input. indicating where the input should be mapped to the original pooling core.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity. at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.142.6 cnmlComputeUnpoolOpForward_V4

cnmlStatus_t `cnmlComputeUnpoolOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t index_tensor, void *index, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeUnpoolOpForward_V4.

It is used to compute the user-specified unpool operator on the MLU.

After creating the unpool operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the unpool operator.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. MLU address pointing to input data.
- `[in] index_tensor`: Input. Index MLU tensor pointer. Pass NULL if not used.
- `[in] index`: Input. MLU address pointing to index data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.

  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.143 Vector2norm Operation

4.143.1 cnmlCreateVector2NormOp

cnmlStatus_t **cnmlCreateVector2NormOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

A function.

According to the base operator pointer given by the user, create a Vector2Norm operator.

After creating a pointer to the base operator address and operator input output tensor, introduce them into the function to create a Vector2Norm operator. The operator can find the L2 norm of input data in channel, height, width dimensions, and each batch will get an output value.

Output is [ni, 1, 1, 1].

Deprecated.

Parameters

- [out] op: Output. A pointer to the base operator address.
- [in] input: Input. A four-dimensional MLU input tensor, the shape is [ni, hi, wi, ci], supports data of float16 type.
- [in] output: Input. A four-dimensional MLU input tensor, the shape is [batch, 1, 1, 1], supports data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Input tensor type is not CNML_TENSOR.

4.143.2 cnmlComputeVector2NormOpForward_V3

cnmlStatus_t **cnmlComputeVector2NormOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeVector2NormOpForward_V4 is recommended to use.

Computethe user-specified Vector2Norm operator.

After creating Vector2Norm operator, input, output and computation stream, introduce them to the function to compute the Vector2Norm operator.

Deprecated.

Parameters

- [out] output: Output. An MLU address that points to the output position.
- [in] op: Output. A pointer to the base operator.
- [in] input: Input. An MLU address that points to the input data.
- [in] compute_forw_param: Input. A pointer to the struct address, which records runtime degree of data parallelism and equipment affinity.
- [in] queue: Input. A computation queue pointer.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions is not met:
  - The operator pointer is null.
  - The output pointer is null.

4.143.3 cnmlComputeVector2NormOpForward_V4

cnmlStatus_t **cnmlComputeVector2NormOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Computethe user-specified Vector2Norm operator.

After creating Vector2Norm operator, input, output and computation queue, introduce them to the function to compute the Vector2Norm operator.

Deprecated.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.144 While Loop Operation

4.144.1 cnmlCreateWhileLoopOp

cnmlStatus_t **cnmlCreateWhileLoopOp**(cnmlBaseOp_t *while_op, cnmlTensor_t cond_input_var)

A function.

The interface is used for creating a while operator.

According to the base operator pointer given by the user, create a while loop operator.

The while operator is used to build a operator that could realize the while control flow function.

The while operator should receive a tensor as the while condition, and would loop when the condition tensor is satisified with the loop requirement(True/False). Each while loop operator should only have one condition input tensor.

DataType

cond_input_var: int8, int16

input datatype is not need to be set

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Output. A pointer pointing to base operator address.

Parameters

- `[in] cond_input_var`: Input. A four-dimensional MLU tensor, the shape is [n, c, h, w] (n = 1, h = 1, w = 1, c = 1), supporting the data of int type.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.144.2 cnmlSetWhileLoopIO

cnmlStatus_t **cnmlSetWhileLoopIO**(cnmlBaseOp_t op, cnmlTensor_t inputs[], int in_num, cnmlTensor_t outputs[], int out_num)

A function.

The interface is used for setting the input and output tensors to a while_loop operator, and the while_loop operator should be already created by cnmlCreateWhileLoopOp function.

DataType

inputs: int8, int16, float16, float32

in_num: int

outputs: same as inputs

out_num: same as in_num

input and output onchip datatypes are not need to be set

Scale Limitation

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[in] op`: Input. A pointer pointing to while_loop operator address.
- `[in] inputs`: Input. A array of four-dimensional MLU input tensors.
- `[in] in_num`: Input. The number of inputs.
- `[in] outputs`: Input. A array of four-dimensional MLU input tensors.

- [in] out_num: Input. The number of outputs.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.144.3 cnmlAddLoopCondOp

cnmlStatus_t **cnmlAddLoopCondOp**(cnmlBaseOp_t while_op, cnmlBaseOp_t loop_cond_op)

A function.

The interface is used for adding the condition operator to a while loop operator, and the while loop operator should be already created by cnmlCreateWhileLoopOp function.

Condition operators are the operators that could determine the computation of the loop requirement, which decide whether to enter a loop branch.

All the condition operators should be added into the controlflow operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] loop_cond_op: Input. A pointer pointing to base operator address.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.144.4 cnmlAddLoopBodyOp

cnmlStatus_t **cnmlAddLoopBodyOp**(cnmlBaseOp_t while_op, cnmlBaseOp_t loop_body_op)

A function.

The interface is used for adding the loop body operator to a while loop operator, and the while loop operator should be already created by cnmlCreateWhileLoopOp function.

Body operators are the operators created by the users and need to be used in the while loop operator.

All the loop body operators should be added into the while loop operator by this interface.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer pointing to base controlflow operator address.
- [in] loop_body_op: Input. A pointer pointing to base operator address.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Op pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.144.5 cnmlAddLoopTensorPair

cnmlStatus_t **cnmlAddLoopTensorPair**(cnmlBaseOp_t while_op, cnmlTensor_t loop_var_init, cnmlTensor_t loop_var_in, cnmlTensor_t loop_next_in, cnmlTensor_t exit_out)

A function.

The interface is used for linking the tensors in a while loop operator, and the while loop operator should be already created by cnmlCreateWhileLoopOp function.

The tensor pairs include the initial input tensor out of a loop, input tensor in a loop, the tensor for next iteration in a loop, and output tensor in a loop. Theses tensors construct a circulation in a loop.

All the tensor pairs in a loop should be added into the while loop operator by this interface.

Formula

loop_val_init = loop_var_in = loop_next_in = exit_out.

DataType

loop_var_init: int8, int16, float16, float32

loop_var_in: same as loop_var_init

loop_next_in: same as loop_var_init

exit_out: same as loop_var_init

input onchip datatype is not need to be set

Scale Limitation

1.all the input tensors datatype should be the same

2.all the input tenosrs shape should be the same

Supports MLU220,MLU270,1M20,and 1M70.

The tensor pairs include the initial input tensor out of a loop, input tensor in a loop, the tensor for next iteration in a loop, and output tensor in a loop. Theses tensors construct a circulation in a loop.

Parameters

- `[in] op`: Input. A pointer pointing to base controlflow operator address.
- `[in] loop_var_init`: Input. A four-dimensional MLU tensor. It' s initial input tensor out of a loop.
- `[in] loop_var_in`: Input. A four-dimensional MLU tensor. It' s input tensor in a loop.
- `[in] loop_next_in`: Input. A four-dimensional MLU tensor. This tensor is the intermediate result of a loop.
- `[in] exit_out`: Input. A four-dimensional MLU output tensor. This tensor is the final result of the loop, it' s input node of successor neural network.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Op pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Task type is invalid at runtime.

4.144.6 cnmlComputeWhileLoopOpForward_V3

cnmlStatus_t `cnmlComputeWhileLoopOpForward_V3`(cnmlBaseOp_t while_op, void *inputs[], int in_num, void *outputs[], int out_num, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

For computing the while loop operator on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] outputs`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operator.
- `[in] inputs`: Input. An MLU address pointing to input data.
- `[in] in_num`: Input. The number of inputs.
- `[in] out_num`: Input. The number of outputs.
- `[in] stream`: Input. A computation stream pointer.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.145 Where Operation

4.145.1 cnmlCreateWhereOp

cnmlStatus_t `cnmlCreateWhereOp`(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t count_tensor)

cnmlCreateWhereOp.

Deprecated. This interface will be deleted in next version and cnmlCreateWhereOp_V2 is recommended to use.

The interface creates a where operator based on the address of the base operator pointer passed by the user.

The approximate operator is used to obtain the coordinates and the values that are true or non-zero in the input data. This operator only support NHWC layout.

Parameters

- `[out] output_tensor[out] :`: Output. 4-dimensional NCHW tensor, non-zero data coordinates, supporting data of int, uint32, and float16 types.
- `[out] count_tensor[out] :`: Output. 4-dimensional NCHW tensor, number of non-zero elements, supporting data of int, uint32, and float16 types.

- `[in] op[in]` :: Input. A pointer to the base operator pointer.
- `[in] input_tensor[in]` :: Input. 4-dimensional NCHW tensor, supporting data of bool and float16 types.

4.145.2 cnmlCreateWhereOp_V2

cnmlStatus_t **cnmlCreateWhereOp_V2**(cnmlBaseOp_t *op, cnmlWhereOpParam_t param, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlTensor_t count_tensor)

cnmlCreateWhereOp_V2.

The interface creates a where operator based on the address of the base operator pointer passed by the user.

The approximate operator is used to obtain the coordinates and the values that are true or non-zero in the input data.

Parameters

- `[out] output_tensor[out]` :: Output. 4-dimensional NCHW tensor, non-zero data coordinates, supporting data of int, uint32, and float16 types.
- `[out] count_tensor[out]` :: Output. 4-dimensional NCHW tensor, number of non-zero elements, supporting data of int, uint32, and float16 types.
- `[in] op[in]` :: Input. A pointer to the base operator pointer.
- `[in] input_tensor[in]` :: Input. 4-dimensional NCHW tensor, supporting data of bool and float16 types.

4.145.3 cnmlCreateWhereOpParam

cnmlStatus_t **cnmlCreateWhereOpParam**(cnmlWhereOpParam_t *param, bool keep_dim)

cnmlCreateWhereOpParam.

The function creates a where operator operation parameter struct according to the pointer given by the user, and fills in the struct with the parameters input by the user.

Supports MLU270.

Parameters

- `[out] param`: Output. A pointer to the address of the where operator operation parameter struct.
- `[in] keep_dim`: Input. bool value, keep_dim, whether to keep output dim

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.

4.145.4 cnmlDestroyWhereOpParam

cnmlStatus_t **cnmlDestroyWhereOpParam**(cnmlWhereOpParam_t *param)

cnmlDestroyWhereOpParam.

Release the where operator operation parameter struct pointer according to the pointer given by the user.

Release the created where operator operation parameter struct pointer after the where operator operation ends.

Supports both MLU100 and MLU270.

Parameters

- `[in] param`: Input. A pointer to the address of the where operator operation parameter struct.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Param is a null pointer.
  - The pointer content pointed to by param has been released.

4.145.5 cnmlComputeWhereOpForward_V3

cnmlStatus_t **cnmlComputeWhereOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, void *count, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeWhereOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeWhereOpForward_V4 is recommended to use.

The interface performs computation on the MLU based on the where operator created by the user.The parameters except the following parameters may refer to cnmlComputeWhereOpForward.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output[out]` :: Output. The address pointing to the MLU coordinate output.
- `[out] count[out]` :: Output. The address pointing to the MLU count ouput.
- `[in] op[in]` :: Input. A pointer to the base operator.
- `[in] input[in]` :: Input. The address pointing to the MLU input data.
- `[in] type[in]` :: Input. An enumeration constant that represents the runtime task type.

- [in] compute_forw_param[in] ::: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- [in] queue[in] ::: Input. A computation queue pointer.

4.145.6 cnmlComputeWhereOpForward_V4

cnmlStatus_t **cnmlComputeWhereOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnmlTensor_t count_tensor, void *count, cnrtQueue_t queue, void *extra)

cnmlComputeWhereOpForward_V4.

The interface performs computation on the MLU based on the where operator created by the user.The parameters except the following parameters may refer to cnmlComputeWhereOpForward.

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] count_tensor: Input. Count MLU tensor pointer. Pass NULL if not used.
- [out] count: Output. An MLU address pointing to count position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.146 Xor Operation

4.146.1 cnmlCreateXorOp

cnmlStatus_t **cnmlCreateXorOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor_1, cnmlTensor_t input_tensor_2, cnmlTensor_t output_tensor)

A function.

Perform element-wise xor operation on two Tensors.

The shapes of two input and one output should be exactly the same.

Formula

c[n c h w] = ((a[n c h w] !=0 && b[n c h w] == 0) || (a[n c h w] == 0 && b[n c h w] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- [out] op: Output. A pointer pointing to base operators address..
- [in] input_tensor_1: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] input_tensor_2: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.
- [in] output_tensor: Input. A 1 to n-dimensional MLU tensor, supporting data of float16 type.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.

4.146.2 cnmlComputeXorOpForward_V3

cnmlStatus_t `cnmlComputeXorOpForward_V3`(cnmlBaseOp_t op, void *input_1, void *input_2, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

A function.

Deprecated. This interface will be deleted in next version and cnmlComputeXorOpForward_V4 is recommended to use.

Compute the Xor operator specified by users on the MLU.

After creating an Xor operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Xor operator.

Formula

c[n c h w] = ((a[n c h w] !=0 && b[n c h w] == 0) || (a[n c h w] == 0 && b[n c h w] != 0)

DataType

MLU270:

float16, float32

Scale Limitation

MLU270:

Unlimited

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] output`: Output. An MLU address pointing to output position.
- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_1`: Input. An MLU address pointing to input data 1.
- `[in] input_2`: Input. An MLU address pointing to input data 2.
- `[in] computue_forw_param`: Input. A pointer pointing to the struct address, which records the degree of data parallelism and device affinity of runtime .
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 the operator pointer is null.
  - Reason2 the output pointer is null.

4.146.3 cnmlComputeXorOpForward_V4

cnmlStatus_t `cnmlComputeXorOpForward_V4`(cnmlBaseOp_t op, cnmlTensor_t input_tensor1, void *input_1, cnmlTensor_t input_tensor2, void *input_2, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

A function.

Compute the Xor operator specified by users on the MLU.

Supports MLU220,MLU270,1M20,and 1M70.

Formula

output[n c h w] = input[n c h w] / sqrt(C) (C is channel dimension size)

DataType

MLU270:

inputDataType float16, float32

outputDataType float16, float32, int16, int8

Scale Limitation

MLU270:

c < 130400

After creating an Xor operator, input, output, runtime parameters, and computation queue, pass them into the function to compute the Xor operator.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor1`: Input. Input MLU tensor pointer1. Pass NULL if not used.
- `[in] input_1`: Input. MLU address pointing to input1 data.
- `[in] input_tensor2`: Input. Input MLU tensor pointer2. Pass NULL if not used.
- `[in] input_2`: Input. MLU address pointing to input2 data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.

- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- `CNML_STATUS_INVALIDARG`: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.147 YUV To RGB Pro Opreation

4.147.1 cnmlCreateYUVtoRGBProOp

cnmlStatus_t **cnmlCreateYUVtoRGBProOp**(cnmlBaseOp_t *op, cnmlTensor_t y_tensor, cnmlTensor_t uv_tensor, cnmlTensor_t output_tensor, cnmlYuvType_t yuv_type, cnmlRgbType_t rgb_type)

cnmlCreateYUVtoRGBProOp.

Create a YUVtoRGB operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, Yuv type, Rgb type, and input and output tensors, pass them to the function to create a YUVtoRGB operator.

Summary

input_y[n, c(1), h, w] + input_uv[n, c(1), h/2, w] compute:

output[n, c(4), h, w] with a certain order(RGB0, BGR0, or ARGB), where

R = 1.164 * Y + 1.596 * V - 222.912

G = 1.164 * Y - 0.392 * U - 0.813 * V + 135.616

B = 1.164 * Y + 2.017 * U - 276.800

A = 0

DataType

Support only UINT8 for both Input/Output

Scale Limitation

H,W must be even

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] y_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, 1, hi, wi] and given by the user.
- `[in] uv_tensor`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, 1, hi/2, wi] and given by the user.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor with a shape of [no, 4, ho, wo].
- `[in] yuv_type`: Input. Yuv type, where CNML_YUV420SP_NV12 and CNML_YUV420SP_NV21 are optional.
- `[in] rgb_type`: Input. Rgb type, where CNML_RGB0, CNML_BGR0 and CNML_ARGB are optional.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - The input input_tensor and output_tensor are empty.

4.147.2 cnmlCreateGrayNormalizeForKcfOp

cnmlStatus_t **cnmlCreateGrayNormalizeForKcfOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor)

cnmlCreateGrayNormalizeForKcfOp.

Create a GrayNormalizeForKcf operator based on the base operator pointer given by the user.

After creating a pointer to the base operator address, input and output tensors, pass them to the function to create a GrayNormalizeForKcf operator.

Summary

output[i] = input[i] / 255.0 - 0.5

DataType

Support only UINT8 for Input and only FLOAT16 for Output

Scale Limitation

H,W must be even

Supports MLU220,MLU270,1M20,and 1M70.

Parameters

- `[out] op`: Output. A pointer to the base operator address.

- [in] input_tensor: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, 1, hi, wi] and given by the user.
- [in] output_tensor: Input. A 4-dimensional MLU output tensor with a shape of [no, 1, ho, wo].

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: (At least one of) the following conditions are not satisfied:
  - The input input_tensor and output_tensor are empty.

4.147.3 cnmlComputeYUVtoRGBProOpForward_V4

cnmlStatus_t **cnmlComputeYUVtoRGBProOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_y_tensor, void *input_y, cnmlTensor_t input_uv_tensor, void *input_uv, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeYUVtoRGBProOpForward_V4.

It is used to compute the user-specified YUVtoRGB operator on the MLU.

After creating the YUVtoRGB operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the YUVtoRGB operator.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_y_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input_y: Input. An MLU address pointing to input_y data.
- [in] input_uv_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input_uv: Input. An MLU address pointing to input_uv data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.147.4 cnmlComputeGrayNormalizeForKcfOpForward

cnmlStatus_t **cnmlComputeGrayNormalizeForKcfOpForward**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeGrayNormalizeForKcfOpForward.

It is used to compute the user-specified GrayNormalizeForKcf operator on the MLU.

After creating the GrayNormalizeForKcf operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the GrayNormalizeForKcf operator.

Parameters

- [in] op: Input. A pointer which points to base operators.
- [in] input_tensor: Input. Input MLU tensor pointer. Pass NULL if not used.
- [in] input: Input. An MLU address pointing to input data.
- [in] output_tensor: Input. Output MLU tensor pointer. Pass NULL if not used.
- [out] output: Output. An MLU address pointing to output position.
- [in] queue: Input. A computation queue pointer.
- [in] extra: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

4.148 YUV To Gray Opreation

4.148.1 cnmlCreateYUVtoGrayOp

cnmlStatus_t **cnmlCreateYUVtoGrayOp**(cnmlBaseOp_t *op, cnmlTensor_t input_tensor, cnmlTensor_t output_tensor, cnmlYuvType_t yuv_type)

cnmlCreateYUVtoGrayOp.

Create a YUVtoGray operator based on the base operator pointer given by the user.

This API is deprecated and will be removed from future release.

After creating a pointer to the base operator address, Yuv type, input and output tensors, pass them to the function to create a YUVtoGray operator.

Supports MLU220 and MLU270.

Parameters

- `[out] op`: Output. A pointer to the base operator address.
- `[in] input`: Input. A 4-dimensional MLU input tensor, of which the shape is [ni, ci, hi, wi] and given by the user.
- `[in] output_tensor`: Input. A 4-dimensional MLU output tensor with a shape of [no, co, ho, wo].
- `[in] yuv_type`: Input. Yuv type, where CNML_YUV420SP_NV12 and CNML_YUV420SP_NV21 are optional.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: (At least one of) the following conditions are not satisfied:
  - The input input_tensor and output_tensor are empty.

4.148.2 cnmlComputeYUVtoGrayOpForward_V3

cnmlStatus_t **cnmlComputeYUVtoGrayOpForward_V3**(cnmlBaseOp_t op, void *input, void *output, cnrtInvokeFuncParam_t *compute_forw_param, cnrtQueue_t queue)

cnmlComputeYUVtoGrayOpForward_V3.

Deprecated. This interface will be deleted in next version and cnmlComputeYUVtoGrayOpForward_V4 is recommended to use.

This API is deprecated and will be removed from future release.

It is used to compute the user-specified YUVtoGray operator on the MLU.

After creating the YUVtoGray operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the YUVtoGray operator.

Supports MLU220 and MLU270.

Parameters

- `[out] output`: Output. An MLU address that points to the output location.
- `[in] op`: Input. A pointer to the base operator.
- `[in] input`: Input. An MLU address that points to the input data.
- `[in] compute_forw_param`: Input. A pointer to the address of the struct, in which the data parallelism and device affinity at runtime are recorded.
- `[in] queue`: Input. A computation queue pointer.

Return Value

- `CNML_STATUS_SUCCESS`: The function ends normally.
- `CNML_STATUS_INVALIDPARAM`: At least one of the following conditions are met:
  - The operator pointer is empty.
  - The output pointer is empty.

4.148.3 cnmlComputeYUVtoGrayOpForward_V4

cnmlStatus_t **cnmlComputeYUVtoGrayOpForward_V4**(cnmlBaseOp_t op, cnmlTensor_t input_tensor, void *input, cnmlTensor_t output_tensor, void *output, cnrtQueue_t queue, void *extra)

cnmlComputeYUVtoGrayOpForward_V4.

It is used to compute the user-specified YUVtoGray operator on the MLU.

This API is deprecated and will be removed from future release.

After creating the YUVtoGray operator, Input, Output, runtime parameters, and computation queue, pass them to the function to It is used to compute the YUVtoGray operator.

Supports MLU220 and MLU270.

Parameters

- `[in] op`: Input. A pointer which points to base operators.
- `[in] input_tensor`: Input. Input MLU tensor pointer. Pass NULL if not used.
- `[in] input`: Input. An MLU address pointing to input data.
- `[in] output_tensor`: Input. Output MLU tensor pointer. Pass NULL if not used.
- `[out] output`: Output. An MLU address pointing to output position.
- `[in] queue`: Input. A computation queue pointer.
- `[in] extra`: Input. Extra parameter pointer. Reserved for future use. Pass NULL is not used.

Return Value

- CNML_STATUS_SUCCESS: The function ends normally.
- CNML_STATUS_INVALIDPARAM: At least one of the following conditions are met:
  - Reason1 The operator pointer is null.
  - Reason2 The output pointer is null.
  - Reason3 The input pointer is null.
- CNML_STATUS_INVALIDARG: At least one of the following conditions are met:
  - Reason1 The task type of runtime is invalid.

5 Error Codes

The error return values and error codes are as follows:

Table 5.1: CNML Error Codes

Return Value	Error Codes	Description
CNML_STATUS_NODEVICE	-1	The device cannot be found.
CNML_STATUS_DOMAINERR	1	A domain error.
CNML_STATUS_INVALIDARG	2	Invalid parameter.
CNML_STATUS_LENGTHERR	3	The length is incorrect.
CNML_STATUS_OUTOFRANGE	4	Out of range.
CNML_STATUS_RANGEERR	5	Out of bounds.
CNML_STATUS_OVERFLOWERR	6	Overflow error.
CNML_STATUS_UNDERFLOWERR	7	Underflow error.
CNML_STATUS_INVALIDPARAM	8	Invalid parameter error.
CNML_STATUS_BADALLOC	9	Memory allocation error.
CNML_STATUS_BADTYPEID	10	TYPE ID error.
CNML_STATUS_BADCAST	11	Type conversion error.
CNML_STATUS_UNSUPPORT	12	Unsupported.