智能计算系统实验五报告

时昌军 221220085

一、实验名称

Ascend C 算子开发

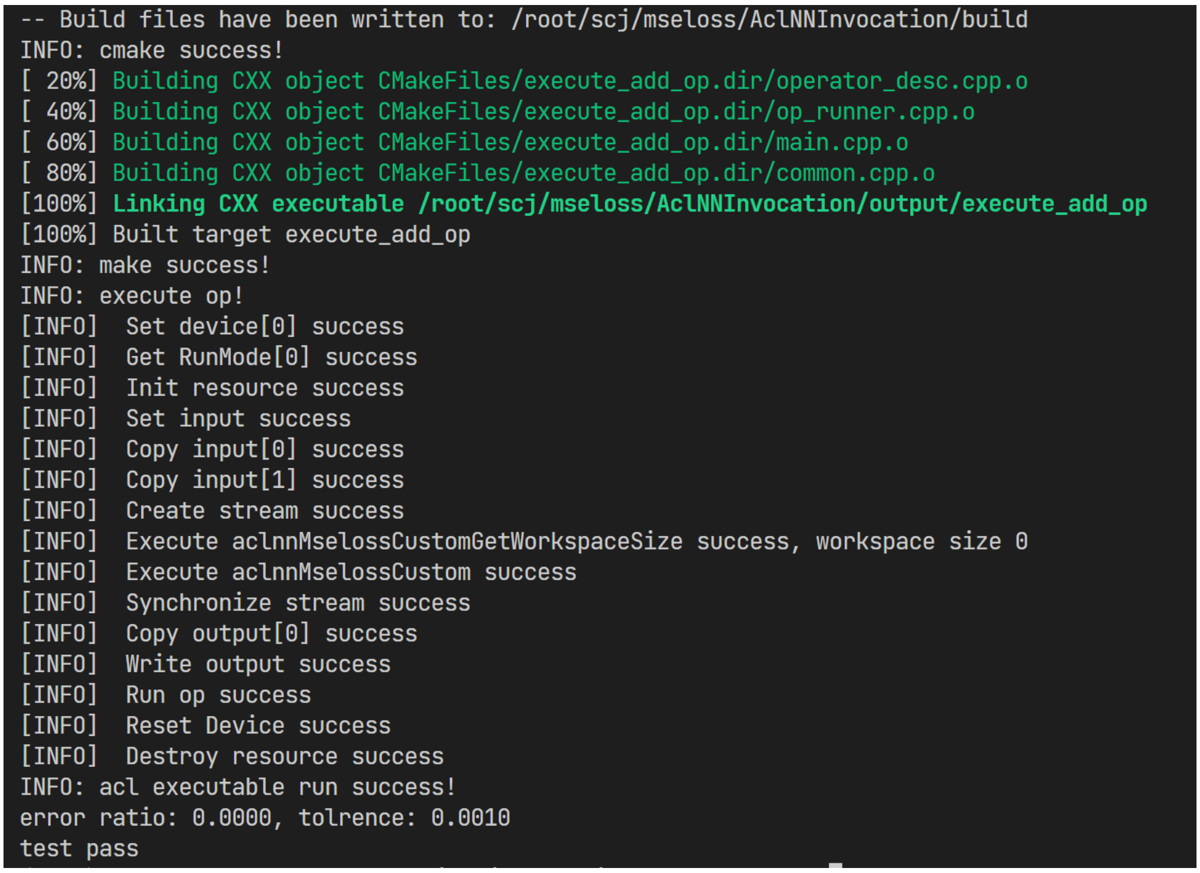
二、实验目的

基于 AscendC 算子开发语言开发运行在 AI Core 上的自定义算子，并对自定义开发的算子进行部署和验证。从而基本掌握 AscendC 的使用，并对让对算子开发工程有宏观的认识。

三、实验结果

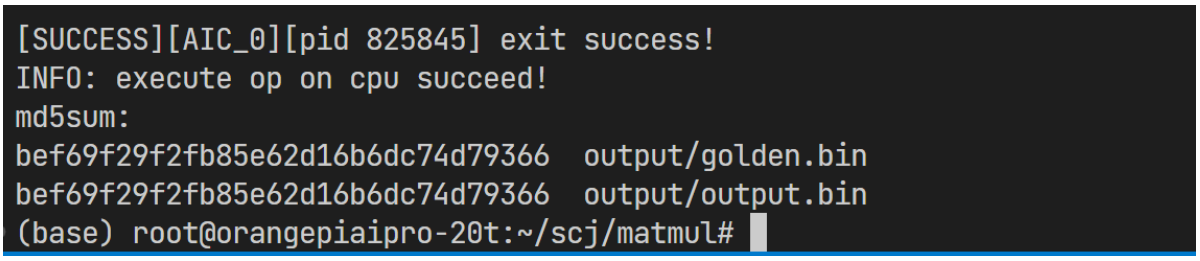
实现给定三个算子的开发和测试，通过 PyTorch 将开发的算子集成到网络中。

1、mseloss三种测试均通过

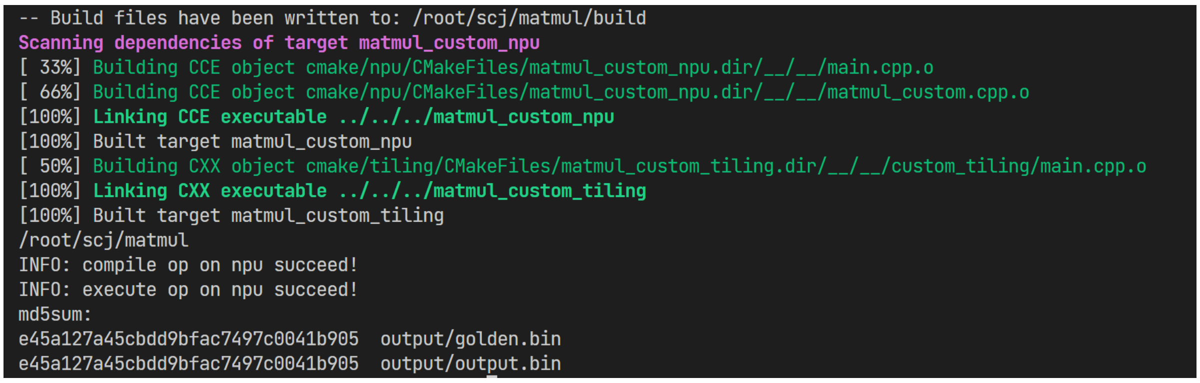
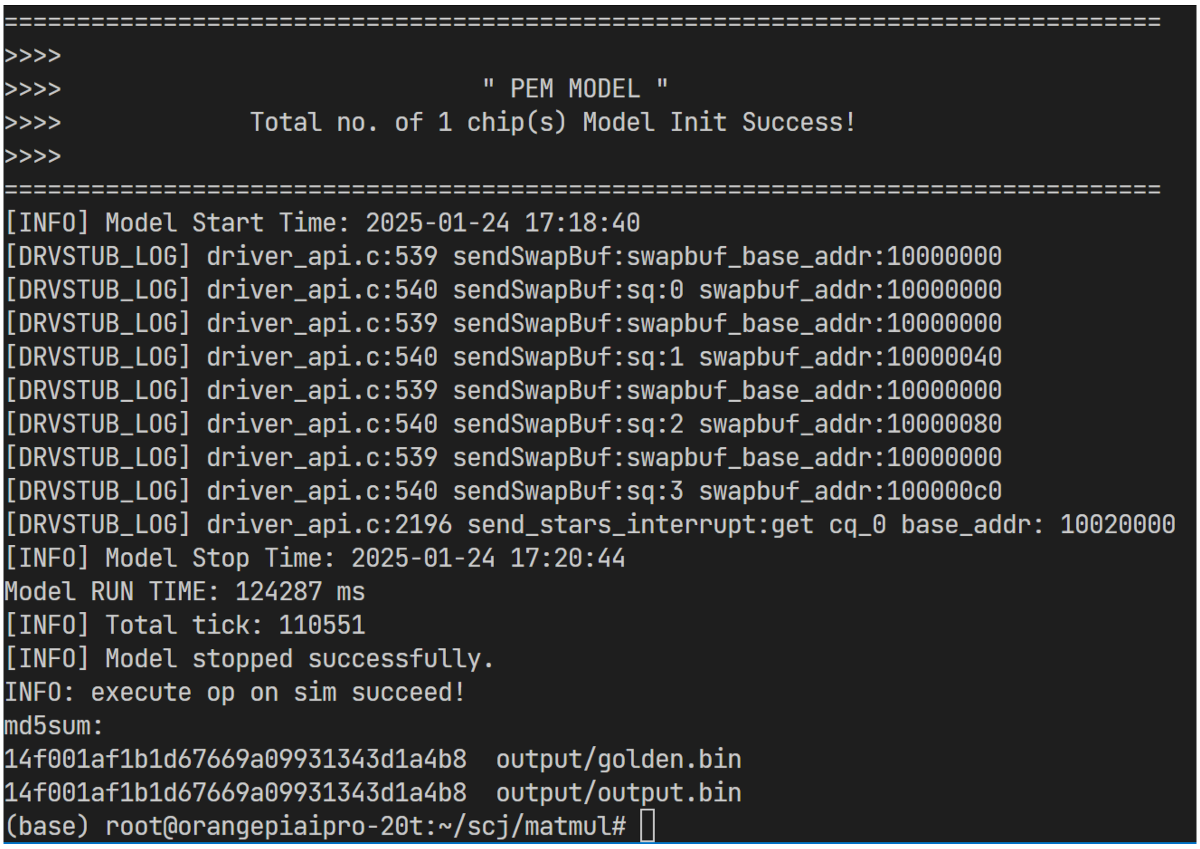


```
-- Build files have been written to: /root/scj/mseloss/AclNNInvocation/build
INFO: cmake success!
[ 20%] Building CXX object CMakeFiles/execute_add_op.dir/operator_desc.cpp.o
[ 40%] Building CXX object CMakeFiles/execute_add_op.dir/op_runner.cpp.o
[ 60%] Building CXX object CMakeFiles/execute_add_op.dir/main.cpp.o
[ 80%] Building CXX object CMakeFiles/execute_add_op.dir/common.cpp.o
[100%] Linking CXX executable /root/scj/mseloss/AclNNInvocation/output/execute_add_op
[100%] Built target execute_add_op
INFO: make success!
INFO: execute op!
[INFO]  Set device[0] success
[INFO]  Get RunMode[0] success
[INFO]  Init resource success
[INFO]  Set input success
[INFO]  Copy input[0] success
[INFO]  Copy input[1] success
[INFO]  Create stream success
[INFO]  Execute aclnnMselossCustomGetWorkspaceSize success, workspace size 0
[INFO]  Execute aclnnMselossCustom success
[INFO]  Synchronize stream success
[INFO]  Copy output[0] success
[INFO]  Write output success
[INFO]  Run op success
[INFO]  Reset Device success
[INFO]  Destroy resource success
INFO: acl executable run success!
error ratio: 0.0000, tolrence: 0.0010
test pass
```

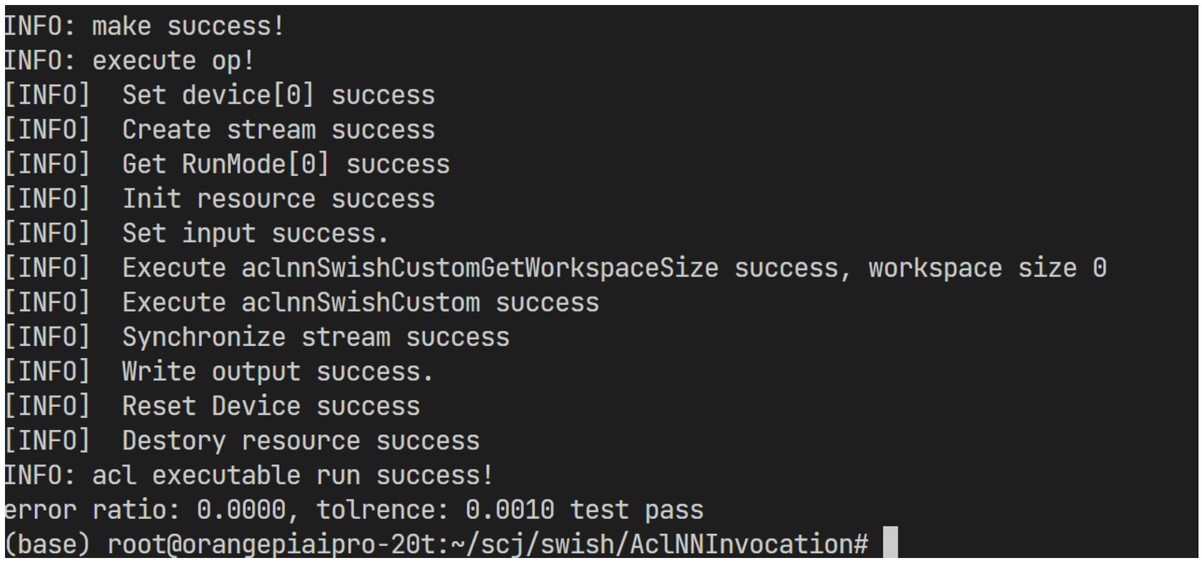
2.matmul三种测试均通过



```
[SUCCESS][AIC_0][pid 825845] exit success!
INFO: execute op on cpu succeed!
md5sum:
bef69f29f2fb85e62d16b6dc74d79366  output/golden.bin
bef69f29f2fb85e62d16b6dc74d79366  output/output.bin
(base) root@orangepiaipro-20t:~/scj/matmul#
```

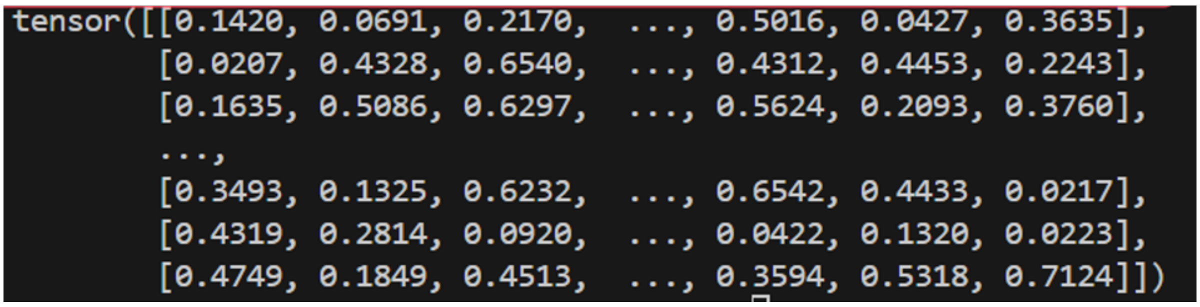

```
-- Build files have been written to: /root/scj/matmul/build
Scanning dependencies of target matmul_custom_npu
[ 33%] Building CCE object cmake/npu/CMakeFiles/matmul_custom_npu.dir/__/__/main.cpp.o
[ 66%] Building CCE object cmake/npu/CMakeFiles/matmul_custom_npu.dir/__/__/matmul_custom.cpp.o
[100%] Linking CCE executable ../../../matmul_custom_npu
[100%] Built target matmul_custom_npu
[ 50%] Building CXX object cmake/tiling/CMakeFiles/matmul_custom_tiling.dir/__/__/custom_tiling/main.cpp.o
[100%] Linking CXX executable ../../../matmul_custom_tiling
[100%] Built target matmul_custom_tiling
/root/scj/matmul
INFO: compile op on npu succeed!
INFO: execute op on npu succeed!
md5sum:
e45a127a45cbdd9bfac7497c0041b905  output/golden.bin
e45a127a45cbdd9bfac7497c0041b905  output/output.bin
```

```
================================================================================
>>>>
>>>>                             " PEM MODEL "
>>>>             Total no. of 1 chip(s) Model Init Success!
>>>>
================================================================================
[INFO] Model Start Time: 2025-01-24 17:18:40
[DRVSTUB_LOG] driver_api.c:539 sendSwapBuf:swapbuf_base_addr:10000000
[DRVSTUB_LOG] driver_api.c:540 sendSwapBuf:sq:0 swapbuf_addr:10000000
[DRVSTUB_LOG] driver_api.c:539 sendSwapBuf:swapbuf_base_addr:10000000
[DRVSTUB_LOG] driver_api.c:540 sendSwapBuf:sq:1 swapbuf_addr:10000040
[DRVSTUB_LOG] driver_api.c:539 sendSwapBuf:swapbuf_base_addr:10000000
[DRVSTUB_LOG] driver_api.c:540 sendSwapBuf:sq:2 swapbuf_addr:10000080
[DRVSTUB_LOG] driver_api.c:539 sendSwapBuf:swapbuf_base_addr:10000000
[DRVSTUB_LOG] driver_api.c:540 sendSwapBuf:sq:3 swapbuf_addr:100000c0
[DRVSTUB_LOG] driver_api.c:2196 send_stars_interrupt:get cq_0 base_addr: 10020000
[INFO] Model Stop Time: 2025-01-24 17:20:44
Model RUN TIME: 124287 ms
[INFO] Total tick: 110551
[INFO] Model stopped successfully.
INFO: execute op on sim succeed!
md5sum:
14f001af1b1d67669a09931343d1a4b8  output/golden.bin
14f001af1b1d67669a09931343d1a4b8  output/output.bin
(base) root@orangepiaipro-20t:~/scj/matmul#
```

3.swish三种测试均通过


```
INFO: make success!
INFO: execute op!
[INFO]  Set device[0] success
[INFO]  Create stream success
[INFO]  Get RunMode[0] success
[INFO]  Init resource success
[INFO]  Set input success.
[INFO]  Execute aclnnSwishCustomGetWorkspaceSize success, workspace size 0
[INFO]  Execute aclnnSwishCustom success
[INFO]  Synchronize stream success
[INFO]  Write output success.
[INFO]  Reset Device success
[INFO]  Destory resource success
INFO: acl executable run success!
error ratio: 0.0000, tolrence: 0.0010 test pass
(base) root@orangepiaipro-20t:~/scj/swish/AclNNInvocation#
```

4.算子集成

把动态库绑定了pytorch，运行出的结果



```
tensor([[0.1420, 0.0691, 0.2170,  ..., 0.5016, 0.0427, 0.3635],
        [0.0207, 0.4328, 0.6540,  ..., 0.4312, 0.4453, 0.2243],
        [0.1635, 0.5086, 0.6297,  ..., 0.5624, 0.2093, 0.3760],
        ...,
        [0.3493, 0.1325, 0.6232,  ..., 0.6542, 0.4433, 0.0217],
        [0.4319, 0.2814, 0.0920,  ..., 0.0422, 0.1320, 0.0223],
        [0.4749, 0.1849, 0.4513,  ..., 0.3594, 0.5318, 0.7124]])
```

四、实验过程

mseloss

1.创建算子工程

`/usr/local/Ascend/ascend-toolkit/latest/python/site-packages/bin/msopgen -i gen MselossCustom.json -out gen -lan cpp -c ai_core-Ascend310B`

2.在 CMakePresets.json 中修改 CANN 安装目录

```
"ASCEND_CANN_PACKAGE_PATH": {
    "type": "PATH",
    "value": "/usr/local/Ascend/ascend-toolkit/latest"
}
```

3.Host 侧算子实现

.h 文件主要用于定义算子的 tiling，其中，totalLength 就表示总计算数据量，tileNum 就表示每个核上总计算数据分块个数。

```
  TILING_DATA_FIELD_DEF(uint32_t, totalLength);
  TILING_DATA_FIELD_DEF(uint32_t, tileNum);
```

.cpp 文件进行 Tiling 的实现。修改“TilingFunc”函数，实现 Tiling 上下文的获取，并通过上下文获取输入输出 shape 信息，并根据 shape 信息设置 TilingData、序列化保存 TilingData ，并设置 TilingKey。

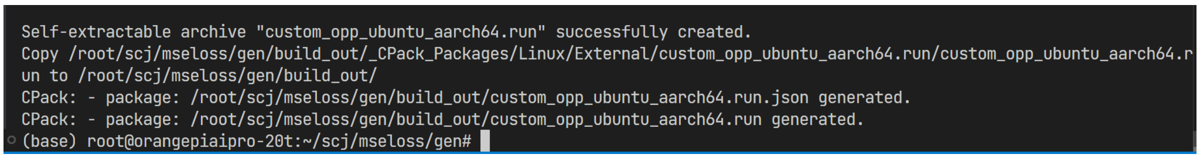
```
...
tiling.set_totalLength(data_sz);
tiling.set_tileNum(8);
...
```

4.算子核函数实现

参考昇腾代码仓库的示例代码实现

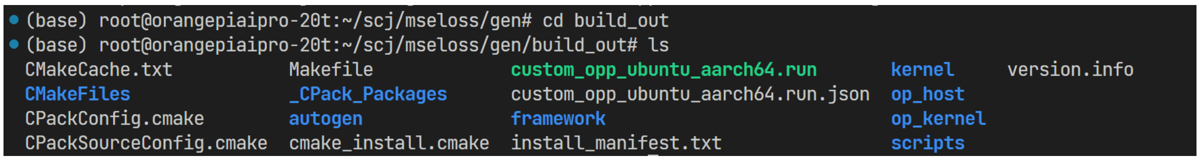
```
extern "C" __global__ __aicore__ void mseloss_custom(GM_ADDR x, GM_ADDR y, GM_ADDR z, GM_ADDR workspace, GM_ADDR tiling) {
    GET_TILING_DATA(tiling_data, tiling);
    KernelMseLoss op;
    op.Init(x, y, z, tiling);
    op.Process();
}
```

5.通过 build.sh 文件编译算子

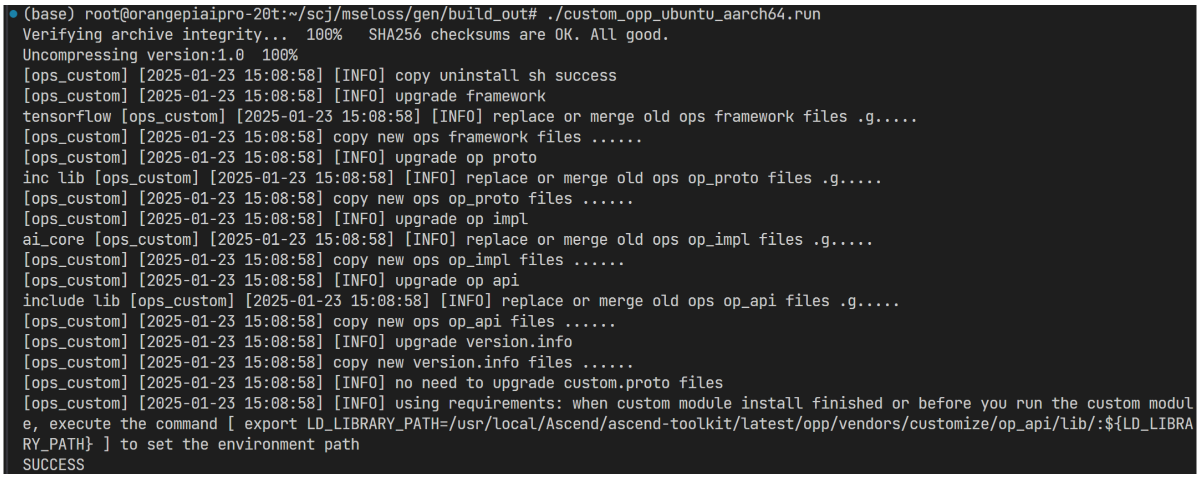


```
Self-extractable archive "custom_opp_ubuntu_aarch64.run" successfully created.
Copy /root/scj/mseloss/gen/build_out/_CPack_Packages/Linux/External/custom_opp_ubuntu_aarch64.run/custom_opp_ubuntu_aarch64.r
un to /root/scj/mseloss/gen/build_out/
CPack: - package: /root/scj/mseloss/gen/build_out/custom_opp_ubuntu_aarch64.run.json generated.
CPack: - package: /root/scj/mseloss/gen/build_out/custom_opp_ubuntu_aarch64.run generated.
(base) root@orangepiaipro-20t:~/scj/mseloss/gen# 
```

6.注册算子

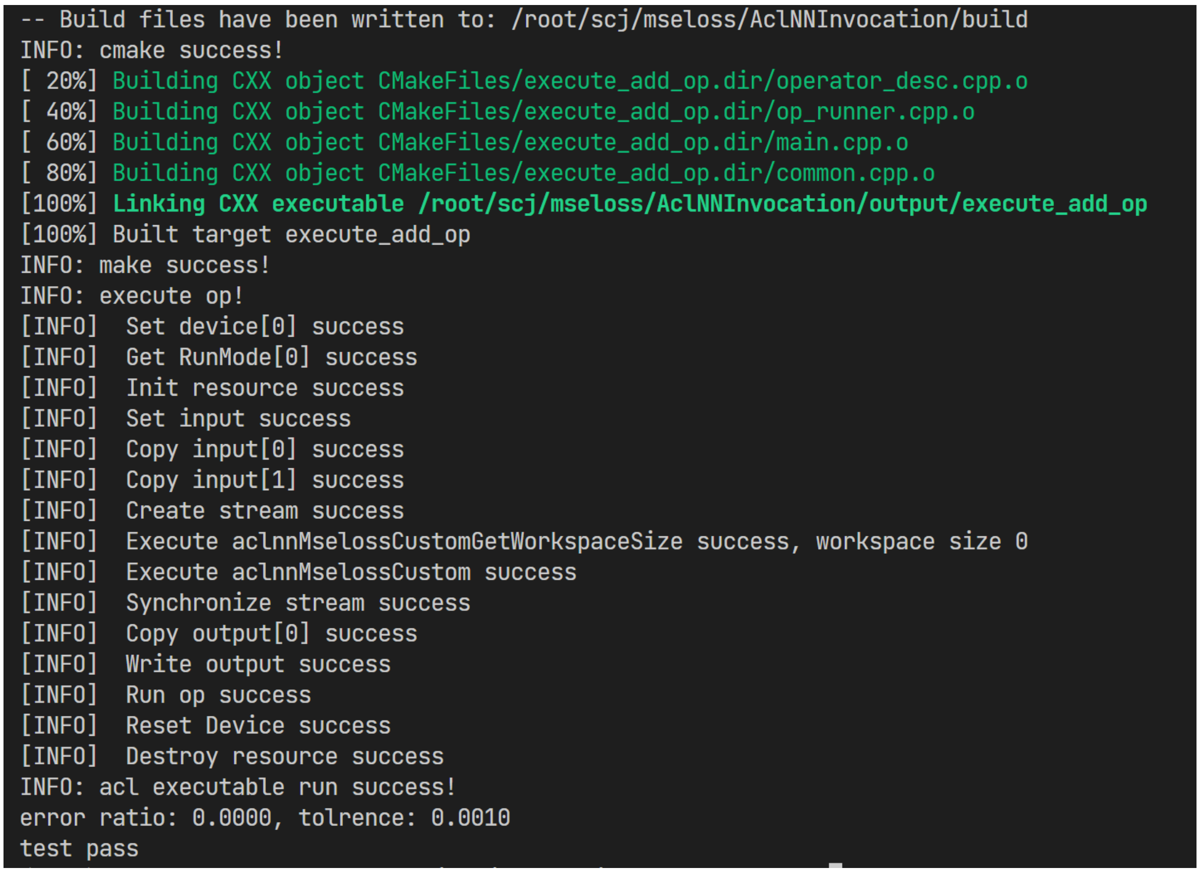


```
(base) root@orangepiaipro-20t:~/scj/mseloss/gen# cd build_out
(base) root@orangepiaipro-20t:~/scj/mseloss/gen/build_out# ls
CMakeCache.txt          Makefile              custom_opp_ubuntu_aarch64.run       kernel     version.info
CMakeFiles              _CPack_Packages       custom_opp_ubuntu_aarch64.run.json  op_host
CPackConfig.cmake       autogen               framework                           op_kernel
CPackSourceConfig.cmake cmake_install.cmake   install_manifest.txt                scripts
```



```
(base) root@orangepiaipro-20t:~/scj/mseloss/gen/build_out# ./custom_opp_ubuntu_aarch64.run
Verifying archive integrity...  100%   SHA256 checksums are OK. All good.
Uncompressing version:1.0  100%
[ops_custom] [2025-01-23 15:08:58] [INFO] copy uninstall sh success
[ops_custom] [2025-01-23 15:08:58] [INFO] upgrade framework
tensorflow [ops_custom] [2025-01-23 15:08:58] [INFO] replace or merge old ops framework files .g.....
[ops_custom] [2025-01-23 15:08:58] copy new ops framework files ......
[ops_custom] [2025-01-23 15:08:58] [INFO] upgrade op proto
inc lib [ops_custom] [2025-01-23 15:08:58] [INFO] replace or merge old ops op_proto files .g.....
[ops_custom] [2025-01-23 15:08:58] copy new ops op_proto files ......
[ops_custom] [2025-01-23 15:08:58] [INFO] upgrade op impl
ai_core [ops_custom] [2025-01-23 15:08:58] [INFO] replace or merge old ops op_impl files .g.....
[ops_custom] [2025-01-23 15:08:58] copy new ops op_impl files ......
[ops_custom] [2025-01-23 15:08:58] [INFO] upgrade op api
include lib [ops_custom] [2025-01-23 15:08:58] [INFO] replace or merge old ops op_api files .g.....
[ops_custom] [2025-01-23 15:08:58] copy new ops op_api files ......
[ops_custom] [2025-01-23 15:08:58] [INFO] upgrade version.info
[ops_custom] [2025-01-23 15:08:58] copy new version.info files ......
[ops_custom] [2025-01-23 15:08:58] [INFO] no need to upgrade custom.proto files
[ops_custom] [2025-01-23 15:08:58] [INFO] using requirements: when custom module install finished or before you run the custom modul
e, execute the command [ export LD_LIBRARY_PATH=/usr/local/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_api/lib/:${LD_LIBRA
RY_PATH} ] to set the environment path
SUCCESS
```

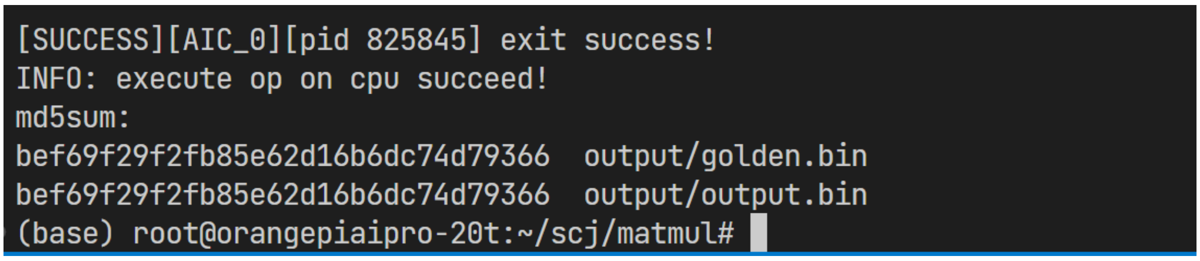
7.运行测试脚本



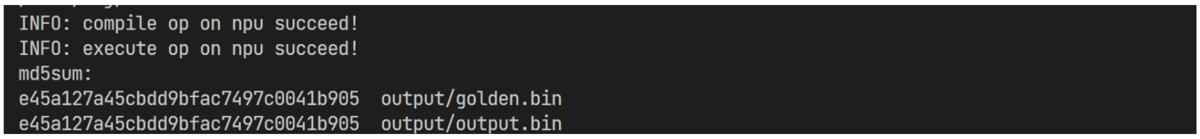
```
-- Build files have been written to: /root/scj/mseloss/AclNNInvocation/build
INFO: cmake success!
[ 20%] Building CXX object CMakeFiles/execute_add_op.dir/operator_desc.cpp.o
[ 40%] Building CXX object CMakeFiles/execute_add_op.dir/op_runner.cpp.o
[ 60%] Building CXX object CMakeFiles/execute_add_op.dir/main.cpp.o
[ 80%] Building CXX object CMakeFiles/execute_add_op.dir/common.cpp.o
[100%] Linking CXX executable /root/scj/mseloss/AclNNInvocation/output/execute_add_op
[100%] Built target execute_add_op
INFO: make success!
INFO: execute op!
[INFO]  Set device[0] success
[INFO]  Get RunMode[0] success
[INFO]  Init resource success
[INFO]  Set input success
[INFO]  Copy input[0] success
[INFO]  Copy input[1] success
[INFO]  Create stream success
[INFO]  Execute aclnnMselossCustomGetWorkspaceSize success, workspace size 0
[INFO]  Execute aclnnMselossCustom success
[INFO]  Synchronize stream success
[INFO]  Copy output[0] success
[INFO]  Write output success
[INFO]  Run op success
[INFO]  Reset Device success
[INFO]  Destroy resource success
INFO: acl executable run success!
error ratio: 0.0000, tolrence: 0.0010
test pass
```

matmul

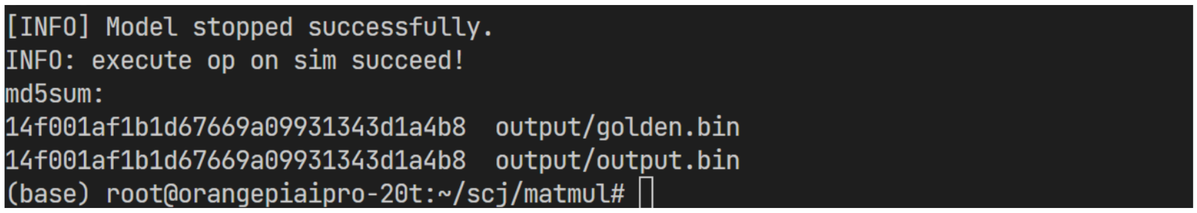
MatmulWithBias 仅要求实现为 Kernel 直调开发方式。且昇腾仓库的示例代码中，已经有完整 Matmul 算子实现。以此为参考即可。



```
[SUCCESS][AIC_0][pid 825845] exit success!
INFO: execute op on cpu succeed!
md5sum:
bef69f29f2fb85e62d16b6dc74d79366  output/golden.bin
bef69f29f2fb85e62d16b6dc74d79366  output/output.bin
(base) root@orangepiaipro-20t:~/scj/matmul#
```



```
INFO: compile op on npu succeed!
INFO: execute op on npu succeed!
md5sum:
e45a127a45cbdd9bfac7497c0041b905  output/golden.bin
e45a127a45cbdd9bfac7497c0041b905  output/output.bin
```



```
[INFO] Model stopped successfully.
INFO: execute op on sim succeed!
md5sum:
14f001af1b1d67669a09931343d1a4b8  output/golden.bin
14f001af1b1d67669a09931343d1a4b8  output/output.bin
(base) root@orangepiaipro-20t:~/scj/matmul#
```

swish

1.创建算子工程

`/usr/local/Ascend/ascend-toolkit/latest/python/site-packages/bin/msopgen gen -i SwishCustom.json -out gen -lan cpp -c ai_core-Ascend310B`

2.在 CMakePresets.json 中修改 CANN 安装目录

```
"ASCEND_CANN_PACKAGE_PATH": {
    "type": "PATH",
    "value": "/usr/local/Ascend/ascend-toolkit/latest"
}
```

3.Host 侧算子实现

.h 文件主要用于定义算子的 tiling，其中，totalLength 就表示总计算数据量，tileNum 就表示每个核上总计算数据分块个数。

```
  TILING_DATA_FIELD_DEF(uint32_t, totalLength);
  TILING_DATA_FIELD_DEF(uint32_t, tileNum);
```

.cpp 文件进行 Tiling 的实现。修改“TilingFunc”函数，实现 Tiling 上下文的获取，并通过上下文获取输入输出 shape 信息，并根据 shape 信息设置 TilingData、序列化保存 TilingData ，并设置 TilingKey。

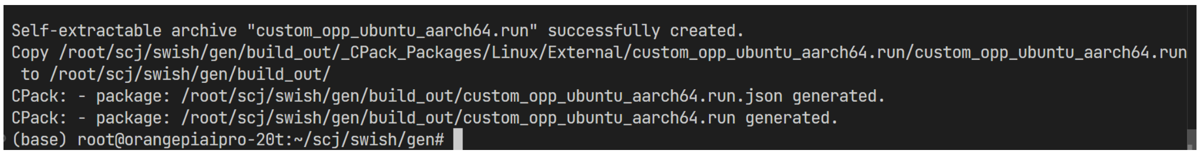
```
...
tiling.set_totalLength(data_sz);
tiling.set_tileNum(8);
...
```

4.算子核函数实现

参考昇腾代码仓库的示例代码实现

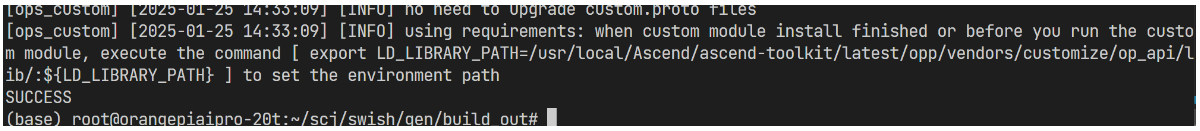
```
__aicore__ inline void Compute(int32_t progress)
{
    // 从输入队列中取出张量
    LocalTensor<float> xLocal = inQueueX.DeQue<float>();
    LocalTensor<float> yLocal = outQueueY.AllocTensor<float>();
    // Leaky ReLU实现：计算 y=x*max(0,x) ,将值1填充到tmpTensor
    LocalTensor<float> tmpTensor = tmpBuffer.Get<float>();
    float one(1), negOne(-1);
    AscendC::Duplicate<float>(tmpTensor, one, this->tileLength);
    // 计算 y=x*(-1) 并取指数, y=y+1,y=1/y,y=x*y
    Muls(yLocal, xLocal, negOne, this->tileLength);
    Exp(yLocal, yLocal, this->tileLength);
    Adds(yLocal, yLocal, one, this->tileLength);
    Div(yLocal, tmpTensor, yLocal, this->tileLength);
    Mul(yLocal, yLocal, xLocal, this->tileLength);
    // 将结果加入输出队列,释放输入张量, 便于重用
    outQueueY.EnQue<float>(yLocal);
    inQueueX.FreeTensor(xLocal);
}
```

5.通过 build.sh 文件编译算子



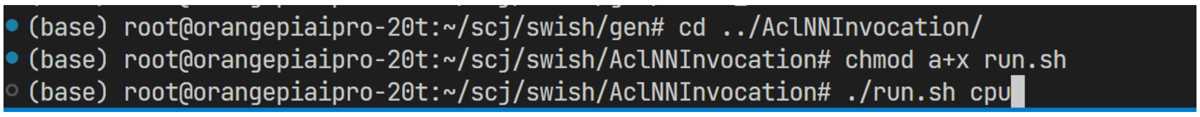
```
Self-extractable archive "custom_opp_ubuntu_aarch64.run" successfully created.
Copy /root/scj/swish/gen/build_out/_CPack_Packages/Linux/External/custom_opp_ubuntu_aarch64.run/custom_opp_ubuntu_aarch64.run
 to /root/scj/swish/gen/build_out/
CPack: - package: /root/scj/swish/gen/build_out/custom_opp_ubuntu_aarch64.run.json generated.
CPack: - package: /root/scj/swish/gen/build_out/custom_opp_ubuntu_aarch64.run generated.
(base) root@orangepiaipro-20t:~/scj/swish/gen#
```

6.注册算子

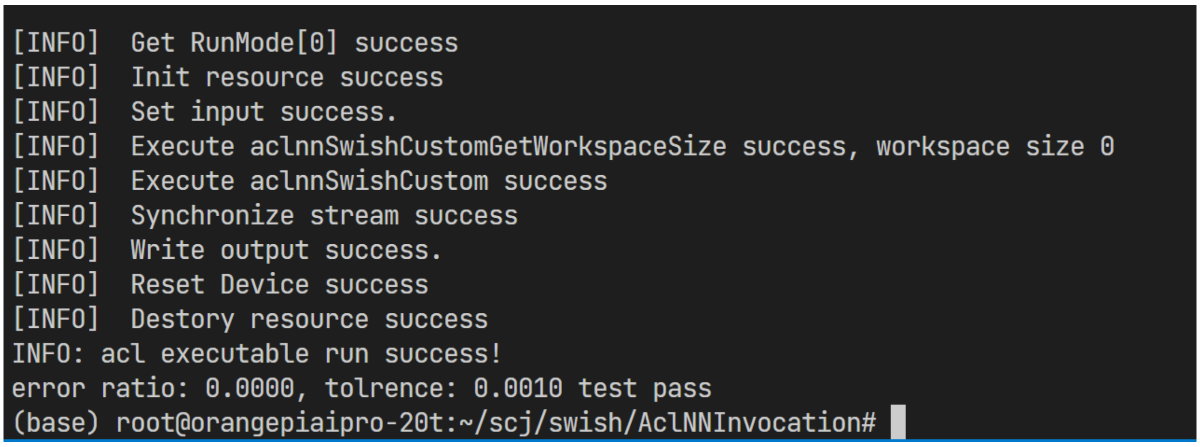


```
[ops_custom] [2025-01-25 14:33:09] [INFO] no need to upgrade custom.proto files
[ops_custom] [2025-01-25 14:33:09] [INFO] using requirements: when custom module install finished or before you run the custo
m module, execute the command [ export LD_LIBRARY_PATH=/usr/local/Ascend/ascend-toolkit/latest/opp/vendors/customize/op_api/l
ib/:${LD_LIBRARY_PATH} ] to set the environment path
SUCCESS
(base) root@orangepiaipro-20t:~/scj/swish/gen/build_out#
```

7.运行测试脚本



```
(base) root@orangepiaipro-20t:~/scj/swish/gen# cd ../AclNNInvocation/
(base) root@orangepiaipro-20t:~/scj/swish/AclNNInvocation# chmod a+x run.sh
(base) root@orangepiaipro-20t:~/scj/swish/AclNNInvocation# ./run.sh cpu
```



```
[INFO]  Get RunMode[0] success
[INFO]  Init resource success
[INFO]  Set input success.
[INFO]  Execute aclnnSwishCustomGetWorkspaceSize success, workspace size 0
[INFO]  Execute aclnnSwishCustom success
[INFO]  Synchronize stream success
[INFO]  Write output success.
[INFO]  Reset Device success
[INFO]  Destory resource success
INFO: acl executable run success!
error ratio: 0.0000, tolrence: 0.0010 test pass
(base) root@orangepiaipro-20t:~/scj/swish/AclNNInvocation#
```

`./run.sh npu` `./run.sh sim`的结果类似

算子集成

1.开一个新环境

`export LD_LIBRARY_PATH=$ASCEND_OPP_PATH/vendors/customize/op_api/lib/:$LD_LIBRARY_PATH`

创建环境变量并克隆`pytorch`

```
git clone https://gitee.com/ascend/pytorch.git -b v6.0.rc2.alpha003-pytorch2.1.0 --recursive
cd pytorch/third_party/op-plugin
```

打开`op_plugin/config/v2r1/op_plugin_functions.yaml`文件

将如下信息拷贝至`op_plugin_functions.yaml`中的`custom:`节点下

```
- func: npu_mseloss_custom(Tensor x, Tensor y) -> Tensor
  impl_ns: op_api
- func: npu_swish_custom(Tensor x) -> Tensor
  impl_ns: op_api
```

2.集成，在pytorch框架中运行

在`pytorch/third_party/op-plugin/op_plugin/ops/v2r1/opapi`目录下，创建`MselossCustomKernelNpu.cpp`、`SwishCustomKernelNpu.cpp`文件并实现算子适配主体函数`npu_mseloss_custom`、`npu_swish_custom`。其核心逻辑为调用`EXEC_NPU_CMD`接口，完成输出结果的计算，`EXEC_NPU_CMD`第一个入参格式为`aclnn+Optype`，之后的参数分别为输入输出。

```
namespace op_api {
using npu_preparation = at_npu::native::OpPreparation;

at::Tensor npu_mseloss_custom(const at::Tensor& x, const at::Tensor& y) {
  // 构造输出tensor
  at::Tensor result = npu_preparation::apply_tensor_without_format(x);
  // 计算输出结果
  // 调用EXEC_NPU_CMD接口，完成输出结果的计算
  // 第一个入参格式为aclnn+Optype，之后的参数分别为输入输出
  EXEC_NPU_CMD(aclnnMselossCustom, x, y, result);
  return result;
}
}
```

```
namespace op_api {
using npu_preparation = at_npu::native::OpPreparation;

at::Tensor npu_swish_custom(const at::Tensor& x) {
  // 构造输出tensor
  at::Tensor result = npu_preparation::apply_tensor_without_format(x);
  // 计算输出结果
  // 调用EXEC_NPU_CMD接口，完成输出结果的计算
  // 第一个入参格式为aclnn+Optype，之后的参数分别为输入输出
  EXEC_NPU_CMD(aclnnSwishCustom, x, result);
  return result;
}
}
```

用指令安装pytorch

```
bash ci/build.sh --python=3.8
pip3 install dist/*.whl --force-reinstall
```

测试代码如下

```python
torch.npu.config.allow_internal_format = False
length = [8, 2048]
x = torch.rand(length, device='cpu', dtype=torch.float32)
y = torch.rand(length, device='cpu', dtype=torch.float32)
print(x, '\n', y)

torch.npu.synchronize()
output1 = torch_npu.npu_swish_custom(x.npu()).cpu()
torch.npu.synchronize()


print(output1)
print(x*torch.sigmoid(x))
```

五、思考题

1、是否可以通过这个实验学习如何查阅文档，解决开发中遇到的问题。

答：通过这个实验，我学会了如何查阅昇腾文档并解决开发中遇到的问题。在实现算子时，我首先查阅了昇腾文档，了解了其对算子融合的支持和具体实现方法，掌握了如何利用昇腾平台的优化工具来提升计算效率。文档中提供的API、性能调优技巧以及示例代码，帮助我快速理解了如何在实际开发中应用算子融合技术。同时，文档中的问题解答和最佳实践也为我在开发过程中遇到的瓶颈提供了有效的解决方案，使我能够高效地优化模型并提升执行性能。

2、算子融合是一种常见的算子优化方法，指将多个算子的计算逻辑级联成一个算子的计算逻辑，可以节省数据从内存中搬入搬出的时间，优化执行效率，从而加速计算。查阅其他相关资料，了解算子融合。

答： 算子融合的基本原理是将多个相邻的算子合并为一个单一的算子，通过减少数据在内存中的传输次数，避免重复的内存读写，降低内存带宽压力，提升计算效率。通过这种方式，多个算子的计算逻辑被串联在一起，从而减少了内存访问和中间数据存储，进而提高了计算性能，尤其是在硬件加速器上。

它被广泛应用于深度学习模型的优化和硬件加速中。在深度学习框架中，算子融合通常用于优化推理性能，减少内存访问，提升计算速度；在硬件加速器上，算子融合能够充分利用硬件的计算能力，减少中间数据存储，提高带宽利用率。算子融合还可以用于定制硬件设计中，以提高整体执行效率，减少能耗。

一个常见的算子融合例子是卷积与激活函数的融合。在没有融合的情况下，卷积操作会生成中间结果并存储，然后激活函数再对其进行处理。通过算子融合，卷积和激活函数的计算可以合并为一个步骤，避免了中间数据的存储和内存传输，从而减少了计算时间和内存带宽消耗。另一个例子是卷积与批归一化操作的融合，避免了额外的批归一化操作，提高了推理速度。

3、你开发的自定义算子有哪些优化空间?

答：在自定义算子的开发中，内存优化是一个关键的性能提升手段。内存访问通常是深度学习计算中最昂贵的操作之一，尤其在GPU或TPU等加速器上，内存带宽的使用直接影响到计算速度。通过优化内存管理，可以减少不必要的内存分配、释放和数据搬移。比如，采用内存池技术管理内存的分配与回收，避免频繁的内存分配和释放操作，减少内存碎片。此外，优化数据的内存布局也能提升性能，选择合适的数据存储格式可以提高内存访问的局部性，减少数据访问的延迟，特别是在GPU或其他并行计算平台上，可以更高效地利用硬件的内存访问模式，进一步加速算子计算。