南京大学

智 能 计 算 系 统
实 验 操 作 手 册

实 验 序 号	五
实 验 名 称	AscendC 算子实现
院　　系	计算机学院

2025 年 12 月 17 日

目　录

实验五　AscendC 算子实现

5.1　实验概览

本实验旨在探索基于华为 AscendC 编程范式的异构计算算子开发流程。实验针对昇腾 AI 处理器架构，设计并实现了三个核心算子：L1Loss、Softplus（向量计算算子）以及 Matmul（矩阵计算算子）。以上算子最终将部署在香橙派开发板上进行测试和执行。

5.2　实验目的

本实验旨在深入理解算子执行背后的原理，包括数据搬运，内存管理，ai-core 的计算过程，以及通信和同步机制。通过本实验，学生将初步了解算子开发的实际流程，熟悉相关的编程范式和常用接口并提升编程能力。深入理解在算力受限的端侧设备及昇腾系列 NPU 上运行大语言模型的优化策略与机制，掌握从框架级导出到硬件执行的完整部署流程，并能根据端侧硬件的特性（如算力、内存约束等）设计针对性的优化方案。

本实验主要目的如下：

1. 理解算子执行的原理和基本流程。掌握 AscendC 开发的算子是如何在昇腾芯片上运行，包括数据流动，计算过程以及异步执行和同步机制。
2. 掌握算子开发的流程，学习搭建算子开发的项目架构，使用 msopgen 工具来创建新的算子工程并逐步完成开发。
3. 学习算子调用过程，在完成算子实现后，安装部署算子，并使用 AscendC 编程语言调用自己实现的算子。

本实验将在搭载昇腾 310B NPU 的香橙派 AI Pro 上开展。实验将实现 3 个算子，每个算子依次完成创建算子工程、实现算子、本地部署算子、调用算子测试。

5.3 环境准备

5.3.1 切换为 root 用户

使用普通 user 可能在部署安装算子等步骤中，会遇到权限问题，为了减少不必要的麻烦，这里推荐切换成 root 用户完成实验 5.

5.3.2 检查环境变量

检查 `ASCEND_HOME_PATH`

是否设置为 /usr/local/Ascend/ascend-toolkit/latest/

如果没有，执行

```
source /usr/local/Ascend/ascend-toolkit/set_env.sh
```

5.3.3 实验项目结构

解压我们提供的压缩包 `lab5.zip` 会得到如图5-1所示的目录结构。其中 Json 文件用于指定算子的属性，如输入输出的数据类型和张量排布等。*AclNNInvocation* 用于算子的测试，无需改动。*xxxCustom* 就是算子工程。

5.4 参考材料

5.4.1 AscendC 文档

1. add 的 AscendC 文档
2. 矩阵乘法的 AscendC 文档
3. AscendC API 列表

!!! 如果你自行到 AscendC 官网上查找相关文档，请注意你使用的文档的版本，本次实验规定的 CANN 版本为 8.0.0，不同文档的 API 可能会有较大差异，我们无法保证实验手册在不同 CANN 版本上的通用性。

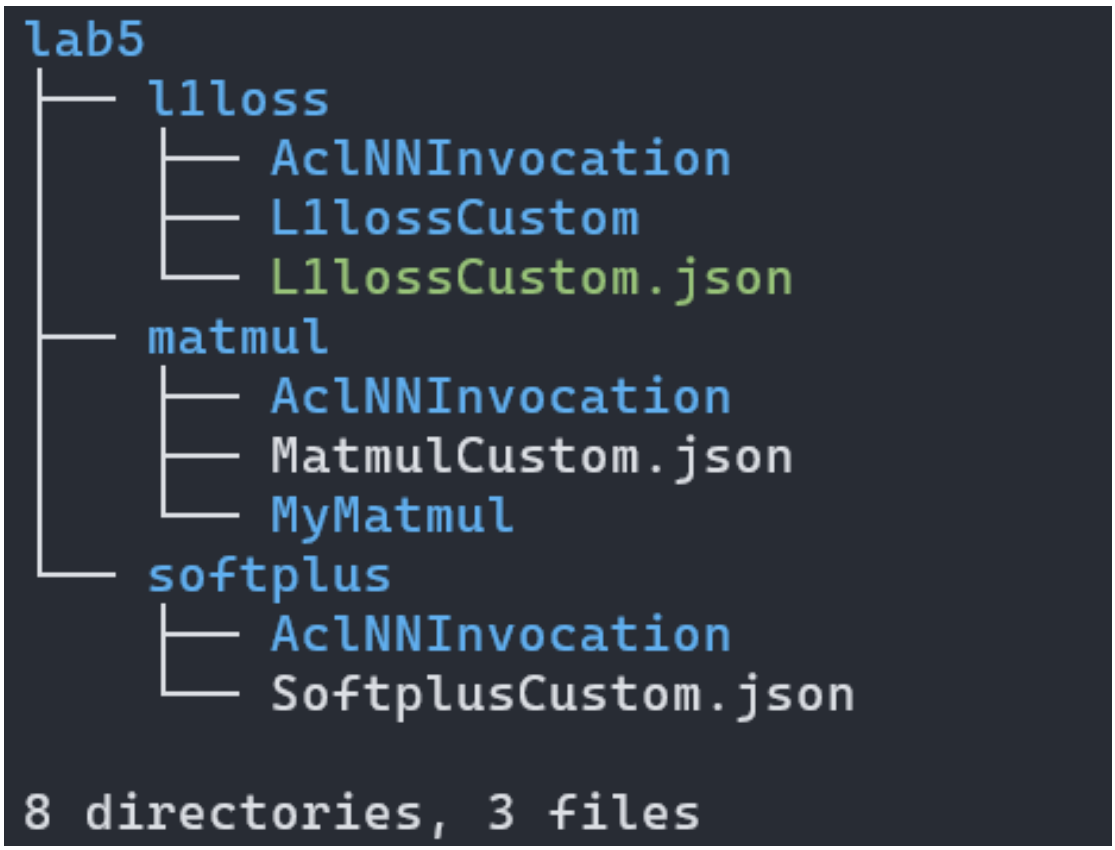

图 5-1 lab5-structure

5.4.2 gitee 上的 demo 仓库

1. add 算子
2. matmul 算子

5.5 L1Loss 算子

5.5.1 算子介绍

L1 loss 的数学定义为

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, l_n = | x_n - y_n |$$

默认的 L1 loss 会对所有元素的 loss 做 reduce（默认为取平均值），我们对此进行了简化，不做 reduce 操作，只计算对应元素的差并取绝对值。

详细定义可以参考 pytorch 的文档

5.5.2 创建算子工程

使用 msopgen 工具创建算子工程

命令如下

```
msopgen gen -i xxx.json -c ai_core-Ascend310B -lan cpp -out CustomOp
# -i 指示 json 文件
# -c 指定 硬件， 这里是 Ascend310B1
# -lan 指示编程语言
# -out 指示 输出目录名称(执行命令前未存在)
```

应该会如图5-2的项目结构

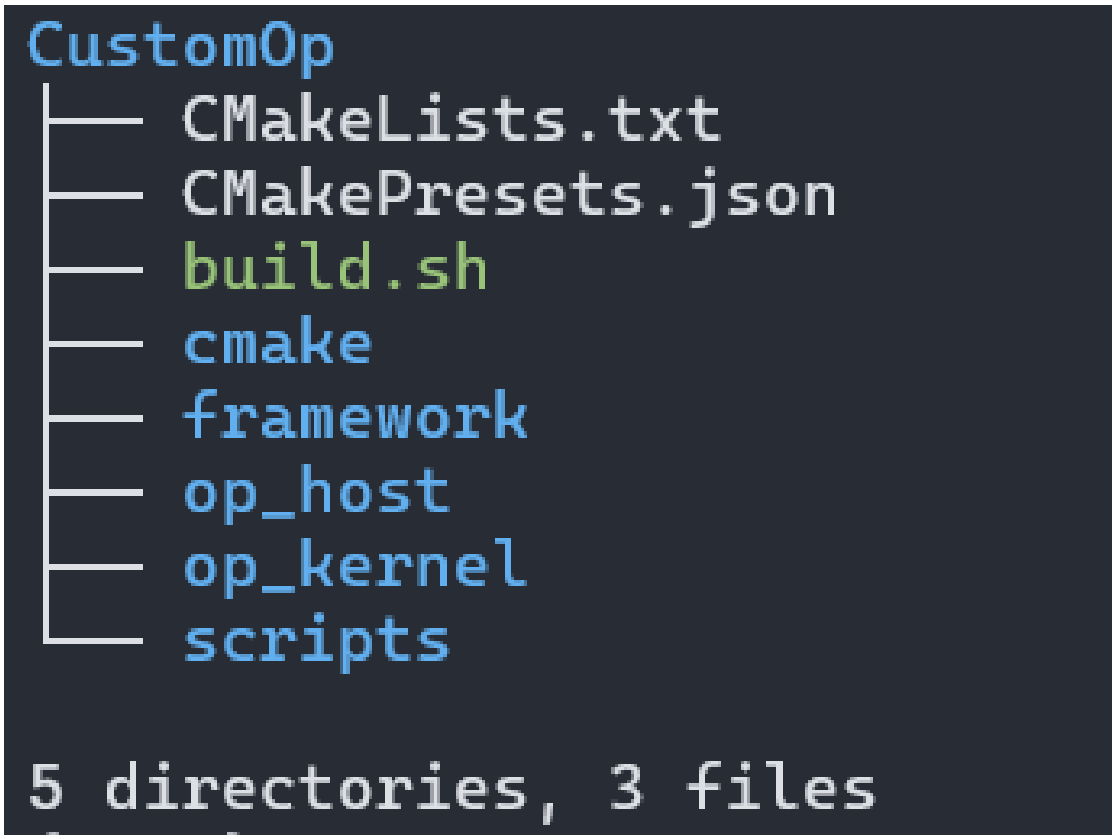

图 5-2 msopgen

尝试理解一下这个项目结构，这里列出需要关注的几个文件

1. `op_host` 实现算子 host 侧的功能，主要是形状推导，以及 tiling 的相关参数设置。
2. `op_kernel` 实现算子 kernel 侧的功能，包括计算，数据移动，内存管理等。

3. build.sh 算子实现完成后，需要使用 build.sh 来编译算子

使用我们提供的部分代码，替换掉 msopgen 自动生成的代码文件。

```
cp -rf L1lossCustom/* CustomOp
```

5.5.3 实现算子

cd 进入 CustomOp 目录（CustomOp 是你在 msopgen 指定的输出目录名，也可以是别的名称，但你要注意区别，后续说明我们假设你没有使用的别的名称）

先理解并实现 op_host 的代码，cd op_host

在这里有 l1loss_custom.cpp 和 l1loss_custom_tiling.h

简单概括：

l1loss_custom_tiling.h 定义了算子的 tiling 结构体

l1loss_custom.cpp 则完成初始化，填充了结构体的数据，以备在 kernel 侧使用

我们已经为大家把需要理解的部分通过注释标出

这里需要做的就是尝试理解 AscendC 的一些宏的行为，以及 tiling 初始化的过程。具体来说，你需要找到所有标记了 TODO 的部分，并将下面的注释的代码解开

完成 host 侧的代码后，返回 CustomOp 目录，并 cd 到 op_kernel 目录我们同样提供了大量的注释，你需要解开一些注释，并结合 AscendC 的手册和参考样例（AddCustom）算子，自己写一些代码，最终实现算子逻辑。

附 AscendC 参考文档 AscendC API 列表

5.5.4 安装算子

使用 CustomOp 中的 build.sh 脚本编译算子

```
bash build.sh
```

如果你的算子实现的没有问题，应该会看到最后一行输出类似

xxx/build_out/custom_opp_ubuntu_aarch64.run generated.

的内容，说明算子编译好了。

这个命令会在 CustomOp 目录创建一个 build_out 目录

在里面找到 custom_opp_ubuntu_aarch64.run 的算子安装包

执行它，安装算子。

```
bash custom_opp_ubuntu_aarch64.run
```

如果一切正常，你会看到如图5-3 的输出 这样你的算子已经在本地成功安装好了

```
Verifying archive integrity...  100%   SHA256 checksums are OK. All good.
Uncompressing version:1.0  100%
[ops_custom] [2025-12-17 23:03:05] [INFO] copy uninstall sh success
[ops_custom] [2025-12-17 23:03:05] [INFO] upgrade framework
tensorflow [ops_custom] [2025-12-17 23:03:05] [INFO] replace or merge old
[ops_custom] [2025-12-17 23:03:05] copy new ops framework files ......
[ops_custom] [2025-12-17 23:03:05] [INFO] upgrade op proto
inc lib [ops_custom] [2025-12-17 23:03:05] [INFO] replace or merge old ops
[ops_custom] [2025-12-17 23:03:05] copy new ops op_proto files ......
[ops_custom] [2025-12-17 23:03:05] [INFO] upgrade op impl
ai_core [ops_custom] [2025-12-17 23:03:05] [INFO] replace or merge old ops
[ops_custom] [2025-12-17 23:03:05] copy new ops op_impl files ......
[ops_custom] [2025-12-17 23:03:05] [INFO] upgrade op api
include lib [ops_custom] [2025-12-17 23:03:05] [INFO] replace or merge old
[ops_custom] [2025-12-17 23:03:05] copy new ops op_api files ......
[ops_custom] [2025-12-17 23:03:05] [INFO] upgrade version.info
[ops_custom] [2025-12-17 23:03:05] copy new version.info files ......
[ops_custom] [2025-12-17 23:03:05] [INFO] no need to upgrade custom.proto
[ops_custom] [2025-12-17 23:03:05] [INFO] using requirements: when custom
export LD_LIBRARY_PATH=/usr/local/Ascend/ascend-toolkit/latest/opp/vendors
SUCCESS
```

图 5-3 operator-install

5.5.5 测试算子

返回到 l1loss 目录

在这个目录下找到 AclNNInvovation，cd 进入该目录。

这个目录就是使用 AscendC 调用算子的最简单的流程，有兴趣的同学可以翻看其中的代码，了解相关 API 的使用方法。本实验不要求大家实现这部分内容。

找到 run.sh，执行它

如果一切顺利，你会看到如图5-4 的输出，说明你的算子通过了测试

```
[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 aclnnL1lossCustomGetWorkspaceSize success, workspace size 0
[INFO]  Execute aclnnL1lossCustom 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, tolerance: 0.0010
test pass
```

图 5-4 test

5.6 Softplus 算子

5.6.1 Softplus 的计算过程

softplus 的详细定义可以参考pytorch 文档

这里作简要介绍

$$Softplus(x) = \frac{1}{\beta} * log(1 + exp(\beta * x))$$

为了简化，我们设定 β 为 1

则公式可以简化为

$$Softplus(x) = log(1 + exp(x))$$

仍然可以视为一个 element-wise 的算子，整体流程以及使用的 API 都与 L1loss 的实验相似

5.6.2 创建算子工程

使用 msopgen 工具创建算子工程。

这一步同 l1loss 的实验，只是我们没有提供提示的代码，因此没有 cp 替代的步骤。

5.6.3 实现算子

我们对算子的实现也做了一些简化，具体来说，因为输入输出的形状都是固定的，（8，2048），因此 tiling 的参数也就固定了，可以将 tiling 的参数直接写进 kernel 侧的代码中，而不用通过 tiling 结构体传递了。这样你就完全不需要改动 tiling 侧的代码，直接使用 msopgen 生成的代码即可。

cd 进入 `CustomOp/op_kernel` 目录，找到 `softplus_custom.cpp` 文件

msopgen 自动生成的脚本会为你生成一个函数定义和一行代码（获取 tiling，你可以不管这行代码，甚至把它注释掉）。如图5-5

```
#include "kernel_operator.h"

extern "C" __global__ __aicore__ void softplus_custom(
    GM_ADDR x, GM_ADDR z, GM_ADDR workspace, GM_ADDR tiling) {
    GET_TILING_DATA(tiling_data, tiling);
    // TODO: user kernel impl
}
```

图 5-5 generated-kernel

你的任务就是实现这个接口。可以仿照 l1loss 中的方式，创建一个算子对象，实现功能，也可以使用函数调用实现数据拷贝和计算等功能。

5.6.4 安装算子

同 l1loss 实验，使用 CustomOp 中的 build.sh 脚本编译算子

找到 `custom_opp_ubuntu_aarch64.run` 的算子安装包

安装算子。

```
bash custom_opp_ubuntu_aarch64.run
```

如果一切正常，你会看到如图5-3 的输出

这样你的算子就安装好了

5.6.5 测试算子

同 l1loss 实验，返回到 `soft_plus` 目录, 并进入 `AclNNInvocation`

执行 `run.sh`

如果顺利的话，可以看到类似的 test pass 的输出

5.7 Matmul 算子

5.7.1 Softplus 的计算过程

Matmul 的计算公式为：C = A * B + bias，其示如图5-6。

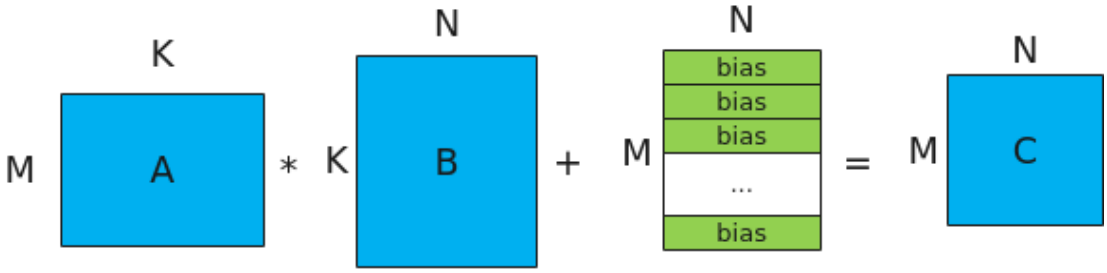

图 5-6 matmul

A、B 为源操作数，A 为左矩阵，形状为 [M, K]；

B 为右矩阵，形状为 [K, N]。

C 为目的操作数，存放矩阵乘结果的矩阵，形状为 [M, N]。

bias 为矩阵乘偏置，形状为 [1, N]。对 A*B 结果矩阵的每一行都采用该 bias 进行偏置。

5.7.2 创建算子工程

使用 msopgen 工具创建算子工程。

这一步同 l1loss 的实验，我们提供了提示的代码, 需要进行替换。

```
cp -rf MyMatmul/* CustomOp
```

5.7.3 实现算子

矩阵乘法的相关编程范式和向量运算很不一样，AscendC 提供了更多缓存的位置来实现矩阵的切块和计算，并提供了一套独立的 tiling 和计算相关的 API，

并且，提供了封装层次不同的 API（基础 API 和高阶 API），因为香橙派的硬件限制，无法使用基础 API，因此我们选择使用高阶 API 实现矩阵乘法算子

你需要参考 matmul 的 AscendC 文档 完成所有的 TODO 标记处的代码。

注意 host 侧和 kernel 侧都有需要完成的内容。

5.7.4 安装算子

同 l1loss 实验，使用 CustomOp 中的 build.sh 脚本编译算子

找到 `custom_opp_ubuntu_aarch64.run` 的算子安装包

安装算子。

```
bash custom_opp_ubuntu_aarch64.run
```

如果一切正常，你会看到如图5-3 的输出

这样你的算子就安装好了

5.7.5 测试算子

同 l1loss 实验，返回到 `soft_plus` 目录

执行 `run.sh`

如果顺利的话，可以看到类似的 test pass 的输出

5.8 评分指标

评分指标分为两个部分，其中实验代码和结果占 85%，实验思考部分和实验报告的撰写占 15%。

实验结果的评分指标如下：

1-49：代码无法跑通，将进行代码审计人工评分。

50：实现至少一个算子，并且通过我们提供的测试。

75: 实现两个算子，并且均通过测试。

85: 实现三个算子，并且均通过测试。

5.9 实验思考

1、参考昇腾的文档材料，你能否将自己开发的算子集成到 CANN 网络或者更上层的 MindSpore 网络中。

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。

3、你开发的自定义算子还有哪些优化空间？