# CPU代码

## 1. 项目目录树

- `libgalois` contains the source code for the shared-memory Galois library, e.g., runtime, graphs, worklists, etc.
- `lonestar` contains the Lonestar benchmark applications and tutorial examples for Galois
- `libdist` contains the source code for the distributed-memory and heterogeneous Galois library
- `lonestardist` contains the source code for the distributed-memory and heterogeneous benchmark applications. Please refer to `lonestardist/README.md` for instructions on building and running these apps.
- `tools` contains various helper programs such as graph-converter to convert between graph file formats and graph-stats to print graph properties
- `libcusp`，包含了名为cusp的分区策略
- `libgluon`，分布式通信库，传递同步信息，根据通信量的不同选择不同的通信策略
- `libpangolin`，This is the Pangolin framework for efficient and flexible graph mining

## 2. 代码中的文件

bfs_push.cpp代码中有多个文件。

- value.txt，cpu根据度数排序判断出热顶点，并将顶点的信息存放在value.txt。数据的格式是： 顶点id 顶点值
- data.txt，存放的是python端传过来的数据，格式是：热点id 热点值
- mind_graph.txt,cpu端统计的热顶点，数据的格式是 源点 目标顶点
- ok.flag，python端执行完毕后，会创建一个文件，如果检测到这个文件，表示python端已经执行完毕
- stop.flag，host的第一个线程会开启mindspore，执行完毕后会创建该文件，这样后续的线程就不用等待mindspore执行。

## 3. 项目执行

### 3.1 方式1：使用指令

```
GALOIS_DO_NOT_BIND_THREADS=1 mpirun -n 1
/home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/bfs-push-dist
/home/hedonghao/graph/dataset/test1/CPU/dataset_1.gr  -runs=1 -startNode=7 -
converge=false -exec=Async -mindspore=false -topN=1000 -output=true -
outputLocation /home/hedonghao/graph/output/ -t=1
```

GALOIS_DO_NOT_BIND_THREADS=1 mpirun -n 1
/home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/bfs-push-dist
../../../data_Gorder.gr -runs=1 -startNode=1 -converge=true -exec=Async -mindspore=true -
topN=1000

## 3.2 方式2：配置launch.json和task.json

launch.json

```
{
// 使用 IntelliSense 了解相关属性。

    // 悬停以查看现有属性的描述。

    // 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387

    "version": "0.2.0",

    "configurations": [

        {

            "name": "Galois",

            "type": "cppdbg",

            "request": "launch",

            "program":
"/home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/bfs-push-dist",

            "args": [

                "/home/hedonghao/graph/dataset/test3/CPU/dataset_3.gr",

                "-runs=1",

                "-t=1",

                "-startNode=4",

                "-output=true",

                "-outputLocation=/home/hedonghao/graph/output/",

                "-exec=Async",

                "-mindspore=false",
```

```
                "-converge=false",
                "-topN=1000"
            ],
            "stopAtEntry": true,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
            ],
            // "preLaunchTask": "Mytask",
            "miDebuggerPath": "/usr/bin/gdb"
        }
    ]
}
}
}
```

task.json

```
{
"version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "Mytask",
            "command": "make",
            "args": [
```

```
                "-C",

"/home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/",
                "bfs-push-dist",
                "-j"
            ],
            "options": {
                // "cwd": "${workspaceFolder}"
                "cwd":
"/home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
        }
    ]
}
```

c_cpp_properties.json

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "/home/hedonghao/openmpi/include/",
                "/home/hedonghao/llvm/include/"
            ],
            "defines": [],
            "compilerPath": "/usr/local/Ascend/ascend-
toolkit/latest/compiler/ccec_compiler/bin//clang",
            "cStandard": "c11",
            "cppStandard": "c++14",
            "intelliSenseMode": "clang-x64"
        }
    ],
```

```
    "version": 4
}
```

## 4. 备忘录

- 增加删除文件，记得修改子目录下的CMakelist.txt。直接复制文件来新建文件，可能会有重复的全局变量造成冲突。
- 使用-outputLocation，最后一项需要写成目录 比如bfs/ 不能省略这个斜杠
- BFS输出结果如果无法到达这个点，结果值就是1073741823（一个极大值，在NPU端的代码极大值是100），一开始选择的其实点不好，是一个度数为0的点，导致所有的点的值都是1073741823
- 许多创建文件，写入内容的操作都需要先重配置文件夹的权限

# NPU代码

生成数据集：

```
python /home/hedonghao/graph/python_code/mind_graph.py
```

bfs测试：

```
python /home/hedonghao/graph/python_code/Matrix_BFS.py
```

合并生成数据集和执行数据集后的文件

```
import numpy as np
import time
import os
import argparse
import mindspore
from mindspore import Tensor
from mindspore import numpy
from mindspore import dtype as mstype
import mindspore.ops as ops
from mindspore import Profiler

def check_file():
    # if
os.path.exists("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/ID.npy"):
    #
os.remove("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/ID.npy")
    # if
os.path.exists("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/data.npy"):
    #
os.remove("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/data.npy")
    # if
os.path.exists("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/table.npy")
:
```

```
    #
os.remove("/home/hedonghao/graph/dataset/"+directory_name+"/NPU/table.npy")
    if
os.path.exists("/home/hedonghao/graph/output/"+directory_name+"python_output.txt
"):

os.remove("/home/hedonghao/graph/output/"+directory_name+"python_output.txt")

def read_argument():
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', type=str, default = None)
    parser.add_argument('--source_node', type=int, default = None)
    directory_name=parser.parse_args().name
    source_node=parser.parse_args().source_node
    return directory_name,source_node

def read_graph():
    #read_graph读取.txt格式的数据，每一行存放的是源点和目的点
    print("执行mind_graph.py代码，生成图数据")
    t1 = time.time()#统计程序耗时
    filename =
'/home/hedonghao/graph/dataset/'+directory_name+'/NPU/dataset.txt'
    f = open(filename)
    edges = 0#统计边信息
    ID=set()#存放所有顶点，定义一个集合避免重复存
    data=[]#存放边信息，每一行存放的是源点和目的点
    table={}#定义一个字典，存放顶点的ID和顶点下标之间的映射关系，key：顶点ID，value：顶点在
列表中的下标
    for file in f.readlines():
        row = [int(line) for line in file.strip().split()]
        data.append(row)
        ID.add(row[0])
        ID.add(row[1])
        edges = edges + 1

    print("num of edges:",edges)

    ID=list(ID)#把集合转化为列表
    length = len(ID)
    print("num of vertex:",length)
    for k in range(length):
        table[ID[k]] = k

    matrix = np.ones((length,length), dtype=np.int16) * 66000 #生成一个矩阵，矩阵的
大小是length*length，默认值是66000，表明两个点之间不可达
    for i in data:
        matrix[table[i[0]]][table[i[1]]] = 1
    for i in range(length):#矩阵的对角线元素都置为0
        matrix[i][i] = 0

    t2 = time.time()
    print("read graph finished in time:",t2-t1)
    return ID,matrix,table

def BFSMatmul(x,y,eye):
    # matmul = ops.MatMul()
    # x = matmul(x, eye)
    add = ops.Add()
```

```
    sum = add(x, y)
    op = ops.ReduceMin(keep_dims=True)
    output = op(sum,0)
    return output


def process_graph(ID,matrix,table,directory_name,source_node):
    print("执行Matrix_BFS.py代码，使用mindspore进行计算")
    matrix = Tensor(matrix,mstype.float16)
    vertex_num=len(ID)
    eye = Tensor(np.eye(vertex_num),mstype.float16)#np.eye用于生成对角阵
    V = np.ones([vertex_num,vertex_num], dtype=np.int16) * 100#np.ones用于生成单位
阵
    V[source_node] = 0
    V = Tensor(V,mstype.int16)
    finished = False
    itr = 0
    t1 = time.time()
    while(not finished):
        res = BFSMatmul(matrix, V, eye)
        transpose = ops.Transpose()
        res_perm = (1,0)
        V1 = transpose(res,res_perm)
        finished = numpy.array_equal(V,V1)
        if finished == False:
            V = V1
        print("itr",itr,"finished")
        itr=itr+1
    t2 = time.time()
    V = V.asnumpy()
    print("BFS finished in:",t2-t1)
    with
open('/home/hedonghao/graph/output/'+directory_name+'/python_output.txt', 'w') as
f:
        for i in range(len(V)):
            f.write('{0} {1} \n'.format(ID[i], int(V[i][0])))
    f.close()


if __name__ == "__main__":
    #读取参数
    directory_name,source_node=read_argument()
    #开启Profiler
    mindspore.set_context(mode=mindspore.GRAPH_MODE, device_target="Ascend")
    profiler =
Profiler(output_path="/home/hedonghao/graph/output/"+directory_name+"/profiler",
profile_memory=True,aicore_metrics=1,l2_cache=True)#初始化分析器
    # 检查文件
    check_file()
    #读图
    ID,matrix,table=read_graph()
    # 处理图
    process_graph(ID,matrix,table,directory_name,source_node)
    profiler.analyse()#关闭分析器
```

# 1. 备忘录

- Python环境装在root用户下，所以使用之前需要

  sudo su  输入密码：wx1186405

- 下载包使用pip3
- 许多创建文件，写入内容的操作都需要先重配置文件夹的权限

# 执行脚本

```
#!/bin/bash
# 定义变量
dateset_name="MyGenerateData_3.edgelist"
node_num=16
edge_num=128
directory_name="test9"
source_node=2
# # 第一步：生成数据集
/home/hedonghao/generate_graph/g500KronGenerator/yche-bin/yche_generator_omp
${node_num} -e ${edge_num} -o
/home/hedonghao/generate_graph/MyGenerateData/${dateset_name}
# 第二步：数据集格式转化
mkdir -p /home/hedonghao/graph/dataset/${directory_name}/CPU
mkdir -p /home/hedonghao/graph/dataset/${directory_name}/NPU
mkdir -p /home/hedonghao/graph/output/${directory_name}/
mkdir -p /home/hedonghao/graph/Mylog/${directory_name}/
/home/hedonghao/graph/build/tools/graph-convert/graph-convert --edgelist2gr --
edgeType=void /home/hedonghao/generate_graph/MyGenerateData/${dateset_name}
/home/hedonghao/graph/dataset/${directory_name}/CPU/dataset.gr
echo "wx1186405"|sudo chmod 777
"/home/hedonghao/graph/dataset/${directory_name}/CPU/dataset.gr"
# 运行代码
# 执行CPU代码
GALOIS_DO_NOT_BIND_THREADS=1 /home/hedonghao/openmpi/bin/mpirun --allow-run-as-
root -n 1 /home/hedonghao/graph/build/lonestar/analytics/distributed/bfs/bfs-
push-dist /home/hedonghao/graph/dataset/${directory_name}/CPU/dataset.gr  -runs=1
-t=1 -startNode=${source_node} -converge=false -exec=Async -mindspore=false -
topN=1000 -output=true -outputLocation
/home/hedonghao/graph/output/${directory_name}/CPU/
1>>/home/hedonghao/graph/Mylog/${directory_name}/bfs_push.txt
# 执行NPU代码
# 先复制一份数据集
cp -i /home/hedonghao/generate_graph/MyGenerateData/${dateset_name}
/home/hedonghao/graph/dataset/${directory_name}/NPU/dataset.txt
echo "wx1186405"|sudo chmod 777
"/home/hedonghao/graph/dataset/${directory_name}/NPU/dataset.txt"
echo "wx1186405"|sudo python /home/hedonghao/graph/python_code/read_and_run.py --
name=${directory_name} --source_node=${source_node}
1>>/home/hedonghao/graph/Mylog/${directory_name}/python_bfs_push.txt
echo "wx1186405"|sudo chmod 777 -R
/home/hedonghao/graph/output/${directory_name}/profiler/profiler/
```

# 数据集

## 1. Graph500数据集

Graph500是对超级计算机系统的评级，专注于数据密集型负载。

- Graph500规则
  - 最终评价指标：GTEPS，即每秒遍历的边数，G代表10^9次
  - 性能评价规则
    - 评价算法：广度优先搜索算法（BFS）和单源最短路算法(SSSP)
    - 执行方式：随机64个根节点，运行64次BFS或SSSP，取平均性能
    - 执行标准：只判断输出结果的正确性，即可以对BFS或SSSP算法本身做修改
    - 评价规则：BFS或SSSP遍历边数/执行时间
    - 时间统计：仅统计BFS或SSSP执行时间，图构建（预处理）时间不计入内
- 最新的BFS性能榜单

| RANK | PREVIOUS RANK | MACHINE | VENDOR | TYPE | NETWORK | INSTALLATION SITE | LOCATION | COUNTRY | YEAR | APPLICATION | USAGE | NUMBER OF NODES | NUMBER OF CORES | MEMORY | IMPLEMENTATION | SCALE | GTEPS | C_TIME | POWER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Supercomputer Fugaku | Fujitsu | Fujitsu A64FX | Tofu Interconnect D | RIKEN Center for Computational Science (R-CCS) | Kobe Hyogo | Japan | 2020 | Various scientific and industrial fields | Academic and industry | 158976 | 7630848 | 5087232 | Custom | 41 | 102955 | 669.564 | 14961115 |
| 2 | new | Pengcheng Cloudbrain-II | HUST-Pengcheng Lab-HUAWEI | Kunpeng 920+Ascend 910 | Custom | Pengcheng Lab | ShenZhen | China | 2022 | Research | Academic | 488 | 93696 | 999424 | Custom (Graph Processing System "YiTu" implemented by Jin Zhao Yujian Liao Yu Zhang etc.) | 40 | 25242.9 | 52872.1 | |
| 3 | 2 | Sunway TaihuLight | NRCPC | Sunway MPP | Sunway | National Supercomputing Center in Wuxi | Wuxi | China | 2015 | research | research | 40768 | 10599680 | 1304580 gigabytes | Custom | 40 | 23755.7 | 961.455 seconds | |
| 4 | 3 | Wisteria/BDEC-01 (Odyssey) | Fujitsu | PRIMEHPC FX1000 | Tofu interconnect D | Information Technology Center The University of Tokyo | Kashiwa Chiba | Japan | 2021 | University | Research | 7680 | 368640 | 245760 | Custom | 37 | 16118 | 90.7395 | |
| 5 | 4 | TOKI-SORA | Fujitsu | PRIMEHPC FX1000 | Tofu interconnect D | Japan Aerospace eXploration Agency (JAXA) | Tokyo | Japan | 2020 | Research | CFD | 5760 | 276480 | 184320 | Custom | 36 | 10813 | 70.5226 | |
| 6 | 5 | LUMI-C | HPE | HPE Cray EX | HPE Slingshot-10 | EuroHPC/CSC | Kajaani | Finland | 2021 | Research | Various | 1492 | 190976 | 381952 | custom | 38 | 8467.71 | 434.819 | |
| 7 | 6 | OLCF Summit (CPU-Only) | IBM | IBM POWER9 | | Oak Ridge National Laboratory | Oak Ridge TN | United States | 2018 | Government | Scientific Research | 2048 | 86016 | 1048576 | Custom | 40 | 7665.7 | 370.647 | |
| 8 | 7 | SuperMUC-NG | Lenovo | ThinkSystem SD530 Xeon Platinum 8174 24C 3.1GHz Intel Omni-Path | | Leibniz Rechenzentrum | Garching | Germany | 2018 | Academic | Research | 4096 | 196608 | 393216 | custom-aml-heavy-diropt | 39 | 6279.47 | 1547.10 | |
| 9 | 8 | Lise | Atos | Bull Intel Cluster Intel Xeon Platinum 9242 48C 2.3GHz Intel Omni-Path | Intel Omni-Path | Zuse Institute Berlin (ZIB) | Berlin | Germany | 2019 | Research | Academic | 1270 | 121920 | 502272 | custom the same code used on Fugaku | 38 | 5423.94 | 228.899 | |
| 10 | 9 | DepGraph Supernode | HUST && Nvidia | DepGraph (+GPU Tesla A100) | Custom | National Engineering Research Center for Big Data Technology and System | Wuhan | China | 2022 | Research | Academic | 1 | 128 | 512 | Custom (Implementation of Yu Zhang) | 33 | 4623.379 | 2294.102304 | |

一般graph500排行榜数据规模在36到40左右，指的是顶点个数是多少次方。2^36顶点，平均度数16的情况下二进制数据都有几十T了，一般规模的集群是处理不了的。

## 2. 生成数据集

项目目录：

```
/home/hedonghao/generate_graph/g500KronGenerator
```

生成指定要求的图：

```
./yche_generator_omp <# of vertices (log 2 base)> <-e intNumber [optional: average # of edges per vertex, defualt to be 16> <-o outputFileName [optional: default to stdout]> <-s intName [optional: default to use the current time]> <-b [optional: default is ascii version, -b for binary version]>
```

示例：

```
/home/hedonghao/generate_graph/g500KronGenerator/yche-bin/yche_generator_omp 15 -e 32 -o /home/hedonghao/generate_graph/MyGenerateData/MyGenerateData.edgelist
```

参数说明：

<# of vertices (log 2 base)>：顶点个数，2^n

<-e intNumber [optional: average # of edges per vertex, defualt to be 16>：顶点的平均度数，对于无向图来说，一条边会被算两次度数，所以要要想让每条边的平均边为16，需要设置为32。

<-o outputFileName [optional: default to stdout]>：输出文件名

<-s intName [optional: default to use the current time]>：

<-b [optional: default is ascii version, -b for binary version]>：数据集格式

生成的数据文件可能有权限，用chmod修改一下。

# 3. 数据格式转化

项目地址：

运行指令：

```
/home/hedonghao/graph/build/tools/graph-convert/graph-convert --edgelist2gr --edgeType=void
/home/hedonghao/generate_graph/MyGenerateData/MyGenerateData.edgelist
/home/hedonghao/graph/dataset/test5/CPU/dataset_5.gr
```

# 4. 数据集测试

| 数据集参数（第一个参数是顶点个数2^n，第二个参数是顶点相连的边的数目m，默认的是无向图，所以这个平均度数为m/2 | CPU测试时间 | NPU测试结果。第一个参数是迭代次数，第二个参数是执行时间 |
| --- | --- | --- |
| 15 32 ($2^{15*32=2}$20=1048576) | 872us | 5 22.9394 |
| 15 64(2^21=2097152) | 1506us | 2 22.7601 |
| 18 16($2^{18*16=2}$24194304) | 2714us | 占用内存过多，分配失败。log:<br>Out of Memory. Request memory size: 60622316544, device free size 32176603136, Memory Statistic:<br>Device HBM memory size: 32768M<br>MindSpore Used memory size: 30694M<br>MindSpore memory base address: 0x120800000000<br>Total Static Memory size: 8M<br>Total Dynamic memory size: 0M<br>Dynamic memory size of this graph: 0M<br>Please try to reduce 'batch_size' or check whether exists extra large shape.<br>More details can be found in MindSpore's FAQ with keyword 'Out of Memory'.<br>[WARNING] DEVICE(22226,7f37b9ff5740,python):2023-05-24-05:26:15.778.862 [mindspore/ccsrc/plugin/device/ascend/hal/device/ascend_memory_pool.cc:74] CalMemBlockAllocSize] Memory Statistics:<br>Device HBM memory size: 32768M<br>MindSpore Used memory size: 30694M<br>MindSpore memory base address: 0x120800000000<br>Total Static Memory size: 8M<br>Total Dynamic memory size: 0M<br>Dynamic memory size of this graph: 0M |

# 性能分析

简介文档：MindSpore

源码文档：MindSpore Profiler

## 1. 生成结果

```
#包含头文件
import mindspore
from mindspore import Profiler

#设置上下文
mindspore.set_context(mode=mindspore.GRAPH_MODE, device_target="Ascend")
#初始化分析器
profiler =
Profiler(output_path="/home/hedonghao/graph/output/"+directory_name+"/profiler",
profile_memory=True,aicore_metrics=1,l2_cache=True)
#关闭分析器
profiler.analyse()
```

上述操作会在output目录下生成Profiler文件夹，里面存放了性能分析器得到的统计数据，但是该文件夹下的文件默认是隐藏的，所以需要改变权限。

MindSpore Insight：MindSpore

CANN性能分析工具：学习向导-性能分析工具-开发工具-开发工具-6.0.1-CANN商用版-文档首页-昇腾社区 (hiascend.com)

# 2. 文件说明

- profile
  - container：空，不知道干啥用
  - PROF_XXX
    - 多Device场景下，若启动单Profiling采集进程，则仅生成一个PROF_XXX目录，若启动多Profiling采集进程则生成多个PROF_XXX目录，其中Device目录在PROF_XXX目录下生成，每个PROF_XXX目录下生成多少个Device目录与用户实际操作有关，不影响Profiling分析
- 性能数据结果文件命名格式为“模块名*{device_id}*{model_id}*{iter_id}.json*”或“模块名{device_id}*{model_id}*{iter_id}.csv”，其中{device_id}表示设备ID，{model_id}表示模型ID，{iter_id}表示某轮迭代的ID。这些字段可以在完成数据解析后，使用数据解析与导出中的“Profiling数据文件信息查询”功能对结果文件进行查询得出，若查询某些字段显示为N/A（为空）则在导出的结果文件名中不展示。
- 单算子场景的性能数据结果文件名格式为：“模块名*{device_id}*{iter_id}.json”或“模块名*{device_id}*{iter_id}.csv”。
- Host侧的性能数据结果文件名格式为：“模块名.json”或“模块名.csv”。
- 由于性能数据结果文件名中{device_id}、{model_id}或{iter_id}可能为空，故下文中的文件名展示为“模块名**.json*”或“模块名*.csv”。
- 获取到的.json文件可以在Chrome浏览器中输入“chrome://tracing”地址，将.json文件拖到空白处打开，通过键盘上的快捷键（w：放大，s：缩小，a：左移，d：右移）进行查看。

## 2.1 profiling变量说明

参考CANN性能分析工具：学习向导-性能分析工具-开发工具-开发工具-6.0.1-CANN商用版-文档首页-昇腾社区 (hiascend.com)

| 字段 | 含义 |
| --- | --- |
| Job Info | 任务名。 |
| Device ID | 设备ID。 |
| Dir Name | 文件夹名称。 |
| Collection Time | 数据采集时间。 |
| Model ID | 模型ID。 |
| Iteration Number | 总迭代数。 |
| Top Time Iteration | 耗时最长的5个迭代。 |
| Rank ID | 集群场景的节点识别ID。仅解析集群场景的数据文件时展示，非集群场景显示N/A。 |

| 字段 | 说明 |
| --- | --- |
| collection_info | 信息收集。 |
| Collection end time | 信息收集结束时间。 |
| Collection start time | 信息收集开始时间。 |
| Result Size | 信息数据大小，单位MB。 |
| device_info | 设备信息。 |
| AI Core Number | AI Core数量。 |
| AI CPU Number | AI CPU数量。 |
| Control CPU Number | Control CPU数量。 |
| Control CPU Type | Control CPU类型。 |
| Device Id | 设备ID。 |
| TS CPU Number | TS CPU数量。 |
| host_info | Host信息。 |
| cpu_info | Host CPU信息。 |
| CPU ID | Host CPU ID。 |
| Name | Host CPU名称。 |
| Type | Host CPU类型。 |
| Frequency | Host CPU频率。 |
| Logical_CPU_Count | Host逻辑CPU数量。 |
| cpu_num | Host CPU数量。 |
| Host Computer Name | Host设备名。 |
| Host Operating System | Host操作系统。 |
| model_info | 模型信息。 |
| Device Id | 设备ID。 |
| iterations | 迭代统计。 |
| Iteration Number | 迭代次数。 |
| Model Id | 模型ID，根据模型数量显示。 |

| 字段名 | 字段含义 |
| --- | --- |
| AscendCL、GE、Runtime、Task Schduler | 各个组件名称。对应右侧显示的为该组件中调用的接口名称及耗时。 |
| Title | 选择某个组件的接口名称，例如本例选择的为AscendCL组件的aclmdlExecute接口。 |
| Start | 显示界面中时间轴上的时刻点，chrome trace自动对齐，单位ms。 |
| Wall Duration | 同Duration Time，表示当前接口调用耗时，单位ms。 |
| Iter ID | 迭代ID。 |
| Start Time | 开始时间，单位ns。 |
| End Time | 结束时间，单位ns。 |
| Duration Time | 当前接口调用耗时，单位ns。 |

## 2.2 其他说明

- 文件名格式为{name}*{device_id}*{model_id}*{iter_id}.json*和*{name}*{device_id}*{model_id}*{iter_id}.csv。其中{device_id}表示设备ID，{model_id}表示模型ID，{iter_id}表示某轮迭代的ID。
- 生成的summary数据文件中某些字段值为“N/A”时，表示此时该值不存在。
- 导出的数据中涉及到的时间节点（非Timestamp）为系统单调时间只与系统有关，非真实时间。

## 2.3 profiling目录

单Profiling进程

```
└── PROF_000001_20220129014731273_KEDKPORHMAGPGD
    ├── device_0
    │   ├── data
    │   ├── summary
    │   └── timeline
    ├── device_1
    │   ├── data
    │   ├── summary
    │   └── timeline
    └── host
```

多Profiling进程

```
├── PROF_000001_20220129023423432_DIRWEHEIKGIRNCGA
│   ├── device_0
│   │   ├── data
│   │   ├── summary
│   │   └── timeline
│   └── host
└── PROF_000001_20220129023534657_PPJBHOJGFGEQMEIA
    ├── device_1
    │   ├── data
    │   ├── summary
    │   └── timeline
    └── host
```

## 2.4 分析生成结构

在Chrome浏览器中输入**chrome://tracing**，然后将ascend_timeline_display_0.json文件拖到空白处打开，通过键盘上的快捷键（w：放大，s：缩小，a：左移，d：右移），查看当前AI任务运行过程中AscendCL接口调用时间线。

使用MindSpore Insight生成结果

启动MindSpore Insight，并通过启动参数指定summary-base-dir目录(**summary-base-dir是Profiler所创建目录的父目录**)，例如训练时Profiler创建的文件夹绝对路径为 `/home/user/code/data`，则summary-base-dir设为 `/home/user/code`。启动成功后，根据IP和端口访问可视化界面，默认访问地址为 `http://127.0.0.1:8080`。

# 工作安排

- 理论分析npu代码的性能
- 要用真实图，稠密度足够高
- 后续如果证明npu对系统有增益，改进cpu，npu之间的文件传输
- 当前问题
    - 目前的代码未对数据集做彻底切分，cpu筛选出稠密点后，并没有剔除稠密点，而是全部一股脑的算了一遍。这样做的话，除非npu比cpu块很多才有增益。