寒武纪性能剖析工具用户手册

版本 *3.13.1*

2021 年 3 月 16 号

Cambricon 寒武纪

目录

Cambricon
寒 武 纪

表格目录

1 版权声明

免责声明

中科寒武纪科技股份有限公司（下称“寒武纪”）不代表、担保（明示、暗示或法定的）或保证本文件所含信息，并明示放弃对可销售性、所有权、不侵犯知识产权或特定目的适用性做出任何和所有暗示担保，且寒武纪不承担因应用或使用任何产品或服务而产生的任何责任。寒武纪不应对因下列原因产生的任何违约、损害赔偿、成本或问题承担任何责任：（1）使用寒武纪产品的任何方式违背本指南；或（2）客户产品设计。

责任限制

在任何情况下，寒武纪都不对因使用或无法使用本指南而导致的任何损害（包括但不限于利润损失、业务中断和信息损失等损害）承担责任，即便寒武纪已被告知可能遭受该等损害。尽管客户可能因任何理由遭受任何损害，根据寒武纪的产品销售条款与条件，寒武纪为本指南所述产品对客户承担的总共和累计责任应受到限制。

信息准确性

本文件提供的信息属于寒武纪所有，且寒武纪保留不经通知随时对本文件信息或对任何产品和服务做出任何更改的权利。本指南所含信息和本指南所引用寒武纪文档的所有其他信息均“按原样”提供。寒武纪不担保信息、文本、图案、链接或本指南内所含其他项目的准确性或完整性。寒武纪可不经通知随时对本指南或本指南所述产品做出更改，但不承诺更新本指南。

本指南列出的性能测试和等级要使用特定芯片或计算机系统或组件来测量。经该等测试，本指南所示结果反映了寒武纪产品的大概性能。系统硬件或软件设计或配置的任何不同会影响实际性能。如上所述，寒武纪不代表、担保或保证本指南所述产品将适用于任何特定用途。寒武纪不代表或担保测试每种产品的所有参数。客户全权承担确保产品适合并适用于客户计划的应用以及对应用程序进行必要测试的责任，以期避免应用程序或产品的默认情况。

客户产品设计的脆弱性会影响寒武纪产品的质量和可靠性并导致超出本指南范围的额外或不同的情况和/或要求。

知识产权通知

寒武纪和寒武纪的标志是中科寒武纪科技股份有限公司在美国和其他国家的商标和/或注册商标。其他公司和产品名称应为与其关联的各自公司的商标。

式复制、重制、修改、出版、上传、发布、传输或分发本指南。除了客户使用本指南信息和产品的权利，根据本指南，寒武纪不授予其他任何明示或暗示的权利或许可。未免疑义，寒武纪不根据任何专利、版权、商标、商业秘密或任何其他寒武纪的知识产权或所有权对客户授予任何（明示或暗示的）权利或许可。

- 版权声明
- © 2021 中科寒武纪科技股份有限公司保留一切权利。

2 前言

2.1 版本记录

表 2.1: 版本记录

文档名称	寒武纪性能剖析工具用户手册
版本号	V 3.13.1
作者	Cambricon
修改日期	2021 年 3 月 16 号

2.2 更新历史

** **V3.13.1**

更新时间: 2021.03.16

更新内容:

- CNPerf 修复 monitor 数据不准的问题。

- **V3.13.0**

  更新时间: 2021.01.15

  更新内容:

  – CNPerf 新增支持监控 memcpy/memset 功能。

  – CNPerf 新增支持通过 cnpx 启停功能。

  – CNPerf monitor/timechart 命令增加显示 device cpu 信息的功能。

  – CNPerf 新增支持提供 kernel name 的功能。

- **V3.12.0**

  更新时间: 2020.11.10

  更新内容:

  - CNPerf layer 命令增强算子理论值预估方法，并提供逻辑 LT 计算量、逻辑 CT 计算量、逻辑 iodma 读写量、逻辑耗时等性能指标。
  - CNPerf layer 命令优化可视化界面展示效果，并提供 SUMMARY 表格折叠展开功能。
  - CNPerf layer 命令支持前融合及 CONV3D 融合情况下的算子理论值计算，并在可视化界面将计算图中发生这两种融合的算子节点删除。
  - CNPerf 支持通过修改配置文件以过滤 cnrt/cndrv 等库的函数。
- **V3.11.0**

  更新时间: 2020.09.21

  更新内容:
  - CNPerf monitor 命令增加显示 jpu 信息的功能。
  - CNPerf 提供支持 cnpx 相关功能。
  - CNPerf show 命令从此版本开始被标记为弃用状态，并将在 CNPerf V3.14.0 版本移除（大约在 3 个月后）。
- **V3.10.1**

  更新时间: 2020.10.15

  更新内容:
  - CNPerf 修复了使用 record –pmu 跟踪 tensorflow 部分程序时会随机发生段错误的问题。
  - CNPerf 修复了部分 pmu 数据不准的问题。
- **V3.10.0**

  更新时间: 2020.09.04

  更新内容:
  - CNPerf 提供多设备支持同时 PMU 采样的功能。
  - replay 命令提供显示 cnrt 的 mem 等函数参数信息功能。
- **V3.9.2**

  更新时间: 2020.08.19

  更新内容:
  - layer 命令导出的 csv 文件中 LT 字段改为显示片上单个 LT 的计算 cycle 数，单位改为 cycles。
  - layer 命令支持解析长时间运行的网络性能。
  - CNPerf timechart 显示所有 memcpy 相关的函数需要单独有一列显示，htod/dtoh/dtod 分开，显示拷贝大小和时间。
  - CNPerf timechart 提供按不同 stream 区分显示每个 stream 的 invoke、memcpy、sync 等操作的功能。
- **V3.9.1**

  更新时间: 2020.08.03

  更新内容:
  - layer 命令新增 CNML 原始图及 CNML 原始图与计算图中算子对应关系展示功能。
  - CNPerf 的 kernel/monitor/show/timechart 命令提供选择支持粗/细粒度显示性能数据的选项。
  - CNPerf timechart 功能提供 LTCTBW 利用率。
  - CNPerf timechart 显示每个 channel 的内存占用情况。

- **V3.8.0**

  更新时间: 2020.06.23

  更新内容:

  - layer 可视化性能数据分析新增网络图 summary 表格，支持展示网络的整体性能。详细内容参见layer 命令可视化性能数据分析 章节。
  - layer 命令新增算子核信息折叠与展开功能，支持用户选择性查看算子某个核信息。详细内容参见layer 命令可视化性能数据分析 章节。
  - layer 命令新增分析离线模型性能功能，详细内容参见layer 命令分析离线模型 章节。
  - record 命令新增 sampler 选项，能够控制是否进行 PMU 数据采样。
  - kernel 命令新增 csv 选项，能够控制是否导出 PMU 数据到 csv 文件。
  - show 命令新增 csv 选项，能够控制是否导出 PMU 数据到 csv 文件。

- **V3.7.0**

  更新时间: 2020.05.28

  更新内容:

  - kernel 命令新增-i 参数，指定跟踪日志的路径。
  - 新增 layer 命令提供可视化性能数据分析功能。
  - replay –tui 命令支持用户交互界面查看函数运行结果。

- **V3.6.0**

  更新时间: 2020.04.09

  更新内容:

  - 更新 monitor 功能，支持多卡选择，使用快捷键 ctrl+c 时正常保存数据以及自定义输出文件的文件名。
  - 新增 timechart 功能，显示函数调用信息、实时功耗和 MLU 利用率等。
  - 优化获取融合模式每层 pmu 数据的功能，可以将获取的数据序列化到 csv 文件，并提供执行时间以及执行核心等信息。
  - 新增控制是否获取 pmu 数据的选项，避免多线程性能分析时因为强制同步导致的性能开销。

- **V3.5.0**

  更新时间: 2020.02.24

  更新内容:

  - 支持部分统计数据导出到 csv 文件。
  - 支持通过配置文件抓取特定函数的参数和返回值。

- **V3.4.0**

  更新时间: 2019.12.31

  更新内容:

  - 提供融合模式下每一层的 pmu 数据，支持多 batch，金字塔融合以及白名单层等功能；
  - 修复获取数据不准的问题；

- **V3.3.0**

  更新时间: 2019.11.28

  更新内容:

  - 在 show 和 kernel 命令中提供 shared/external mem 带宽、功耗和利用率等信息。
  - 新增 monitor 命令，提供旁路指令，查看 JPU/VPU 利用率，以及 VPU 读写的带宽和累计值。
  - 在 show 命令中提供融合模式下每层 IO 量、网络信息和时间等信息。
- **V3.2.2**

  更新时间: 2019.11.8

  更新内容:
  - 提升整体稳定性。
- **V3.2.1**

  更新时间: 2019.10.18

  更新内容:
  - 新增 cnperf-cli show 命令，将相关性能数据实时输出到终端。
- **V3.2.0**

  更新时间: 2019.09.17

  更新内容:
  - 改用 header only json lib 实现配置文件，避免遇到追踪的程序因为 protobuf 版本不一致导致追踪失败的问题。
  - 新增 cnperf-cli kernel 命令，展示更多 PMU 数据。
- **V3.1.0**

  更新时间: 2019.08.13

  更新内容:
  - 支持 MLU270 设备。
- **V3.0.0**

  更新时间: 2019.02.25

  更新内容:
  - 不再需要 root 权限启动。
  - 减少内存占用。
  - 优化日志结构。
- **V2.6.2**

  更新时间: 2018.12.30

  更新内容:
  - 修复跟踪部分多线程程序可能导致的崩溃问题。
- **V2.6.1**

  更新时间: 2018.11.30

  更新内容:
  - 重构图形化分析界面。
  - 支持新增的 CNML 算子函数。
- **V2.6.0**

  更新时间: 2018.10.31

  更新内容:

  - 支持跟踪 CNCC 编译出的 kernel 函数。
  - 支持获取某一个 layer 中所有的 kernel 函数信息。

- **V2.5.0**

  更新时间: 2018.09.27

  更新内容:

  - 获取精确 kernel 函数执行时间。
  - 减少内存占用。
  - 支持新增的 CNRT 内存拷贝函数。

- **V2.4.2**

  更新时间: 2018.08.30

  更新内容:

  - 初始版本。

3 概述

CNPerf（Cambricon Neuware Performance，寒武纪性能剖析工具）是一款针对用户层的性能剖析工具，它以性能事件为基础，可用于 MLU200 系列产品上程序性能瓶颈查找和热点函数定位。

CNPerf 支持的功能如下：

- 精确获得用户程序及部分依赖库中每个函数的执行时间。
- 获取函数调用栈信息。
- 获取并展示 PMU 数据信息。
- 获取并展示 VPU/JPU 利用率，以及 VPU 读写带宽等信息。
- 获取函数调用、实时功耗、MLU 利用率等信息。
- 命令行分析日志文件。
- 可视化网络性能信息。

CNPerf 命令行主要支持以下命令：

- **record**： 记录性能数据并保存到数据文件中。数据文件默认保存在当前目录下的 dltrace_data 文件夹中，该命令可以使用相关参数改变该数据文件所在的默认路径。
- **report**： 在终端上显示目标程序的总体信息。
- **replay**： 在终端上显示所有日志信息。
- **kernel**： 在终端上展示更多 PMU 数据。
- **show**： 不记录日志，直接将数据打印到终端。
- **monitor**: 提供旁路指令，查看 JPU/VPU 利用率，以及 VPU 读写的带宽和累计值。
- **info**： 显示本次 **record** 运行环境相关的信息。
- **layer**: 导出融合模式下每层的 PMU 性能数据，并生成 html 和 json 文件，用于可视化性能分析。
- **timechart**: 生成包含函数调用、实时功耗、MLU 利用率等信息的 json 文件，可使用 chrome://tracing 查看。

具体使用方法请参考命令行接口 章节。

注意:

- MLU220 Edge 平台目前不支持 layer 命令。
- 无法统计以下 BANG C 接口的 ctram 数据：__bang_rotate90、__bang_rotate180、__bang_rotate_270、__bang_mirror、__bang_transpose。

3.1 运行环境

3.1.1 硬件要求

- CPU：Intel i5-4570 或 AMD 同等配置以上。
- Memory：8GB 或以上。

3.1.2 软件要求

CNPERF 可在如下系统中运行：

- Ubuntu 16.04，操作系统版本为 x86-64 位版本。
- Debian 9，操作系统版本为 x86-64 位版本。
- CentOS 7.2，操作系统版本为 x86-64 位版本。
- 依赖：
  1. MLU270 驱动版本必须为 4.9.0 及以上版本。
  2. 其它依赖库在用户执行完驱动安装步骤后会自动安装。

3.1.3 免责声明

CNPerf 只保证能在以上系统中正常运行，其它系统中运行若出现问题，请联系技术支持。

3.2 安装方法

安装方法有两种：

1. 直接在/usr/local/neuware/bin 目录下执行。
2. 将/usr/local/neuware/bin 目录添加到环境变量中，在任意位置执行。

3.3 使用方法

CNPerf 提供命令行供用户使用，命令行的详细使用方法参见命令行接口 章节。

4 命令行接口

本章详细介绍命令行的功能和使用方法。通过使用以下命令，用户可以生成并查看性能数据。

生成并查看性能数据的步骤如下所示：

1. 使用 **record** 命令生成性能数据，并保存到指定文件中。若要执行 **kernel** 命令，则需要添加 **--pmu** 参数，数据文件默认保存到生成的 dltrace_data 文件夹下。
2. 以 dltrace_data 文件夹下的数据文件为输入，执行 **report** 命令查看性能数据信息。
3. 以 dltrace_data 文件夹下的数据文件为输入，执行 **replay** 命令显示函数调用信息。
4. 以 dltrace_data 文件夹下的数据文件为输入，执行 **kernel** 命令显示所有监测到的 PMU 性能信息。
5. 使用 **show** 命令生成性能数据，不记录日志，直接在终端显示设备相关的性能信息。
6. 使用 **layer** 命令导出融合模式下每层的 PMU 性能数据，并生成 html 和 json 文件，用于可视化性能分析。
7. 使用 **monitor** 命令查看 MLU 工作时的各种性能指标，如 VPU 读写带宽等信息。
8. 使用 **timechart** 命令生成包含函数调用、实时功耗、MLU 利用率等信息的 json 文件，可使用 chrome://tracing 查看。

注意:

- CNPerf 跟踪的目标程序需要加-pg 编译。不加-pg 编译的目标程序 CNPerf 无法跟踪其主程序函数调用，举例如下: gcc test.c –pg –o test。
- MLU220 Edge 平台目前不支持 layer 命令。

4.1 命令格式

通用命令格式如下所示：

```
cnperf-cli <command>  -<argument> "program command"
```

CNPerf 的所有命令都采用该格式执行。

通用命令格式参数解释如下表所示：

表 4.1: 通用命令参数解释

参数	参数解释
<command>	CNPerf 支持的命令行，命令的详细介绍参见以下章节内容。
-<argument>	<command> 命令可以携带的参数。每个命令的参数都不同，详细参数信息参见以下章节内容。
“program command”	需要跟踪的目标程序以及目标程序支持的命令及参数。

小技巧: 可以使用 **cnperf-cli -help** 查看 CNPerf 支持的所有命令行及相关参数信息。

4.2 record 命令

record 命令用来生成性能数据，并保存到指定文件中。数据文件默认保存到生成的 dltrace_data 文件夹下，可以通过-o 参数改变数据文件保存的路径。

4.2.1 record 命令参数及解释

表 4.2: record 命令参数及解释

参数	参数解释
-c <card>	指定命令运行的卡号，数值范围应在 0 到该机器所拥有的板卡数目减去 1。当取值为-1 时，表示指定所有卡号。默认值为 0。
-o <path>	指定跟踪日志输出路径。
-t <interval>	设定 PMU 数据采样循环周期，数值范围为 5-1000。默认为 500 毫秒。
- -filter_config_file <file path>	指定 cnperf.json 配置文件的路径，可以是相对路径也可以是绝对路径。
- -pmu	监测 PMU 性能信息。
- -sampler	开启 PMU 数据采样，默认为开启状态。

使用 record 命令注意事项:

1. 使用- -pmu 参数时会大幅度降低被跟踪程序的性能，因此不需要监测 PMU 信息时不建议使用。
2. 使用- -pmu 参数时，被跟踪程序调用的 cnrtInvokeFunction/cnrtInvokeKernel/cnrtInvokeRuntimeContext 系列函数的行为会发生改变，变为同步执行，即相当于函数内部调用了 cnrtSyncQueue。
3. 不能同时使用”- -pmu”和”-c -1”参数，即不支持同时检测所有卡号的 PMU 性能信息。
4. 不能跟踪被抹除（strip）符号的程序。
5. 不能跟踪脚本程序，只能跟踪可执行程序，例如想要跟踪 abc.py，应写为 python abc.py。
6. 不能跟踪多进程程序，如被跟踪程序创建了其他进程，则只有被跟踪程序的函数调用会被记录下来。
7. **record** 命令会默认跟踪 CNRT 和 CNML 中的部分 API 函数，不受配置文件影响。涉及的 API 参见 CNPAPI 开发者手册中 cnpapi_CallbackIdCNRT 和 cnpapi_CallbackIdCNML 两个枚举类型的定义。
8. **record** 命令会默认过滤掉下列函数，不受配置文件影响：
   - std 标准库下的函数
   - boost 库中的函数
   - opencv 库中的函数
   - __gnu_cxx 系列函数
   - operator new
   - allocator 相关函数

4.2.2 record 命令使用方法

- 跟踪 test 可执行文件，使用如下命令:

```
cnperf-cli record test
```

- 若 test 目标程序自带参数，则使用如下命令:

```
cnperf-cli record "test -a -b"
```

- 追踪 Caffe 使用样例，使用命令如下:

```
cnperf-cli record "caffe test -model conv.prototxt -mmode MLU -mcore MLU270"
```

执行完毕后该目录会生成 dltrace_data 文件夹。

4.3 report 命令

report 命令用来查看 dltrace_data 下保存的数据信息。

4.3.1 report 命令参数及解释

表 4.3: report 命令参数及解释

参数	参数解释
- -before_demangle	显示还原前的函数名。
-d <num>	不显示小于指定函数调用栈层数限制的函数信息。
-D <num>	不显示超过指定函数调用栈层数限制的函数信息。
-i <path>	指定跟踪日志路径。
-n <num>	不显示调用次数小于指定值的函数。
-N <num>	不显示调用次数大于指定值的函数。
-t <num>	不显示执行时间小于指定值的函数信息，单位：us。
- -sortby <str>	按照总时间（total）、自身执行时间（self）、调用次数（calls）排序。默认按照总时间排序。
- -csv	将 report 命令提供的统计数据写入 CSV 文件。

4.3.2 report 命令使用方法

1. 进入放置 dltrace_data 的目录，一般是运行 **record** 命令时所在的目录。
2. 输入命令如下：

```
cnperf-cli report
```

3. 显示结果如下所示：

```
PID    Total time   Self time     Calls  Function
=====  ==========  ==========   =======  ========================================
25150  918.722 us  918.722 us       228  cnrtInvokeFunction
       698.348 us  698.348 us       269  cnmlBindConstData
       666.434 us  666.434 us       228  cnrtCreateKernelHeapAllocInfo
       623.647 us  623.647 us       228  cnrtUpdateBarrierInstV3
       ...
         3.204 us    3.204 us         2  cnrtInvokeKernelRecordList_clear
         3.044 us    3.044 us         1  cnmlDestroyConvFirstOpParam
-----  ----------  ----------   -------  ----------------------------------------
```


回显字段详细解释如下表所示：

表 4.4: report 命令回显显字段解释

回显字段	字段解释
PID	被跟踪程序的线程号。
Total time	调用 Function 的累计执行时间，包含子函数调用时间。
Self time	调用 Function 自身的执行时间，不包含子函数调用时间。
Calls	调用函数的次数。
Function	被调用函数名称。

4.4 replay 命令

replay 命令是在终端显示所有监测日志信息，使用方法和 **report** 命令类似。

4.4.1 replay 命令参数及解释

表 4.5: replay 命令参数及解释

参数	参数解释
- -before_demangle	显示还原前的函数名。
-d <num>	不显示小于指定函数调用栈层数限制的函数信息。
-D <num>	不显示超过指定函数调用栈层数限制的函数信息。
-i <path>	指定跟踪日志路径。
- -name <str>	只显示函数名包含指定字符串的函数。
-t <time>	不显示执行时间小于指定值的函数信息，单位：us。
- -tui	交互式 GUI 显示监测日志信息。

4.4.2 replay 命令使用方法

1. 进入放置 dltrace_data 的目录，一般是运行 **record** 命令时所在的目录。
2. 输入如下命令：

```
cnperf-cli replay
```

3. 显示结果如下：

```
PID     TIME                  Duration      Function
======  ====================  ============  ========================
25150   [  0sec,  0ms,  0us]                 cnmlInit{
        [  0sec,  0ms,693us]                  cnrtInit{
        [  0sec,  0ms,878us]                   cnrtGetBoolEnv{
        [  0sec,  0ms,886us]  [  8.196 us]     }  /*cnrtGetBoolEnv*/
        [  0sec,  0ms,889us]                   cnrtGetBoolEnv{
        [  0sec,  0ms,891us]  [  2.041 us]     }  /*cnrtGetBoolEnv*/
        ...
```

回显字段详细解释如下表所示：

表 4.6: replay 命令回显字段解释

回显字段	字段解释
PID	被跟踪程序的线程号。
TIME	Function 的执行时间点。
Duration	该函数的累计执行时间。
Function	被调用函数名称。

replay 命令支持显示每一次 cnrtMalloc 以及 cnrtFree 的大小及地址，支持显示每一次 cnrtMemcpy 的大小和拷贝速率。

4.4.3 replay GUI 使用方法

1. 进入放置 dltrace_data 的目录，一般是运行 **record** 命令时所在的目录。
2. 输入如下命令：

```
cnperf-cli replay --tui
```

3. 显示结果如下：


```
ThreadID    11389     12649     12650     12654     ...
  7.61 ms     > cnmlInit
 13.29 us     - fwrite
  2.08 us     - fwrite
  6.39 us     - fflush
 14.37 us     - fwrite
 11.57 us     - cnrtGetDeviceCount
 ...
```

回显字段详细解释如下表所示：

表 4.7: replay GUI 回显字段解释

回显字段	字段解释
ThreadID	被跟踪程序的线程号。
xx.xx us/ms	累计执行时间，包含内部调用函数时间。
> func_name	被调用函数名称。包含其他函数调用，可展开。
- func_name	被调用函数名称。不含其他函数调用，不可展开。

GUI 操作详细解释如下表所示：

表 4.8: replay GUI 操作解释

操作按键	操作解释
n	选择下一个线程号。
p	选择上一个线程号。
arrow down	选择下一个执行函数。
arrow up	选择上一个执行函数。
arrow left	折叠函数内其他函数调用信息。
arrow right	展开函数内其他函数调用信息。
page down	下翻页显示函数内容。
page up	上翻页显示函数内容。
h	用户按键帮助信息。

4.5 kernel 命令

kernel 命令在终端显示丰富的 PMU 数据，细化性能数据，使用方法和 **report** 命令类似。

4.5.1 kernel 命令参数及解释

表 4.9: kernel 命令参数及解释

参数	参数解释
-i <path>	可选参数，指定跟踪日志路径。
- -csv	将 kernel 命令提供的统计数据写入 CSV 文件。
- -csv - -name	将 kernel 命令提供的统计数据写入 CSV 文件中，并且允许用户指定文件名。
- -detail	展示详细的性能数据。

4.5.2 kernel 命令使用方法

kernel 命令使用方法如下步骤所示：

1. 使用 **record** 时，需要添加 **- -pmu [-c 数字]** 参数，生成 dltrace_data 相关数据，中括号的内容可选。-c 指定运行的卡号，数值范围应为 0 到该机器所拥有的板卡数目减去 1。当取值为-1 时，表示指定所有卡号, 默认值为 0。
2. 进入放置 dltrace_data 的目录，一般是运行 **record** 命令时所在的目录。
3. 输入如下命令：

```
cnperf-cli kernel
```

4. 显示结果如下：

```
Kernels Info:
================================================================================
cnrtInvokeRuntimeContext_V2[1]:
TID     : 31275
Duration: 52.131 ms
--------------------------------------------------------------------------------
Card ID : 0
                    write_bytes              read_bytes                          ␣
↪ write_bytes              read_bytes
```

(下页继续)

(续上页)

```
vpu                 0                   0                   dram                ␣
↪ 168550336                 168550336
tp_cluster          166491392           166491392           tp_cdma             ␣
↪ 0                         0
llc                 168566400           168566400           pcie                ␣
↪ 40192                     40192

                    tagram_hit          tagram_miss
llc                 113676              0

                    tlb_write_access    tlb_read_access     tlb_write_miss      ␣
↪ tlb_read_miss
tp_core             328411              1371527             443                 ␣
↪ 2394
jpu                 0                   0                   0                   ␣
↪ 0
tp_memcore          0                   0                   0                   ␣
↪ 0

                    lt_cycles           ct_cycles           alu_cycles          ␣
↪ io_write_bytes            io_read_bytes
tp_core             342218752           8884271             104055              ␣
↪ 166491392                 364897792


...
```

回显字段详细解释如下表所示：

表 4.10: kernel 命令回显字段解释

回显字段	字段解释
TID	线程号。
Duration	函数在设备上执行的总时长。
Card ID	板卡号。
llc	cluster 上所连接的 LLC 端口模块。
dram	cluster 上所连接的 DDR 端口模块。
tp_core	即为 tensor processor 模块。
tp_cdma	为 tensor processor 中 cdma 模块。
write_bw	各端口写入数据的吞吐量，单位为字节。
read_bw	各端口读取数据的吞吐量，单位为字节。
tagram_hit	LLC 的命中次数。
tagram_miss	LLC 的失败命中次数。
tlb_write_access	运算核心发起的写入请求，在 SMMU 处查询页表的次数。
tlb_read_access	运算核心发起的读取请求，在 SMMU 处查询页表的次数。
tlb_write_miss	tbu_wr_access 中发生缺页的次数。
tlb_read_miss	tbu_rd_access 中发生缺页的次数。
lt_cycles	乘法累加指令的执行次数。
ct_cycles	矢量计算指令的执行次数。
alu_cycles	保持标量计算指令的执行次数。
io_write_bytes	计算单元写入 DDR 数据大小的数量，单位为字节。
io_read_bytes	计算单元读取 DDR 数据大小的数量，单位为字节。
cnrtInvokeRuntimeContext_V2[1]	被跟踪函数。[1] 指 depth 为 1。


```
cnperf-cli kernel --detail
```

5. 显示结果如下：

```
Kernels Info:
================================================================================
cnrtInvokeRuntimeContext_V2[1]:
TID     : 31275
Duration: 52.131 ms
--------------------------------------------------------------------------------
Card ID : 0
                        write_bytes             read_bytes                      ␣
↪  write_bytes            read_bytes
vpu0                    0                       0                       vpu1    ␣
↪  0                      0
vpu2                    0                       0                       vpu3    ␣
↪  0                      0
vpu4                    0                       0                       vpu5    ␣
↪  0                      0
dram_channel0           167067840               314335936               dram_channel1 ␣
↪  521536                 30497856
dram_channel2           491968                  33167872                dram_channel3 ␣
↪  468992                 30407680
tp_cluster0             166491392               388392000               tp_cluster1 ␣
↪  0                      0
tp_cluster2             0                       0                       tp_cluster3 ␣
↪  0                      0
tp_cdma0                0                       0                       tp_cdma1 ␣
↪  0                      0
tp_cdma2                0                       0                       tp_cdma3 ␣
↪  0                      0
llc0                    167084096               316223552               llc1    ␣
↪  521536                 32465792
llc2                    491776                  35125440                llc3    ␣
↪  468992                 32382656
pcie0                   40192                   1322484


                        tagram_hit              tagram_miss                     ␣
↪  tagram_hit             tagram_miss
llc0                    28442                   0                       llc1    ␣
↪  28375                  0
llc2                    28382                   0                       llc3    ␣
↪  28477                  0
```

(下页继续)

(续上页)

```
                      tlb_write_access        tlb_read_access         tlb_write_miss
↪  tlb_read_miss
tp_core_cluster0_core0  328411                  1371527                 443
↪  2394
tp_core_cluster0_core1  0                       0                       0
↪  0
tp_core_cluster0_core2  0                       0                       0
↪  0
tp_core_cluster0_core3  0                       0                       0
↪  0
tp_core_cluster1_core0  0                       0                       0
↪  0
tp_core_cluster1_core1  0                       0                       0
↪  0
tp_core_cluster1_core2  0                       0                       0
↪  0
tp_core_cluster1_core3  0                       0                       0
↪  0
tp_core_cluster2_core0  0                       0                       0
↪  0
tp_core_cluster2_core1  0                       0                       0
↪  0
tp_core_cluster2_core2  0                       0                       0
↪  0
tp_core_cluster2_core3  0                       0                       0
↪  0
tp_core_cluster3_core0  0                       0                       0
↪  0
tp_core_cluster3_core1  0                       0                       0
↪  0
tp_core_cluster3_core2  0                       0                       0
↪  0
tp_core_cluster3_core3  0                       0                       0
↪  0
jpu0                    0                       0                       0
↪  0
jpu1                    0                       0                       0
↪  0
jpu2                    0                       0                       0
↪  0
```

(下页继续)

(续上页)

```
jpu3                    0                    0                    0
↪  0
jpu4                    0                    0                    0
↪  0
jpu5                    0                    0                    0
↪  0
tp_memcore_cluster0     0                    0                    0
↪  0
tp_memcore_cluster1     0                    0                    0
↪  0
tp_memcore_cluster2     0                    0                    0
↪  0
tp_memcore_cluster3     0                    0                    0
↪  0



                        lt_cycles            ct_cycles            alu_cycles
↪  io_write_bytes          io_read_bytes
tp_core_cluster0_core0  342218752            8884271              8884271
↪  166491392               364897792
tp_core_cluster0_core1  0                    0                    0
↪  0                       0
tp_core_cluster0_core2  0                    0                    0
↪  0                       0
tp_core_cluster0_core3  0                    0                    0
↪  0                       0
tp_core_cluster1_core0  0                    0                    0
↪  0                       0
tp_core_cluster1_core1  0                    0                    0
↪  0                       0
tp_core_cluster1_core2  0                    0                    0
↪  0                       0
tp_core_cluster1_core3  0                    0                    0
↪  0                       0
tp_core_cluster2_core0  0                    0                    0
↪  0                       0
tp_core_cluster2_core1  0                    0                    0
↪  0                       0
tp_core_cluster2_core2  0                    0                    0
↪  0                       0
tp_core_cluster2_core3  0                    0                    0
↪  0                       0
```

(下页继续)

(续上页)

```
tp_core_cluster3_core0  0                       0                       0                       ␣
↪  0                       0
tp_core_cluster3_core1  0                       0                       0                       ␣
↪  0                       0
tp_core_cluster3_core2  0                       0                       0                       ␣
↪  0                       0
tp_core_cluster3_core3  0                       0                       0                       ␣
↪  0                       0


...
```

回显字段详细解释如下表所示：

表 4.11: kernel 命令回显字段解释

回显字段	字段解释
tp_core_cluster0_core0	cluster：设备上丛集数。core：设备上核编号。

4.6 show 命令

show 命令不记录日志，直接将性能数据输出到终端上，使用方法和 **record** 命令类似。

4.6.1 show 命令参数及解释

表 4.12: show 命令参数及解释

参数	参数解释
-c <card>	指定命令运行的卡号，数值范围为 0 到该机器所拥有的板卡数目减去 1。默认值为 0。
- -csv	将 show 命令提供的统计数据写入 CSV 文件。
- -csv - -name	将 show 命令提供的统计数据写入 CSV 文件中，并且允许用户指定文件名。
- -detail	展示详细的性能数据。

show 命令使用举例如下所示：

1. 追踪 Caffe 使用样例，使用命令如下：


```
cnperf-cli show "caffe test -model conv.prototxt -mmode MLU -mcore MLU270"
```

2. 显示结果如下：

```
cnrtMemcpy:
                    StartTime           Duration            TID                 Size
↪         Speed
                    0m,3s,568ms,204us   0m,0s,0ms,58us      19201               200704
↪         3.21733 GiB/s


================================================================================
cnrtInvokeRuntimeContext_V2:
                    StartTime           Duration            TID
                    0m,3s,570ms,193us   0m,0s,4ms,755us     19201

                    write_bytes         read_bytes                              write_bytes
↪         read_bytes
 vpu                0                   0                   dram                24445952
↪         63804480
 tp_core            23927808            59544896            tp_cdma             0
↪         0
 llc                24445632            63524032            pcie                9120
↪         489756

                    tagram_hit          tagram_miss
 llc                3627                0

                    tlb_write_access    tlb_read_access     tlb_write_miss      tlb_read_
↪miss
 tp_core            46739               119902              31                  106
 jpu                0                   0                   0                   0

                    lt_cycles           ct_cycles           alu_cycles          io_write_
↪bytes      io_read_bytes
 tp_core            55778304            860005              5654                23927808
↪         58548224


================================================================================
...
```

回显字段详细解释如下表所示：

表 4.13: show 命令回显字段解释

回显字段	字段解释
cnrtMemcpy	被追踪函数的名称。
cnrtInvokeRuntimeContext_V2	被追踪函数的名称。
TID	线程号。
Size	CNRT 中函数内存拷贝大小，单位为字节。
Speed	CNRT 中函数内存拷贝速度。
Duration	函数在设备上执行的总时长。
dram	cluster 上所连接的 DDR 端口模块。
tp_core	即为 tensor processor 模块。
tp_cdma	为 tensor processor 中 cdma 模块。
llc	cluster 上所连接的 LLC 端口模块。
write_bw	各端口写入数据的吞吐量，单位为字节。
read_bw	各端口读取数据的吞吐量，单位为字节。
tagram_hit	LLC 的命中次数。
tagram_miss	LLC 的失败命中次数。
tlb_write_access	运算核心发起的写入请求，在 SMMU 处查询页表的次数。
tlb_read_access	运算核心发起的读取请求，在 SMMU 处查询页表的次数。
tlb_write_miss	tlb_write_access 中发生缺页的次数。
tlb_read_miss	tlb_read_access 中发生缺页的次数。
lt_cycles	乘法累加指令的执行次数。
ct_cycles	矢量计算指令的执行次数。
alu_cycles	标量计算指令的执行次数。
io_write_bytes	计算单元写入 dram 数据大小的数量，单位为字节。
io_read_bytes	计算单元读取 dram 数据大小的数量，单位为字节。

```
cnperf-cli show --detail "caffe test -model conv.prototxt -mmode MLU -mcore MLU270"
```

1. 显示结果如下：

```
cnrtInvokeRuntimeContext_V2:
                        StartTime               Duration                TID
                        0m,33s,8ms,177us        0m,0s,52ms,174us        21077

                        write_bytes             read_bytes                                  ␣
↪    write_bytes              read_bytes
vpu0                    0                       0                       vpu1                ␣
↪    0                        0
vpu2                    0                       0                       vpu3                ␣
↪    0                        0
vpu4                    0                       0                       vpu5                ␣
↪    0                        0
dram_channel0           167312064               313171456               dram_channel1       ␣
↪    762944                   30639424
dram_channel2           732480                  33312192                dram_channel3       ␣
↪    715904                   30552576
tp_cluster0             166491392               388392000               tp_cluster1         ␣
↪    0                        0
tp_cluster2             0                       0                       tp_cluster3         ␣
↪    0                        0
tp_cdma0                0                       0                       tp_cdma1            ␣
↪    0                        0
tp_cdma2                0                       0                       tp_cdma3            ␣
↪    0                        0
llc0                    167082688               315053568               llc1                ␣
↪    500800                   32608960
llc2                    471360                  35281984                llc3                ␣
↪    453760                   32535168
pcie0                   63872                   1949784

                        tagram_hit              tagram_miss                                 ␣
↪    tagram_hit               tagram_miss
llc0                    28444                   480                     llc1                ␣
↪    28375                    512
llc2                    28382                   510                     llc3                ␣
↪    28477                    512
```

(下页继续)

(续上页)

```
                        tlb_write_access        tlb_read_access         tlb_write_miss
↪    tlb_read_miss
tp_core_cluster0_core0  328411                  1371527                 443
↪    2394
tp_core_cluster0_core1  0                       0                       0
↪    0
tp_core_cluster0_core2  0                       0                       0
↪    0
tp_core_cluster0_core3  0                       0                       0
↪    0
tp_core_cluster1_core0  0                       0                       0
↪    0
tp_core_cluster1_core1  0                       0                       0
↪    0
tp_core_cluster1_core2  0                       0                       0
↪    0
tp_core_cluster1_core3  0                       0                       0
↪    0
tp_core_cluster2_core0  0                       0                       0
↪    0
tp_core_cluster2_core1  0                       0                       0
↪    0
tp_core_cluster2_core2  0                       0                       0
↪    0
tp_core_cluster2_core3  0                       0                       0
↪    0
tp_core_cluster3_core0  0                       0                       0
↪    0
tp_core_cluster3_core1  0                       0                       0
↪    0
tp_core_cluster3_core2  0                       0                       0
↪    0
tp_core_cluster3_core3  0                       0                       0
↪    0
jpu0                    0                       0                       0
↪    0
jpu1                    0                       0                       0
↪    0
jpu2                    0                       0                       0
↪    0
jpu3                    0                       0                       0
↪    0
```

(下页继续)

(续上页)

```
jpu4                    0                    0                    0                    ␣
↪    0
jpu5                    0                    0                    0                    ␣
↪    0
tp_memcore_cluster0     0                    0                    0                    ␣
↪    0
tp_memcore_cluster1     0                    0                    0                    ␣
↪    0
tp_memcore_cluster2     0                    0                    0                    ␣
↪    0
tp_memcore_cluster3     0                    0                    0                    ␣
↪    0

                        lt_cycles            ct_cycles            alu_cycles           ␣
↪    io_write_bytes           io_read_bytes
tp_core_cluster0_core0  342218752            8884271              8884271              ␣
↪    166491392                364897792
tp_core_cluster0_core1  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster0_core2  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster0_core3  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster1_core0  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster1_core1  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster1_core2  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster1_core3  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster2_core0  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster2_core1  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster2_core2  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster2_core3  0                    0                    0                    ␣
↪    0                        0
tp_core_cluster3_core0  0                    0                    0                    ␣
↪    0                        0
```

(下页继续)

(续上页)

```
 tp_core_cluster3_core1  0                       0                       0                       ␣
↪     0                       0
 tp_core_cluster3_core2  0                       0                       0                       ␣
↪     0                       0
 tp_core_cluster3_core3  0                       0                       0                       ␣
↪     0                       0


================================================================================
...
```

回显字段详细解释如下表所示：

表 4.14: show 命令回显字段解释

回显字段	字段解释
tp_core_cluster0_core0	cluster：设备上丛集数。core：设备上核编号。

4.7 monitor 命令

monitor 命令用于查看 MLU 工作时的性能指标，如 VPU 读写带宽等信息。

4.7.1 monitor 命令参数及解释

表 4.15: monitor 命令参数及解释

参数	参数解释
-c <card>	指定命令运行的卡号，数值范围应为 0 到该机器所拥有的板卡数目减去 1。默认值为 0。
- -csv	将监测到的性能数据直接记录到 CSV 文件中，而不是输出到屏幕上。
- -csv - -name	将监测到的性能数据直接记录到 CSV 文件中，并且允许用户指定文件名。
- -detail	展示详细的性能数据。
-r <count>	设定循环次数，数值范围应在 1 到 60000，默认次数为 1000。
-t <interval>	设定循环周期，数值范围应在 5 到 1000，默认为 500 毫秒。

4.7.2 monitor 命令使用方法

1. 在任意目录输入如下命令

```
cnperf-cli monitor
```

2. 显示结果如下：

```
CNPERF Monitor:


  CLOCK_MONOTONIC time(ns) :  [767852s,333ms,144us,132ns]
  execution time(ns) :  [0s,589ms,839us,826ns]


  DEV Information:
  power(W)                 : 21
  board_temperature(C)     : 45
  ipu_utils(%)             : 0
  jpu_utils(%)             : 0
  vpu_utils(%)             : 0
  device_cpu_utils(%)      : 51
  footprint_channel(MB)    : 2758
```

(下页继续)

(续上页)

```
  Monitor Increment Information(MB/s):
                          write_bytes             read_bytes                  ␣
↪        write_bytes             read_bytes
  vpu                     0                       0                       dram ␣
↪        29.5734                 29.5734
  tp_cluster              0                       0                       tp_cdma ␣
↪        0                       0
  llc                     30.8801                 30.8801                 pcie ␣
↪        0.10653                 0.10653

  Monitor Total Information(MB):
                          write_bytes             read_bytes                  ␣
↪        write_bytes             read_bytes
  vpu                     0                       0                       dram ␣
↪        17.4435                 17.4435
  tp_cluster              0                       0                       tp_cdma ␣
↪        0                       0
  llc                     18.2143                 18.2143                 pcie ␣
↪        0.0628357               0.0628357

  LLC Increment Information:
                          tagram_hit              tagram_miss
  llc                     0                       0

  LLC Total Information:
                          tagram_hit              tagram_miss
  llc                     0                       0

  SMMU Increment Information:
                          tlb_write_access        tlb_read_access         tlb_write_miss ␣
↪        tlb_read_miss
  tp_core                 0                       0                       0 ␣
↪        0
  jpu                     0                       0                       0 ␣
↪        0
  tp_memcore              0                       0                       0 ␣
↪        0

  SMMU Total Information:
                          tlb_write_access        tlb_read_access         tlb_write_miss ␣
↪        tlb_read_miss
```

(下页继续)

(续上页)

```
  tp_core                 0                       0                       0
↪         0
  jpu                     0                       0                       0
↪         0
  tp_memcore              0                       0                       0
↪         0

  IPU Increment Information:
                          lt_cycles               ct_cycles               alu_cycles
↪         io_write_bytes          io_read_bytes
  tp_core                 0                       0                       0
↪         0                       0

  IPU Total Information:
                          lt_cycles               ct_cycles               alu_cycles
↪         io_write_bytes          io_read_bytes
  tp_core                 0                       0                       0
↪         0                       0

  ..............................................................................
```

回显字段详细解释如下表所示：

表 4.16: monitor 命令回显字段解释

回显字段	字段解释
CLOCK_MONOTONIC time(ns)	系统当前时间。
execution time(ns)	当前刷新数据的时间戳（相对于程序启动时刻来说）。
board_temperature(C)	板卡温度。
ipu_utils(%)	IPU 的利用率。
jpu_utils(%)	JPU 的利用率。
vpu_utils(%)	VPU 的利用率。
device_cpu_utils(%)	DEVICE CPU 的利用率。
scaler_utils(%)	scaler 的利用率。
write_bw	各端口写入数据的吞吐量，单位为字节。
read_bw	各端口读取数据的吞吐量，单位为字节。
tagram_hit	LLC 的命中次数。
tagram_miss	LLC 的失败命中次数。
tlb_write_access	运算核心发起的写入请求，在 SMMU 处查询页表的次数。
tlb_read_access	运算核心发起的读取请求，在 SMMU 处查询页表的次数。
tlb_write_miss	tlb_write_access 中发生缺页的次数。
tlb_read_miss	tlb_read_access 中发生缺页的次数。
lt_cycles	乘积累加单元的运算周期数。
ct_cycles	向量运算单元的运算周期数。
alu_cycles	标量运算单元的运算周期数。
io_write_bytes	计算单元写入 dram 数据大小的数量，单位为字节。
io_read_bytes	计算单元读取 dram 数据大小的数量，单位为字节。
Monitor Increment Information(MB/s)	当前周期内数据的读写带宽，单位为 MB/s。
Monitor Total Information(MB)	当前硬件上统计到的读写数据量，单位为 MB。

```
cnperf-cli monitor --detail
```

1. 显示结果如下：

```
CNPERF Monitor:
card index number: 0
card type: MLU270
CLOCK_MONOTONIC time(ns) :  [768313s,203ms,833us,727ns]
execution time(ns) :  [0s,591ms,782us,553ns]

DEV Information:
power(W)                 : 21
board_temperature(C)     : 44
ipu_utils(%)             : 0 0 0 0 0 0 0 0
                         : 0 0 0 0 0 0 0 0
jpu_utils(%)             : 0 0 0 0 0 0
vpu_utils(%)             : 0 0 0 0 0 0
device_cpu_utils(%)      : 0 0 100 100
footprint_channel(MB)    : 690 689 690 689

Monitor Increment Information(MB/s):
                         write_bytes              read_bytes                          ␣
↪        write_bytes              read_bytes
vpu0                     0                        0                       vpu1        ␣
↪        0                        0
vpu2                     0                        0                       vpu3        ␣
↪        0                        0
vpu4                     0                        0                       vpu5        ␣
↪        0                        0
dram_channel0            8.15304                  364.179                 dram_channel1 ␣
↪        7.49853                  3.75535
dram_channel2            6.97036                  3.03473                 dram_channel3 ␣
↪        6.67941                  2.9714
tp_cluster0              0                        0                       tp_cluster1 ␣
↪        0                        0
tp_cluster2              0                        0                       tp_cluster3 ␣
↪        0                        0
tp_cdma0                 0                        0                       tp_cdma1    ␣
↪        0                        0
tp_cdma2                 0                        0                       tp_cdma3    ␣
↪        0                        0
```

(下页继续)

(续上页)

```
 llc0                    8.53063                 377.477                 llc1
↪        7.85869                 3.78629
 llc2                    7.3233                  3.16427                 llc3
↪        6.98119                 3.04112
 pcie0                   0.0958666               0.99916

 Monitor Total Information(MB):
                         write_bytes             read_bytes
↪        write_bytes             read_bytes
 vpu0                    0                       0                       vpu1
↪        0                       0
 vpu2                    0                       0                       vpu3
↪        0                       0
 vpu4                    0                       0                       vpu5
↪        0                       0
 dram_channel0           4.82483                 215.515                 dram_channel1
↪        4.4375                  2.22235
 dram_channel2           4.12494                 1.7959                  dram_channel3
↪        3.95276                 1.75842
 tp_cluster0             0                       0                       tp_cluster1
↪        0                       0
 tp_cluster2             0                       0                       tp_cluster3
↪        0                       0
 tp_cdma0                0                       0                       tp_cdma1
↪        0                       0
 tp_cdma2                0                       0                       tp_cdma3
↪        0                       0
 llc0                    5.04828                 223.385                 llc1
↪        4.65063                 2.24066
 llc2                    4.3338                  1.87256                 llc3
↪        4.13135                 1.79968
 pcie0                   0.0567322               0.591286

 LLC Increment Information:
                         tagram_hit              tagram_miss
↪        tagram_hit              tagram_miss
 llc0                    0                       0                       llc1
↪        0                       0
 llc2                    0                       0                       llc3
↪        0                       0
```

(下页继续)

(续上页)

```
 LLC Total Information:
                          tagram_hit              tagram_miss                    ␣
↪        tagram_hit              tagram_miss
 llc0                     0                       0                       llc1   ␣
↪        0                       0
 llc2                     0                       0                       llc3   ␣
↪        0                       0

 SMMU Increment Information:
                          tlb_write_access        tlb_read_access         tlb_write_miss ␣
↪        tlb_read_miss
 tp_core_cluster0_core0   0                       0                       0      ␣
↪        0
 tp_core_cluster0_core1   0                       0                       0      ␣
↪        0
 tp_core_cluster0_core2   0                       0                       0      ␣
↪        0
 tp_core_cluster0_core3   0                       0                       0      ␣
↪        0
 tp_core_cluster1_core0   0                       0                       0      ␣
↪        0
 tp_core_cluster1_core1   0                       0                       0      ␣
↪        0
 tp_core_cluster1_core2   0                       0                       0      ␣
↪        0
 tp_core_cluster1_core3   0                       0                       0      ␣
↪        0
 tp_core_cluster2_core0   0                       0                       0      ␣
↪        0
 tp_core_cluster2_core1   0                       0                       0      ␣
↪        0
 tp_core_cluster2_core2   0                       0                       0      ␣
↪        0
 tp_core_cluster2_core3   0                       0                       0      ␣
↪        0
 tp_core_cluster3_core0   0                       0                       0      ␣
↪        0
 tp_core_cluster3_core1   0                       0                       0      ␣
↪        0
 tp_core_cluster3_core2   0                       0                       0      ␣
↪        0
```

(下页继续)

(续上页)

```
  tp_core_cluster3_core3  0                       0                       0
↪        0
  jpu0                    0                       0                       0
↪        0
  jpu1                    0                       0                       0
↪        0
  jpu2                    0                       0                       0
↪        0
  jpu3                    0                       0                       0
↪        0
  jpu4                    0                       0                       0
↪        0
  jpu5                    0                       0                       0
↪        0
  tp_memcore_cluster0     0                       0                       0
↪        0
  tp_memcore_cluster1     0                       0                       0
↪        0
  tp_memcore_cluster2     0                       0                       0
↪        0
  tp_memcore_cluster3     0                       0                       0
↪        0

  SMMU Total Information:
                          tlb_write_access        tlb_read_access         tlb_write_miss
↪        tlb_read_miss
  tp_core_cluster0_core0  0                       0                       0
↪        0
  tp_core_cluster0_core1  0                       0                       0
↪        0
  tp_core_cluster0_core2  0                       0                       0
↪        0
  tp_core_cluster0_core3  0                       0                       0
↪        0
  tp_core_cluster1_core0  0                       0                       0
↪        0
  tp_core_cluster1_core1  0                       0                       0
↪        0
  tp_core_cluster1_core2  0                       0                       0
↪        0
  tp_core_cluster1_core3  0                       0                       0
↪        0
```

(下页继续)

(续上页)

```
  tp_core_cluster2_core0  0                       0                       0
↪        0
  tp_core_cluster2_core1  0                       0                       0
↪        0
  tp_core_cluster2_core2  0                       0                       0
↪        0
  tp_core_cluster2_core3  0                       0                       0
↪        0
  tp_core_cluster3_core0  0                       0                       0
↪        0
  tp_core_cluster3_core1  0                       0                       0
↪        0
  tp_core_cluster3_core2  0                       0                       0
↪        0
  tp_core_cluster3_core3  0                       0                       0
↪        0
  jpu0                    0                       0                       0
↪        0
  jpu1                    0                       0                       0
↪        0
  jpu2                    0                       0                       0
↪        0
  jpu3                    0                       0                       0
↪        0
  jpu4                    0                       0                       0
↪        0
  jpu5                    0                       0                       0
↪        0
  tp_memcore_cluster0     0                       0                       0
↪        0
  tp_memcore_cluster1     0                       0                       0
↪        0
  tp_memcore_cluster2     0                       0                       0
↪        0
  tp_memcore_cluster3     0                       0                       0
↪        0

  IPU Increment Information:
                          lt_cycles               ct_cycles               alu_cycles
↪        io_write_bytes          io_read_bytes
  tp_core_cluster0_core0  0                       0                       0
↪        0                       0
```

(下页继续)

(续上页)

```
  tp_core_cluster0_core1  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster0_core2  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster0_core3  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster1_core0  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster1_core1  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster1_core2  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster1_core3  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster2_core0  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster2_core1  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster2_core2  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster2_core3  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster3_core0  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster3_core1  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster3_core2  0                       0                       0                       ␣
↪        0                       0
  tp_core_cluster3_core3  0                       0                       0                       ␣
↪        0                       0

  IPU Total Information:
                          lt_cycles               ct_cycles               alu_cycles              ␣
↪        io_write_bytes          io_read_bytes
  tp_core_cluster0_core0   0                       0                       0                       ␣
↪         0                       0
  tp_core_cluster0_core1   0                       0                       0                       ␣
↪         0                       0
  tp_core_cluster0_core2   0                       0                       0                       ␣
↪         0                       0
  tp_core_cluster0_core3   0                       0                       0                       ␣
↪         0                       0
```

(下页继续)

(续上页)

```
  tp_core_cluster1_core0   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster1_core1   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster1_core2   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster1_core3   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster2_core0   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster2_core1   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster2_core2   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster2_core3   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster3_core0   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster3_core1   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster3_core2   0                    0                    0                    ␣
↪         0                    0
  tp_core_cluster3_core3   0                    0                    0                    ␣
↪         0                    0

  ..........................................................................

...
```

回显字段详细解释如下表所示：

表 4.17: 部分硬件模块介绍（因设备不同而不同）

显示名	字段解释
vpuX	第 X 个 vpu（及 jpu）。
clusterX	第 X 个 cluster。
llcX	第 X 个 llc。
ddrX	第 X 个 ddr channel。
tp_core	即为 tensor processor 模块。
tp_cdmaX	第 X 个 cluster 间的 sharememory 访存总线。
scalerX	第 X 个 scaler。
dramX	第 X 个 ddr bank。
pcieX	pcie 的 link X。
tagram_hit	tagram 发生 hit 的次数。
tagram_miss	发生 eviction 的次数。
tlb_write_access	运算核心发起的写入请求，在 SMMU 处查询页表的次数。
tlb_read_access	运算核心发起的读取请求，在 SMMU 处查询页表的次数。
tlb_write_miss	tlb_write_access 中发生缺页的次数。
tlb_read_miss	tlb_read_access 中发生缺页的次数。

4.8 layer 命令

layer 命令用于输出融合模式下每层的性能数据到 CSV 文件，并生成 html 和 json 文件，用于可视化性能分析。

> **注意:**
>
> layer 生成的 json 文件以及图形化界面使用介绍，详见layer 命令可视化性能数据分析 章节。

4.8.1 layer 命令参数及解释

表 4.18: layer 命令参数及解释

参数	参数解释
- -nglog <name>	指定用户已有的 nglog.json 文件，使用 layer 命令跟踪网络时不需要指定 nglog。nglog 为包含网络信息的日志文件。
- -perfdata <name>	指定用户已有的 perfData 文件，使用 layer 命令跟踪网络时不需要指定 perfdata。perfdata 为网络中包含每层 PMU 数据的日志文件。
- -device <type>	指定生成 nglog 和 perfdata 时使用的设备类型，目前只支持填写 220 和 270。

4.8.2 layer 命令使用方法

1. layer 命令提供以下两种使用方法：
   - 执行 cnperf-cli layer “xxx”命令跟踪目标网络，生成目标网络运行时的性能分析结果。推荐使用该命令跟踪目标网络，避免因分析日志时的环境差异无法生成分析结果。
   - 执行 cnperf-cli layer –nglog nglog.json –perfdata perfData –device <type> 分析已有的原始网络性能日志。

   layer 命令的使用举例如下，以 MLU270 为例：
   - 追踪 caffe 在线网络使用样例，在任意目录可直接执行如下命令：

```
cnperf-cli layer "clas_online_multicore -model inception-v3_int8_scale_dense_1batch.
↪prototxt -weights inception-v3_int8_dense.caffemodel -labels synset_words.txt -images␣
↪file_list -mcore MLU270 -mmode MFUS -batchsize 16 -core_num 16"
```

   - 在任意目录下输入以下命令分析已有的 perf log：

```
cnperf-cli layer --nglog nglog.json --perfdata perfData --device 270
```

2. 显示结果如下：

```
PMU Data has been written into cnperf_layer/csv/xxx_ipu_core.csv and cnperf_layer/csv/xxx_
↪mem_core.csv
web data has been written into cnperf_layer/json/xxx_data.json and cnperf_layer/json/xxx_
↪fused_data.json
trace complete.
```

3. 使用该命令后，会在当前目录下生成 cnperf_layer 文件夹，将融合模式下每层的性能数据输出到

cnperf_layer 下的 CSV 文件中，并生成 html 和 json 文件用于可视化性能分析。CSV 文件统一放在 cnperf_layer/csv 文件夹内，json 文件统一放在 cnperf_layer/json 文件夹内。

4.8.3 layer 命令分析离线模型

因为离线模型的编译和运行是分开进行的，所以使用 layer 命令时应按照如下步骤进行：

1. 使用 cnperf-cli layer “xxx” 编译离线模型，在当前目录下生成 xxx_CNPERF_DATA_1_xxx 和 xxx_CNPERF_DATA_4 性能信息文件。
2. 使用 cnperf-cli layer “xxx” 运行离线模型，在当前目录下生成 xxx_CNPERF_DATA_2_xxx 文件。
3. 使用 cnperf-cli layer –nglog xxx_CNPERF_DATA_1_xxx –perfdata xxx_CNPERF_DATA_2_xxx –device <type> 分析性能日志。

注解:

“xxx_CNPERF_DATA_1_xxx”、” xxx_CNPERF_DATA_2_xxx” 和 “xxx_CNPERF_DATA_4” 文件的命名规则参见layer 命令可视化性能数据分析 章节。

4.8.4 显示 CNML 原始图信息及原始图与计算图算子对应关系

如果使用的 CNML 库支持导出 CNML 网络原始图信息，那么 layer 命令可以解析原始图日志，给用户展示原始图的网络结构及算子节点信息，此时 layer 命令还可以整理原始图与计算图的算子对应关系，将算子对应关系展示给用户。可视化结果查看方法参见layer 命令可视化性能数据分析 章节。

4.8.5 CSV 文件各字段解释

表 4.19: layer 命令生成的 CSV 文件中各字段解释

字段名	字段解释
core_id	IPU 核号。
layer_group_id	该核上运行的网络层 ID。
layer_group_type	该核上运行的网络层类型。
duration	该层的运行时间。
alu	标量计算单元运算的 cycle 数，单位为 cycles。
inst_num	当前层执行的指令总条数。
ct	向量运算计算的 cycle 数，单位为 cycles。
lt	卷积运算计算的 cycle 数，单位是 cycles，显示的是片上单个 LT 的计算 cycle 数。
iodma_rd	IPU_CORE 访问 DRAM 的读入数据量，单位为字节。
iodma_wr	IPU_CORE 访问 DRAM 的写入数据量，单位为字节。
shrm_rd	IPU_CORE 访问 MEM_CORE 的读入数据量，单位是字节。
shrm_wr	IPU_CORE 访问 MEM_CORE 的写入数据量，单位是字节。
boardcast	MEM_CORE 向 IPU_CORE 发起广播的数据量，单位是字节。
gdma_rd	MEM_CORE 访问 DRAM 的读入数据量，单位为字节。
gdma_wr	MEM_CORE 访问 DRAM 的写入数据量，单位为字节。
cdma0_recv	其它 cluster 的 MEM_CORE 通过 CDMA0 向当前 cluster 的 MEM_CORE 发送的数据量，单位是字节。
cdma0_wr	当前 cluster 的 MEM_CORE 通过 CDMA0 向其它 cluster 的 MEM_CORE 发送的数据量，单位是字节。
cdma1_recv	其它 cluster 的 MEM_CORE 通过 CDMA1 向当前 cluster 的 MEM_CORE 发送的数据量，单位是字节。
cdma1_wr	当前 cluster 的 MEM_CORE 通过 CDMA1 向其它 cluster 的 MEM_CORE 发送的数据量，单位是字节。
cache_miss	ICACHE 上 miss 的次数。

4.8.6 layer 命令注意事项

1. web 端绘图使用的 G6 库存在 BUG，使用 miniMap 移动网络时，若将网络移到当前界面外，则无法移回当前界面，只能重新加载 json 文件。

4.9 timechart 命令

timechart 命令用来生成 json 文件，文件中包含函数调用、实时功耗、MLU 利用率等信息，用户可使用 chrome://tracing 查看。

> **注意:**
>
> timechart 生成的 json 文件以及 Chrome 图形化接口使用介绍，参见timechart 命令可视化性能数据分析 章节。

4.9.1 timechart 命令参数及解释

表 4.20: timechart 命令参数及解释

参数	参数解释
-c <card>	写入指定卡号的 PMU 采样数据，数值范围应在 0 到该机器所拥有的板卡数目减去 1。当取值为-1 时，表示指定所有卡号，默认值为-1。
- -detail	写入详细的性能数据。
-i <path>	指定跟踪日志路径。默认为当前目录下的 dltrace_data 文件夹。
- -name	指定 json 文件名称。
-o <path>	指定 json 文件输出路径。
- -sampler	开启 PMU 数据采样，默认为开启状态。

4.9.2 timechart 命令使用方法

1. 进入放置 dltrace_data 的目录，一般是运行 **record** 时所在的目录。
2. 输入命令如下：

```
cnperf-cli timechart
```

3. 显示结果如下所示:

```
CNPERF timechart:
write timechart json success!
```

4.10 info 命令

info 命令主要用来查看上次 **record** 命令执行环境的相关信息。

4.10.1 info 命令参数及解释

表 4.21: info 命令参数及解释

参数	参数解释
-i <path>	指定跟踪日志路径。

4.10.2 info 命令使用方法

1. 进入放置 dltrace_data 的目录，一般是运行 **record** 时所在的目录。
2. 输入命令如下：

```
cnperf-cli info
```

3. 显示结果如下所示:

```
# system information
# ==================
# recorded on           : Tue Nov 17 19:01:11 2020
# cmdline               : ./cnperf-cli$record$/opt/shared/cnperf/hello$--pmu$
# cpu info              :  Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz
# number of cpus        : 12 / 12 (online / possible)
# memory info           : 5.4GB / 7.5GB (free / total)
```

(下页继续)

(续上页)

```
# kernel version        : Linux 3.10.0-957.27.2.el7.x86_64
# hostname              : localhost.localdomain
# distro                : "CentOS Linux 7 (Core)"
# driver version        : 4.7.0
# card name             : MLU270
# cnperf version        : 3.12.0 cf63d53
```

process information # ================== # log version : 3 # exe image : /opt/shared/cnperf/hello # exit status : 0 # task start time : 4225107230117463(ns) # trace time : 5319179115(sec)
param information # ================== # exec : /opt/shared/cnperf/hello # c : 0 <default> # cpu : true <default> # filter_config_file : /projs/rootzone/systools/cnperf.json # o : dltrace_data/ <default> # pmu : true # sampler : true <default> # t : 100 <default>

回显字段详细解释如下：

表 4.22: info 命令回显字段解释

回显字段	字段解释
system information	系统相关信息。
recorded on	生成性能数据时间。
cmdline	执行的命令行。
cpu info	主机 CPU 信息。
number of cpus	使用的 cpu 数量。online 指在线可以调度的 CPU 数量。possible 指可能存在的 CPU 数量。
memory info	内存信息。free 指空闲内存信息，total 指整个内存信息。
kernel version	Linux 内核版本号。
hostname	主机域名信息。
distro	操作系统版本号。
driver version	驱动版本号。
card name	板卡名称。
cnperf version	CNPerf 版本号。
process information	进程信息。
log version	日志版本。
exe image	被跟踪程序路径。
exit status	进程状态。0 代表进程正常，其它值说明进程异常。
task start time	任务启动时间。
trace time	进程跟踪时间。
param information	上一次 record 命令设置的参数信息。
exec	指定的待追踪程序路径。
c	指定的卡号, 默认值为 0, <default> 代表未设置。
cpu	指定的待追踪程序，默认为 true, <default> 代表未设置。

4.11 配置 cnperf.json

cnperf.json 是使用 json 语法的配置文件，是由一系列 file_list 对象组成，用户可以通过配置 cnperf.json 筛选可执行程序中的部分函数进行跟踪。执行 **record** 命令时可以通过使用 `--filter_config_file` 参数指定配置文件的位置。若不指定该参数，则默认使用存放在/etc 目录下 (x86_64 环境) 的配置文件。

4.11.1 配置文件说明

- **file_list**： 包含需要跟踪的目标文件列表。每个元素对应一个文件，每个文件包含一个 file_name 和 trace、notrace 两个列表。trace、notrace 中包含由 func_name、func_args、func_ret 组成的元素。
  - **file_name**： 被跟踪的文件名。对于可执行程序，该路径可留空（或保留默认的“exe filter”）。
  - **trace**： 记录需要跟踪的函数列表。
  - **notrace**： 记录不需要跟踪的函数。
  - **func_name**： 函数名，一个函数对应一个函数名。
  - **func_args**: 为一个字符串数组，数组中每一项都是一个字符串, 表示参数的类型。参数支持 `u8`, `u16`, `u32`, `u64`, `i8`, `i16`, `i32`, `i64` (8 种整形), `p` (指针), `b` (bool), `s` (字符串), `c` (字符), `f` (float), `lf` (double)。
  - **func_ret** : 对应项是一个字符串, 也表示参数类型, 同上但是额外支持 v, 表示 void 返回类型。

注意事项

1. 对于可执行文件，默认只会跟踪 main 函数，如有其它需求，请修改配置文件，默认第一个 file_list 对应主程序的跟踪规则。
2. func_name 填写的函数名可以是某个函数的全部名称或部分名称，也可以使用正则表达式进行匹配或者部分匹配。如跟踪 C++ 程序，由于编译器会进行 mangling，如需表示 ClassA::func，请写 func 或者 ClassA，不支持 ClassA::func 的写法。
3. 用户所跟踪的可执行文件建议使用本地编译生成。如果跟踪程序出现问题，请尝试在本地编译可执行程序后，再进行跟踪。
4. 运行 **record** 命令时，可以指定配置文件路径（详见—help 命令的显示信息），如不指定，默认使用/etc/cnperf.json。
5. trace、notrace 只需设置其中一个。设置 trace 则其他函数不跟踪，只跟踪 trace 中配置的函数；设置 notrace，则 notrace 中配置的函数不跟踪，其他函数全跟踪。如果二者都设置，则以 trace 为准。
6. func_args 从第一个参数开始，且必须连续。最多支持 6 个参数，其中最多 2 位为浮点类型。暂不支持 CNML、CNRT 相关函数跟踪参数信息。
7. 跟踪 cnrt, cndrv, cnml, cnnl 中的应用编程接口函数, file_name 必须为”cambricon apis”.

4.11.2 配置文件编写举例

- 跟踪所有函数：

```
{
  "file_list": [
    {
      "file_name": "exe filter",

      "notrace": []
    }
  ]
}
```

- 只跟踪 main 函数和 func 函数：

```
{
  "file_list": [
    {
      "file_name": "exe filter",

      "trace": [
        { "func_name": "^main$" },
        { "func_name": "^func$" }
      ]
    }
  ]
}
```

- 只跟踪 main 函数和函数名以 test 为前缀的所有函数：

```
{
  "file_list": [
    {
      "file_name": "exe filter",

      "trace": [
        { "func_name": "^main$" },
        { "func_name": "^test.*" }
      ]
    }
  ]
}
```


- 不跟踪 abc 开头的所有函数，剩余函数全部跟踪：

```
{
  "file_list": [
    {
     "file_name": "exe filter",

      "notrace": [
        { "func_name": "^abc.*" }
      ]
    }
  ]
}
```

- 跟踪 abc 开头的所有函数，但同时写了 trace 和 notrace，这种情况下，只会跟踪 trace 中所写的函数：

```
{
  "file_list": [
    {
      "file_name": "exe filter",

      "trace": [
        { "func_name": "^abc.*" }
      ],
      "notrace": [
        { "func_name": "^abc.*" }
      ]
    }
  ]
}
```

- 针对特定函数追踪函数参数及返回值信息：

```
{
  "file_list": [
    {
      "file_name": "exe filter",

      "trace": [
        {
          "func_name": "^.*func_1.*$",
          "func_args": ["b", "c", "u64"],
          "func_ret": "u64"
```

(下页继续)

(续上页)

```
        }, {
          "func_name": "^.*func_2.*$",
          "func_args": ["s", "f", "lf"],
          "func_ret": "u64"
        }, {
          "func_name": "^.*func_3.*$",
          "func_ret": "u64"
        }
      ]
    }
  ]
}
```

- 追踪所有 cndrv, cnml, cnnl APIs; 对于 cnrt APIS, 除了 cnrtConvertDoubleToHalf, cnrtConvertFloatToHalf, cnrtConvertHalfToDouble, cnrtConvertHalfToFloat 函数，其它的都会进行追踪：

```
{
    "file_name": "cambricon apis",
    "trace": null,
    "notrace": [
        { "func_name": "cnrtConvertDoubleToHalf" },
        { "func_name": "cnrtConvertFloatToHalf" },
        { "func_name": "cnrtConvertHalfToDouble" },
        { "func_name": "cnrtConvertHalfToFloat" }
    ]
}
```


5 layer 命令可视化性能数据分析

5.1 layer 命令可视化性能数据分析

本章节主要介绍性能剖析工具 layer 的可视化分析功能，该功能导出融合模式下每层的 PMU 性能数据，并以图形化界面展示融合模式下 CNML 原始图和计算图的网络结构和参数信息。

5.2 生成性能分析结果

layer 命令生成性能分析结果的方法参见layer 命令使用方法 章节。性能分析完成后，会在当前目录下生成 cnperf_layer 文件夹，cnperf_layer 文件夹中各文件和子文件夹的含义如下所示：

- origin_cnperf_data：该文件夹存放网络的原始性能日志，包括 [tid]_[compile_count]_CNPERF_DATA_1_[suffix]、[tid]_[compute_count]_CNPERF_DATA_2_[suffix] 和 [tid]_[compile_count]_CNPERF_DATA_4 文件。
- csv：该文件夹下存放 xxx_ipu_core.csv 和 xxx_mem_core.csv 文件。
- json：该文件夹下存放 xxx_fused_data.json 和 xxx_data.json 文件。
- cnml_json：该文件夹下存放描述 CNML 原始图信息的 json 文件。
- [tid]_[compile_count]_CNPERF_DATA_1_[suffix]：即 nglog.json 文件，存放网络结构信息。
- [tid]_[compute_count]_CNPERF_DATA_2_[suffix]：即 perfData 文件，存放网络运行时参数信息。
- [tid]_[compile_count]_CNPERF_DATA_4：存放 CNML 原始图网络结构及原始图和计算图算子对应关系。
- xxx_ipu_core.csv：保存网络 ipu core 的性能数据。
- xxx_mem_core.csv：保存网络 mem core 的性能数据。
- xxx_fused_data.json：包含融合后网络信息的数据文件。
- xxx_data.json：包含融合前网络信息的数据文件。

上述文件名中 [suffix] 是文件名的后缀，suffix 可能为空。xxx 由 [tid]_[compute_count]_[first_group_id]_[first_group_layer_type] 组成，以上涉及的参数含义如下所示：

- suffix：该后缀由 ID_[算子 ID]_[Shape]_[用户输入的形状]_[日期]_[时间] 组成。
- tid：被跟踪的产生性能数据的线程号。

- compute_count：被跟踪网络计算次数。
- compile_count：被跟踪网络编译次数。
- first_group_id：被跟踪网络第一层的 ID。
- first_group_layer_type：被跟踪网络第一层的类型。

5.3 查看计算图性能分析结果

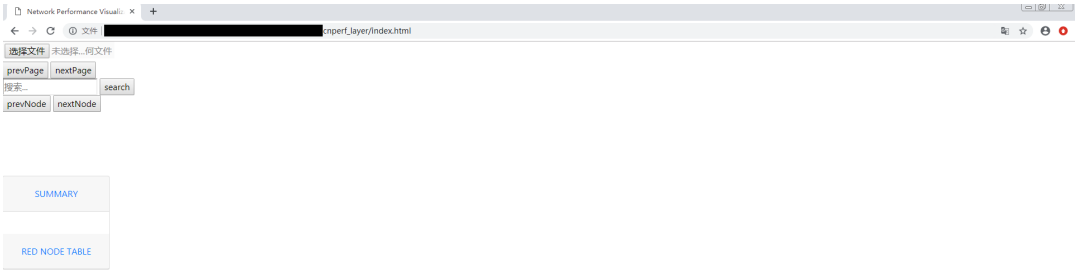
本节主要介绍如何使用图形化界面查看融合模式下计算图的网络结构和参数。

提示:

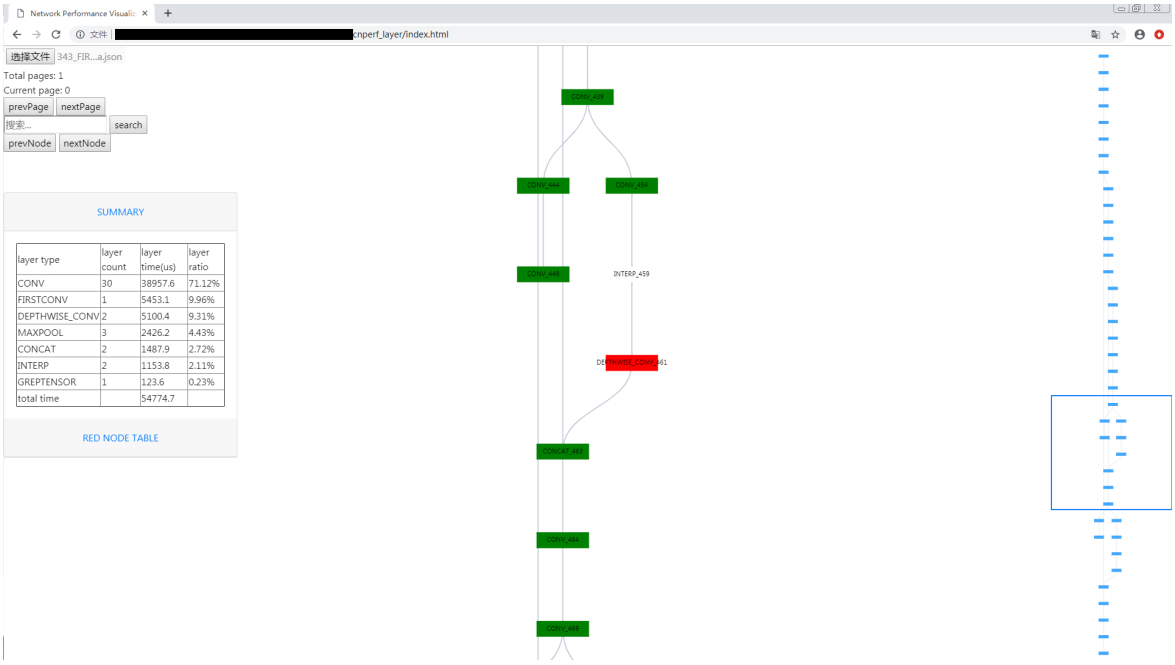
ipu core 和 mem core 的性能数据存放在 csv 文件中，文件内各字段的解释请参考“layer 命令-csv 各字段解释”章节。

---

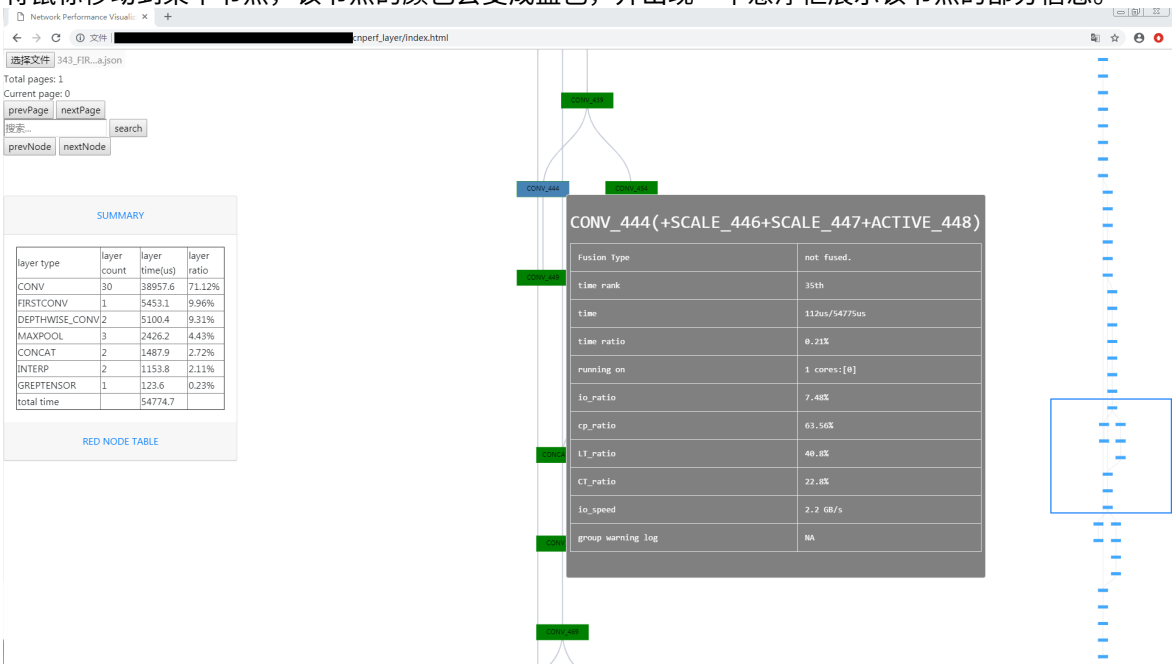
1. 使用 Chrome 浏览器打开 cnperf_layer/index.html。



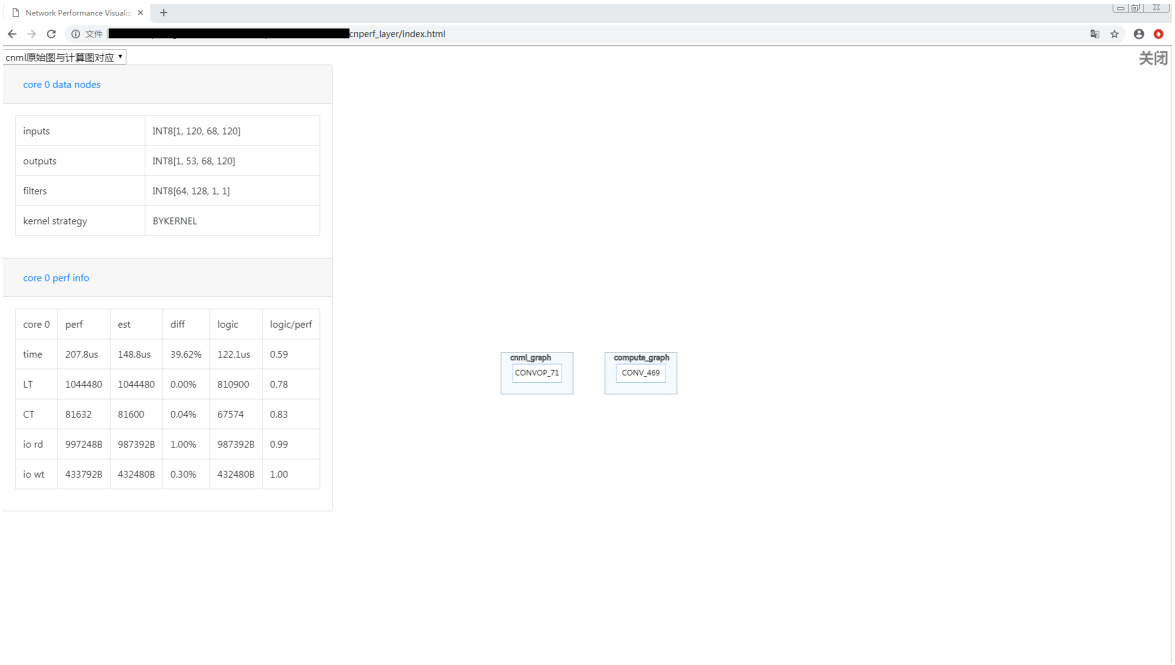
2. 点击“选择文件”，加载 cnperf_layer/json 文件夹下的 json 数据文件，界面展示了网络结构和网络算子总结信息。



3. 将鼠标移动到某个节点，该节点的颜色会变成蓝色，并出现一个悬浮框展示该节点的部分信息。



4. 左键单击该节点出现的新界面中有该节点使用的 ipu 核的 data nodes 和 perf 信息。

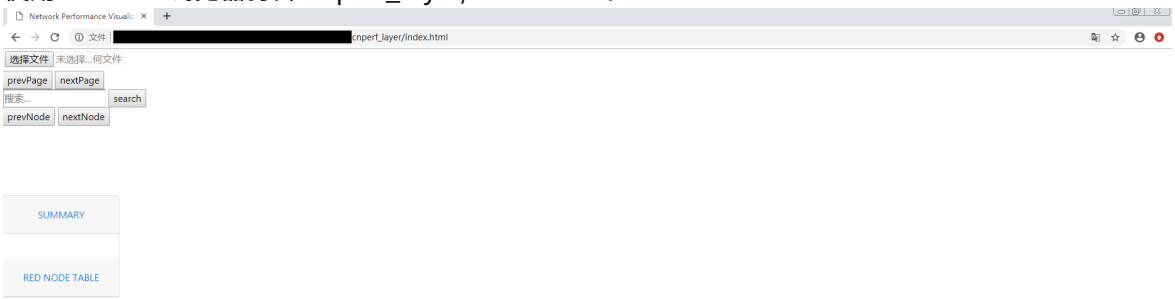


网络图界面还支持以下操作：

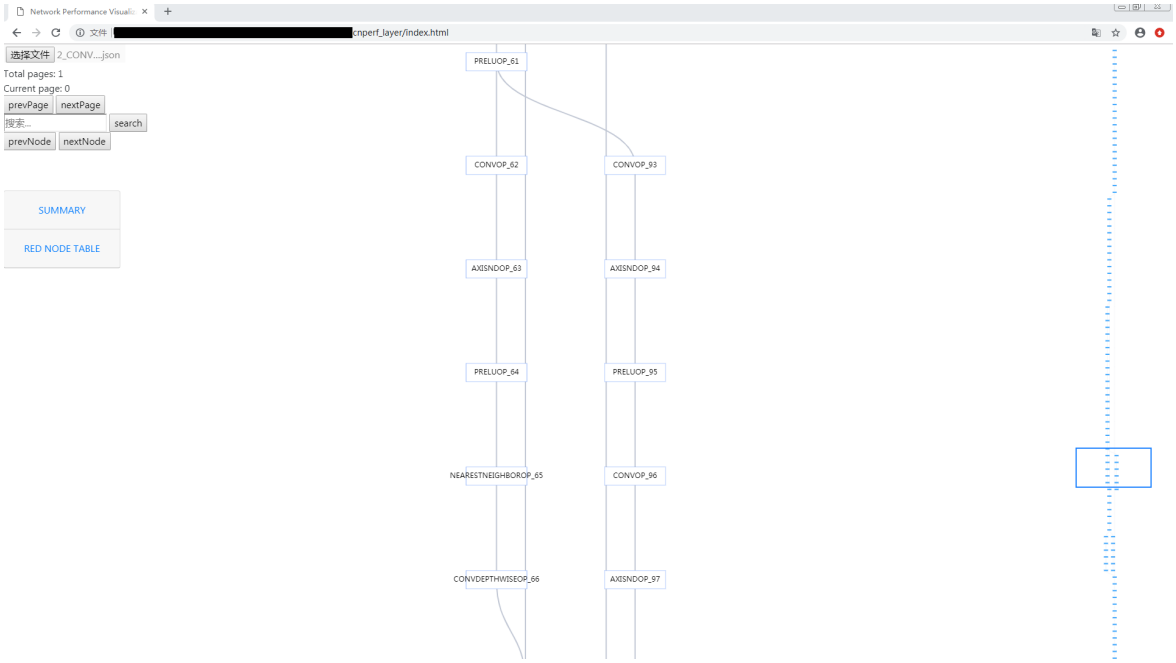
- 滚动鼠标滚轮实现页面缩放。
- 按住鼠标左键可以拖动网络。
- 使用 miniMap 快速拖动网络。
- 鼠标左键单击“SUMMARY”和“RED NODE TABLE”按键可展开或折叠对应表格。

5.4 查看 CNML 原始图信息

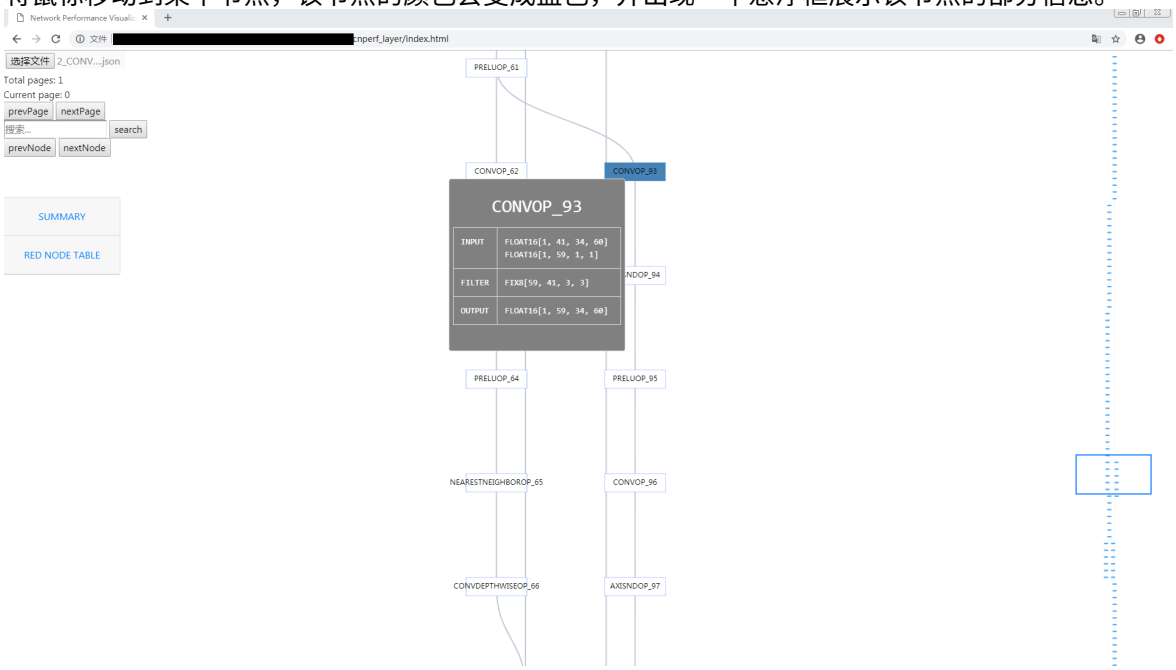
1. 使用 Chrome 浏览器打开 cnperf_layer/index.html。



2. 点击“选择文件”，加载 cnperf_layer/cnml_json 文件夹下的 json 数据文件，界面展示了 CNML 原始图的网络结构信息。



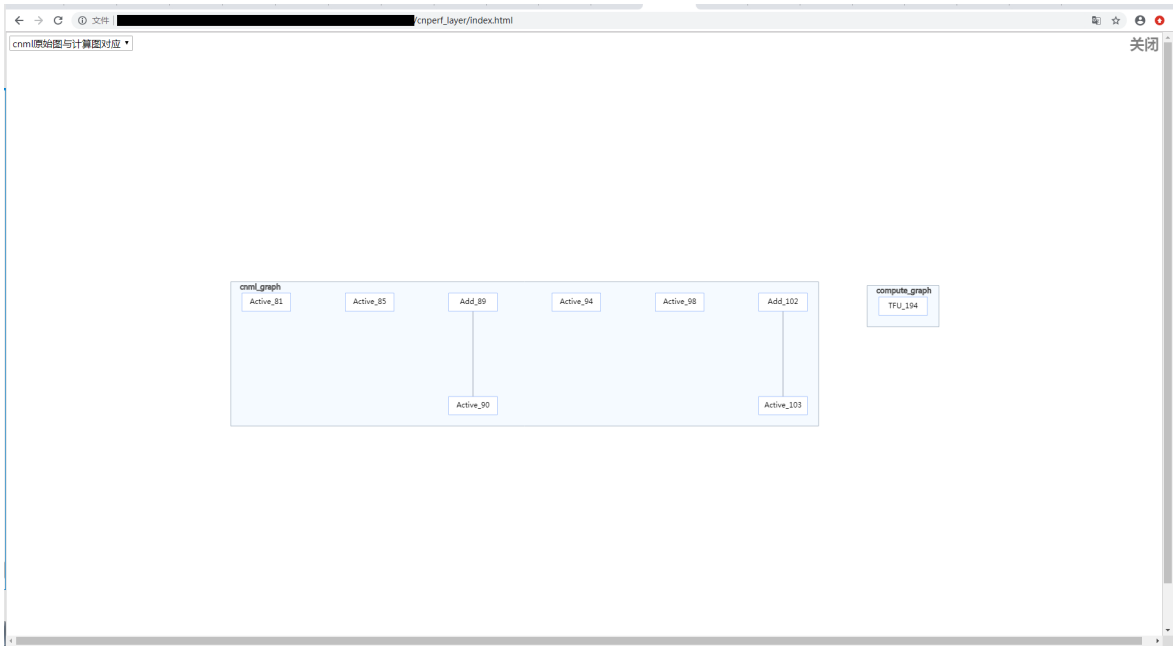
3. 将鼠标移动到某个节点，该节点的颜色会变成蓝色，并出现一个悬浮框展示该节点的部分信息。



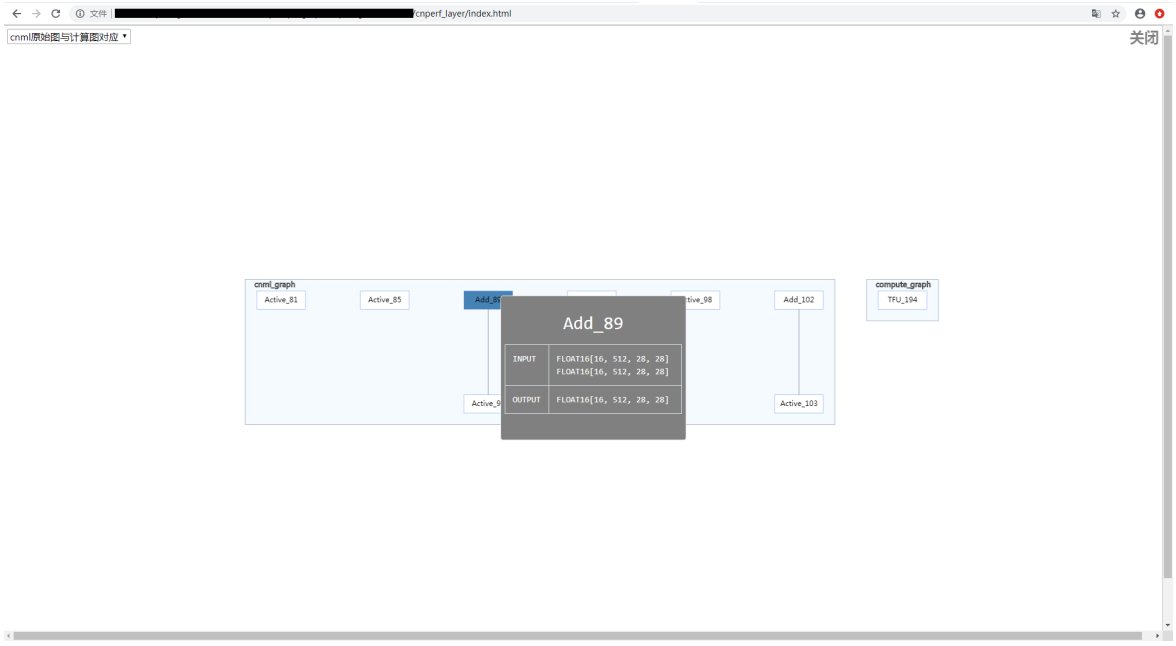
5.5 查看原始图与计算图算子对应关系

从原始图界面和计算图界面均可以查看算子对应关系，查看方式如下：

1. 鼠标左键单击算子，会出现一个新的界面，默认展示的是“CNML 原始图和计算图对应关系”。



2. 子图界面会出现两个框，其中一个框名为“cnml_graph”，里面展示的是原始图中的算子节点，另一个框名为“compute_graph”，里面展示的是计算图中的算子节点。
3. 将鼠标移动到子图的节点上，该节点颜色会变成蓝色，并出现一个悬浮框展示该节点的部分信息。



提示:

- 计算图中在同一方框中且节点形状被置为菱形的节点为一组，该组节点与原始图中节点存在“多对一”、“一对多”或“多对多”关系。
- 原始图中在同一方框中且节点形状被置为菱形的节点为一组，该组节点与计算图中的节点存在“一对多”、“多对一”或“多对多”关系。
- 矩形节点表示该节点与另一图中节点存在“一对一”或“一对无”关系。

5.6 分析结果界面介绍

生成的分析结果界面主要包含以下信息：

- 网络结构
- 节点信息
  - 节点名称：以“节点层类型 _ 节点层 ID（+ 被融合层类型 _ 被融合层 ID）”格式命名。
  - time rank：网络层消耗的时间排名，消耗时间越多排名越靠前。
  - time：以“该层消耗时间/整个网络消耗时间”格式显示，单位为 us。
  - running on：该网络层跑在哪些 ipu core 上。
  - cp_ratio：网络层的 lt 和 ct 计算时间和占网络层实际消耗时间的比例。
  - LT_ratio：网络层的 lt 计算时间和占网络层实际消耗时间的比例。
  - CT_ratio：网络层的 ct 计算时间和占网络层实际消耗时间的比例。
  - io_ratio：网络层的 io 时间占网络层实际消耗时间的比例。
  - inputs：网络层在某个核上的 inputs 数据类型及数值，以“数据类型 [n，c，h，w，logicC]”格式展示。
  - outputs：网络层在某个核上的 outputs 数据类型及数值，以“数据类型 [n，c，h，w，logicC]”格式展示。
  - filters：网络层在某个核上的 filters 数据类型及数值，以“数据类型 [n，c，h，w，logicC]”格式展示。
  - kernel strategy：网络层在某个核上采用的策略。
  - time：网络层消耗时间。
  - LT：lt 运算的 cycle 数。
  - CT：ct 运算的 cycle 数。
  - io rd：iodma 读取量，单位为字节。
  - io wt：iodma 写入量，单位为字节。
- 网络节点颜色

  layer 命令采用默认阈值标记节点颜色。当节点对应参数超过阈值时，会将节点标记为红色，当节点对应参数不超过阈值时，将节点标记为绿色，当节点暂不支持参数理论值计算或者该节点是被融合的节点时，将节点标记为白色。

说明：

- time、LT、CT、id rd 和 io wt 还分为 perf（实际值）、est（理论值）、diff（实际值与理论值的差值占理论值的百分比）、logic（逻辑值）和 logic/perf（逻辑值与实际值的比值）。

6 timechart 命令可视化性能数据分析

6.1 timechart 命令可视化性能数据分析

本章节主要介绍性能剖析工具的 timechart 功能，以图形化的方式呈现给用户，使其更直观地理解和优化相关应用程序的性能。

6.2 追踪目标程序

本节主要是对可执行目标进行分析并将分析结果存放在数据文件中，然后将数据结果以可视化的形式展示。此部分需要先使用 record 功能再使用 timechart 功能，将跟踪目标程序的结果存放在 json 数据文件中。

6.3 生成 json 数据文件

生成 json 数据文件的步骤如下所示：

1. 执行 cnperf-cli record 追踪目标程序，默认情况下生成的数据会存放在当前目录下的 dltrace_data 文件夹中。
2. 然后执行程序 cnperf-cli timechart，生成 json 文件。
3. 若执行程序 cnperf-cli timechart –detail，会细化到每个核上的输出数据，生成 json 文件。
4. 若使用程序 cnperf-cli timechart -c + 卡号，会将指定卡号上的 PMU 采样数据，写入到生成的 json 文件。默认情况是将所有卡号的 PMU 采样数据写入 json 文件。

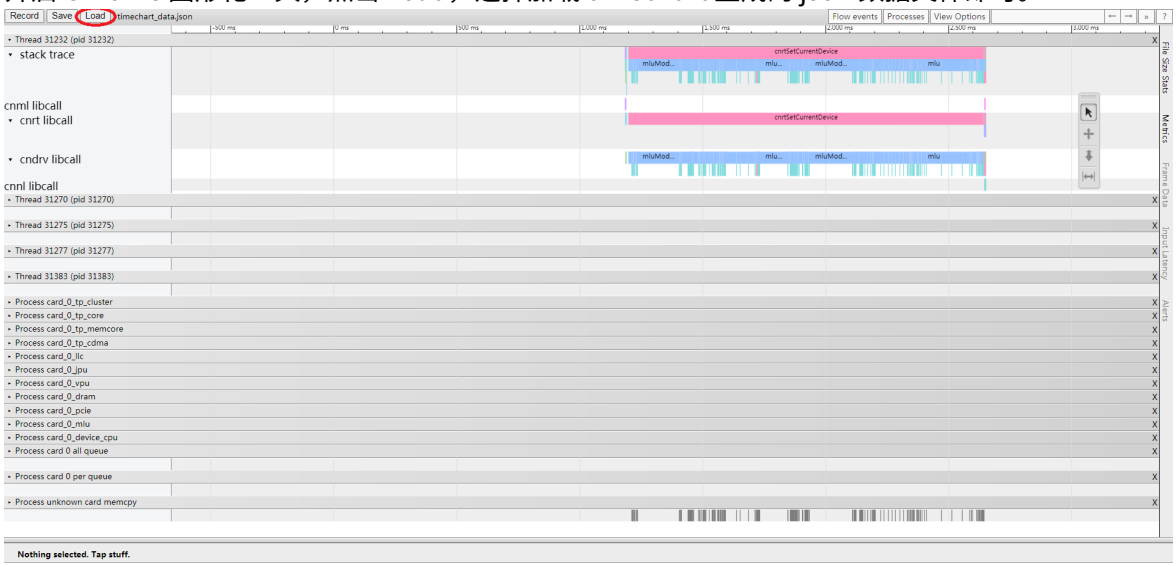
提示:

record 和 timechart 命令的使用参见record 命令 和timechart 命令 章节。

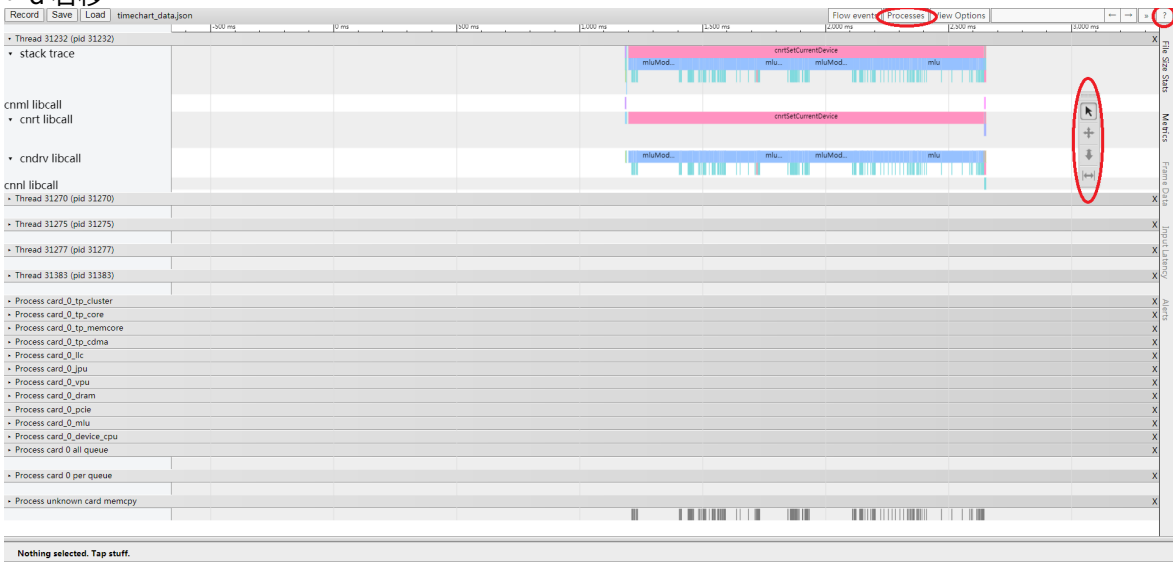
6.4 分析性能数据

本节主要分析已有的数据文件，将其以 Chrome 浏览器自带的图形化工具打开。

1. 打开 Chrome 浏览器，输入链接：chrome://tracing。
2. 开启 Chrome 图形化工具，点击 Load，选择加载 timechart 生成的 json 数据文件即可。



3. Chrome 图形化工具使用按键 [w] [a] [s] [d] 或者点击页面的悬浮按钮即可实现图形的移动和缩放。右上角“问号”查看帮助，或者参考 Chrome 图形化工具的相关文档。
   - w 放大
   - a 左移
   - s 缩小
   - d 右移



4. 所有的数据都是分 Process 显示的，可以点击右上角 Processes 选项，筛选自己想要显示的数据。

6.5 分析结果界面介绍

生成的分析结果界面主要包含以下信息：

- 函数信息图：

  该窗口呈现的是被跟踪程序中所有导出函数的执行时间、调用栈深度和起止时间等信息。每一个长方形表示一个函数在相应时间内的调用，长方形的长度就是函数的持续时间。

  在窗口中鼠标单击某一函数块，窗口左下侧就显示该函数相关详细数据，包含函数名、具体执行时间、起止时间等等。

  在该列表中，主要展示以下内容：

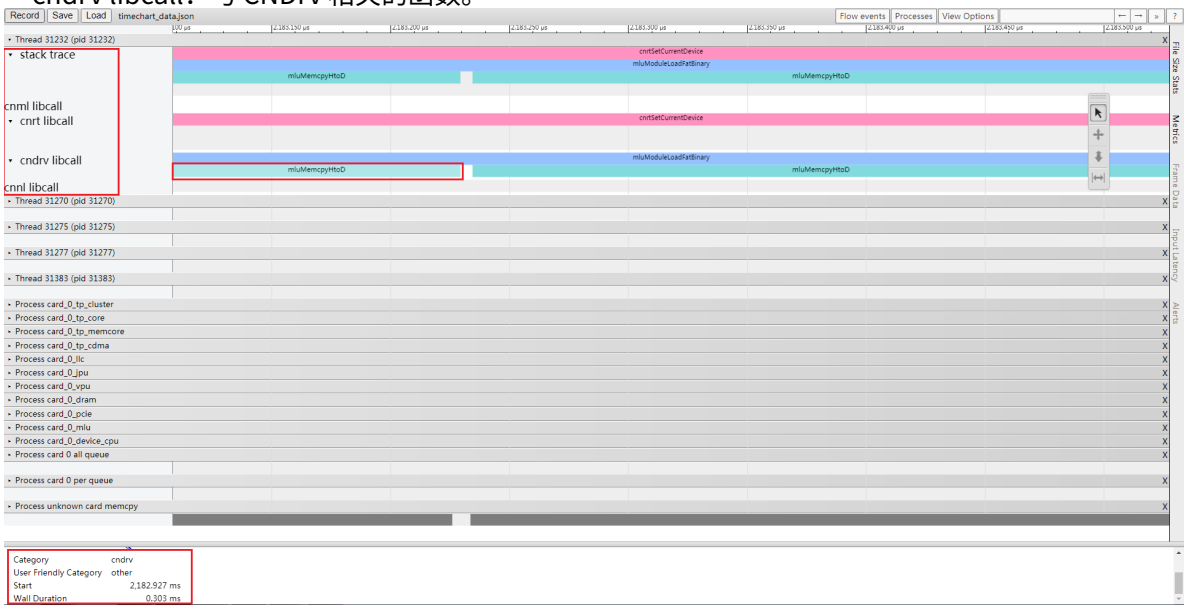
  – stack trace: 基本调用函数，包含所有追踪到的函数。

  – cnml libcall：与 CNML 相关的函数。

  – cnrt libcall：与 CNRT 相关的函数。

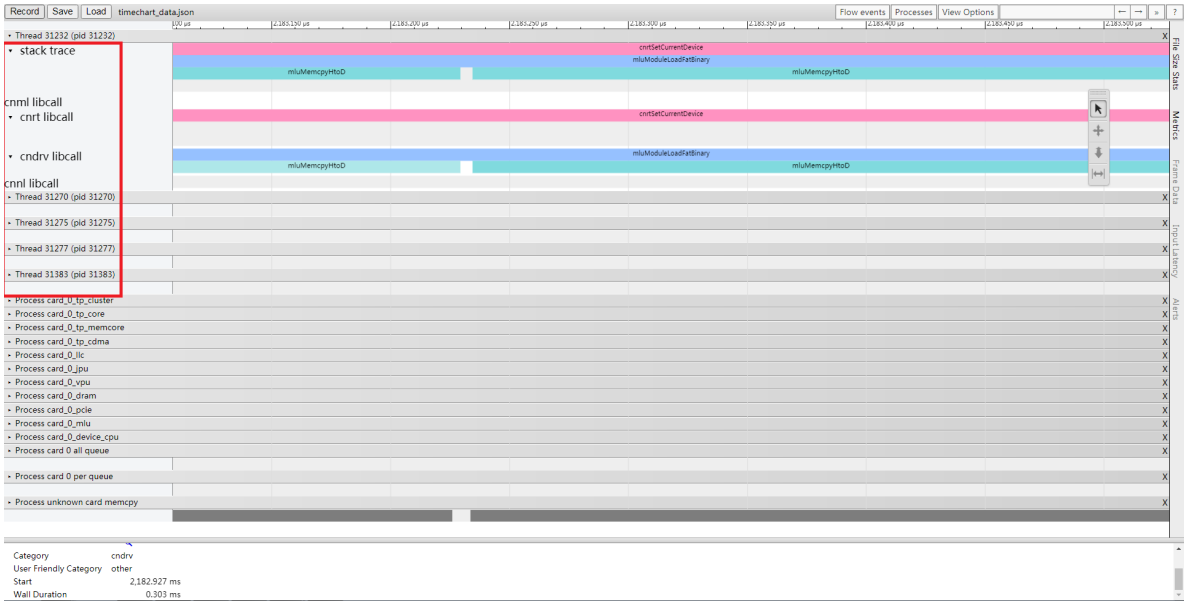
  – cnnl libcall：与 CNNL 相关的函数。

  – cndrv libcall：与 CNDrv 相关的函数。



- 多线程函数信息图：

  对于多线程函数，该窗口会分线程显示，在 Threads 双击相应的线程号，则在 Host 下显示此线程下执行的所有函数。详细信息如下图所示：



- queue 信息图：

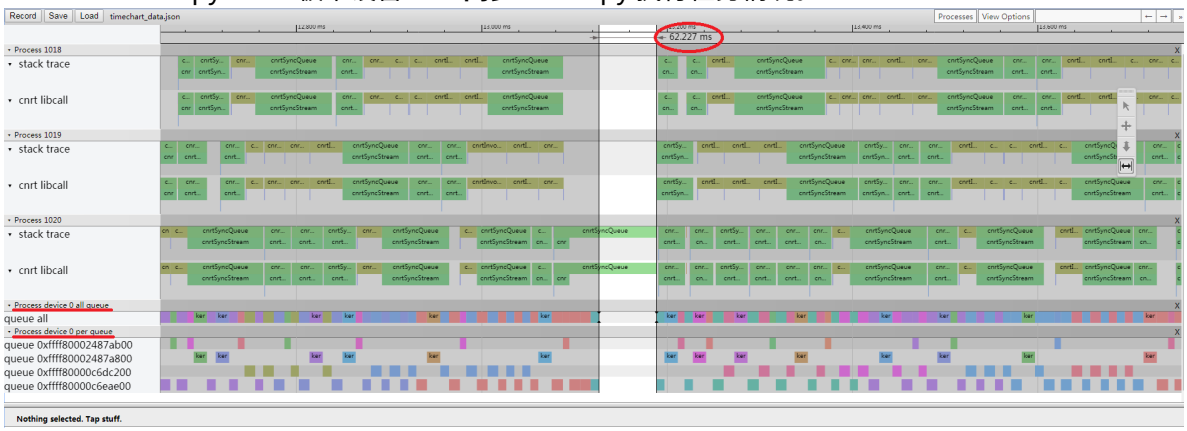
  在该窗口中，主要展示以下内容：

  – device X all queue: MLU 板卡设备 X 上所有 queue 的任务执行情况，包括 kernel，**异步** memcpy，**异步** memset 任务。

  – device X per queue: MLU 板卡设备 X 上每一个 queue 的任务执行情况, 包括 kernel，**异步** memcpy，**异步** memset 任务。

  – X card memset: MLU 板卡设备 X 上 **同步** memset 执行任务情况。

  – X card memcpy: MLU 板卡设备 X 上 **同步** memcpy 执行任务情况。



该窗口可以显示任务在 MLU 上准确的执行情况，用户可以通过此窗口判断是否由于 CPU 函数阻塞等原因导致了 MLU 出现空闲。例如上图中就出现了 MLU 空闲的情况，通过观察 device 0 all queue 窗口可发现，出现了 62.227ms 时间长度的空闲。

queue 队列中每一个事件代表着一次任务，可能是 kernel，memcpy，memset 等任务，详细信息如下：

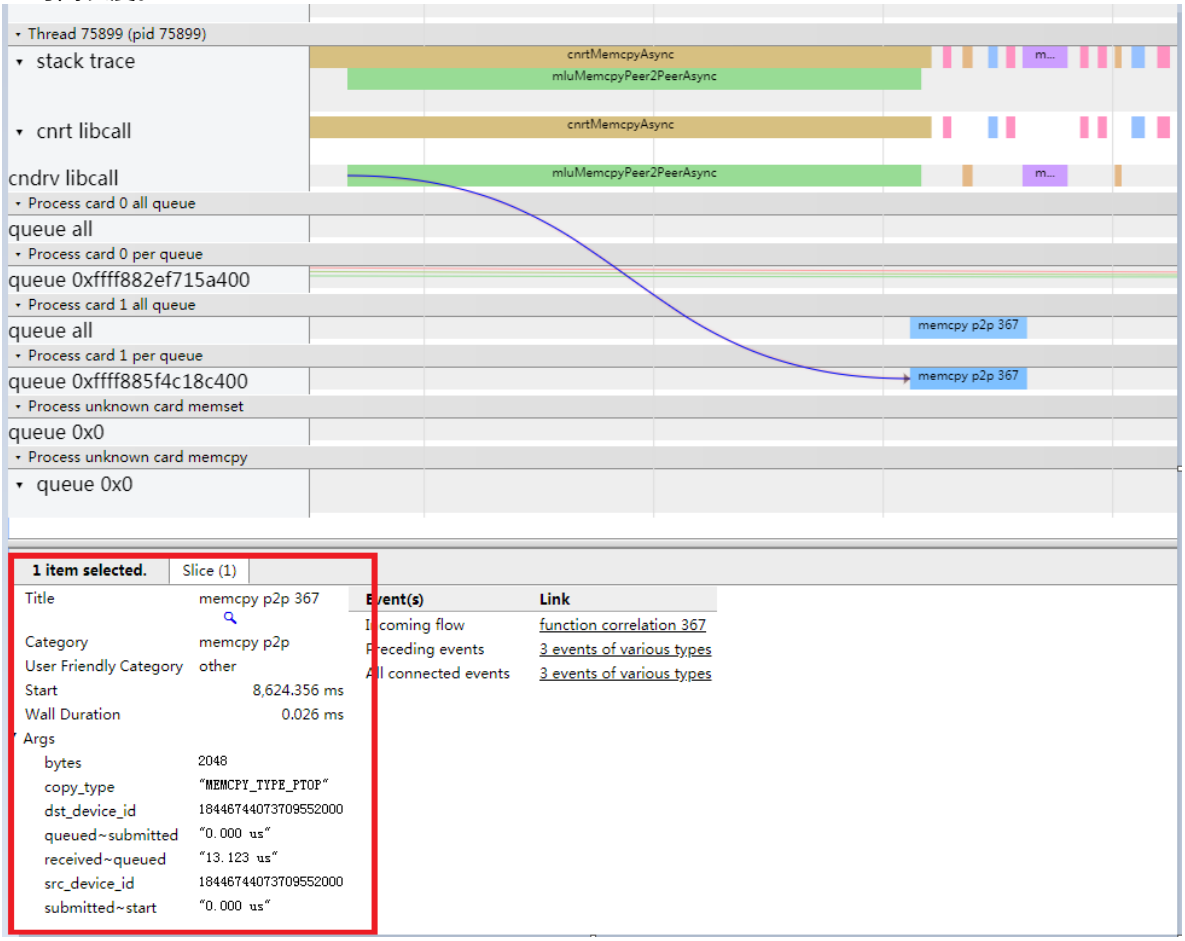
– Start: 设备记录的起始时间。

– Wall Duration: 设备完成任务开始到结束的时间。

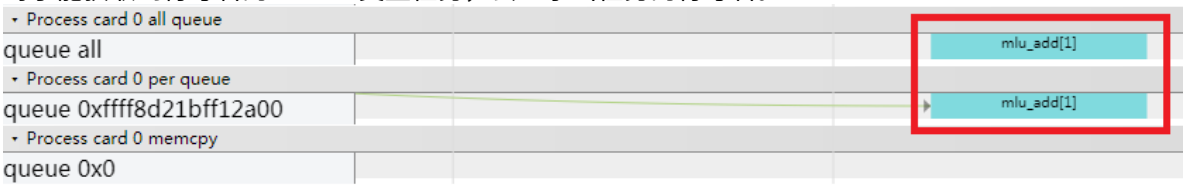
– Bytes: 当任务类型为 memcpy/memset 时，表示该任务参数中的字节数。

– copy_type: memcpy 的类型。

- received~queued: 从 MLU 驱动收到任务，到 MLU 驱动将任务放进 command buffer 的时间长度。
- queued~submitted: 从 MLU 驱动将任务放进 command buffer，到 MLU 驱动将 command buffer 中的任务提交到 MLU 的时间长度。
- submitted~start: 从 MLU 驱动将 command buffer 中的任务提交到 MLU，到任务真正开始执行的时间长度。



对于能获取到符号名的 kenrel 类型任务，会显示出任务的符号名。

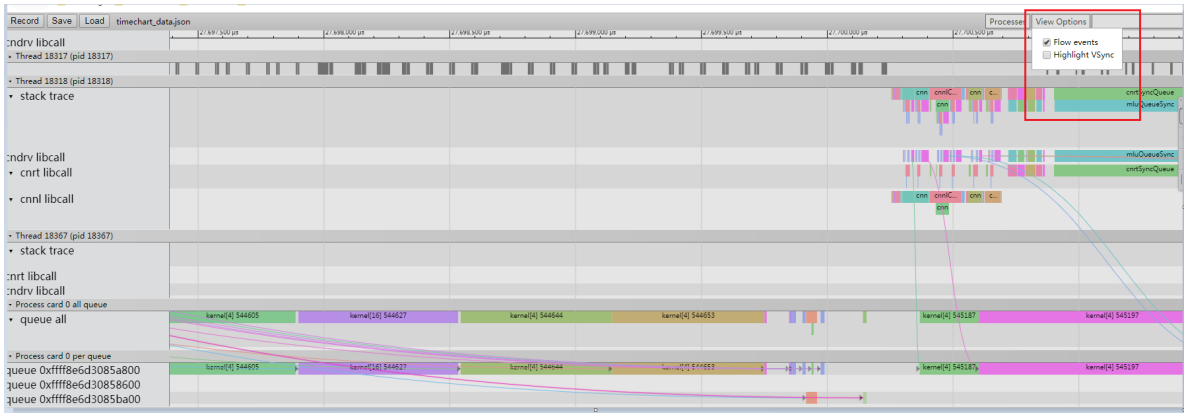


注解:

command buffer 是 MLU 驱动内部维护的一个 buffer，用于将 kernel、memcpy 等任务下发到 MLU 上。

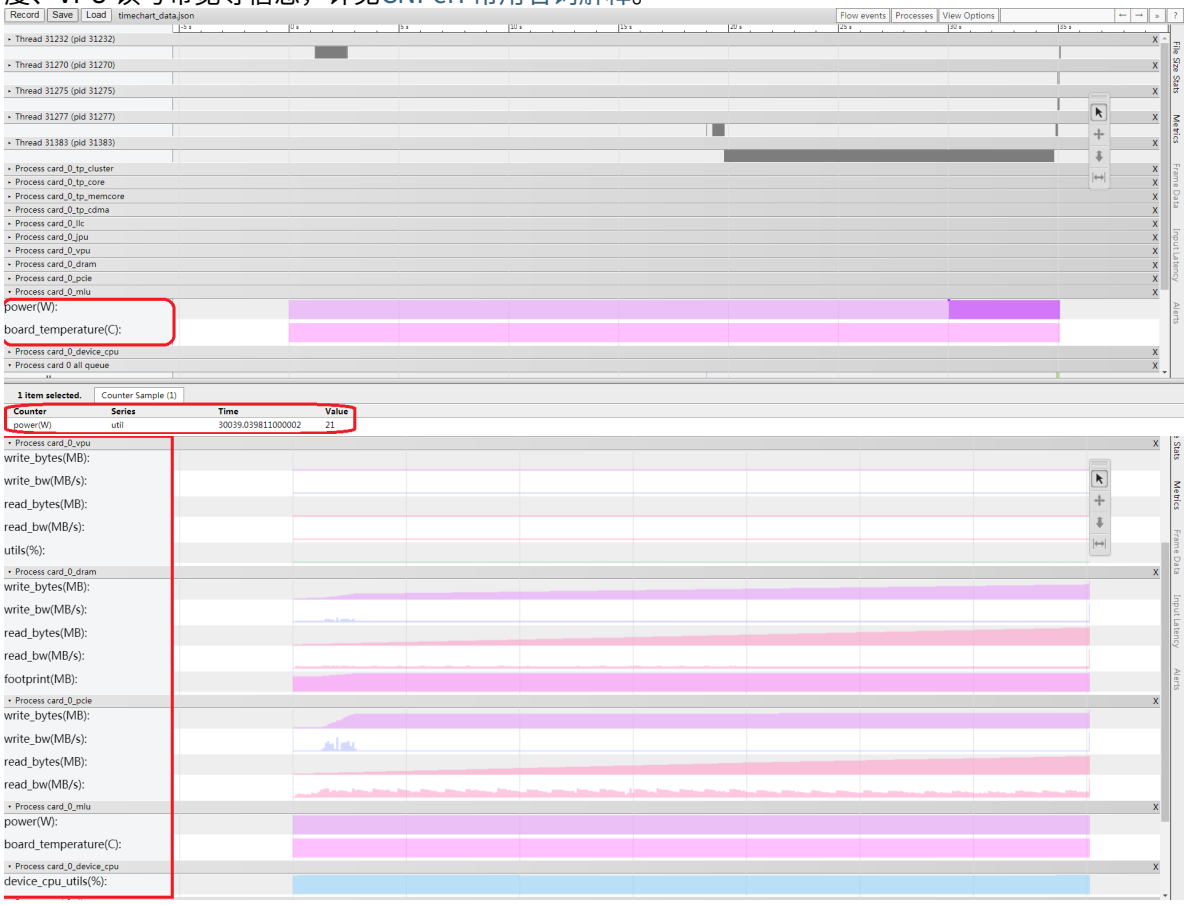
- 事件流 (flow event) 图:

  用户可以通过带箭头的连线，匹配 CPU 函数和 MLU 设备端相关操作的对应关系。用户可以通过网页浏览器右上角 view option 菜单键中的 flow event 开关，选择是否显示事件流的连线。



- 设备性能指标：

  这部分功能展示 CNPerf 在执行 record 功能的时候，与设备相关的性能指标，包括板卡功耗、板卡温度、VPU 读写带宽等信息，详见CNPerf 常用名词解释。



7 附录-常用名词解释

本章节主要介绍性能剖析工具的名词解释，方便理解和使用。

7.1 CNPerf 常用名词解释

1. 设备性能数据。
   - power(W): 板卡实时功耗。
   - board_temperature(C): 板卡温度。
   - ipu_utils(%): ipu 的利用率。
   - jpu_utils(%): jpu 的利用率。
   - vpu_utils(%): vpu 的利用率。
   - device_cpu_utils(%): device_cpu 的利用率。
   - footprint_channel(MB): 内存通道的占用情况。
   - scaler_utils(%): scaler 的利用率。
2. 读写带宽数据。
   - write_bytes: 写入数据统计，单位为字节。
   - read_bytes: 读取数据统计，单位为字节。
   - write_bw: 写入带宽。
   - read_bw: 读取带宽。
   - io_read_bytes: 计算单元读取 ddr 数据大小的数量，单位为字节。
   - io_write_bytes: 计算单元写入 ddr 数据大小的数量，单位为字节。
   - vpuX_read_bw: 第 X 个 VPU 的读取带宽。
   - vpuX_write_bw: 第 X 个 VPU 的写入带宽。
   - tp_clusterX_read_bw: 第 X 个 cluster 的读取带宽。
   - tp_clusterX_write_bw: 第 X 个 cluster 的写入带宽。
   - llcX_read_bw: 第 X 个 cluster 上所连接的 LLC 端口模块的读取带宽。
   - llcX_write_bw: 第 X 个 cluster 上所连接的 LLC 端口模块的写入带宽。
   - dram_channelX_read_bw: 第 X 个 cluster 上所连接的 DDR 端口模块的读取带宽。
   - dram_channelX_write_bw: 第 X 个 cluster 上所连接的 DDR 端口模块的写入带宽。
   - tp_cdmaX_read_bw: 第 X 个 cluster 间的 sharememory 访存总线的读取带宽。
   - tp_cdmaX_write_bw: 第 X 个 cluster 间的 sharememory 访存总线的写入带宽。

3. 运算单元数据。
   - lt_cycles: 乘积累加单元的运算周期数。
   - ct_cycles: 向量运算单元的运算周期数。
   - alu_cycles: 标量运算单元的运算周期数。
4. SMMU 读写请求数据。
   - tp_core_clusterX_coreY_tlb_read_access: 第 X 个 cluster 中第 Y 个 core 的运算核心发起的读取请求，在 SMMU 处查询页表的次数。
   - tp_core_clusterX_coreY_tlb_read_miss: 第 X 个 cluster 中第 Y 个 core 的 tlb_read_access 中发生缺页的次数。
   - tp_core_clusterX_coreY_tlb_write_access: 第 X 个 cluster 中第 Y 个 core 的运算核心发起的写入请求，在 SMMU 处查询页表的次数。
   - tp_core_clusterX_coreY_tlb_write_miss: 第 X 个 cluster 中第 Y 个 core 的 tlb_write_access 中发生缺页的次数。
5. 其它。
   - llcX_tagram_hit: 第 X 个 llc 中 tagram 发生 hit 的次数。
   - llcX_tagram_miss: 第 X 个 llc 中 tagram 发生 miss 的次数。

8 FAQ

问题 1：CNPerf 性能开销如何?

性能开销一般在 10% -15%，硬盘读写等操作会影响性能开销。可以通过调整配置文件或调整 record 参数来减少开销, 具体参见配置 cnperf.json 章节。

问题 2：为什么有时候 timechart 的 json 文件打不开?

chrome://tracing 对内存需求较多，实测下来所需内存是文件大小的 10 倍左右。建议使用内存更大的机器或者调整配置文件，减少追踪内容。

