

# 寒武纪思元MLU270测试

MLU：Machine Learning Unit，MLU。

## 一、寒武纪加速库：

### 1、寒武纪机器学习库（Cambricon Machine Learning Library，CNML）：

是一个基于寒武纪机器学习单元（Machine Learning Unit，MLU）并针对机器学习以及深度学习的编程库。CNML 为用户提供简洁、高效、通用、灵活并且可扩展的编程接口，用于在 MLU 上加速用户各种机器学习和深度学习算法。CNML 提供了丰富的基本算子。通过组合基本算子可以实现多样的机器学习和深度学习算法。

Cambricon 机器学习库 CNML 提供了一套高效、通用、灵活、可扩展的编程接口，用于在 MLU 上加速各种机器学习和深度学习算法。CNML 主要包含以下几个特性：

支持丰富的基本算子。

– 常见神经网络算子：

\* 卷积、反卷积；

\* 池化；

\* 激活算子，如 ReLU、Sigmoid、TANH 等；

\* Local Response Normalization（LRN）、批规范化；

\* Softmax；

\* 全连接；

– 矩阵、向量、标量算子：

\* 矩阵乘；

\* 张量加和减；

\* 张量逻辑运算；

\* 张量变换，如 Crop、Reshape、Slice、Concat 等；

– 神经网络算子：

\* Long Short-Term Memory（LSTM）；

\* BasicRNNPro、RNN；

### 2、CNPlugin：

在 CNML（Cambricon Machine Learning Library 寒武纪机器学习库）层提供一个接口，将 BANG C 语言生成的算子与 CNML 的执行逻辑统一起来。BANG C 语言的 MLU 异构编程解决方案运行在 CNRT (Cambricon Runtime Library，寒武纪运行时库) 之上。因此为了让 BANG C 实现的算子更好

的与 CNML 协同工作，提供了插件算子 CNPlugin，实现了用 BANG C 语言对 CNML 的操作进行扩展。除此之外，还可以支持 CNML 的特性及多种运行模式，如在线、离线、逐层、融合等。

### 3、寒武纪 CNNL（寒武纪神经网络计算库）：

CNNL 是一个基于寒武纪机器学习单元（Machine Learning Unit，MLU）并针对 DNN（深度神经网络）的计算库。CNCL 针对 DNN 应用场景，提供了高度优化的常用算子，同时也为用户提供简洁、高效、通用、灵活并且可扩展的编程接口。

### 4、CNCL（Cambricon Communications Library，寒武纪通信库）：

CNCL 是面向 MLU（Machine Learning Unit）设计的高性能通信库：

- CNCL 帮助应用开发者优化了基于 MLU 进行多机多卡的集合通信（Collective）操作；
- CNCL 支持多种 MLU 处理芯片的互联技术，包括 PCIe、Interlaken、RoCE、Infiniband Verbs 以及 Sockets；
- CNCL 能够根据芯片的互联拓扑关系，自动的选择最优的通信算法和数据传输路径，从而最大化利用传输带宽完成不同的通信操作；

## 二、寒武纪AI框架Cambricon\_PyTorch

当前的 Cambricon PyTorch 适配的**原生 PyTorch 版本为 v1.3.0**。

对原生PyTorch做了进一步扩展，主要修改有：添加MLU设备、实现MLU特有的在线融合模式、部分算子的分发方式，torchvision访问权限和Cache拓展包。

为支持寒武纪 MLU（Machine Learning Unit，机器学习处理器），寒武纪定制了开源深度学习编程框架 PyTorch（以下简称 Cambricon PyTorch）。

Cambricon PyTorch 借助 PyTorch 自身提供的设备扩展接口将 MLU 后端库中所包含的算子操作动态注册到 PyTorch 中，MLU 后端库可处理 MLU 上的张量和神经网络算子的运算。Cambricon PyTorch 会基于 CNML 库在 MLU 后端实现一些常用神经网络算子，并完成一些数据拷贝操作。

Cambricon PyTorch 兼容原生 PyTorch 的 Python 编程接口和原生 PyTorch 网络模型，支持在线逐层、在线融合和离线三种方式执行推理，同时还支持在线逐层训练。网络的权重可以从 pth 格式文件中读取，已支持的分类和检测网络结构由 torchvision 管理，可以从 torchvision 中读取。对于推理任务，Cambricon PyTorch 不仅支持 float16、float32 等网络模型，而且在寒武纪机器学习处理器上能高效地支持 int8 和 int16 网络模型。对于训练任务，支持 float32 及自适应量化网络模型。

## 三、寒武纪CNToolKit

### 1、Compilers

编译类组件主要支持 BANG C 和 BANG C++ 编程语言的编译，生成可以和 Host 端目标平台 obj 文件链接的对象文件并进行优化，还提供丰富的编译选项配合调试和性能分析工具使用。

## 1.1 CNCC

CNCC (Cambricon Compiler Collection, 寒武纪 BANG C 语言编译器)，是基于 Clang 和 LLVM 开发的 BANG C 和 BANG C++ 的编译器主驱动程序，负责将 \*.mlu 的 C 或 C++ 源码文件编译为 \*.s 的 MLISA 汇编文件。

## 1.2 CNAS

CNAS (Cambricon Assembler, 寒武纪 MLISA 语言编译器)，负责将 \*.s 的 MLISA 汇编文件编译为 \*.cncode (REL-Device 端可重定位) 或 \*.cnbin (EXE-Device 端链接后) 或 \*.cnfatbin (集合多个 cnbin 的胖二进制) 或 \*.o (可与 Host 端目标平台链接的) 的 ELF 格式的 obj 文件。

## 2、Tools

工具类组件主要帮助用户解决开发中遇到的功能和性能调试问题。

### 2.1 CNGDB

CNGDB (Cambricon GNU Debugger, 寒武纪 BANG 语言调试工具) 基于 GDB 修改，添加了对 BANG C 和 C++ 语言的支持，可同时调试 Host 端和 Device 端的代码，并遵循 GPL 协议。目前已开放源代码。源代码信息，参见 <https://github.com/Cambricon/CN-GDB>。

### 2.2 CNPerf

CNPerf (Cambricon Performance, 寒武纪性能剖析工具) 是一款针对寒武纪软件栈全栈设计的性能剖析工具。以性能事件为基础，可对 MLU 异构并行编程查找性能瓶颈和热点函数。

## 3、Libraries

运行库类组件主要为用户提供部署环境和开发环境中的运行时支持。各个组件可以定制化独立安装在部署环境或开发环境中，但需要注意各组件之间的版本依赖，以及开发部署两种环境的版本要尽量保持一致。

### 3.1 CNCCodec

CNCCodec (Cambricon Codec Library, 寒武纪硬件编解码库)，为带有视频或图片编解码能力的 MLU 设备提供 Host 端的 API 支持。

### 3.2 CNDev

CNDev (Cambricon Device Interface Library, 寒武纪设备接口库)，主要为上层库或工具提供公共统一的获取硬件设备信息的 API。

### 3.3 CNDrv

CNDrv (Cambricon Driver Interface Library, 寒武纪驱动接口库)，主要为上层库或工具提供异构编程中 Host 和 Device 内存管理、异步执行和控制、设备管理、多卡多机协同能力。

### 3.4 CNRT

CNRT（Cambricon Runtime Library，寒武纪运行时库）提供和 CNDrv 类似的 Host 和 Device 内存管理、异步执行和控制、设备管理、多卡多机协同等功能 API。二者差异在于 CNRT 将 CNDrv 若干 API 做了流程上或逻辑上的友好封装，帮助用户降低编程复杂度。

寒武纪软件栈中在 CNToolkit 之上还有很多高层级的库或框架。例如 CNML（Cambricon Machine Learning Library，寒武纪机器学习库）是一个面向推理场景的引擎算子库，它会依赖 CNRT 提供一些离线模型的解析和执行能力。

### 3.5 CNRTC

CNRTC（Cambricon Runtime Compilation Library，寒武纪运行时编译库），它是 BANG C/C++ 的运行时编译库，通过 API 接收字符串形式的 BANG 语言源码，并创建可在多架构 MLU 上执行的 cnfatbin，然后配合 cndrv 层 Module 解析和 Kernel 执行接口实现 JIT 方式的编译执行。

这个库多用于 AI 框架中的 Runtime 层，用来实现框架级编译器的 JIT 功能。

### 3.6 CNStudio

CNStudio（Cambricon Studio，寒武纪集成开发环境）为用户开发和调试 BANG C/C++ 语言提供 IDE 级的环境。当前 CNStudio 只提供了 Visual Studio Code 的插件，帮助用户在 VSCode 中快速开发、构建、调试 BANG C/C++ 的工程。

### 3.7 CNJPU

CNJPU（Cambricon Edge JPEG Processing Unit Library，寒武纪 JPEG 处理单元库）为 Edge 形态的用户提供调用 JPEG 处理单元的运行支持，在 CNCodec 的编程中会使用或链接此库。

此库只支持 aarch64 平台，提供给 Edge 形态的产品使用，用法为拷贝动态库至 Edge 操作系统对应的目录，或拷贝至 x86\_64 的主机测进行交叉编译和链接使用。

### 3.8 CNION

CNION（Cambricon Edge ION Library，寒武纪 ION 库）为 Edge 形态的用户提供调用 ION 用户态支持，此库是基于 AOSP ION 添加了寒武纪相关函数功能的 ION 库，在 CNCodec 的编程中会使用或链接此库。

此库只支持 aarch64 平台，提供给 Edge 形态的产品使用，用法为拷贝动态库至 Edge 操作系统对应的目录，或拷贝至 x86\_64 的主机测进行交叉编译和链接使用。

## 四、测试整套流程：

**环境准备：**这部分是后期加的，因为考虑到后人还要用，所以标号可能比较混乱。

(1) pytorch要使用图下面第一个容器，我把这个容器外放了端口，用于网络通信，其他的也都能用，如果使用tensorflow另外研究一下。

```
ict@ict-Precision-3630-Tower:~$ su root
密码:
root@ict-Precision-3630-Tower:/home/ict# docker ps -a
```

CONTAINER ID	IMAGE	PORTS	NAMES	COMMAND	CREATED
933a2b8c520a	yellow.hub.cambricon.com/pytorch/pytorch:0.15.0-ubuntu16.04_3		dev_cn	"/bin/bash"	12 days ago
a84d9ef269f9	yellow.hub.cambricon.com/tensorflow/tensorflow:1.4.0-tf1-ubuntu16.04-py2		keen_franklin	"/bin/bash"	2 months ago
ef3bbec19207	yellow.hub.cambricon.com/tensorflow/tensorflow:1.4.0-tf1-ubuntu16.04-py3		ecstatic_wozniak	"/bin/bash"	2 months ago
c2e0e359e31b	yellow.hub.cambricon.com/pytorch/pytorch:0.15.0-ubuntu16.04		pedantic_cannon	"/bin/bash"	2 months ago

(2) 利用容器的ID启动容器，并且进入他给分配的终端，就是root@933a2b8c520a。

```
root@ict-Precision-3630-Tower:/home/ict# docker start 933a2b8c520a
933a2b8c520a
root@ict-Precision-3630-Tower:/home/ict# docker exec -it 933a2b8c520a /bin/bash
root@933a2b8c520a:/# ls
bin    dep_libs_extract  etc    lib    lib64  media  opt    root  sbin  sys  torch  var
boot  dev               home  lib32  libx32  mnt    proc   run   srv   tmp  usr
```

(3) 进入pytorch环境，运行：

```
1 source /torch/venv3/pytorch/bin/activate
```

```
root@933a2b8c520a:/torch/venv3/pytorch/bin# source activate
(pytorch) root@933a2b8c520a:/torch/venv3/pytorch/bin# cd ..
```

可以看到前边多了一个（pytorch）。

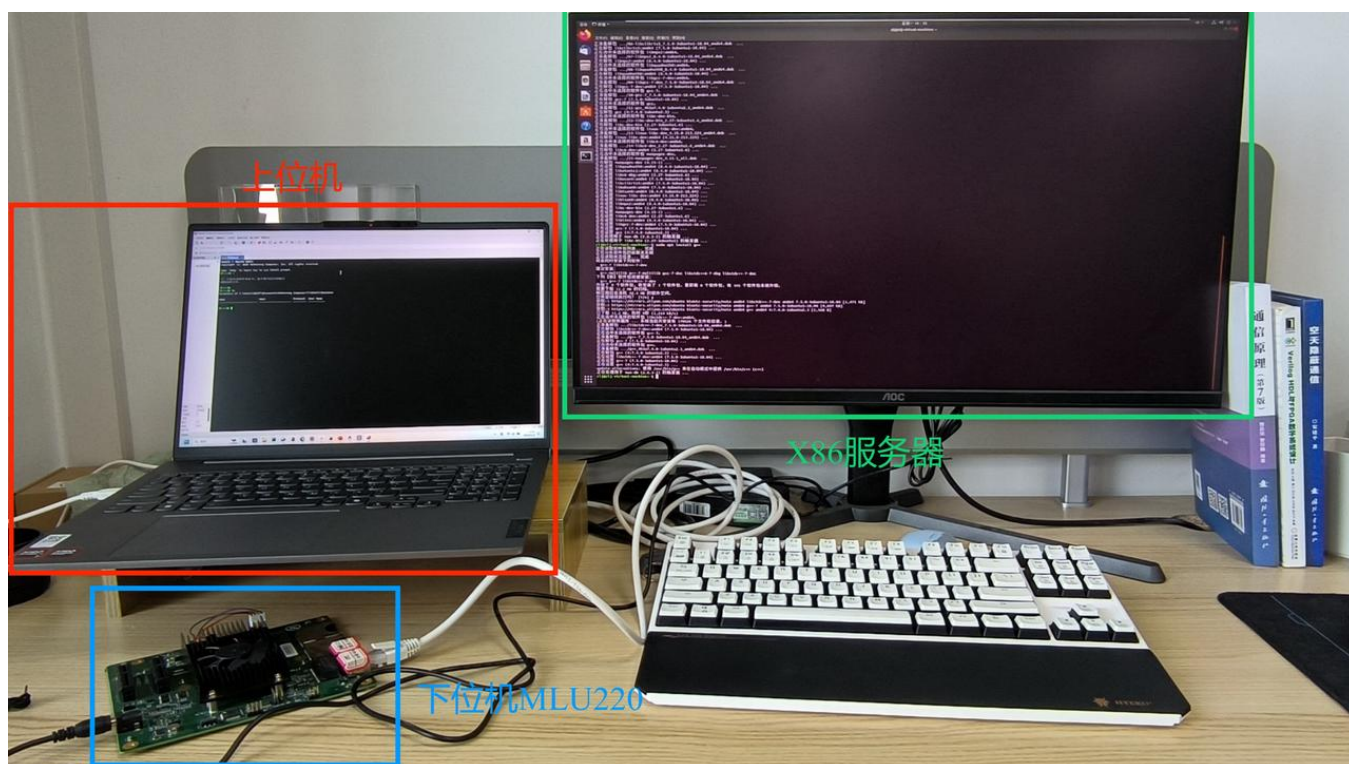
开发、训练环境：

PyTorch:1.12.1

CUDA: 12.3

Python:3.9





1、在x86服务器 + MLU270上的docker容器里，进行CPU模式的推理、量化、MLU逐层推理，MLU融合推理，并生成MLU270的离线模型。（参考文档：寒武纪PyTorch用户手册Cambricon-PyTorch-User-Guide-CN-v0.15.0.pdf 存放位置：cambricon/1.7.0/PyTorch/Cambricon-PyTorch-User-Guide-CN-v0.15.0.pdf）

导入必要模块：

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import torch_mlu.core.mlu_quantize as mlu_quantize
5 import torch_mlu.core.mlu_model as ct
```

1.1 将训练好后的模型保存为.pth权重文件，然后拷贝到x86服务器 + MLU270容器内。

注意：PyTorch 1.6.0后的版本中torch.save默认使用zip文件保存权重.pth，因此PyTorch 1.5及以下无法直接加载，会报错。

```
RuntimeError: ./checkpoint_epoch50.pth is a zip archive (did you mean to use torch.jit.load())?
```

解决方法：最好直接使用与Cambricon\_PyTorch适配的**原生PyTorch 1.3.0**，或者在PyTorch 1.6.0及以上版本中修改参数\_use\_new\_zipfile\_serialization=False，代码如下（重新加载权重，并修改权重文件保存的格式）：

```

1  #在torch 1.6以及更高版本版本中重新加载一下网络参数
2  policy_net = policy_net().to(device) #实例化模型并加载到cpu或GPU中
3  policy_net.load_state_dict(torch.load(model_cp_pth)) #加载模型参数，model_cp
  为之前训练好的模型参数（zip格式）
4  #重新保存网络参数，此时注意改为非zip格式
5  torch.save(policy_net.state_dict(), model_cp_pth,
  _use_new_zipfile_serialization=False)

```

## 1.2 CPU模式模型量化

使用模型量化工具对模型权，并保存量化后的权重，此处命名为

`policyNet_cp_quantization.pth`，代码如下：

```

1  # CPU模式模型量化
2  checkpoint = torch.load('./cp_epoch50.pth', map_location='cpu')
3  # 加载
4  policy_net.load_state_dict(checkpoint)
5  # 为了加速推理，量化为int8、int16
6  net_quantization = mlu_quantize.quantize_dynamic_mlu(policy_net,
  dtype='int8', gen_quant=True)
7  # 保存量化模型
8  torch.save(net_quantization.state_dict(), 'policyNet_cp_quantization.pth')

```

## 1.3 MLU逐层推理

MLU逐层推理，代码如下：

```

1  # step 1 使用quantize_dynamic_mlu接口替换网络中需要量化的算子为Cambricon PyTorch对
  应的自定义算子
2  net_quantization = mlu_quantize.quantize_dynamic_mlu(policy_net)
3  # step 2 加载量化权重
4  net_quantization.load_state_dict(torch.load('policyNet_cp_quantization.pth'),
  False)
5  # step 3 模型和数据拷贝到MLU上
6  net_quantization_mlu = net_quantization.to(ct.mlu_device())
7  input_h_mlu = input_h.to(ct.mlu_device())
8  input_o_mlu = input_o.to(ct.mlu_device())
9  # step 4 逐层推理
10 x, h = net_quantization_mlu(input_o_mlu, input_h_mlu)
11 print(type(x), type(h)) #<class 'torch.Tensor'>
12 print(x.cpu()) #放回当CPU输出

```

**结果：出现[warning]。**

```

MLU
[WARNING][pytorch/catch/torch_mlu/csrc/aten/operators/op_methods.cpp][line:173][addmm][thread:140419487147776][process:2959]:
addmm Op cannot run on MLU device, start running on CPU!
[WARNING][pytorch/catch/torch_mlu/csrc/aten/operators/op_methods.cpp][line:173][addmm][thread:140419487147776][process:2959]:
addmm Op cannot run on MLU device, start running on CPU!
[WARNING][pytorch/catch/torch_mlu/csrc/aten/operators/op_methods.cpp][line:1936][sigmoid_][thread:140419487147776][process:2959]:
sigmoid_ Op cannot run on MLU device, start running on CPU!
[WARNING][pytorch/catch/torch_mlu/csrc/aten/operators/op_methods.cpp][line:1936][sigmoid_][thread:140419487147776][process:2959]:
sigmoid_ Op cannot run on MLU device, start running on CPU!

```

查看文档，MLU暂不支持addmm、sigmoid\_算子，因此涉及这种算子的代码会把数据放到CPU上运行，然后把结果拷贝回MLU，因此不影响代码运行。

### 7.2.1.1 对 MLU 不支持算子的处理

对于 MLU 暂不支持的算子，如果未在 catch/torch\_mlu/tools/mlu\_functions.yaml 中声明，程序会直接终止，并抛出无法分发到 MLU 设备的异常；如果已经在 catch/torch\_mlu/tools/mlu\_functions.yaml 中声明但在运行至 wrapper 或 kernel 时失败，输入数据将会拷贝到 CPU 上，然后调用 CPU 相关算子，使其在 CPU 上运行，最后再将输出结果拷回到 MLU 上。具体实现，可以查询 op\_methods.cpp，该文件在 catch/torch\_mlu/csrc/aten/operators/ 目录下。

## 1.4 融合推理

```

1  # fusion infer
2  ct.save_as_cambricon("policyNet_offline")
3  traced_model = torch.jit.trace(net_quantization.forward, x_mlu,
    check_trace=False)
4  print(type(traced_model))
5  x, h = traced_model(x_mlu)

```

出现问题：RNN中使用了MLU不适配的算子，无法进行融合推理，无法生成寒武纪离线模型，但是在mlu\_functions.yaml中这两个算子均有声明。

```

[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140554987390720][process:28453]:
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: addmm
%33 : Float(*, *) = aten::addmm(%16, %input.5, %30, %31, %31), scope: RnnAgent/GRUCell[rnn] # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/nn/modules/rnn.py:1023:0

[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140554987390720][process:28453]:
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: sigmoid_
%52 : Tensor = aten::sigmoid_(%51), scope: RnnAgent/GRUCell[rnn] # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/nn/modules/rnn.py:1023:0

```



```
20000].
Graph is segmented into 4 subgraphs.
Please check the above WARNING logs which include "[Fusion Segment]" to see whether there are MLU unsupported ops exists in pytorch model.

Traceback (most recent call last):
  File "test_mlu_copy.py", line 87, in <module>
    x, h = traced_model(x_mlu)
  File "/torch/venv3/pytorch/lib/python3.6/site-packages/torch/nn/modules/module.py", line 541, in __call__
    result = self.forward(*input, **kwargs)
RuntimeError: outputs_[i]->uses().empty() INTERNAL ASSERT FAILED at /pytorch/torch/csrc/jit/ir.cpp:1027, please report a bug to PyTorch. (eraseOutput at /pytorch/torch/csrc/jit/ir.cpp:1027)
```

## TracedModule打印:

```
TracedModule[RnnAgent](
  original_name=RnnAgent
  (enc): TracedModule[Sequential](
    original_name=Sequential
    (0): TracedModule[MLULinear](original_name=MLULinear)
    (1): TracedModule[ReLU](original_name=ReLU)
    (2): TracedModule[MLULinear](original_name=MLULinear)
    (3): TracedModule[ReLU](original_name=ReLU)
  )
  (rnn): TracedModule[GRUCell](original_name=GRUCell)
  (f_out): TracedModule[MLULinear](original_name=MLULinear)
)
```

## 解决方案:

- 在Python 3.6、PyTorch 1.3.0、cudatoolkit 9.2环境重新训练，尽可能保证算子适配:

训练出现问题:

```
1 RuntimeError: CUDA error: CUBLAS_STATUS_EXECUTION_FAILED when calling
  cublasSgemm( handle, opa, opb, m, n, k, &alpha, a, lda, b, ldb, &beta, c,
  ldc)
```

将代码放到CPU、高版本CUDA跑，没有出现问题，所以不是代码的原因，升级CUDA版本10.1，可以运行，因此**推荐开发、训练环境：python 3.7、Pytorch 1.3.0、cudatoolkit 10.1。**

在**python 3.7、Pytorch 1.3.0、cudatoolkit 10.1**该环境中重新训练，报相同的错误。

- 把不适配的算子重新修改一下，拆成几个操作的组合，并且用现有的PyTorch函数的实现，防止无法计算梯度等各种问题，**难度比较大。**

2024.3.12

使用论文[Reconfigurable Intelligent Surface Assisted Multiuser MISO Systems Exploiting Deep Reinforcement Learning](#)中模型，**state**是传输功率，接收功率，还有t-1时隙的信道矩阵，输入尺寸为(1, 128)，**action**是传输波束赋形矩阵和相位移矩阵，尺寸为(1, 40)，更换模型：

```
1  def __init__(self, state_dim, action_dim, M, N, K, power_t, device,
2      max_action=1):
3      super(Actor, self).__init__()
4      hidden_dim = 1 if state_dim == 0 else 2 ** (state_dim -
5          1).bit_length()
6
7      self.device = device
8
9      self.M = M
10     self.N = N
11     self.K = K
12     self.power_t = power_t
13
14     self.l1 = nn.Linear(state_dim, hidden_dim)
15     self.l2 = nn.Linear(hidden_dim, hidden_dim)
16     self.l3 = nn.Linear(hidden_dim, action_dim) #会被量化为cnq.MLULinear
17
18     self.bn1 = nn.BatchNorm1d(hidden_dim) #不会被量化
19     self.bn2 = nn.BatchNorm1d(hidden_dim)
20
21     self.max_action = max_action
```

逐层推理和融合推理都没有出现问题，并逐层推理和融合推理结果一样，但是与cpu推理有一定精度差距（因为量化模型需要将模型权重参数为int8、int16等整形数据），最后生成寒武纪**MLU270离线模型**actor\_offline.cambricon。

之后一般不会降低多少精度，有的甚至会提高精度。寒武纪软件栈针对卷积、全连接算子等必须要进行量化后才能运行；而其他如激活算子、BN 算子等不需要量化，直接使用浮点型计算。

而且通过文档得知，BN算子BatchNorm1d不会被量化，直接使用浮点型计算。

原生的 PyTorch 算子	替换后的 MLU 算子
nn.Linear	cnq.MLULinear
nn.Conv3d	cnq.MLUConv3d
nn.Conv2d	cnq.MLUConv2d
nn.Conv1d	cnq.MLUConv1d
nn.ConvTranspose3d	cnq.MLUConvTranspose3d
nn.ConvTranspose2d	cnq.MLUConvTranspose2d
nn.LocalResponseNorm	cnq.MLULocalResponseNorm
nn.LSTM	cnq.MLULSTM
nn.GRU	cnq.MLUGRU
nn.InstanceNorm2d	cnq.MLUInstanceNorm2d
nn.MaxUnpool2d	cnq.MLUMaxUnpool2d

文档里原生与模型量化后替换的算子表格中并没有nn.GRUcell，只有nn.GRU，所以尝试使用GRU重新训练（TODO）。

2、编写运行离线模型的CNRT（C++）们在x86服务器+MLU270上的容器中运行正常后，生成MLU220的离线模型。（参考文档：寒武纪运行时库用户手册 Cambricon-CNRT-User-Guide-CN-v4.10.1.pdf 位置 cambricon/1.7.0/CNtoolkit/cnrt/Cambricon-CNRT-User-Guide-CN-v4.10.1.pdf）

文档中说的开发样例位于/usr/local/neuware/samples/cnrt/下。

## 2.1 环境准备

- CNRT驱动、CNRT头文件 `/usr/local/neuware/include`、动态链接库 `/usr/local/neuware/lib64` 或者 `/usr/local/neuware/lib` 目录下。

- 已经生成的离线模型文件：`model.cambricon` 和 `model.cambricon_twins`（`cat odel.cambricon_twins` 可以查看离线模型的具体信息），把`core_version`设置为MLU220，然后利用前边的步骤保存为MLU220离线模型即可。

```
1  ct.set_core_number(4)
2  ct.set_core_version("MLU220")
3  torch.set_grad_enabled(False)
```

- CMake：用于编译
- glog（用于日志记录，可选）

## 2.2 编写CMakeList.txt

- 文件结构整理如下：

bin：存放输出文件（可执行文件）

build：用于存放编译后的文件

src：存放源文件

```
(pytorch) root@c2e0e359e31b:/home/ict/test_mlu270/newtry/cnrt_test# tree
.
├── CMakeLists.txt
├── bin
├── build
└── src
    ├── CMakeLists.txt
    └── main.cpp
```

- 外层CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2  project(demo_cnrt)
3
4  add_subdirectory(src)
```

- src/CMakeLists.txt

```
1  # add cnrt
2  include_directories(/usr/local/neuware/include) #x86
3  link_directories(/usr/local/neuware/lib64) #x86
4  link_libraries(cnrt)
```

```

5
6 aux_source_directory(. SRC_LIST)
7 add_executable(democnrt ${SRC_LIST}) #执行
8
9 set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin) #将输出文件保存到bin目录
10 target_link_libraries(democnrt cnrt) # (可选)

```

- 在build文件中：

```

1 cmake .. && make
2
3 """"
4 -- The C compiler identification is GNU 7.3.0
5 -- The CXX compiler identification is GNU 7.3.0
6 -- Check for working C compiler: /usr/bin/cc
7 -- Check for working C compiler: /usr/bin/cc -- works
8 -- Detecting C compiler ABI info
9 -- Detecting C compiler ABI info - done
10 -- Detecting C compile features
11 -- Detecting C compile features - done
12 -- Check for working CXX compiler: /usr/bin/c++
13 -- Check for working CXX compiler: /usr/bin/c++ -- works
14 -- Detecting CXX compiler ABI info
15 -- Detecting CXX compiler ABI info - done
16 -- Detecting CXX compile features
17 -- Detecting CXX compile features - done
18 -- Configuring done
19 -- Generating done
20 -- Build files have been written to:
   /home/ict/test_mlu270/newtry/cnrt_test/build
21 Scanning dependencies of target democnrt
22 [ 50%] Building CXX object src/CMakeFiles/democnrt.dir/main.cpp.o
23 [100%] Linking CXX executable ../../bin/democnrt
24 [100%] Built target democnrt
25 """"

```

- 运行无误之后在bin文件中生成可执行文件，运行：

```

1 ./democnrt

```

## 2.3 编写CNRT代码，运行、DEBUG MLU270离线模型



根据官方给的例程修改即可，

```
1  #include "cnrt.h"
2  #include <iostream>
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  using namespace std;
8
9  int offline_test(const char *name) {
10     // This example is used for MLU270 and MLU220. You need to choose
    the corresponding offline model.
11     // when generating an offline model, u need cnml and cnrt both
12     // when running an offline model, u need cnrt only
13     //在调用任何 CNRT API 之前需要调用此 API 初始化环境,保证线程安全。
14     //1. init
15     cnrtInit(0);
16
17     //2. check mlu device, set device
18     cnrtDev_t dev;
19     cnrtGetDeviceHandle(&dev, 0);
20     cnrtSetCurrentDevice(dev);
21
22     //3. load model
23     cnrtModel_t model;
24     cnrtLoadModel(&model, "../model/actor_mlu270_offline.cambricon");
25     cout << "model:" << model << "\n";
26
27     //4. get model propertys
28     //get model total memory
29     int64_t totalMem;
30     cnrtGetModelMemUsed(model, &totalMem);
31     printf("total memory used:%ld Bytes\n", totalMem);
32     //get model parallelism
33     int model_parallelism;
34     cnrtQueryModelParallelism(model, &model_parallelism);
35     printf("model parallelism:%d.\n", model_parallelism);
36     //5. build inference logic flow
37     cnrtFunction_t function;
38     cnrtCreateFunction(&function);
39     char *func_name = NULL;
40     int func_name_size = 0;
41     cnrtGetFunctionSymbol(model, 0, &func_name, &func_name_size);
42     cout << "func_name:" << func_name << "\n";
43     //生成的模型推理结构其实被当做一个内核函数，函数名就是这个subnet0
```

```

44         cnrtExtractFunction(&function, model, func_name);
45
46         //6. get size of input, output
47         int inputNum, outputNum;
48         int64_t *inputSizeS, *outputSizeS;
49         cnrtGetInputDataSize(&inputSizeS, &inputNum, function);
50         cnrtGetOutputDataSize(&outputSizeS, &outputNum, function);
51         cout << "inputNum:" << inputNum << " inputSizeS:" << *inputSizeS <<
"\n";
52         cout << "outputNum:" << outputNum << " outputSizeS:" << *outputSizeS
<< "\n";
53
54         //7. prepare data on cpu
55         //申请CPU上的输入输出内存
56         void **inputCpuPtrS = (void **)malloc(inputNum * sizeof(void *));
57         void **outputCpuPtrS = (void **)malloc(outputNum * sizeof(void *));
58
59         //8. allocate I/O data memory on MLU
60         //申请MLU上的输入输出内存, 申请数据节点的句柄, 没有真正申请内存
61         void **inputMluPtrS = (void **)malloc(inputNum * sizeof(void *));
62         void **outputMluPtrS = (void **)malloc(outputNum * sizeof(void *));
63
64         //prepare input buffer
65         //真正申请内存空间, 类比cuda的话就是申请显存
66         for (int i = 0; i < inputNum; i++) {
67             //converts data format when using new interface model
68             inputCpuPtrS[i] = malloc(inputSizeS[i]);
69             //malloc mlu memory
70             cnrtMalloc(&(inputMluPtrS[i]), inputSizeS[i]);
71             cnrtMemcpy(inputMluPtrS[i], inputCpuPtrS[i], inputSizeS[i],
CNRT_MEM_TRANS_DIR_HOST2DEV);
72         }
73
74         //prepare output buffer
75         for (int i = 0; i < outputNum; i++) {
76             outputCpuPtrS[i] = malloc(outputSizeS[i]);
77             //malloc mlu memory
78             cnrtMalloc(&(outputMluPtrS[i]), outputSizeS[i]);
79         }
80
81         //prepare parameters for cnrtInvokeRuntimeContext
82         void **param = (void **)malloc(sizeof(void *) * (inputNum +
outputNum));
83         for (int i = 0; i < inputNum; ++i) {
84             param[i] = inputMluPtrS[i];
85         }
86         for (int i = 0; i < outputNum; ++i) {

```

```
87         param[inputNum + i] = outputMluPtrS[i];
88     }
89
90     //setup runtime ctx
91     cnrtRuntimeContext_t ctx;
92     cnrtCreateRuntimeContext(&ctx, function, NULL);
93
94     //bind device
95     cnrtSetRuntimeContextDeviceId(ctx, 0);
96     cnrtInitRuntimeContext(ctx, NULL);
97
98     //compute offline
99     cnrtQueue_t queue;
100    cnrtRuntimeContextCreateQueue(ctx, &queue);
101
102    // invoke
103    cnrtInvokeRuntimeContext(ctx, param, queue, NULL);
104
105    // sync
106    cnrtSyncQueue(queue);
107
108    // copy mlu result to cpu
109    for (int i = 0; i < outputNum; i++) {
110        cnrtMemcpy(outputCpuPtrS[i], outputMluPtrS[i],
111        outputSizeS[i], CNRT_MEM_TRANS_DIR_DEV2HOST);
112    }
113
114    //free memory space
115    for (int i = 0; i < inputNum; i++) {
116        free(inputCpuPtrS[i]);
117        cnrtFree(inputMluPtrS[i]);
118    }
119    for (int i = 0; i < outputNum; i++) {
120        free(outputCpuPtrS[i]);
121        cnrtFree(outputMluPtrS[i]);
122    }
123    free(inputCpuPtrS);
124    free(outputCpuPtrS);
125    free(param);
126
127    cnrtDestroyQueue(queue);
128    cnrtDestroyRuntimeContext(ctx);
129    cnrtDestroyFunction(function);
130    cnrtUnloadModel(model);
131    cnrtDestroy();
132
133    return 0;
```

```

133 }
134
135 int main() {
136     printf("offline test\n");
137     offline_test("mlp");
138
139     return 0;
140 }

```

3、上一步运行没问题之后，在上面代码中把cnrtLoadModel加载的模型改称MLU220的离线模型actor\_mlu220\_offline.cambricon。准备在X86服务器上交叉编译，生成aarch64可执行文件，将aarch64可执行文件和mlu220离线模型文件通过网络拷贝到MLU220 SOC开发板的硬盘上。

### 3.1 安装交叉编译工具链gcc-linaro-6.2.1-2016.11-x86\_64\_aarch64-linux-gnu

```

1  mkdir /usr/local/arm #创建arm目录
2  mv ./gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-gnu.tar.xz ./usr/local/arm
   #把交叉编译工具链拷贝过来
3  cd /usr/local/arm
4  tar -vxf gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-gnu.tar.xz #解压
5  cd ./gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-gnu/bin
6  export PATH=$PATH:$PWD/ #配置环境变量，推荐写入~/.bashrc，否则关闭终端丢失。
7  aarch64-linux-gnu-gcc -v #检测是否安装成功
8
9  ""打印下面信息即安装成功
10 Using built-in specs.
11 COLLECT_GCC=aarch64-linux-gnu-gcc
12 COLLECT_LTO_WRAPPER=/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-
   linux-gnu/bin/../../libexec/gcc/aarch64-linux-gnu/6.2.1/lto-wrapper
13 Target: aarch64-linux-gnu
14 Configured with: /home/tcwg-buildslave/workspace/tcwg-make-
   release/label/docker-trusty-amd64-tcwg-build/target/aarch64-linux-
   gnu/snapshots/gcc-linaro-6.2-2016.11/configure SHELL=/bin/bash --with-
   mpc=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-
   amd64-tcwg-build/target/aarch64-linux-gnu/_build/builds/destdir/x86_64-
   unknown-linux-gnu --with-mpfr=/home/tcwg-buildslave/workspace/tcwg-make-
   release/label/docker-trusty-amd64-tcwg-build/target/aarch64-linux-
   gnu/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gmp=/home/tcwg-
   buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-
   build/target/aarch64-linux-gnu/_build/builds/destdir/x86_64-unknown-linux-gnu
   --with-gnu-as --with-gnu-ld --disable-libstdcxx-pch --disable-libmudflap --
   with-cloog=no --with-ppl=no --with-isl=no --disable-nls --enable-c99 --enable-
   gnu-indirect-function --disable-multilib --with-arch=armv8-a --enable-fix-

```

```
cortex-a53-835769 --enable-fix-cortex-a53-843419 --enable-multiarch --with-
build-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-
trusty-amd64-tcwg-build/target/aarch64-linux-gnu/_build/sysroots/aarch64-
linux-gnu --enable-lto --enable-linker-build-id --enable-long-long --enable-
shared --with-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-
release/label/docker-trusty-amd64-tcwg-build/target/aarch64-linux-
gnu/_build/builds/destdir/x86_64-unknown-linux-gnu/aarch64-linux-gnu/libc --
enable-languages=c,c++,fortran,lto --enable-checking=release --disable-
bootstrap --build=x86_64-unknown-linux-gnu --host=x86_64-unknown-linux-gnu --
target=aarch64-linux-gnu --prefix=/home/tcwg-buildslave/workspace/tcwg-make-
release/label/docker-trusty-amd64-tcwg-build/target/aarch64-linux-
gnu/_build/builds/destdir/x86_64-unknown-linux-gnu
15 Thread model: posix
16 gcc version 6.2.1 20161016 (Linaro GCC 6.2-2016.11)
17 ""
```

3.2 因为交叉编译需要链接MLU220的crt动态链接库（aarch64架构），因此通过ssh拷贝MLU220的crt动态链接库到X86服务器，在宿主机端输入：

```
1 scp -r root@192.168.100.50:/neuware C:\Users\Lenovo\neuware #mlu220 -> 宿主机
```

再通过U盘拷贝到X86服务器，在X86服务器创建目录 `/usr/local/neuware220/`，将动态链接库文件拷贝进去。

3.3 编写CMakeLists文件：

- 文件目录结构如下：

```
root@c2e0e359e31b:/home/ict/test_mlu270/newtry/cnrt_test# tree
.
├── CMakeLists.txt
├── bin
├── build
├── model
│   ├── actor_mlu270_offline.cambricon
│   └── actor_mlu270_offline.cambricon_twins
└── src
    ├── CMakeLists.txt
    └── main.cpp
```

- 外层CMakeLists：



```

1  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2  project(demo_cnrt)
3
4  SET(CMAKE_SYSTEM_NAME Linux)
5  SET(CMAKE_SYSTEM_PROCESSOR aarch64)
6
7  add_compile_options(-std=c++11)
8  SET(CMAKE_C_COMPILER "aarch64-linux-gnu-gcc")
9  SET(CMAKE_CXX_COMPILER "aarch64-linux-gnu-g++")
10
11  add_subdirectory(src)

```

- src/CMakeLists:

```

1  # add cnrt
2  include_directories(/usr/local/neuware/include)
3  link_directories(/usr/local/neuware220/neuware/lib64/) #mlu220 cnrt动态链接库的
   目录
4  link_libraries(cnrt)
5
6  # add aarch64-linux-gnu-gcc
7  include_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-
   linux-gnu/include/)
8  link_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-
   gnu/aarch64-linux-gnu/lib64/)
9
10  aux_source_directory(. SRC_LIST)
11  add_executable(democnrt ${SRC_LIST})
12
13  set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
14
15  target_link_libraries(democnrt cnrt)

```

### 3.3 编译、生成arm可执行文件。

```

1  cd build/
2  cmake .. && make
3
4  -- The C compiler identification is GNU 7.3.0
5  -- The CXX compiler identification is GNU 7.3.0
6  -- Check for working C compiler: /usr/bin/cc
7  -- Check for working C compiler: /usr/bin/cc -- works
8  -- Detecting C compiler ABI info

```

```

9  -- Detecting C compiler ABI info - done
10 -- Detecting C compile features
11 -- Detecting C compile features - done
12 -- Check for working CXX compiler: /usr/bin/c++
13 -- Check for working CXX compiler: /usr/bin/c++ -- works
14 -- Detecting CXX compiler ABI info
15 -- Detecting CXX compiler ABI info - done
16 -- Detecting CXX compile features
17 -- Detecting CXX compile features - done
18 -- Configuring done
19 -- Generating done
20 -- Build files have been written to:
    /home/ict/test_mlu270/newtry/cnrt_test/build
21 Scanning dependencies of target democnrt
22 [ 50%] Building CXX object src/CMakeFiles/democnrt.dir/main.cpp.o
23 [100%] Linking CXX executable ../../bin/democnrt
24 [100%] Built target democnrt
25 """"
26 file ../../bin/democnrt #下边打印ARM aarch64即交叉编译成功
27 """"
28 ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked,
29 interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0,
    BuildID[sha1]=12a9f3ba441d8326d83f030d67d1ce084a0ac019, not stripped
30 """"

```

### 3.4 将可执行文件与离线模型拷贝至MLU220

- 压缩aarch64可执行文件与离线模型。

```

1  tar -czvf mlu220_run.tar.gz model bin
2  """"
3  model/
4  model/actor_mlu220_offline.cambricon_twins
5  model/actor_mlu220_offline.cambricon
6  bin/
7  bin/democnrt
8  """"

```

- 在宿主机ssh拷贝到MLU220的/cambricon目录下，防止掉电丢失。

```

1  scp -r G:\mlu220_run.tar.gz root@192.168.100.50:/cambricon #宿主机 -> mlu220

```

- 在MLU220中，解压，运行aarch64可执行文件。

```
1 cd /cambricon
2 mv mlu220_run.tar.gz ./mlu220_run
3 cd ./mlu220_run
4 tar -xzvf ./mlu220_run.tar.gz
5 cd bin
6 ./democnrt #运行
```

```
[root@cambricon /cambricon/mlu220_run/bin]# ./democnrt
offline test
CNRT: 4.10.1 a884a9a
model:0x1c8bfd70
total memory used:140448 Bytes
model parallelism:4.
func_name:subnet0
inputNum:1 inputSizeS:448
outputNum:1 outputSizeS:160
[root@cambricon /cambricon/mlu220_run/bin]#
```

流程完成。

## 五、使用CNRT（CNRT代码：离线推理流程）

主要依据例程：/torch/examples/offline/test\_forward\_offline/test\_forward\_offline.cpp，并查阅寒武纪CNRT开发者文档（Cambricon-CNRT-Developer-Guide-EN-v4.10.1.pdf）。

### 1、初始化：初始化环境，并且获取设备句柄，设置当前使用的设备。

```
1 cnrtInit(0);
2 cnrtDev_t dev;
3 cnrtGetDeviceHandle(&dev, 0);
4 cnrtSetCurrentDevice(dev);
```

### 2、加载模型，并且获取一些必要信息

2.1 测试就加载270的离线模型，交叉编译则加载220的离线模型，可以使用一些API获取模型的一些信息，比如占用内存 `cnrtGetModelMemUsed`，并行度 `cnrtQueryModelParallelism` 等等。

```
1 cnrtModel_t model;
2 cnrtLoadModel(&model, "../model/actor_mlu270_offline.cambricon");
3 cnrtGetModelMemUsed(model, &totalMem);
```

```
4 cnrtQueryModelParallelism(model, &model_parallelism);
```

## 2.2 把模型转成一个内核函数，用于调用，这里函数名字就是function。

`cnrtCreateRuntimeContext` 在MLU上基于CNRT function创建CNRT function的上下文描述，存放到`rt_ctx_`中。作用：将计算资源与function本身解除绑定，实现同一个CNRT function可以根据不同的配置，实例化function到不同的MLU硬件设备。

```
1 cnrtFunction_t function;
2 cnrtCreateFunction(&function);
3 cnrtGetFunctionSymbol(model, 0, &func_name, &func_name_size);
4 cnrtExtractFunction(&function, model, func_name);
5 cnrtCreateRuntimeContext(&rt_ctx_, function, NULL);
```

## 2.3 获取模型的输入输出规模（Bytes）、输入输出数据类型

```
1 int inputNum, outputNum;
2 int64_t *inputSizeS, *outputSizeS;
3 cnrtGetInputDataSize(&inputSizeS, &inputNum, function);
4 cnrtGetOutputDataSize(&outputSizeS, &outputNum, function);
5 cnrtDataType_t* input_data_type = nullptr;
6 cnrtDataType_t* output_data_type = nullptr;
7 cnrtGetInputDataType(&input_data_type, &inputNum, function);
8 cnrtGetOutputDataType(&output_data_type, &outputNum, function);
```

# 3、在系统内存上分配 I/O 数据空间并准备输入数据

## 3.1 分配输入空间

- `cnrtDataTypeSize` 可以获取输入数据类型的大小，这里是4 Bytes，因此输入规模（Bytes） / 数据类型大小（Bytes） = 输入尺寸（无量纲），本模型输入尺寸为112。
- 申请输入尺寸大小的databuf，然后把数据读入databuf（这里采用随机数），然后放到申请好的内存空间`inputCpuPtrS`上。
- `cnrtGetInputDataShape` 获取输入数据的尺寸，正常数据输入的格式是NCHW，这个API返回的是NHWC，注意调整。

```
1 void **inputCpuPtrS = reinterpret_cast<void **>(malloc(inputNum * sizeof(void *)));
2 void **outputCpuPtrS = reinterpret_cast<void **>(malloc(outputNum * sizeof(void *)));
```

```

3  vector<int> out_count;
4  unsigned int in_n, in_c, in_h, in_w;
5  for (int i = 0; i < inputNum; i++) {
6      int ip = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
7      auto databuf = reinterpret_cast<float *>(malloc(sizeof(float) * ip));
8      rand_data(databuf, ip); //data in databuf
9      inputCpuPtrS[i] = reinterpret_cast<void*>(databuf); // NCHW
10     vector<int> shape(4, 1);
11     int dimNum = 4;
12     cnrtGetInputDataShape((int**)&shape, &dimNum, i, function); // NHWC
13     in_n = shape[0];
14     in_c = (input_data_type[i] == CNRT_UINT8) ? (shape[3] - 1) : shape[3];
15     in_h = shape[1];
16     in_w = shape[2];
17 }

```

## 3.2 分配输出空间

与上面流程类似。

```

1  vector<float *> output_cpu;
2  unsigned int out_n, out_c, out_h, out_w;
3  for (int i = 0; i < outputNum; i++) {
4      int op = outputSizeS[i] / cnrtDataTypeSize(output_data_type[i]);
5      float* outcpu = reinterpret_cast<float*>(malloc(op * sizeof(float)));
6      out_count.push_back(op);
7      output_cpu.push_back(outcpu);
8      outputCpuPtrS[i] = reinterpret_cast<void*>(outcpu);
9      std::vector<int> shape(4, 1);
10     int dimNum = 4;
11     cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function); // NHWC
12     out_n = shape[0];
13     out_c = shape[3];
14     out_h = shape[1];
15     out_w = shape[2];
16     cout << "out_n " << out_n << " out_c " << out_c
17          << " out_h " << out_h << " out_w " << out_w << "\n";
18 }

```

## 4、在MLU内存上分配I/O数据空间并且准备数据

- 使用 `cnrtMalloc` 在mlu内存上分配数据空间。
- param后便会用到。



```

1 void** inputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
  inputNum));
2 void** outputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
  outputNum));
3 void **param =
4     reinterpret_cast<void **>(malloc(sizeof(void *) * (inputNum +
  outputNum)));
5 for (int i = 0; i < inputNum; i++) {
6     cnrtMalloc(&inputMluPtrS[i], inputSizes[i]);
7 }
8 for (int i = 0; i < outputNum; i++) {
9     cnrtMalloc(&outputMluPtrS[i], outputSizes[i]);
10 }
11
12 for (int i = 0; i < inputNum; i++) {
13     param[i] = inputMluPtrS[i];
14 }
15 for (int i = 0; i < outputNum; i++) {
16     param[inputNum + i] = outputMluPtrS[i];
17 }

```

4.1 接下来要把数据从CPU拷贝到MLU进行离线推理了，这里就需要用到上面提到的function的上下文rt\_ctx，需要设置使用该function上下文的MLU设备，然后初始化function上下文。为了计算任务，还要在此context上创建计算队列。

队列queue：在 CNRT 中,一个队列(cnrtQueue\_t)是由主机端发布的一系列对 MLU 设备的执行操作。**所有的设备操作都是通过队列进行的**，其中包括计算任务、通讯任务以及事件等。**同一个队列可以容纳多个任务。在一个队列中，操作是按顺序串行执行的,不同的队列中的操作可以并发执行。**队列内部无论是执行Notifier 任务还是计算任务,始终遵循 FIFO(First In First Out,先进先出)调度原则。当没有指定队列的时候，CNRT 会使用默认的队列。

用法：可以使用cnrtCreateQueue创建一个队列然后cnrtInvokeRuntimeContext调用function上下文时调用这个队列，也可以使用cnrtRuntimeContextCreateQueue直接在MLU上为function创建一个队列。

```

1 //create cnrt_queue
2 cnrtQueue_t cnrt_queue;
3 //cnrtCreateQueue(&cnrt_queue);
4 cnrtSetRuntimeContextDeviceId(rt_ctx_, dev);
5 cnrtInitRuntimeContext(rt_ctx_, NULL);
6 cnrtRuntimeContextCreateQueue(rt_ctx_, &cnrt_queue);
7 void** tempPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) * inputNum));
8 void* temp_input_cpu_data = nullptr;
9 for (int i = 0; i < inputNum; i++) {
10     int ip = inputSizes[i] / cnrtDataTypeSize(input_data_type[i]);

```

```

11     auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) * ip));
12     tempPtrS[i] = reinterpret_cast<void*>(databuf);
13     std::vector<int> shape(4, 1);
14     int dimNum = 4;
15     cnrtGetInputDataShape((int**)&shape, &dimNum, i, function);
16     int dim_order[4] = {0, 2, 3, 1};
17     int dim_shape[4] = {shape[0], shape[3],
18                         shape[1], shape[2]}; // NCHW
19     cnrtTransDataOrder(inputCpuPtrS[i], CNRT_FLOAT32, tempPtrS[i],
20                        4, dim_shape, dim_order); //transfer order
21     temp_input_cpu_data = (void*)malloc(inputSizeS[i]);
22     int input_count = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
23     if (input_data_type[i] != CNRT_FLOAT32) {
24         cnrtCastDataType(tempPtrS[i],
25                          CNRT_FLOAT32,
26                          temp_input_cpu_data,
27                          input_data_type[i],
28                          input_count,
29                          nullptr);
30     } else {
31         temp_input_cpu_data = tempPtrS[i];
32     }
33     cnrtMemcpy(inputMluPtrS[i],
34               temp_input_cpu_data,
35               inputSizeS[i],
36               CNRT_MEM_TRANS_DIR_HOST2DEV);
37     if (temp_input_cpu_data) {
38         free(temp_input_cpu_data);
39         temp_input_cpu_data = nullptr;
40     }

```

在把CPU上的数据拷贝到MLU输入数据前，使用 `cnrtTransDataOrder` 对数据进行了预处理，NCHW->NHWC，并且使用 `cnrtCastDataType` 对输入数据进行类型转换，转换为 CNRT\_FLOAT32 类型，然后使用 `cnrtMemcpy` 将数据从CPU拷贝到MLU上，这里 `temp_input_cpu_data` 其实就是一个中间变量。

## 4.2 运行上下文

```

1  cnrtNotifier_t notifierBeginning, notifierEnd;
2  cnrtCreateNotifier(&notifierBeginning);
3  cnrtCreateNotifier(&notifierEnd);
4  float event_time_use;
5  // run MLU
6  // place start_event to cnrt_queue

```

```

7  cnrtPlaceNotifier(notifierBeginning, cnrt_queue);
8  CNRT_CHECK(cnrtInvokeRuntimeContext(rt_ctx_, param, cnrt_queue, nullptr));
9  // place end_event to cnrt_queue
10 cnrtPlaceNotifier(notifierEnd, cnrt_queue);

```

这里使用 `cnrtInvokeRuntimeContext` 调用function上下文执行队列里的运算任务，这里用到了上面提到的param。

- CNRT\_CHECK用于打印Debug信息。
- Notifier机制：CNRT使用Notifier机制统计硬件执行时间。Notifier 是一种特殊的任务,它和计算任务一样可以放置到队列中执行。相比计算任务,Notifier 不执行实际的硬件操作。如果相邻的两个或者多个 Notifier 之间包含计算任务,可以通过 Notifier 来实现对计算任务耗时的精确统计。任务执行时间=notifier\_end - notifier\_begin。详情参考寒武纪CNRT用户手册和开发者手册。

### 4.3 把计算的结果从MLU拷贝到CPU，并且转换数据格式NHWC->NCHW。

这里数据保存到了outPtrS中。

```

1  void** outPtrS = reinterpret_cast<void*>(malloc(sizeof(void*) * outputNum));
2  void* temp_output_cpu_data = nullptr;
3  for (int i = 0; i < outputNum; i++) {
4      temp_output_cpu_data = (void*)malloc(outputSizeS[i]);
5      cnrtMemcpy(temp_output_cpu_data,
6                  outputMluPtrS[i],
7                  outputSizeS[i],
8                  CNRT_MEM_TRANS_DIR_DEV2HOST); //MLU->temp_output_cpu
9      int output_count = outputSizeS[i] / cnrtDataTypeSize(output_data_type[i]);
10     if (output_data_type[i] != CNRT_FLOAT32) {
11         cnrtCastDataType(temp_output_cpu_data,
12                           output_data_type[i],
13                           outputCpuPtrS[i],
14                           CNRT_FLOAT32,
15                           output_count,
16                           nullptr);
17     } else {
18         memcpy(outputCpuPtrS[i], temp_output_cpu_data, outputSizeS[i]);
19         //temp_output_cpu_data -> outputCpuPtrS
20     }
21     auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) *
22     output_count));
23     outPtrS[i] = reinterpret_cast<void*>(databuf);
24     std::vector<int> shape(4, 1);
25     int dimNum = 4;
26     cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function);
27     int dim_order[4] = {0, 3, 1, 2};

```

```

26     int dim_shape[4] = {shape[0], shape[1],
27                           shape[2], shape[3]}; // NHWC
28     cnrtTransDataOrder(outputCpuPtrS[i], CNRT_FLOAT32, outPtrS[i],
29                          4, dim_shape, dim_order);
30     if (temp_output_cpu_data) {
31         free(temp_output_cpu_data);
32         temp_output_cpu_data = nullptr;
33     }
34 }

```

## 4.4 打印数据

out\_count是数据尺寸。

```

1  for (int i = 0; i < outputNum; i++) {
2      for (int j = 0; j < out_count[i]; ++j) {
3          cout << (reinterpret_cast<float*>(outPtrS[i]))[j] << "\n";
4      }
5  }

```

## 5、释放MLU的内存空间

```

1  output_cpu.clear();
2  //free memory space
3  free(inputCpuPtrS);
4  free(outputCpuPtrS);
5  free(outPtrS);
6  for (int i = 0; i < inputNum; i++) {
7      cnrtFree(inputMluPtrS[i]);
8  }
9  for (int i = 0; i < outputNum; i++) {
10     cnrtFree(outputMluPtrS[i]);
11 }
12
13 cnrtDestroyQueue(cnrt_queue);
14 cnrtDestroyFunction(function);
15 cnrtUnloadModel(model);
16 cnrtDestroy();

```

## 6、输出结果

尺寸为1x40输出。

```
copying output data of 0th function:out_count:40
-1
0.857186
1
1
-1
-1
1
1
1
1
-1
1
1
1
1
1
1
-1
-1
1
-1
1
0.997366
-1
-0.998165
-1
1
1
-1
-1
-1
-1
-1
-1
1
1
1
1
-1
1
-1
-1
```

7、代码：

```
1  #include "crt.h"
2  #include <iostream>
```



```

3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <ctime>
7  #include <vector>
8  #include "glog/logging.h"
9  #include <fstream>
10
11
12  using namespace std;
13
14  void rand_data(float *data, int length) {
15      unsigned int seed = 1024;
16      for (int i = 0; i < length; i++) {
17          if (i % 5 == 4) {
18              data[i] = rand_r(&seed) % 100 / 100. + 0.0625;
19          } else if (i % 5 >= 2) {
20              data[i] = data[i - 2] + (rand_r(&seed) % 100) / 100.0 + 0.0625;
21          } else {
22              data[i] = (rand_r(&seed) % 100) / 100. + 0.0625;
23          }
24      }
25  }
26
27  int offline_test(const char *name) {
28      // This example is used for MLU270 and MLU220. You need to choose
the corresponding offline model.
29      // when generating an offline model, u need cnml and cnrt both
30      // when running an offline model, u need cnrt only
31      //在调用任何 CNRT API 之前需要调用此 API 初始化环境,保证线程安全。
32      //1. init
33      cnrtInit(0);
34      //2. check mlu device, set device
35      cnrtDev_t dev;
36      cnrtGetDeviceHandle(&dev, 0);
37      cnrtSetCurrentDevice(dev);
38
39      //3. load model
40      cnrtModel_t model;
41      cnrtLoadModel(&model, "../model/actor_mlu270_offline.cambricon");
42      cout << "model:" << model << "\n";
43
44      //4. get model propertys
45      //get model total memory
46      int64_t totalMem;
47      cnrtGetModelMemUsed(model, &totalMem);
48      printf("total memory used:%ld Bytes\n", totalMem);

```

```

49     //get model parallelism
50     int model_parallelism;
51     cnrtRuntimeContext_t rt_ctx_;
52     cnrtQueryModelParallelism(model, &model_parallelism);
53     printf("model parallelism:%d.\n", model_parallelism);
54     //init struct variable
55     cnrtFunction_t function;
56     cnrtCreateFunction(&function);
57     char *func_name = NULL;
58     int func_name_size = 0;
59     cnrtGetFunctionSymbol(model, 0, &func_name, &func_name_size);
60     cout << "func_name:" << func_name << "\n";
61     //生成的模型推理结构其实被当做一个内核函数，函数名就是这个subnet0
62     cnrtExtractFunction(&function, model, func_name);
63     cnrtCreateRuntimeContext(&rt_ctx_, function, NULL);
64
65     //6. get size of input, output
66     int inputNum, outputNum;
67     int64_t *inputSizeS, *outputSizeS;
68     cnrtGetInputDataSize(&inputSizeS, &inputNum, function);
69     cnrtGetOutputDataSize(&outputSizeS, &outputNum, function);
70     cout << "inputNum:" << inputNum << " inputSizeS:" << *inputSizeS <<
"\n";
71     cout << "outputNum:" << outputNum << " outputSizeS:" << *outputSizeS
<< "\n";
72     cnrtDataType_t* input_data_type = nullptr;
73     cnrtDataType_t* output_data_type = nullptr;
74     cnrtGetInputDataType(&input_data_type, &inputNum, function);
75     cnrtGetOutputDataType(&output_data_type, &outputNum, function);
76
77     //7. prepare data on cpu
78     //申请CPU上的输入输出内存
79     void **inputCpuPtrS = reinterpret_cast<void **>(malloc(inputNum *
sizeof(void *)));
80     void **outputCpuPtrS = reinterpret_cast<void **>(malloc(outputNum *
sizeof(void *)));
81     vector<int> out_count;
82     unsigned int in_n, in_c, in_h, in_w;
83     for (int i = 0; i < inputNum; i++) {
84         int ip = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
85         cout << "ip:" << ip << " " << "cnrtDataTypeSize:" <<
cnrtDataTypeSize(input_data_type[i]) << "\n";
86         auto databuf = reinterpret_cast<float *>(malloc(sizeof(float) *
ip));
87         rand_data(databuf, ip); //data in databuf
88         inputCpuPtrS[i] = reinterpret_cast<void*>(databuf); // NCHW
89         vector<int> shape(4, 1);

```

```

90         int dimNum = 4;
91         cnrtGetInputDataShape((int**)&shape, &dimNum, i, function); //
NHWC
92         in_n = shape[0];
93         in_c = (input_data_type[i] == CNRT_UINT8) ? (shape[3] - 1) :
shape[3];
94         in_h = shape[1];
95         in_w = shape[2];
96         cout << "in_n: " << in_n << " in_c: " << in_c
97                 << " in_h: " << in_h << " in_w: " << in_w << "\n";
98     }
99     vector<float *> output_cpu;
100     unsigned int out_n, out_c, out_h, out_w;
101     for (int i = 0; i < outputNum; i++) {
102         int op = outputSizeS[i] / cnrtDataTypeSize(output_data_type[i]);
103         float* outcpu = reinterpret_cast<float*>(malloc(op *
sizeof(float)));
104         out_count.push_back(op);
105         output_cpu.push_back(outcpu);
106         outputCpuPtrS[i] = reinterpret_cast<void*>(outcpu);
107         std::vector<int> shape(4, 1);
108         int dimNum = 4;
109         cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function); //
NHWC
110         out_n = shape[0];
111         out_c = shape[3];
112         out_h = shape[1];
113         out_w = shape[2];
114         cout << "out_n " << out_n << " out_c " << out_c
115                 << " out_h " << out_h << " out_w " << out_w
<< "\n";
116     }
117
118     // allocate I/O data space on MLU memory and copy Input data
119     void** inputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
inputNum));
120     void** outputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*)
* outputNum));
121     void **param =
122         reinterpret_cast<void **>(malloc(sizeof(void *) * (inputNum +
outputNum)));
123     for (int i = 0; i < inputNum; i++) {
124         cnrtMalloc(&inputMluPtrS[i], inputSizeS[i]);
125     }
126     for (int i = 0; i < outputNum; i++) {
127         cnrtMalloc(&outputMluPtrS[i], outputSizeS[i]);
128     }

```

```

129
130     for (int i = 0; i < inputNum; i++) {
131         param[i] = inputMluPtrS[i];
132     }
133     for (int i = 0; i < outputNum; i++) {
134         param[inputNum + i] = outputMluPtrS[i];
135     }
136
137     //create cnrt_queue
138     cnrtQueue_t cnrt_queue;
139     //cnrtCreateQueue(&cnrt_queue);
140     cnrtSetRuntimeContextDeviceId(rt_ctx_, dev);
141     cnrtInitRuntimeContext(rt_ctx_, NULL);
142     cnrtRuntimeContextCreateQueue(rt_ctx_, &cnrt_queue);
143     void** tempPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
inputNum));
144     void* temp_input_cpu_data = nullptr;
145     for (int i = 0; i < inputNum; i++) {
146         int ip = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
147         auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) *
ip));
148         tempPtrS[i] = reinterpret_cast<void*>(databuf);
149         std::vector<int> shape(4, 1);
150         int dimNum = 4;
151         cnrtGetInputDataShape((int**)&shape, &dimNum, i, function);
152         int dim_order[4] = {0, 2, 3, 1};
153         int dim_shape[4] = {shape[0], shape[3],
154                             shape[1], shape[2]}; // NCHW
155         cnrtTransDataOrder(inputCpuPtrS[i], CNRT_FLOAT32, tempPtrS[i],
156                             4, dim_shape, dim_order); //transfer
order
157         temp_input_cpu_data = (void*)malloc(inputSizeS[i]);
158         int input_count = inputSizeS[i] /
cnrtDataTypeSize(input_data_type[i]);
159         if (input_data_type[i] != CNRT_FLOAT32) {
160             cnrtCastDataType(tempPtrS[i],
161                             CNRT_FLOAT32,
162                             temp_input_cpu_data,
163                             input_data_type[i],
164                             input_count,
165                             nullptr);
166         } else {
167             temp_input_cpu_data = tempPtrS[i];
168         }
169         cnrtMemcpy(inputMluPtrS[i],
temp_input_cpu_data,

```

```

170             inputSizes[i],
171             CNRT_MEM_TRANS_DIR_HOST2DEV);
172         if (temp_input_cpu_data) {
173             free(temp_input_cpu_data);
174             temp_input_cpu_data = nullptr;
175         }
176     }
177
178     // create start_event and end_event
179     cnrtNotifier_t notifierBeginning, notifierEnd;
180     cnrtCreateNotifier(&notifierBeginning);
181     cnrtCreateNotifier(&notifierEnd);
182     float event_time_use;
183     // run MLU
184     // place start_event to cnrt_queue
185     cnrtPlaceNotifier(notifierBeginning, cnrt_queue);
186     CNRT_CHECK(cnrtInvokeRuntimeContext(rt_ctx_, param, cnrt_queue,
187     nullptr));
188     // place end_event to cnrt_queue
189     cnrtPlaceNotifier(notifierEnd, cnrt_queue);
190     if (cnrtSyncQueue(cnrt_queue) == CNRT_RET_SUCCESS) { //Waits for a
191     queue operations to be completed.
192         //get start_event and end_event elapsed time
193         cnrtNotifierDuration(notifierBeginning, notifierEnd,
194         &event_time_use); //Computes the time duration between starting and ending
195         notifiers.
196         cout << " hardware time: " << event_time_use << "\n";
197     } else {
198         cout << " SyncQueue Error " << "\n";
199     }
200     void** outPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
201     outputNum));
202     void* temp_output_cpu_data = nullptr;
203     for (int i = 0; i < outputNum; i++) {
204         temp_output_cpu_data = (void*)malloc(outputSizes[i]);
205         cnrtMemcpy(temp_output_cpu_data,
206             outputMluPtrS[i],
207             outputSizes[i],
208             CNRT_MEM_TRANS_DIR_DEV2HOST); //MLU->temp_output_cpu
209         int output_count = outputSizes[i] /
210         cnrtDataTypeSize(output_data_type[i]);
211         if (output_data_type[i] != CNRT_FLOAT32) {
212             cnrtCastDataType(temp_output_cpu_data,
213                 output_data_type[i],
214                 outputCpuPtrS[i],
215                 CNRT_FLOAT32,
216                 output_count,

```

```

211                 nullptr);
212         } else {
213             memcpy(outputCpuPtrS[i], temp_output_cpu_data,
outputSizeS[i]); //temp_output_cpu_data -> outputCpuPtrS
214         }
215         auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) *
output_count));
216         outPtrS[i] = reinterpret_cast<void*>(databuf);
217         std::vector<int> shape(4, 1);
218         int dimNum = 4;
219         cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function);
220         int dim_order[4] = {0, 3, 1, 2};
221         int dim_shape[4] = {shape[0], shape[1],
222                             shape[2], shape[3]}; // NHWC
223         cnrtTransDataOrder(outputCpuPtrS[i], CNRT_FLOAT32, outPtrS[i],
224                             4, dim_shape, dim_order);
225         if (temp_output_cpu_data) {
226             free(temp_output_cpu_data);
227             temp_output_cpu_data = nullptr;
228         }
229     }
230
231     for (int i = 0; i < outputNum; i++) {
232         cout << "copying output data of " << i << "th" << " function:";
233         stringstream ss;
234         ss << "hello.txt";
235         string output_name = ss.str();
236         std::ofstream fout(output_name, std::ios::out);
237         fout << std::flush;
238         cout << "out_count:" << out_count[i] << "\n";
239         for (int j = 0; j < out_count[i]; ++j) {
240             fout <<(reinterpret_cast<float*>(outPtrS[i]))[j] << std::endl;
241             cout << (reinterpret_cast<float*>(outPtrS[i]))[j] << "\n";
242         }
243         fout << std::flush;
244         fout.close();
245     }
246     for (auto flo : output_cpu) {
247         free(flo);
248     }
249     output_cpu.clear();
250     //free memory space
251     free(inputCpuPtrS);
252     free(outputCpuPtrS);
253     free(outPtrS);
254     for (int i = 0; i < inputNum; i++) {
255         cnrtFree(inputMluPtrS[i]);

```

```

256     }
257     for (int i = 0; i < outputNum; i++) {
258         cnrtFree(outputMluPtrS[i]);
259     }
260
261     cnrtDestroyQueue(cnrt_queue);
262     cnrtDestroyFunction(function);
263     cnrtUnloadModel(model);
264     cnrtDestroy();
265
266     return 0;
267 }
268
269 int main() {
270     printf("offline test\n");
271     offline_test("mlp");
272     return 0;
273 }

```

- 在CNRT中，本质上是将模型转成了一个函数function。
- 原模型输入规模是112，输出规模是40，在使用 `cnrtGetInputDataSize` 获取的inputSizeS为448，是因为此函数返回的是字节，使用malloc申请内存空间也是需要单位为字节，一个int4个字节，可以使用 `cnrtGetInputDataType` 获得function的输入输出数据类型datatype，然后使用 `cnrtDataTypeSize` 获得数据类型的大小datatypesize，那么输入规模就是inputSizeS/datatypesize。
- 队列queue：在 CNRT 中,一个队列(cnrtQueue\_t)是由主机端发布的一系列对 MLU 设备的执行操作。**所有的设备操作都是通过队列进行的**,其中包括计算任务、通讯任务以及事件等。**同一个队列可以容纳多个任务。在一个队列中,操作是按顺序串行执行的,不同的队列中的操作可以并发执行。**队列内部无论是执行Notifier 任务还是计算任务,始终遵循 FIFO(First In First Out,先进先出)调度原则。当没有指定队列的时候,CNRT 会使用默认的队列。指定队列就使用 `cnrtCreateQueue` 这个API。MLU270 单卡队列的数量上限为 4,096。MLU220 单卡队列的数量上限为 1,024。
- 数据格式转化：`cnrtTransDataOrder`用于转换输入数据的格式，NHWC->NCHW，把inputCpuPtrS转换格式后的数据放到tempPtrS。

```

1  cnrtTransDataOrder(inputCpuPtrS[i], CNRT_FLOAT32, tempPtrS[i],
2                      4, dim_shape, dim_order);

```

- 数据类型转换：`cnrtCastDataType`用于转换数据类型，输入到MLU上的数据一般是FP15、FP32类型。
- Notifier机制：



CNRT使用Notifier机制统计硬件执行时间。Notifier 是一种特殊的任务,它和计算任务一样可以放置到队列中执行。相比计算任务,Notifier 不执行实际的硬件操作。如果相邻的两个或者多个 Notifier 之间包含计算任务,可以通过 Notifier 来实现对计算任务耗时的精确统计。任务执行时间=notifier\_end - notifier\_begin。详情参考寒武纪CNRT用户手册和开发者手册。

- 离线模型不依赖于 PyTorch 框架,只基于 CNRT(Cambricon Neuware Runtime Library,寒武纪运行时库)单独运行。离线模型为.cambricon文件,生成离线模型可使用 Cambricon PyTorch 的 Python 接口将模型转为离线模型。**离线模型的生成以及离线推理教程在寒武纪PyTorch用户手册中写到,离线模型的使用在《寒武纪运行时库用户手册（CNRT开发者手册）》（cambricon/1.7.0/CNToolkit/cnrt）中写到。**

## 六、上位机与下位机交互

### 1、方案1：

#### 1.1 x86服务器：

把推理程序编译成一个动态链接库，拷贝到MLU220。

#### 1.2 上位机（作为client）：

使用Python运行强化学习环境（state），向server通过http请求的方式发送数据，接收server返回的结果（action）与环境交互。

#### 1.3 MLU220（server）：

安装Python，编写Python程序接收client发送的数据，然后调用动态链接库，输入数据，得到结果再返回给server。

#### 问题：

- MLU220里只有与业务相关的程序和库，是相当于一个极其精简的Linux系统，python、g++、gcc 以及Linux很多命令都没有，所以在上边写程序不容易，除非重新在MLU220装一个完整的系统。

### 2、方案2：

#### 2.1 x86服务器：

**（1）寒武纪开发环境运行在x86服务器的docker容器中，外部网络与docker容器通信需要将docker容器端口映射到主机端口，外部网络才能通过主机端口访问到docker容器。**

- 把已经装好开发环境的容器commit为一个新的镜像，c2e0e359e31b是当前可用容器ID，后边是新镜像的名字：

```
1 docker commit c2e0e359e31b yellow.hub.cambricon.com/pytorch/pytorch:0.15.0-
```

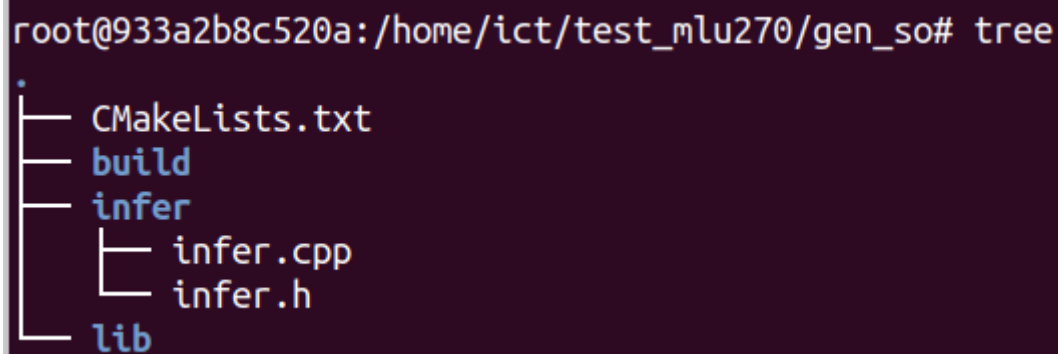
- 基于新镜像启动一个新的容器（把系统家目录挂载到容器家目录，把mlu设备挂载到容器，并把容器的8050端口映射到主机8050端口）

```
1 docker run --name dev_cn -p 8050:8050 --privileged=true -dit --
  device=/dev/cambricon_dev0 --device=/dev/cambricon_ctl --user root -v
  /usr/bin/cnmon:/usr/bin/imagecnmon -v /home:/home
  yellow.hub.cambricon.com/pytorch/pytorch:0.15.0-ubuntu16.04_3 /bin/bash
```

新创建的容器ID为933a2b8c520a，现在外部网络可以通过访问主机的8050端口访问容器，容器内需要监听8050端口的请求。

## （2）把推理程序编译成一个动态链接库：

- 文件结构：



```
root@933a2b8c520a:/home/ict/test_mlu270/gen_so# tree
.
├── CMakeLists.txt
├── build
├── infer
│   ├── infer.cpp
│   └── infer.h
└── lib
```

- infer.cpp：封装了推理程序，两个参数：databuf是输入，outPtrS用于存储输出并返回。

```
1  #include "infer.h"
2  #include "cnrt.h"
3  #include <iostream>
4  #include <stdio.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <ctime>
8  #include <vector>
9  #include <fstream>
10 #include <cstring>
11
12 using namespace std;
13
14 int offline_infer(float *databuf, void** outPtrS, bool init, int num) {
```

```

15         // This example is used for MLU270 and MLU220. You need to choose
the corresponding offline model.
16         // when generating an offline model, u need cnml and cnrt both
17         // when running an offline model, u need cnrt only
18         //在调用任何 CNRT API 之前需要调用此 API 初始化环境,保证线程安全。
19         //1. init
20         cnrtInit(0);
21         //2. check mlu device, set device
22         cnrtDev_t dev;
23         cnrtGetDeviceHandle(&dev, 0);
24         cnrtSetCurrentDevice(dev);
25
26         //3. load model
27         cnrtModel_t model;
28         cnrtLoadModel(&model, "../model/actor_mlu270_offline.cambricon");
29         if (init) {
30             cout << "model:" << model << "\n";
31         }
32
33         //4. get model propertys
34         //get model total memory
35         int64_t totalMem;
36         cnrtGetModelMemUsed(model, &totalMem);
37         if (init) {
38             printf("model total memory used:%ld Bytes\n", totalMem);
39         }
40         //get model parallelism
41         int model_parallelism;
42         cnrtRuntimeContext_t rt_ctx_;
43         cnrtQueryModelParallelism(model, &model_parallelism);
44         if (init) {
45             printf("model parallelism:%d.\n", model_parallelism);
46         }
47         //init struct variable
48         cnrtFunction_t function;
49         cnrtCreateFunction(&function);
50         char *func_name = NULL;
51         int func_name_size = 0;
52         cnrtGetFunctionSymbol(model, 0, &func_name, &func_name_size);
53         if (init) {
54             cout << "model func_name:" << func_name << "\n";
55         }
56         //生成的模型推理结构其实被当做一个内核函数, 函数名就是这个subnet0
57         cnrtExtractFunction(&function, model, func_name);
58         cnrtCreateRuntimeContext(&rt_ctx_, function, NULL);
59

```

```

60         //6. get size of input, output
61         int inputNum, outputNum;
62         int64_t *inputSizeS, *outputSizeS;
63         cnrtGetInputDataSize(&inputSizeS, &inputNum, function);
64         cnrtGetOutputDataSize(&outputSizeS, &outputNum, function);
65         if (init) {
66             cout << "model inputNum:" << inputNum << " model
inputSizeS(Bytes):" << *inputSizeS << "\n";
67             cout << "model outputNum:" << outputNum << " model
outputSizeS(Bytes):" << *outputSizeS << "\n";
68         }
69         cnrtDataType_t* input_data_type = nullptr;
70         cnrtDataType_t* output_data_type = nullptr;
71         cnrtGetInputDataType(&input_data_type, &inputNum, function);
72         cnrtGetOutputDataType(&output_data_type, &outputNum, function);
73
74         //7. prepare data on cpu
75         //申请CPU上的输入输出内存
76         void **inputCpuPtrS = reinterpret_cast<void **>(malloc(inputNum *
sizeof(void *)));
77         void **outputCpuPtrS = reinterpret_cast<void **>(malloc(outputNum *
sizeof(void *)));
78         vector<int> out_count;
79         unsigned int in_n, in_c, in_h, in_w;
80         for (int i = 0; i < inputNum; i++) {
81             int ip = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
82             if (init) {
83                 cout << "model ipsize:" << ip << " " <<
"cnrtDataTypeSize(Bytes):" << cnrtDataTypeSize(input_data_type[i]) << "\n";
84             }
85             //auto databuf = reinterpret_cast<float *>(malloc(sizeof(float) *
ip));
86             //rand_data(databuf, ip); //data in databuf
87             inputCpuPtrS[i] = reinterpret_cast<void*>(databuf); // NCHW
88             vector<int> shape(4, 1);
89             int dimNum = 4;
90             cnrtGetInputDataShape((int**)&shape, &dimNum, i, function); //
NHWC
91             in_n = shape[0];
92             in_c = (input_data_type[i] == CNRT_UINT8) ? (shape[3] - 1) :
shape[3];
93             in_h = shape[1];
94             in_w = shape[2];
95             if (init) {
96                 cout << "in_n: " << in_n << " in_c: " << in_c
<< " in_h: " << in_h << " in_w: " << in_w << "\n";
97             }
98         }

```

```

99         }
100         vector<float*> output_cpu;
101         unsigned int out_n, out_c, out_h, out_w;
102         for (int i = 0; i < outputNum; i++) {
103             int op = outputSizeS[i] / cnrtDataTypeSize(output_data_type[i]);
104             float* outcpu = reinterpret_cast<float*>(malloc(op *
NHWC sizeof(float)));
105             out_count.push_back(op);
106             output_cpu.push_back(outcpu);
107             outputCpuPtrS[i] = reinterpret_cast<void*>(outcpu);
108             std::vector<int> shape(4, 1);
109             int dimNum = 4;
110             cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function); //
111             out_n = shape[0];
112             out_c = shape[3];
113             out_h = shape[1];
114             out_w = shape[2];
115             if (init) {
116                 cout << "out_n " << out_n << " out_c " << out_c
117                     << " out_h " << out_h << " out_w " << out_w
<< "\n";
118             }
119         }
120
121         // allocate I/O data space on MLU memory and copy Input data
122         void** inputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
inputNum));
123         void** outputMluPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
outputNum));
124         void **param =
125             reinterpret_cast<void **>(malloc(sizeof(void *) * (inputNum +
outputNum)));
126         for (int i = 0; i < inputNum; i++) {
127             cnrtMalloc(&inputMluPtrS[i], inputSizeS[i]);
128         }
129         for (int i = 0; i < outputNum; i++) {
130             cnrtMalloc(&outputMluPtrS[i], outputSizeS[i]);
131         }
132
133         for (int i = 0; i < inputNum; i++) {
134             param[i] = inputMluPtrS[i];
135         }
136         for (int i = 0; i < outputNum; i++) {
137             param[inputNum + i] = outputMluPtrS[i];
138         }
139

```

```

140         //create cnrt_queue
141         cnrtQueue_t cnrt_queue;
142         //cnrtCreateQueue(&cnrt_queue);
143         cnrtSetRuntimeContextDeviceId(rt_ctx_, dev);
144         cnrtInitRuntimeContext(rt_ctx_, NULL);
145         cnrtRuntimeContextCreateQueue(rt_ctx_, &cnrt_queue);
146         void** tempPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
inputNum));
147         void* temp_input_cpu_data = nullptr;
148         for (int i = 0; i < inputNum; i++) {
149             int ip = inputSizeS[i] / cnrtDataTypeSize(input_data_type[i]);
150             auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) *
ip));
151             tempPtrS[i] = reinterpret_cast<void*>(databuf);
152             std::vector<int> shape(4, 1);
153             int dimNum = 4;
154             cnrtGetInputDataShape((int**)&shape, &dimNum, i, function);
155             int dim_order[4] = {0, 2, 3, 1};
156             int dim_shape[4] = {shape[0], shape[3],
157                                 shape[1], shape[2]}; // NCHW
158             cnrtTransDataOrder(inputCpuPtrS[i], CNRT_FLOAT32, tempPtrS[i],
159                                4, dim_shape, dim_order); //transfer
order
160             temp_input_cpu_data = (void*)malloc(inputSizeS[i]);
161             int input_count = inputSizeS[i] /
cnrtDataTypeSize(input_data_type[i]);
162             if (input_data_type[i] != CNRT_FLOAT32) {
163                 cnrtCastDataType(tempPtrS[i],
164                                   CNRT_FLOAT32,
165                                   temp_input_cpu_data,
166                                   input_data_type[i],
167                                   input_count,
168                                   nullptr);
169             } else {
170                 temp_input_cpu_data = tempPtrS[i];
171             }
172             cnrtMemcpy(inputMluPtrS[i],
173                        temp_input_cpu_data,
174                        inputSizeS[i],
175                        CNRT_MEM_TRANS_DIR_HOST2DEV);
176             if (temp_input_cpu_data) {
177                 free(temp_input_cpu_data);
178                 temp_input_cpu_data = nullptr;
179             }
180         }

```

```

181
182     // create start_event and end_event
183     cnrtNotifier_t notifierBeginning, notifierEnd;
184     cnrtCreateNotifier(&notifierBeginning);
185     cnrtCreateNotifier(&notifierEnd);
186     float event_time_use;
187     // run MLU
188     // place start_event to cnrt_queue
189     cnrtPlaceNotifier(notifierBeginning, cnrt_queue);
190     CNRT_CHECK(cnrtInvokeRuntimeContext(rt_ctx_, param, cnrt_queue,
nullptr));
191     // place end_event to cnrt_queue
192     cnrtPlaceNotifier(notifierEnd, cnrt_queue);
193     if (cnrtSyncQueue(cnrt_queue) == CNRT_RET_SUCCESS) { //Waits for a
queue operations to be completed.
194         //get start_event and end_event elapsed time
195         cnrtNotifierDuration(notifierBeginning, notifierEnd,
&event_time_use); //Computes the time duration between starting and ending
notifiers.
196         cout << " hardware time: " << event_time_use << "\n";
197     } else {
198         cout << " SyncQueue Error " << "\n";
199     }
200     cout << num << "th inference." << "\n\n\n";
201     //void** outPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
outputNum));
202     void* temp_output_cpu_data = nullptr;
203     for (int i = 0; i < outputNum; i++) {
204         temp_output_cpu_data = (void*)malloc(outputSizeS[i]);
205         cnrtMemcpy(temp_output_cpu_data,
206                     outputMluPtrS[i],
207                     outputSizeS[i],
208                     CNRT_MEM_TRANS_DIR_DEV2HOST); //MLU->temp_output_cpu
209         int output_count = outputSizeS[i] /
cnrtDataTypeSize(output_data_type[i]);
210         if (output_data_type[i] != CNRT_FLOAT32) {
211             cnrtCastDataType(temp_output_cpu_data,
212                             output_data_type[i],
213                             outputCpuPtrS[i],
214                             CNRT_FLOAT32,
215                             output_count,
216                             nullptr);
217         } else {
218             memcpy(outputCpuPtrS[i], temp_output_cpu_data,
outputSizeS[i]); //temp_output_cpu_data -> outputCpuPtrS
219         }

```



```

220         auto databuf = reinterpret_cast<float*>(malloc(sizeof(float) *
output_count));
221         outPtrS[i] = reinterpret_cast<void*>(databuf);
222         std::vector<int> shape(4, 1);
223         int dimNum = 4;
224         cnrtGetOutputDataShape((int**)&shape, &dimNum, i, function);
225         int dim_order[4] = {0, 3, 1, 2};
226         int dim_shape[4] = {shape[0], shape[1],
227                             shape[2], shape[3]}; // NHWC
228         cnrtTransDataOrder(outputCpuPtrS[i], CNRT_FLOAT32, outPtrS[i],
229                             4, dim_shape, dim_order);
230         if (temp_output_cpu_data) {
231             free(temp_output_cpu_data);
232             temp_output_cpu_data = nullptr;
233         }
234     }
235
236     for (auto flo : output_cpu) {
237         free(flo);
238     }
239
240     output_cpu.clear();
241     //free memory space
242     free(inputCpuPtrS);
243     free(outputCpuPtrS);
244     //free(outPtrS);
245     for (int i = 0; i < inputNum; i++) {
246         cnrtFree(inputMluPtrS[i]);
247     }
248     for (int i = 0; i < outputNum; i++) {
249         cnrtFree(outputMluPtrS[i]);
250     }
251
252     cnrtDestroyQueue(cnrt_queue);
253     cnrtDestroyFunction(function);
254     cnrtUnloadModel(model);
255     cnrtDestroy();
256
257     return 0;
258 }

```

- infer.h:

```

1  /*
2  infer.h

```

```

3  */
4
5  #ifndef _INFER_H_
6  #define _INFER_H_
7
8  int offline_infer(float *databuf, void** outPtrS, bool init, int num);
9
10 #endif

```

- CMakeLists.txt:

```

1  cmake_minimum_required (VERSION 2.8)
2
3  project (infer)
4
5  include_directories(/usr/local/neuware/include)
6  link_directories(/usr/local/neuware/lib64/)
7  link_libraries(cnrt)
8
9  set (SRC_LIST ${PROJECT_SOURCE_DIR}/infer/infer.cpp)
10
11 add_library (infer_shared SHARED ${SRC_LIST})
12 add_library (infer_static STATIC ${SRC_LIST})
13
14 set_target_properties (infer_shared PROPERTIES OUTPUT_NAME "infer")
15 set_target_properties (infer_static PROPERTIES OUTPUT_NAME "infer")
16
17 set (LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)

```

- 在build中执行 `cmake .. && make` 编译生成动态链接库与静态链接库libinfer.so libinfer.a

```

root@933a2b8c520a:/home/ict/test_mlu270/gen_so# tree
.
├── CMakeLists.txt
├── build
├── infer
│   ├── infer.cpp
│   └── infer.h
└── lib
    ├── libinfer.a
    └── libinfer.so

```

### (3) socket网络通信、推理调试：

- 文件结构

```
root@933a2b8c520a:/home/ict/test_mlu270/newtry3/cnrt_test# tree
.
├── CMakeLists.txt
├── bin
│   └── democnrt
├── build
├── infer
│   ├── inc
│   │   └── infer.h
│   └── lib
│       ├── libinfer.a
│       └── libinfer.so
├── mlu220_run.tar.gz
├── model
│   ├── actor_mlu220_offline.cambricon
│   ├── actor_mlu220_offline.cambricon_twins
│   ├── actor_mlu270_offline.cambricon
│   └── actor_mlu270_offline.cambricon_twins
└── src
    ├── CMakeLists.txt
    └── main.cpp
```

- CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2  project(demo_cnrt)
3
4  add_subdirectory(src)
```

- bin：用于存放编译链接后的可执行程序。
- build：用于存放编译后生成的相关内容。
- infer：
  - inc/infer.h：声明推理函数 `offline_infer`

```
1  /*
2  ** infer.h
3  */
4
5  #ifndef _INFER_H_
6  #define _INFER_H_
```

```

7
8  int offline_infer(float *databuf, void** outPtrS);
9
10 #endif

```

- lib/: 存放生成的静态与动态链接库（上面步骤生成的直接拷贝到这里）。
- mlu220\_run.tar.gz: 最终要拷贝到MLU220中的文件打包。
- model: 存放离线模型
- src:
  - CMakeLists.txt: 要链接上面生成的动态链接库

```

1  # add cnrt
2  include_directories(/usr/local/neuware/include)
3  link_directories(/usr/local/neuware/lib64/)
4  link_libraries(cnrt)
5
6  # add infer
7  include_directories(${PROJECT_SOURCE_DIR}/infer/inc)
8  MESSAGE(STATUS "PROJECT_SOURCE_DIR=${PROJECT_SOURCE_DIR}")
9  link_directories(${PROJECT_SOURCE_DIR}/infer/lib)
10 link_libraries(infer)
11
12 aux_source_directory(. SRC_LIST)
13 add_executable(democnrt ${SRC_LIST})
14
15 set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
16
17 # link
18 target_link_libraries(democnrt cnrt infer)

```

- main.cpp:
  - socket\_init: 初始化网络链接，client链接后返回客户端标识符。
  - deal\_recv\_str: 处理client发送的数据（字符串），处理为float数组，并返回float数组。
  - deal\_send\_str: 处理server要发送的数据（数字），处理为char \*字符串，并返回char \*数组。
  - inference: 推理函数，调用了deal\_recv\_str、deal\_send\_str和offline\_infer，返回处理好的待发送数据。offline\_infer在头文件 `#include "infer.h"` 中声明，因为在上边步骤中将其编译成了动态链接库，所以在这里直接调用（需要在CMakeLists.txt中链接）。

```
1  #include "cnrt.h"
2  #include <iostream>
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <ctime>
7  #include <vector>
8  #include <fstream>
9  #include <cstring>
10 #include <sys/fcntl.h>
11 #include <sys/socket.h>
12 #include <unistd.h>
13 #include <netinet/in.h>
14 #include <errno.h>
15 #include <sys/types.h>
16 #include <arpa/inet.h>
17 #include "infer.h"
18 #include <string>
19
20 using namespace std;
21
22 int socket_init(int *fd) {
23     int listenfd;
24     listenfd = socket(AF_INET, SOCK_STREAM, 0);
25     if (listenfd == -1) {
26         printf("socket create fail\n");
27         return -1;
28     }
29     struct sockaddr_in serveraddr;
30     memset(&serveraddr, 0, sizeof(serveraddr));
31     serveraddr.sin_family = AF_INET;
32     serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
33     serveraddr.sin_port = htons(8050); //specify port
34     if (bind(listenfd, (struct sockaddr *)&serveraddr, sizeof(serveraddr))
    != 0) {
35         printf("bind failed\n");
36         return -1;
37     }
38
39     if (listen(listenfd, 5) != 0) {
40         printf("Listen failed\n");
41         close(listenfd);
42         return -1;
43     }
44     printf("...waiting for client...\n");
45
46     int clientfd;
```

```

47     int socklen = sizeof(struct sockaddr_in);
48     struct sockaddr_in client_addr;
49     clientfd = accept(listenfd, (struct sockaddr*)&client_addr, (socklen_t
*)&socklen);
50     if (clientfd == -1) {
51         printf("connect failed\n");
52     } else {
53         printf("client %s has connected\n",
inet_ntoa(client_addr.sin_addr));
54     }
55     fd[0] = listenfd;
56     fd[1] = clientfd;
57
58     return 0;
59 }
60
61 void deal_recv_str(char *recvBuf, float *data) {
62     char *p;
63     p = strtok(recvBuf, " ");
64     string s;
65     int i = 0;
66     while (p != NULL) {
67         s = p;
68         data[i++] = stof(s);
69         // cout << data[i] << "\n";
70         p = strtok(NULL, " ");
71     }
72 }
73
74 void deal_send_str(char *sendBuf, float *data, int length) {
75     string sendstream;
76     for (int i = 0; i < length; i++) {
77         sendstream += to_string(data[i]) + " ";
78     }
79     strcpy(sendBuf, const_cast<char *>(sendstream.c_str()));
80 }
81
82 void inference(char *recvBuf, int input_len, char *sendBuf, int
output_len, bool init, int num) {
83     cout << "recvBufLen:" << strlen(recvBuf) << "\n";
84     auto databuf = reinterpret_cast<float *>(malloc(sizeof(float) *
input_len));
85     deal_recv_str(recvBuf, databuf);
86     //cout << "input:" << "\n";
87     //for (int i = 0; i < input_len; i++) {
88     //    cout << databuf[i] << "\n";
89     //}

```

```

90
91     int outputNum = 1;
92     void** outPtrS = reinterpret_cast<void**>(malloc(sizeof(void*) *
outputNum));
93     offline_infer(databuf, outPtrS, init, num);
94     //cout << "\noutput:\n";
95     //for (int i = 0; i < outputNum; i++) {
96     //    for (int j = 0; j < output_len; ++j) {
97     //        cout << (reinterpret_cast<float*>(outPtrS[i]))[j] <<
"\n";
98     //    }
99     //}
100     deal_send_str(sendBuf, reinterpret_cast<float*>(outPtrS[0]), 40);
101     //cout << "sendBuf: " << sendBuf << "\n";
102 }
103
104 int main() {
105     int *fd = (int *)malloc(sizeof(int) * 2);
106     int ok_socket = socket_init(fd);
107     int clientfd = fd[1], listenfd = fd[0];
108     cout << "clienfd = " << clientfd << " listenfd = " << listenfd <<
"\n";
109     if (ok_socket == -1) {
110         printf("socket init unsuccessfully!\n");
111         return 0;
112     }
113
114     int iret;
115     char recvBuf[30000], sendBuf[30000];
116     char looprecv[1025];
117     memset(recvBuf, 0, sizeof(recvBuf));
118     memset(sendBuf, 0, sizeof(sendBuf));
119     bool init = true;
120     int num = 1;
121     while (true) {
122         while (true) {
123             iret = recv(clientfd, looprecv, 1024, 0); //waiting until
new info
124             strcat(recvBuf, looprecv);
125             if (iret < 1024) {
126                 break;
127             }
128             memset(looprecv, 0, sizeof(looprecv));
129         }
130         //cout << "recvBuf:" << recvBuf << "\n";
131         int ip_len = 112, op_len = 40;
132         inference(recvBuf, ip_len, sendBuf, op_len, init, num);

```



```

133         //cout << "sendBuf::" << sendBuf << "\n";
134         if ((iret = send(clientfd, sendBuf, strlen(sendBuf), 0)) <= 0)
        {
135             perror("send");
136             break;
137         }
138         //printf("send:%s\n", sendBuf);
139         memset(recvBuf, 0, sizeof(recvBuf));
140         memset(sendBuf, 0, sizeof(sendBuf));
141         memset(looprecv, 0, sizeof(looprecv));
142         num++;
143         init = false;
144     }
145
146     close(listenfd);
147     close(clientfd);
148
149     return 0;
150 }

```

#### (4) 交叉编译MLU220上运行的动态链接库：

- 修改代码：把infer.cpp中要加载的模型改为220的离线模型

```

1  cnrtLoadModel(&model, "../model/actor_mlu220_offline.cambricon");

```

- 目录结构

```

root@933a2b8c520a:/home/ict/test_mlu270/gen_220_so# tree
.
├── CMakeLists.txt
├── build
├── infer
│   ├── infer.cpp
│   └── infer.h
├── lib
│   ├── libinfer.a
│   └── libinfer.so

```

- CMakeLists.txt

```

1  cmake_minimum_required (VERSION 2.8)
2
3  project (infer)
4
5
6  SET(CMAKE_SYSTEM_NAME Linux)
7  SET(CMAKE_SYSTEM_PROCESSOR aarch64)
8
9  add_compile_options(-std=c++11)
10 SET(CMAKE_C_COMPILER "aarch64-linux-gnu-gcc")
11 SET(CMAKE_CXX_COMPILER "aarch64-linux-gnu-g++")
12
13 # add cnrt
14 include_directories(/usr/local/neuware/include)
15 link_directories(/usr/local/neuware220/neuware/lib64/) #mlu220 cnrt动态链接库的
    目录
16 link_libraries(cnrt)
17
18 # add aarch64-linux-gnu-gcc
19 include_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-
    linux-gnu/include/)
20 link_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-
    gnu/aarch64-linux-gnu/lib64/)
21
22 set (SRC_LIST ${PROJECT_SOURCE_DIR}/infer/infer.cpp)
23
24 add_library (infer_shared SHARED ${SRC_LIST})
25 add_library (infer_static STATIC ${SRC_LIST})
26
27 set_target_properties (infer_shared PROPERTIES OUTPUT_NAME "infer")
28 set_target_properties (infer_static PROPERTIES OUTPUT_NAME "infer")
29
30
31 set (LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)

```

- 编译

```

1  cd build
2  cmake .. && make
3  """"
4  root@933a2b8c520a:/home/ict/test_mlu270/gen_220_so/build# cmake .. && make
5  -- The C compiler identification is GNU 7.3.0
6  -- The CXX compiler identification is GNU 7.3.0
7  -- Check for working C compiler: /usr/bin/cc
8  -- Check for working C compiler: /usr/bin/cc -- works

```

```

 9  -- Detecting C compiler ABI info
10  -- Detecting C compiler ABI info - done
11  -- Detecting C compile features
12  -- Detecting C compile features - done
13  -- Check for working CXX compiler: /usr/bin/c++
14  -- Check for working CXX compiler: /usr/bin/c++ -- works
15  -- Detecting CXX compiler ABI info
16  -- Detecting CXX compiler ABI info - done
17  -- Detecting CXX compile features
18  -- Detecting CXX compile features - done
19  -- Configuring done
20  -- Generating done
21  -- Build files have been written to: /home/ict/test_mlu270/gen_220_so/build
22  Scanning dependencies of target infer_shared
23  [ 25%] Building CXX object CMakeFiles/infer_shared.dir/infer/infer.cpp.o
24  [ 50%] Linking CXX shared library ../lib/libinfer.so
25  [ 50%] Built target infer_shared
26  Scanning dependencies of target infer_static
27  [ 75%] Building CXX object CMakeFiles/infer_static.dir/infer/infer.cpp.o
28  [100%] Linking CXX static library ../lib/libinfer.a
29  [100%] Built target infer_static
30  """"

```

- 查看编译结果

```

root@933a2b8c520a:/home/ict/test_mlu270/gen_220_so/lib# ls
libinfer.a  libinfer.so
root@933a2b8c520a:/home/ict/test_mlu270/gen_220_so/lib# file libinfer.so
libinfer.so: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically linked, BuildID[sha1]=abccef39f928c103f32ee65236a70f7fd2801d82, not stripped

```

## (5) 交叉编译运行在MLU220上的程序

- 把编译好的动态链接库copy过来，文件目录结构如下：

```

root@933a2b8c520a:/home/ict/test_mlu270/mlu220_debug/cnrt_test# tree
.
├── CMakeLists.txt
├── bin
│   └── democnrt
├── build
├── infer
│   ├── inc
│   │   └── infer.h
│   └── lib
│       ├── libinfer.a
│       └── libinfer.so
├── mlu220_run.tar.gz
├── model
│   ├── actor_mlu220_offline.cambricon
│   ├── actor_mlu220_offline.cambricon_twins
│   ├── actor_mlu270_offline.cambricon
│   └── actor_mlu270_offline.cambricon_twins
└── src
    ├── CMakeLists.txt
    └── main.cpp

```

- CMakeLists.txt

```

1  cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
2  project(demo_cnrt)
3
4  SET(CMAKE_SYSTEM_NAME Linux)
5  SET(CMAKE_SYSTEM_PROCESSOR aarch64)
6
7  add_compile_options(-std=c++11)
8  SET(CMAKE_C_COMPILER "aarch64-linux-gnu-gcc")
9  SET(CMAKE_CXX_COMPILER "aarch64-linux-gnu-g++")
10
11 add_subdirectory(src)

```

- src/CMakeLists.txt

```

1  # add cnrt
2  include_directories(/usr/local/neuware/include)
3  link_directories(/usr/local/neuware220/neuware/lib64/) #mlu220 cnrt动态链接库的
   目录
4  link_libraries(cnrt)

```

```

5
6 # add aarch64-linux-gnu-gcc
7 include_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-
linux-gnu/include/)
8 link_directories(/usr/local/arm/gcc-linaro-6.2.1-2016.11-x86_64_aarch64-linux-
gnu/aarch64-linux-gnu/lib64/)
9
10 # add infer
11 include_directories(${PROJECT_SOURCE_DIR}/infer/inc)
12 MESSAGE(STATUS "PROJECT_SOURCE_DIR=${PROJECT_SOURCE_DIR}")
13 link_directories(${PROJECT_SOURCE_DIR}/infer/lib)
14 link_libraries(infer)
15
16 aux_source_directory(. SRC_LIST)
17 add_executable(democnrt ${SRC_LIST})
18
19 set (EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
20
21 # link
22 target_link_libraries(democnrt cnrt infer)

```

- 编译

```

1 cd build
2 cmake .. && make
3
4 root@933a2b8c520a:/home/ict/test_mlu270/mlu220_debug/cnrt_test/build# cmake
.. && make
5 -- The C compiler identification is GNU 7.3.0
6 -- The CXX compiler identification is GNU 7.3.0
7 -- Check for working C compiler: /usr/bin/cc
8 -- Check for working C compiler: /usr/bin/cc -- works
9 -- Detecting C compiler ABI info
10 -- Detecting C compiler ABI info - done
11 -- Detecting C compile features
12 -- Detecting C compile features - done
13 -- Check for working CXX compiler: /usr/bin/c++
14 -- Check for working CXX compiler: /usr/bin/c++ -- works
15 -- Detecting CXX compiler ABI info
16 -- Detecting CXX compiler ABI info - done
17 -- Detecting CXX compile features
18 -- Detecting CXX compile features - done
19 -- PROJECT_SOURCE_DIR=/home/ict/test_mlu270/mlu220_debug/cnrt_test
20 -- Configuring done
21 -- Generating done

```

```
22  -- Build files have been written to:
    /home/ict/test_mlu270/mlu220_debug/cnrt_test/build
23  Scanning dependencies of target democnrt
24  [ 50%] Building CXX object src/CMakeFiles/democnrt.dir/main.cpp.o
25  [100%] Linking CXX executable ../../bin/democnrt
26  [100%] Built target democnrt
27  """"
```

- 查看结果

```
root@933a2b8c520a:/home/ict/test_mlu270/mlu220_debug/cnrt_test/build# file ../bin/democnrt
../bin/democnrt: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, BuildID[sha1]=35772f94d197c46a9ad135eaf6dbb4e70288a418, not stripped
```

## (6) 打包、拷贝

- 在X86服务器上，把生成的可执行文件democnrt、离线模型，以及上面的动态链接库libinfer.so（跟可执行文件一块放到bin目录下即可）打包为mlu220\_run.tar.gz，命令如图所示：

```
root@933a2b8c520a:/home/ict/test_mlu270/mlu220_debug/cnrt_test# tar -czvf mlu220_run.tar.gz model bin
model/
model/actor_mlu220_offline.cambricon_twins
model/actor_mlu220_offline.cambricon
bin/
bin/democnrt
```

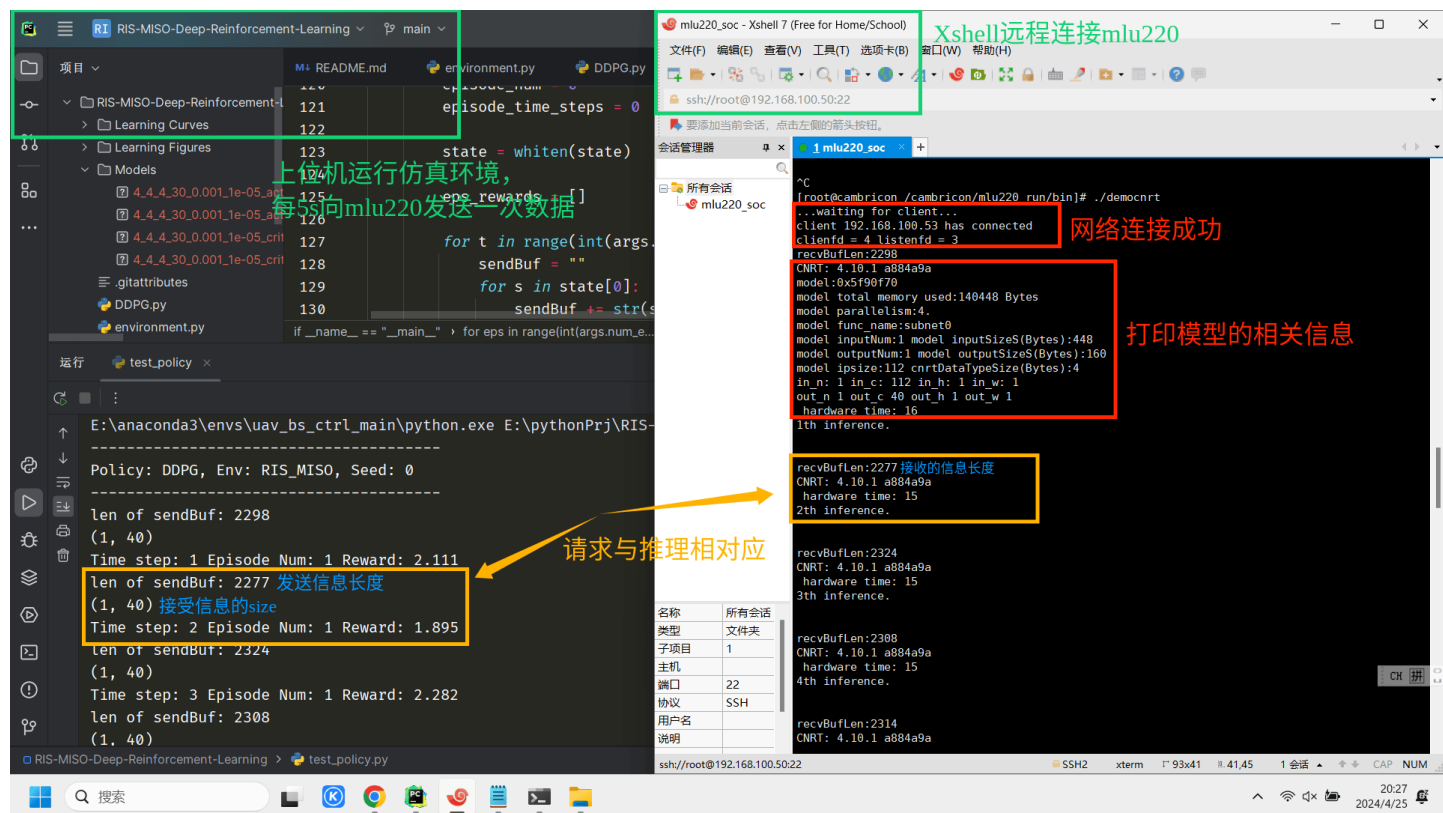
- 在上位机通过ssh拷贝到MLU220的/cambricon目录下，防止掉电丢失，在cmd中运行下面命令。

```
1  scp -r G:\mlu220_run.tar.gz root@192.168.100.50:/cambricon #宿主机 -> mlu220
```

- 在MLU220中，解压，运行aarch64可执行文件。

```
1  cd /cambricon
2  mv mlu220_run.tar.gz ./mlu220_run
3  cd ./mlu220_run
4  tar -xzf ./mlu220_run.tar.gz
5  cd bin
6  ./democnrt #运行
```

## (7) 结果展示：



## 2.2 上位机（作为client）：

使用Python运行强化学习环境（state），向server通过http请求的方式发送数据，接收server返回的结果（action）与环境交互。

- 在于MLU270或者MLU220交互的时候要记得修改为相应的IP。

## 2.3 MLU220（server）：

MLU220仅作为一个运行程序的平台，不做开发，开发全部在x86服务器完成。

# 七、完善

## 1、解决算子不适配的问题。

### 1.1 问题

因为在调用GRUCell时，报错提示addmm与sigmoid\_不支持，但是在mlu\_functions.yaml中均已声明。

```
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140363448362752][process:8786]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: addmm  
%21 : Float(*, *) = aten::addmm(%7, %hx, %18, %9, %9), scope: RnnAgent/GRUCell[grucell] # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/nn/modules/rnn.py:1023:0  
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140363448362752][process:8786]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: sigmoid  
%30 : Tensor = aten::sigmoid_%29, scope: RnnAgent/GRUCell[grucell] # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/nn/modules/rnn.py:1023:0
```

mlu\_functions.yaml中两个算子的声明。

```

- name: addmm
  use_mlu_dispatcher: unboxed_only
  derived_type: cnnl
  schema_string: aten::addmm(Tensor self, Tensor mat1, Tensor mat2, *, Scalar beta=1, Scalar alpha=1) -> Tensor
  arguments:
    - name: self
      type: const at::Tensor &
    - name: mat1
      type: const at::Tensor &
    - name: mat2
      type: const at::Tensor &
    - name: beta
      type: at::Scalar
    - name: alpha
      type: at::Scalar
  return_type: at::Tensor

```

```

- name: sigmoid_
  use_mlu_dispatcher: unboxed_only
  derived_type: cnnl
  schema_string: aten::sigmoid_(Tensor(a!) self) -> Tensor(a!)
  arguments:
    - name: self
      type: at::Tensor &
  return_type: at::Tensor &

```

在pytorch中，RNN有非常多的变体，比如RNN、LSTM、LSTMCell、GRU、GRUCell，在文档中明确说了已经自定义了GRU和LSTM：

▼ 6.5 自定义算子说明	56
6.5.1 Bert_Squad	56
6.5.2 Bert_Embedding	57
6.5.3 Big_topk	57
6.5.4 MLUConv2d	58
6.5.5 MLUConv3d	58
6.5.6 MLUConvBnReLU3d	59
6.5.7 MLUConvTranspose2d	60
6.5.8 MLUConvTranspose3d	60
6.5.9 MLUConvTransposeBnReLU...	61
6.5.10 Crop_resize	62
6.5.11 GRU	62
6.5.12 Image_detect	63
6.5.13 Linear	63
6.5.14 LSTM	64

经过测试了一下GRU、LSTM，发现是可行的，所以原因就是MLU不支持GRUCell，在**算子分发**的时候出现了问题：



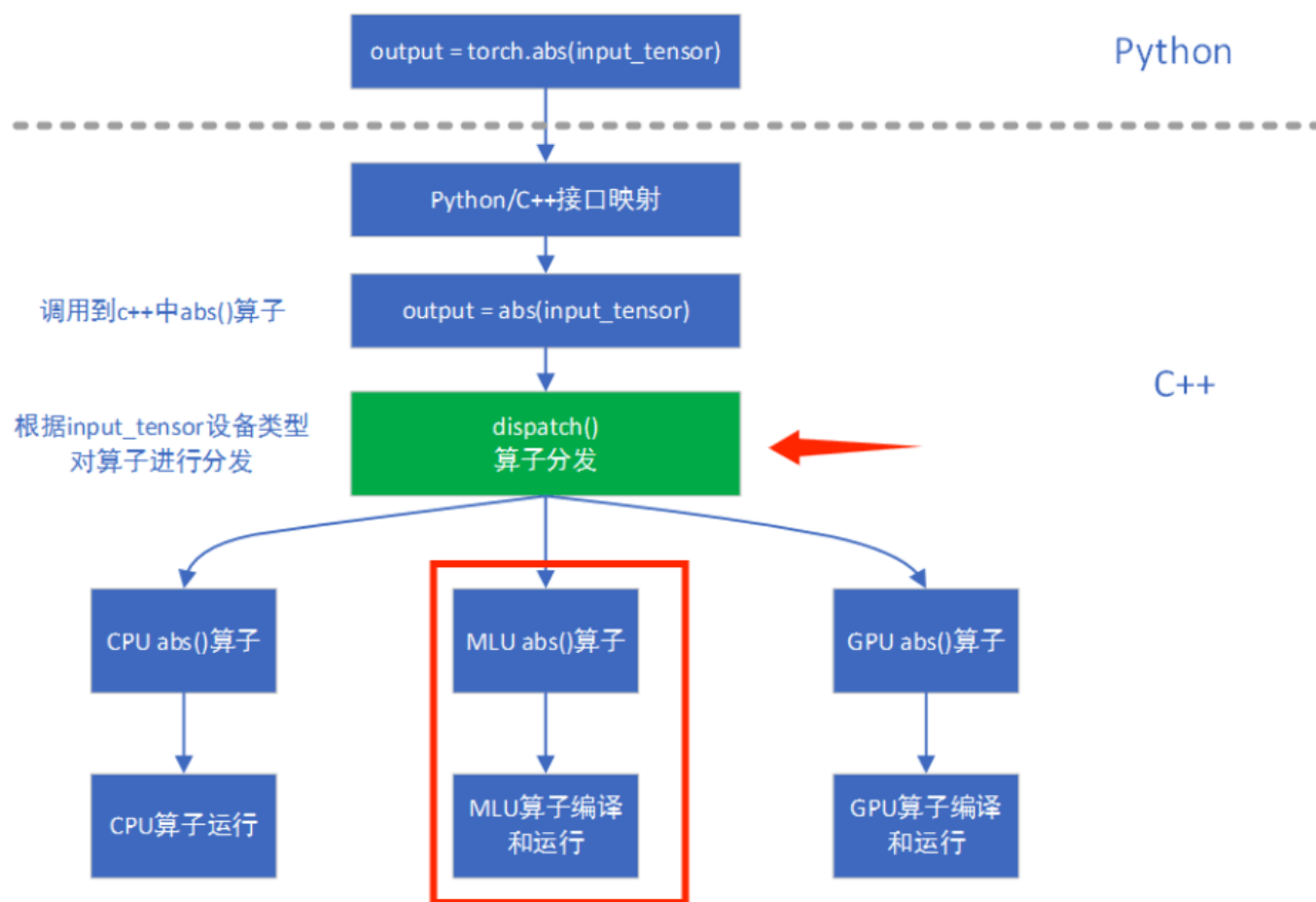


图 7.2: 逐层算子分发图

但是对于mlu不支持算子的处理有两种情况：

对于 MLU 暂不支持的算子，如果未在 `catch/torch_mlu/tools/mlu_functions.yaml` 中声明，程序会直接终止，并抛出无法分发到 MLU 设备的异常；如果已经在 `catch/torch_mlu/tools/mlu_functions.yaml` 中声明但在运行至 `wrapper` 或 `kernel` 时失败，输入数据将会拷贝到 CPU 上，然后调用 CPU 相关算子，使其在 CPU 上运行，最后再将输出结果拷回到 MLU 上。具体实现，可以查询 `op_methods.cpp`，该文件在 `catch/torch_mlu/csrc/aten/operators/` 目录下。

应该不是第一种，因为已经在 `mlu_functions.yaml` 文件中声明了，说明算子成功分发到 mlu 上，而从 `mlu_function.yaml` 中由发现两个算子的派生类型是 **CNNL**，偶然在论坛里看到有个帖子 (<https://forum.cambricon.com/index.php?m=content&c=index&a=show&catid=156&id=2974>) 说了 MLU270 不支持 CNNL，所以才猜测是不是 mlu270 不支持 CNNL 的原因。

在寒武纪论坛询问得到官方人员回复，不支持 GRUCell。



LV.1 #1 踏雪寻梅 回复

尊敬的开发者您好，目前暂不支持 GRUCell，已支持 GRU 算子。  
如需使用 GRUCell，可参考 Pytorch User Guide 添加自定义算子实现。

2024-05-10 11:09:04

0 | 回复

并且不支持寒武纪神经网络计算库CNNL（3系列才支持），只支持寒武纪机器学习库CNML，因此在所有在mlu\_functions.yaml中声明的，派生类型(derived\_type)里不包CNML的的算子都无法使用：



LV.1 #4 踏雪寻梅 回复

#3 zzlj 回复

您好，麻烦您，我还有一个问题，MLU270是不是不支持cnnl？

尊敬的开发者您好，cnnl主要用于3系列相关产品，270暂不支持。

2024-05-14 15:23:23

0 | 回复

## 1.2 解决办法：

（1）根据GRUCell的计算公式，把GRUCell拆成寒武纪支持的算子，直接在代码文件中利用寒武纪支持的算子实现一个GRUCell。

- 优点：方便、简单
- 缺点：由python实现，效率比较低，但是完全满足科研需求。

a. 根据pytorch 1.3.0 [官方文档](<https://pytorch.org/docs/1.3.0/nn.html?highlight=grucell#torch.nn.GRUCell>)中的公式，把forward实现：

A gated recurrent unit (GRU) cell

$$\begin{aligned}r &= \sigma(W_{ir}x + b_{ir} + W_{hr}h + b_{hr}) \\z &= \sigma(W_{iz}x + b_{iz} + W_{hz}h + b_{hz}) \\n &= \tanh(W_{in}x + b_{in} + r * (W_{hn}h + b_{hn})) \\h' &= (1 - z) * n + z * h\end{aligned}$$

where  $\sigma$  is the sigmoid function, and  $*$  is the Hadamard product.

b. 根据要求初始化权重和偏置：

- `~GRUCell.weight_ih` – the learnable input-hidden weights, of shape  $(3*hidden\_size, input\_size)$
- `~GRUCell.weight_hh` – the learnable hidden-hidden weights, of shape  $(3*hidden\_size, hidden\_size)$
- `~GRUCell.bias_ih` – the learnable input-hidden bias, of shape  $(3*hidden\_size)$
- `~GRUCell.bias_hh` – the learnable hidden-hidden bias, of shape  $(3*hidden\_size)$

#### • NOTE

All the weights and biases are initialized from  $\mathcal{U}(-\sqrt{k}, \sqrt{k})$  where  $k = \frac{1}{hidden\_size}$

### c. 代码实现：

```

1  class GRUCell(nn.Module):
2      def __init__(self, input_size, hidden_size):
3          super(GRUCell, self).__init__()
4          stdv = 1.0 / math.sqrt(hidden_size)
5          self.in2hid_w = nn.ParameterList([self.__init(stdv, input_size,
6              hidden_size) for _ in range(3)])
7          self.hid2hid_w = nn.ParameterList([self.__init(stdv, hidden_size,
8              hidden_size) for _ in range(3)])
9          self.in2hid_b = nn.ParameterList([self.__init(stdv, hidden_size) for
10             _ in range(3)])
11         self.hid2hid_b = nn.ParameterList([self.__init(stdv, hidden_size) for
12             _ in range(3)])
13
14     @staticmethod
15     def __init(stdv, dim1, dim2=None):
16         if dim2 is None:
17             return nn.Parameter(nn.init.uniform_(torch.Tensor(dim1), -stdv,
18                 stdv)) # 按照官方的初始化方法来初始化网络参数
19         else:
20             return nn.Parameter(nn.init.uniform_(torch.Tensor(dim1, dim2), -
21                 stdv, stdv))
22
23     def forward(self, x, hid):
24         r = torch.sigmoid(torch.mm(x, self.in2hid_w[0]) + self.in2hid_b[0] +
25             torch.mm(hid, self.hid2hid_w[0]) + self.hid2hid_b[0])
26         z = torch.sigmoid(torch.mm(x, self.in2hid_w[1]) + self.in2hid_b[1] +
27             torch.mm(hid, self.hid2hid_w[1]) + self.hid2hid_b[1])
28         n = torch.tanh(torch.mm(x, self.in2hid_w[2]) + self.in2hid_b[2] +
29             torch.mul(r, (torch.mm(hid, self.hid2hid_w[2]) + self.hid2hid_b[2])))
30         next_hid = torch.mul(-(z - 1), n) + torch.mul(z, hid)

```

除此之外，为了保证算子结果准确性，初始化的时候可以直接把官方算子GRUCell初始化的权重和偏执直接导入到自己写的算子中，代码如下：

```

1  def init_rnn_wb(self):
2      """load initial weights and bias from official torch.nn.grucell"""
3      net = torch.nn.GRUCell(self._hidden_size, self._hidden_size)
4      p = self.rnn.state_dict()
5      print(net.state_dict()['weight_ih'].shape)
6      p['in2hid_w.0'] = net.state_dict()['weight_ih'][0:self._hidden_size,
7      :].transpose(0, 1)
8      p['in2hid_w.1'] = net.state_dict()['weight_ih']
9      [self._hidden_size:self._hidden_size*2, :].transpose(0, 1)
10     p['in2hid_w.2'] = net.state_dict()['weight_ih']
11     [self._hidden_size*2:self._hidden_size*3, :].transpose(0, 1)
12
13     p['hid2hid_w.0'] = net.state_dict()['weight_hh'][0:self._hidden_size,
14     :].transpose(0, 1)
15     p['hid2hid_w.1'] = net.state_dict()['weight_hh']
16     [self._hidden_size:self._hidden_size*2, :].transpose(0, 1)
17     p['hid2hid_w.2'] = net.state_dict()['weight_hh']
18     [self._hidden_size*2:self._hidden_size*3, :].transpose(0, 1)
19
20     p['in2hid_b.0'] = net.state_dict()['bias_ih'][0:self._hidden_size]
21     p['in2hid_b.1'] = net.state_dict()['bias_ih']
22     [self._hidden_size:self._hidden_size*2]
23     p['in2hid_b.2'] = net.state_dict()['bias_ih']
24     [self._hidden_size*2:self._hidden_size*3]
25
26     p['hid2hid_b.0'] = net.state_dict()['bias_hh'][0:self._hidden_size]
27     p['hid2hid_b.1'] = net.state_dict()['bias_hh']
28     [self._hidden_size:self._hidden_size*2]
29     p['hid2hid_b.2'] = net.state_dict()['bias_hh']
30     [self._hidden_size*2:self._hidden_size*3]
31     self.rnn.load_state_dict(p)

```

(2) 使用Bang C写一个C++算子，然后修改算子调用过程中的各种文件。

- 优点：效率高，符合工业界要求。
- 缺点：比较麻烦。

### 1.3 使用上述解决方法1出现新的问题：

- 高维矩阵乘法算子torch.matmul不支持：

```
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140439080888064][process:2157]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: matmul  
%28 : Float(*, *) = aten::matmul(%x, %5), scope: GRUCell1 # test_mlu.py:32:0  
  
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140439080888064][process:2157]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: matmul  
%37 : Float(*, *) = aten::matmul(%x, %6), scope: GRUCell1 # test_mlu.py:33:0
```

暂时不用解决，不要需要。

- z是一个20\*30规模的矩阵，由公式中可以看到有1 - z运算，这样就会调用rsub算子，导致不兼容：

```
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140439080888064][process:2157]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: rsub  
%49 : Tensor = aten::rsub(%z, %20, %20), scope: GRUCell1 # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/tensor.py:355:0  
  
[WARNING][pytorch/catch/torch_mlu/csrc/jit/passes/segment_graph.cpp][line:41][MLUSupport][thread:140439080888064][process:2157]:  
[Fusion Segment] Please check mlu_functions.yaml && Maybe MLU fusion does NOT supports op: rsub  
%49 : Tensor = aten::rsub(%z, %20, %20), scope: GRUCell1 # /torch/venv3/pytorch/lib/python3.6/site-packages/torch/tensor.py:355:0
```

**解决问题：**因为1是标量，z是张量，pytorch为了方便处理调用了sub的反向算子rsub算子，所以可以改为 `next_hid = torch.mul(-(z-1), n) + torch.mul(z, hid)` 即可。

## 1.4 生成并且测试离线模型，代码如下：

```
1  # encoding: utf-8  
2  import torch  
3  
4  import torch.nn as nn  
5  
6  import math  
7  
8  import torch_mlu  
9  import torch_mlu.core.mlu_model as ct  
10 import torch_mlu.core.mlu_quantize as mlu_quantize  
11 import torchvision.models as models  
12  
13 ct.set_core_number(4)  
14 ct.set_core_version("MLU270")  
15 torch.set_grad_enabled(False)  
16  
17 class GRUCell(nn.Module):  
18     def __init__(self, input_size, hidden_size):  
19         super(GRUCell, self).__init__()  
20         stdv = 1.0 / math.sqrt(hidden_size)  
21         self.weight_ih = nn.Parameter(nn.init.uniform_(torch.Tensor(3 *  
22             hidden_size, input_size), -stdv, stdv))  
23         self.in2hid_w = nn.ParameterList([self.__init__(stdv, input_size,  
24             hidden_size) for _ in range(3)])
```

```

23         self.hid2hid_w = nn.ParameterList([self.__init(stdv, hidden_size,
hidden_size) for _ in range(3)])
24         self.in2hid_b = nn.ParameterList([self.__init(stdv, hidden_size) for
_ in range(3)])
25         self.hid2hid_b = nn.ParameterList([self.__init(stdv, hidden_size) for
_ in range(3)])
26
27     @staticmethod
28     def __init(stdv, dim1, dim2=None):
29         if dim2 is None:
30             return nn.Parameter(nn.init.uniform_(torch.Tensor(dim1), -stdv,
stdv)) # 按照官方的初始化方法来初始化网络参数
31         else:
32             return nn.Parameter(nn.init.uniform_(torch.Tensor(dim1, dim2), -
stdv, stdv))
33
34     def forward(self, x, hid):
35         r = torch.sigmoid(torch.mm(x, self.in2hid_w[0]) + self.in2hid_b[0] +
torch.mm(hid, self.hid2hid_w[0]) + self.hid2hid_b[0])
36         z = torch.sigmoid(torch.mm(x, self.in2hid_w[1]) + self.in2hid_b[1] +
torch.mm(hid, self.hid2hid_w[1]) + self.hid2hid_b[1])
37         n = torch.tanh(torch.mm(x, self.in2hid_w[2]) + self.in2hid_b[2] +
torch.mul(r, (torch.mm(hid, self.hid2hid_w[2]) + self.hid2hid_b[2])))
38         next_hid = torch.mul(-(z - 1), n) + torch.mul(z, hid)
39
40         return next_hid
41
42 class RnnAgent(nn.Module):
43     def __init__(self, obs_shape, n_actions, hidden_size, n_layers):
44         super(RnnAgent, self).__init__()
45
46         self._n_layers = n_layers
47         self._hidden_size = hidden_size
48
49         layers = [nn.Linear(obs_shape, self._hidden_size), nn.ReLU()]
50         for l in range(self._n_layers - 1):
51             layers += [nn.Linear(self._hidden_size, self._hidden_size),
nn.ReLU()]
52
53         self.enc = nn.Sequential(*layers)
54         self.rnn = GRUCell(self._hidden_size, self._hidden_size)
55         self.init_rnn_wb()
56         self.f_out = nn.Linear(self._hidden_size, n_actions)
57
58     def init_hidden(self):
59         return torch.zeros(1, self._hidden_size)
60

```

```

61     def init_rnn_wb(self):
62         """load initial weights and bias from official torch.nn.grucell"""
63         net = torch.nn.GRUCell(self._hidden_size, self._hidden_size)
64         p = self.rnn.state_dict()
65         # print(net.state_dict()['weight_ih'].shape)
66         p['in2hid_w.0'] = net.state_dict()['weight_ih'][0:self._hidden_size,
:] .transpose(0, 1)
67         p['in2hid_w.1'] = net.state_dict()['weight_ih']
[ self._hidden_size:self._hidden_size*2, :].transpose(0, 1)
68         p['in2hid_w.2'] = net.state_dict()['weight_ih']
[ self._hidden_size*2:self._hidden_size*3, :].transpose(0, 1)
69
70         p['hid2hid_w.0'] = net.state_dict()['weight_hh'][0:self._hidden_size,
:] .transpose(0, 1)
71         p['hid2hid_w.1'] = net.state_dict()['weight_hh']
[ self._hidden_size:self._hidden_size*2, :].transpose(0, 1)
72         p['hid2hid_w.2'] = net.state_dict()['weight_hh']
[ self._hidden_size*2:self._hidden_size*3, :].transpose(0, 1)
73
74         p['in2hid_b.0'] = net.state_dict()['bias_ih'][0:self._hidden_size]
75         p['in2hid_b.1'] = net.state_dict()['bias_ih']
[ self._hidden_size:self._hidden_size*2]
76         p['in2hid_b.2'] = net.state_dict()['bias_ih']
[ self._hidden_size*2:self._hidden_size*3]
77
78         p['hid2hid_b.0'] = net.state_dict()['bias_hh'][0:self._hidden_size]
79         p['hid2hid_b.1'] = net.state_dict()['bias_hh']
[ self._hidden_size:self._hidden_size*2]
80         p['hid2hid_b.2'] = net.state_dict()['bias_hh']
[ self._hidden_size*2:self._hidden_size*3]
81         self.rnn.load_state_dict(p)
82
83     def forward(self, obs, h):
84         x = self.enc(obs)
85         h = self.rnn(x, h)
86         x = self.f_out(h)
87         return x, h
88
89     def setup_seed(seed):
90         torch.manual_seed(seed)
91         torch.cuda.manual_seed(seed)
92         torch.cuda.manual_seed_all(seed)
93         torch.backends.cudnn.deterministic = True
94         torch.backends.cudnn.benchmark = False
95
96     if __name__ == '__main__':
97         # 设置随机数种子

```

```

98     setup_seed(4)
99
100     rnn_agent = RnnAgent(obs_shape=82, n_actions=5, hidden_size=256,
n_layers=2)
101     model = torch.load('./checkpoint_ep50_unzip.pth',
102                        map_location='cpu')
103     rnn_agent.load_state_dict(model, False)
104     rnn_agent.eval().float()
105     input_obs = torch.randn(1, 82)
106     input_h = torch.randn(1, 256)
107     # common infer
108     x, h = rnn_agent(input_obs, input_h)
109     # print(x.shape)
110     # print(type(x))
111     # print(h.shape)
112     # print(type(h))
113     print("common infer:", x, h)
114     # quantization and save quantization model
115     torch.set_grad_enabled(False)
116     mean = [0]
117     std = [1/255]
118     rnn_agent_quantization = mlu_quantize.quantize_dynamic_mlu(rnn_agent,
{'mean':mean, 'std':std, 'firstconv':True}, dtype='int8', gen_quant=True)
119     torch.save(rnn_agent_quantization.state_dict(),
'rnn_agent_quantization.pth')
120     print(ct.mlu_device())
121     # load quantization model
122     rnn_agent_quantization = mlu_quantize.quantize_dynamic_mlu(rnn_agent)
123
rnn_agent_quantization.load_state_dict(torch.load('rnn_agent_quantization.pth'
), False)
124     # print(rnn_agent_quantization.state_dict())
125     # send data and model to mlu_device
126     rnn_agent_quantization.to(ct.mlu_device())
127     input_obs_mlu = input_obs.to(ct.mlu_device())
128     input_h_mlu = input_h.to(ct.mlu_device())
129     # layer-by-layer infer:
130     x, h = rnn_agent_quantization(input_obs_mlu, input_h_mlu)
131     print("layer-by-layer infer:", x.cpu(), h.cpu())
132     # generate offline model
133     ct.save_as_cambricon('rnn_agent270')
134     input_data = (input_obs_mlu, input_h_mlu)
135     trace_model = torch.jit.trace(rnn_agent_quantization, input_data,
check_trace=False)
136     # fus infer:
137     x, h = trace_model(input_obs_mlu, input_h_mlu)
138     print('fus infer:', x.cpu(), h.cpu())

```



```

139
140 # sim quant infer
141 from torch_mlu.core.utils import sim_quant_utils
142 rnn_agent = RnnAgent(obs_shape=82, n_actions=5, hidden_size=256,
n_layers=2)
143 rnn_agent_quantization = mlu_quantize.quantize_dynamic_mlu(rnn_agent,
{'mean':mean, 'std':std, 'firstconv':True}, dtype='int8', gen_quant=True)
144 sim_quant_utils.register_quant_hook(rnn_agent_quantization)
145
rnn_agent_quantization.load_state_dict(torch.load('rnn_agent_quantization.pth'
), False)
146 x, h = rnn_agent_quantization(input_obs, input_h)
147 print("sim quantization infer:", x, h)

```

经过测试发现，模型量化后的CPU推理与MLU逐层推理差异较大，图如下（上边是CPU推理，下边是MLU逐层推理）：

```

common infer: tensor([[-15.5791, 16.3302, 14.8993, 15.6386, 15.9942]]) tensor([[ 1.9706,  0.1102, -0.1319, -0.9933,  0.5838,  0.8257,  0.3525,  0.7256,
-0.8925,  0.9634,  0.3923, -0.9960, -0.2284,  0.4821, -0.1611,  0.7931,
-0.9842, -0.5601,  0.9894,  0.9193, -0.5992,  0.7702,  0.0129, -0.1357,
 0.1404, -0.0496,  0.6611, -0.9978, -0.7844, -0.2506,  0.2758,  1.6201,
 0.9688,  0.6207,  2.1797,  0.0107,  0.2271, -0.0463,  0.0479,  1.2438,
-0.8410,  0.2167, -0.9133,  0.1369, -0.6296, -0.9610, -0.5352,  0.9992,
 0.0429,  0.9899, -0.9747,  0.2070, -0.5813, -0.1463, -0.9888, -0.7106,
-0.8674,  0.9999,  0.7710, -0.2903,  0.9627,  0.9987, -0.4474, -0.9276,
 0.5897,  0.3678, -0.9235,  1.1345, -0.8066, -0.9997, -0.9727, -0.9676,
-0.7077,  0.9626,  0.3847, -1.0679,  0.8433,  0.5793, -0.9996, -0.4887,
 0.6614,  0.7508, -0.9406,  0.7393,  0.0068, -0.9798,  0.2819,  0.9326,
 0.5443,  0.9543,  0.9998,  0.1064, -0.7978, -1.0000, -0.5680,  0.3360,
-0.0504,  0.1219, -0.9520,  0.0470,  0.8428,  0.9832,  0.9914, -0.8447,
-0.0533,  0.3459, -0.9578,  0.5467,  1.0000,  0.9319, -0.1111,  0.5452,
 0.9944, -0.8398,  0.7884,  0.2305,  0.2455, -0.6509, -0.8824,  1.0842,
-0.3004,  0.2716, -0.9545, -0.1933, -0.4521,  0.3226,  0.6656,  0.0163,
-0.8464, -0.2345,  0.5629, -0.2497,  0.3149, -0.8739,  0.7410, -1.0683,
 0.2331,  0.7194,  0.6811,  0.8535, -0.9489,  1.3085,  0.4521, -0.7530,
-0.4710, -0.5114,  0.9316, -0.6176, -0.9984, -0.3081, -0.7611,  0.3254,
-1.0003, -0.4477,  0.7502,  0.8475, -0.9988,  0.1701, -0.6619, -0.4681,
-0.0220, -0.9991, -0.6949, -0.5006, -0.1002,  0.9738,  0.0348, -0.4650,
 0.4283, -0.9980, -0.8744, -0.2211, -0.9980, -0.4215,  0.5231,  0.1596,
-0.8722, -0.8193, -0.8674,  0.1850,  0.0448,  0.5637,  0.9993,  0.9999,
 0.2740,  0.4494, -0.7988, -0.9781, -0.7309,  0.8970, -0.8607,  0.3709,
 0.1728, -0.6509, -0.9888,  0.6999, -0.9803,  0.0823,  0.9026,  0.9764,
 0.1966, -0.6044,  0.7799,  0.9952,  0.1143, -1.7053, -1.2173,  0.9754,
-0.2165, -0.2952,  0.4226, -0.6355,  0.2572,  0.3864, -0.7035, -1.3023,
 0.9996, -1.4722, -0.1548, -0.3573, -0.2934, -2.0458, -0.9779,  0.2036,
 0.7885, -0.6443, -0.0598,  0.7566,  0.8553, -1.0000,  0.6106, -0.1187,
-0.9235, -0.5138, -0.4315, -0.6856,  0.3707, -0.9465,  0.2328, -0.9975,
-0.2525, -0.1977,  0.9688, -0.7922, -0.3662,  0.6767,  0.3484, -0.9786,
-0.9624, -1.6913, -0.8227,  0.4081, -0.9933,  1.0426,  0.7788, -0.0490]])
/torch/venv3/pytorch/lib/python3.6/site-packages/torch_mlu/core/quantized/observer.py:88: UserWarning: Must run observer before calling calculate_qparams.
Returning default scale and zero point
Returning default scale and zero point
mlu
layer-by-layer infer: tensor([[-18.0470, 18.2829, 18.0914, 17.3267, 17.0471]]) tensor([[ 1.9375,  0.1102, -0.1319, -0.9993,  0.6110, -0.0870,  0.3558,  0.9291,
-0.9360,  0.9635,  0.6380, -0.9998, -0.2930,  0.5981,  0.0874,  0.9669,
-0.9394, -0.7750,  0.9276,  0.9866, -0.9664,  0.8760, -0.1274, -0.5456,
 0.3033, -0.4042,  0.7675, -0.9992, -0.9307, -0.2275,  0.3027,  1.6201,
 0.9816,  0.6325,  1.9704, -0.1333, -0.3235, -0.5474, -0.0514,  1.2448,
-0.6302,  0.1904, -0.9545, -0.3017,  0.0742, -0.9751,  0.0502,  0.9994,
 0.4746,  0.9961, -0.9228, -0.1006, -0.7207, -0.1460, -0.9895, -0.7787,
-0.8218,  0.9995,  0.7794, -0.3583,  0.9933,  0.9987, -0.6987, -0.8874,
 0.6153, -0.3958, -0.9846,  1.1345, -0.6685, -0.9998, -0.9759, -0.9978,
-0.7818,  0.9888,  0.6612, -1.1483,  0.4634,  0.5687, -0.9994, -0.4887,
 0.5602,  0.7508, -0.8092,  0.7335,  0.2307, -0.9772,  0.5072,  0.9651,
 0.3848,  0.9562,  0.9996,  0.6954, -0.8795, -0.9991, -0.5693,  0.7498,
-0.0480,  0.1219, -0.9752,  0.0977,  0.8600,  0.9956,  0.9994, -0.8461,
-0.0536, -0.2528, -0.9867,  0.5574,  1.0003,  0.9676, -0.1620,  0.5452,
 0.9966,  0.2491,  0.9407,  0.2304, -0.1037, -0.2876, -0.6797,  1.0843,
-0.6420,  0.2716, -0.8495, -0.2846, -0.2936,  0.3225, -0.4802,  0.0193,
-0.8431, -0.2179,  0.8132, -0.2497,  0.1469, -0.8785,  0.9286, -0.9913,
 0.4439,  0.7989,  0.7415,  0.5395, -0.9905,  1.4072,  0.4375, -0.7526,
-0.6531,  0.0751,  0.9759, -0.6437, -0.9963, -0.5954, -0.5713,  0.2031,
-1.0005,  0.1241,  0.8365,  0.8601, -0.9975,  0.0608,  0.3834, -0.0989,
-0.0575, -0.9996, -0.8809, -0.5048, -0.0434, -1.0046, -0.5154, -0.5369,
 0.6660, -0.9987, -0.7921,  0.6330, -0.9973, -0.4353,  0.9106,  0.2646,
-0.8630, -0.8206, -0.8291,  0.1839, -0.1167,  0.9514,  0.9986,  0.9988,
 0.2749,  0.7155, -0.7877, -0.9864, -0.5029,  0.7532, -0.8581,  0.6681,
 0.1735, -0.2262, -0.9885,  0.6641, -0.9942,  0.0815,  0.9921,  0.9590,
 0.1755, -0.4895,  0.7895,  0.9971,  0.1160, -1.6983, -1.3781,  0.9978,
-0.3544, -0.3636,  0.4776, -0.6303,  0.0706,  0.8397, -0.5012, -1.4890,
 0.9997, -1.4736, -0.1544,  0.0243, -0.1511, -2.0609, -0.9884,  0.3453,
 0.6906, -0.6441,  0.0601,  0.7852,  0.7141, -0.9990,  0.6106, -0.2658,
-0.8306, -0.6779, -0.7421, -0.0774,  0.6582, -0.9913, -0.2653, -0.9991,
-0.2397, -0.4949,  0.9950, -0.7909, -0.1712,  0.6682, -0.9191, -0.9911,
-0.9997, -1.6977, -0.1730,  0.2552, -0.9960,  1.0633,  0.3299, -0.0258]])

```

## 1.5

## 2、使用http传输数据

### 2.1 客户端使用http

客户端用python实现，可以直接安装requests库即可。

### 2.2 服务端使用http

使用httplib开源库，github地址：<https://github.com/yhirose/cpp-httplib?tab=readme-ov-file#server-multi-threaded>

这是一个header-only library，因此可以直接下载头文件放到项目中使用即可。

### 3、使用json

#### 3.1 服务端使用json

cpp没有原生的json库，所以需要使用开源的库，这里选用了jsoncpp，github地址：<https://github.com/open-source-parsers/jsoncpp>

(1) 下载源码然后按照如下步骤编译生成静态和动态连接库即可：

```
1  mkdir jsoncpp-Sandbox
2  cd jsoncpp-sandbox
3  git clone https://github.com/open-source-parsers/jsoncpp.git # or download &
   unpack the source tarball
4  mkdir jsoncpp-build
5  # this will create the following directory structure:
6  # jsoncpp-Sandbox
7  #   |--jsoncpp
8  #   |--jsoncpp-build
9
10 Then you can proceed to configure and build
11 by using the following commands
12
13 cd jsoncpp-build
14 cmake ../jsoncpp # or ccmake, or cmake-gui
15 make
```

(2) 结果：用的就是libjsoncpp.a和libjsoncpp.so：

```
-- Build files have been written to: /home/tct/jsoncpp-sandbox/jsoncpp-build
tct@ict-Precision-3630-Tower:~/jsoncpp-Sandbox/jsoncpp-build$ make
Scanning dependencies of target jsoncpp_static
[ 5%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_static.dir/json_reader.cpp.o
[ 11%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_static.dir/json_value.cpp.o
[ 17%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_static.dir/json_writer.cpp.o
[ 23%] Linking CXX static library ../../lib/libjsoncpp.a
[ 23%] Built target jsoncpp_static
Scanning dependencies of target jsoncpp_lib
[ 29%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_lib.dir/json_reader.cpp.o
[ 35%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_lib.dir/json_value.cpp.o
[ 41%] Building CXX object src/lib_json/CMakeFiles/jsoncpp_lib.dir/json_writer.cpp.o
[ 47%] Linking CXX shared library ../../lib/libjsoncpp.so
[ 47%] Built target jsoncpp_lib
```

