2023 全国大学生计算机系统能力大赛

编译系统挑战赛昇腾算子开发赛题技术方案

详情访问大赛技术平台：https://compiler.educg.net

一、评价方式的基本说明

第1条　大赛编译系统挑战赛昇腾算子开发赛题（以下简称“赛题”）要求参赛队基于毕昇 C++异构编程语言和昇腾架构，综合使用架构特点及异构编程等技术，实现算子 Softmax 和 GEMM，实现在目标平台上算子的高效运行。

第2条　本赛题鼓励各参赛队充分了解毕昇 C++语言与昇腾架构特点，充分使能向量化、矩阵乘单元等加速计算单元，并从数据复用、双缓冲优化等角度尽最大可能发挥昇腾架构的硬件能力，提升算子实现的性能。

第3条　除本技术方案特别要求、规定和禁止事项外，各参赛队可在参考实现源码中规定的实现算子功能的函数内部，自行决定采用的实现技术。

二、初赛评分标准

第4条　比赛内容。在给定的参考实现源码基础上，通过扩展算子实现函数的具体实现代码，开发 Softmax 和 GEMM 算子，尽可能利用异构编程能力和昇腾架构硬件能力，提升算子在目标平台上的性能。

1. 赛题提供参考实现代码基于毕昇 C++语言开发，标准主程序中已经完成读取数据、写回数据、正确性检查、数据搬运等部分，不作为考察部分。
2. 参赛队需要在算子程序接口下实现算子函数，相关代码应能够在组委会提供的运行欧拉操作系统（EulerOS）的服务器上，由毕昇 C++编译系统编译，生成面向 AArch64 + Ascend 910 的二进制代码。
3. 赛题提供 Python 脚本生成基准测试用例，通过变化尺寸或者维度等参数，为每个算子各生成多组基准测试程序，考察算子实现的覆盖率；每组基准测试程序包含多个输入和期望输出作为测例，用于对参赛队实现的算子进行功能和性能测试。

第5条　功能测试。基于参赛队提交的算子实现源代码，经过赛题提供的毕昇 C++编译系统（具体版本另行公告）进行编译，并在规定的模拟器（规定配置参数，测试功能为主）或物理硬件平台（测试功能与性能）上依次运行每

个基准测试程序的测例输入，对比输出结果，计算功能正确性得分。所有测试点均未通过计 0 分；所有测试点都通过计 100 分；部分测试点通过的，按所通过测试点的比例计算功能测试得分。参赛队的最终功能测试成绩为每个基准测试程序功能测试分数的平均值。

1. 功能正确标准：参赛队提供的算子实现运行基准测试程序得到的输出结果，与基准测试程序提供的期望输出结果逐元素比较，满足双千标准（即不超过千分之一的数据出现超过千分之一的相对误差），则认为结果正确。
2. 基准测试程序的单个测试点运行结果不正确，即未达到双千标准，或者结果存在随机错误，判定错误结果（WA），该测试点不得分。
3. 基准测试程序的单个测试点的 Kernel 运行时间超过 10 秒，判定运行超时（TLE），该测试点不得分。

第6条　性能测试。在通过功能测试的前提下，记录每一个基准测试程序的测试点在昇腾硬件上的执行时间作为评价依据。

1. 每个基准测试程序的单个测试点在目标平台上运行 3 次，分别记录执行时间，取 3 次执行时间的平均值作为该算子实现在该基准测试程序测试点上的最终执行时间。
2. 每个基准测试程序的单个测试点按照执行时间最短者的性能测试得分被定义为 100 分，其余各参赛队依据 Kernel 在该基准测试程序上的执行时间与最短执行时间的比值除 100 计算参赛队在该基准测试用例上的性能得分（性能得分=100 /（执行时间/最短执行时间））。性能测试分值越大越好。
3. 每个基准测试程序的性能分数，为该基准测试程序所有测试点性能测试分值的平均值。
4. 参赛队在某一算子实现上的最终性能测试成绩为每个基准测试程序性能成绩的平均值。

第7条　初赛总成绩 100 分，各分项成绩权重如下：

1. Softmax 算子的性能成绩：40%。
2. GEMM 算子的性能成绩：60%

三、决赛评分标准

第8条　决赛阶段赛题要求及目标硬件平台特征与初赛保持一致，允许参赛团队

继续优化算子代码，并提交最终版本。

第9条　决赛阶段，组委会将生成新的基准测试用例，对参赛队算子实现进行功能和性能测试。

第10条　决赛阶段参赛队团队协作及现场答辩主要考察参赛队的团队协作及提交作品的代码完成度、技术创新性及代码规范性等因素，具体评分标准在决赛阶段另行公布。

第11条　决赛总成绩 100 分，各分项成绩权重如下：

1. Softmax 算子的性能成绩：24%。
2. GEMM 算子的性能成绩：36%
3. 参赛队团队协作及现场答辩：40%

四、参赛作品提交

第12条　各参赛队在初赛和决赛阶段需要在大赛的竞赛平台提交如下内容：

1. 完整的项目代码。其中，Softmax 算子实现提交 softmax.cpp，其中 softmaxFunc()函数由参赛队扩展实现；GEMM 算子实现提交 gemm.cpp，其中，gemmFunc()函数由参赛队扩展实现。
2. 参赛队在竞赛平台中至少有一次完整通过功能和性能测试的记录和有效成绩。
3. 相关设计文档（PDF 格式），内容包括但不限于对功能、性能及创新性的说明、相关测试结果及与类似项目的对比分析、遇到的瓶颈问题和解决方法等。

第13条　参赛队在初赛阶段需要提供时长不超过 5 分钟的答辩视频（MP4 格式）。

第14条　如果需要使用第三方 IP 或者借鉴他人的部分源码，必须在设计文档和源代码的头部予以明确说明，并确保内容符合相关法律法规和开源协议之规定。

第15条　参赛队必须严守学术诚信。一经发现存在代码抄袭或使用他人代码不做说明等学术不端行为，修改的代码重复率在 20%以上，取消参赛队的参赛资格。

第16条　参赛队在赛后，按编译系统设计赛要求公开代码和文档。

五、竞赛平台与测试程序

第17条　大赛提供的竞赛平台和测试程序包括：

1. 代码托管平台，支持各参赛队的群体协作与版本控制。
2. 竞赛评测系统，根据参赛队的申请从代码托管平台获取指定版本，生成编译系统，并加载基准测试程序，自动进行功能、性能及代码规模测试。
3. 生成基准测试程序的 Python 脚本，用于参赛队自行生成测试程序，对实现算子的功能与性能进行测试评估

六、 软硬件系统规范

第18条 赛题使用毕昇 C++作为开发语言。毕昇 C++是面向昇腾芯片的一种编程抽象，遵循标准 C++17 规范，并提供向量、矩阵运算接口或相关语法形式，支持基于昇腾架构的算子高性能开发。

第19条 赛道提供给参赛队的毕昇 C++语言、编译器及相关文档为定向版本，仅用于本竞赛用途，未经授权禁止扩散。

第20条 赛题指定的硬件平台为 AArch64 + Ascend 910，可通过 PMU（Performance Monitor Unit）和 profiling 数据进行数据分析。参赛队的最终功能和性能评分均基于该硬件平台。参赛队通过大赛提供的云服务访问该硬件平台。

第21条 赛题提供与 AArch64 + Ascend 910 对应的昇腾仿真器平台，供参赛队进行算子的功能和性能调试，仿真器的详细内容请参阅毕昇 C++技术文档。

第22条 参赛队可通过大赛组委会提供的指南对算子实现进行编译与调试，并在仿真器平台或指定硬件平台上运行测试，组委会提供环境变量配置、编译、运行脚本，与主程序同步发布。

七、大赛网站

第23条 编译系统赛技术平台网址：https://compiler.educg.net。

第24条 大赛网站提供多种软件开发工具及设计资料，包括但不限于下列内容：

1. 毕昇 C++技术文档及算子实现框架。
2. 竞赛平台（代码托管平台、竞赛测试系统）使用文档。
3. 生成基准测试程序的 Python 脚本。
4. 参赛队远程调试及本地调试指南。
5. 标准环境变量配置、编译、运行脚本。

附：Softmax 算子及 GEMM 算子说明

本附件中 Softmax 和 GEMM 算子描述中所述半精度均指 IEEE 754-2008 标准指定的 binary16。

每个算子均给出主程序和算子程序接口。主程序的逻辑为读取和准备输入数据——调用算子——读出结果数据和正确性校验。每个算子程序接口各参数的意义和尺寸范围限制请参考详细说明。基准测试用例是通过变化尺寸或者维度等参数生成的，所有用例的数据由统一的数据生成脚本生成。该脚本会统一发布，选手可以自行生成输入输出数据并提前自测。

算子 1（Softmax 算子）：

Softmax 函数是一种常用的激活函数，它的数学描述如下：

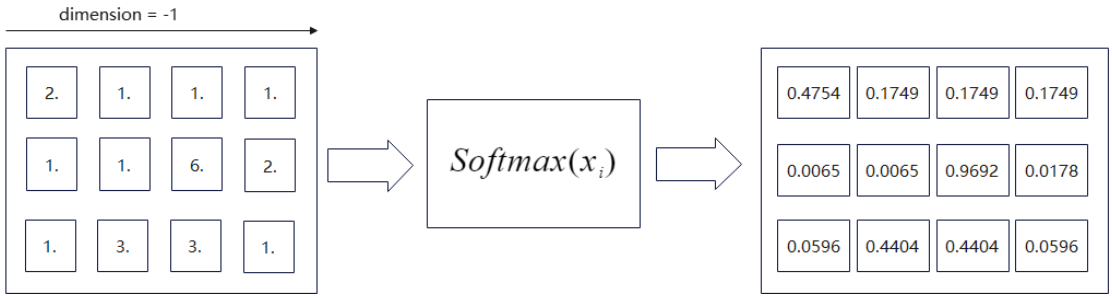
$$\mathrm{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Softmax 输出的是一个概率分布向量，且在指定向量方向(即指定维度)上的各个概率数值之和为 1。

给定一个 $a_1 \times a_2 \times \ldots \times a_n$ 的 n（$n \geq 2$）维向量，在 Softmax 函数中这个向量的维度值范围就是[-n, n - 1], 如果 $i - j = n$ 则表示维度值 i 和维度值 j 是表示相同的维度，如 -1 和 n - 1 表示按这个向量的第 n 维方向进行 Softmax 计算，0 和 -n 表示按这个向量的第 1 维进行 Softmax 计算。

例如，一个 3 ×4 矩阵，数据类型为半精度浮点数，求 Softmax 函数在指定维度方向上的各个概率分布。

该矩阵的每个元素记为 $a_{[i][j]}$,当 dimension = -1 时，固定 i 值，j 值从 0 到 3 遍历，计算每一行中每个元素 $a_{[i][j]}$在该行中的概率分布。具体输入输出如下图所示：



当 dimension = 0 时，则计算每一列中每个元素在该列中的概率分布。

现在给定一个 N 维的向量($1 \leq N \leq 10$)和指定维度，每一维上的尺寸范围为[1,100000]，数据类型限定为半精度浮点数类型 half，N 维向量的总占用字节数

不超过 2GB。请使用毕昇 C++实现 Softmax 算子。

接口说明

Softmax 算子的开发与优化需要基于如下模板进行实现。选手完成 softmaxFunc 内部实现即可，其他代码不需要修改。

```
void softmaxWrapper(sycl::queue &myQueue, size_t numCores,
                    sycl::global_ptr<sycl::half> inputPtr,
                    sycl::global_ptr<sycl::half> outputPtr,
                    sycl::global_ptr<size_t> shapePtr,
                    size_t shapeSize,
                    size_t dataSize,
                    int dim) {
    auto softmaxFunc = [inputPtr, outputPtr, shapePtr, shapeSize, dataSize, dim](sycl::group<1> myGroup) {
      // TODO: 请在这里完成 Softmax 算子的代码
    };
    myQueue.launch<class softmax_ascend>(numCores, softmaxFunc);
    myQueue.wait();
}

int main(int argc, char *argv[]) {
    // 解析参数，获取输入文件路径、输入数据形状、维度值等数据
    std::vector<sycl::half> inputVec;
    std::vector<sycl::half> outputVec(inputVec.size());
    std::vector<size_t> shapeVec;
    size_t shapeSize = shapeVec.size();
    size_t dataSize = 1;
    int dimension = -1;
    // …
    // 创建一个 queue 对象
    sycl::queue myQueue(sycl::ascend_selector{});
    // 在昇腾设备侧申请 Global Memory
    auto *gmInputPtr = sycl::malloc_device<sycl::half>(inputVec.size(), myQueue);
    auto *gmOutputPtr = sycl::malloc_device<sycl::half>(outputVec.size(), myQueue);
    auto *gmShapePtr = sycl::malloc_device<size_t>(shapeVec.size(), myQueue);
    // 从主机侧 Memory 拷贝数据到昇腾设备侧 Global Memory
    myQueue.memcpy(gmInputPtr, inputVec.data(), inputVec.size() * sizeof(sycl::half));
    myQueue.memcpy(gmShapePtr, shapeVec.data(), shapeVec.size() * sizeof(size_t));
    myQueue.wait();
    constexpr size_t numCores = 32;
    softmaxWrapper(myQueue, numCores, gmInputPtr, gmOutputPtr, gmShapePtr, shapeSize,
                   dataSize, dimension);
    // 从昇腾设备侧 Global Memory 拷贝数据到主机侧 Memory
    myQueue.memcpy(outputVec.data(), gmOutputPtr, dataSize * sizeof(sycl::half));
    myQueue.wait();
    // 释放昇腾设备侧 Global Memory
    // …
    // 将输出数据保存到本地文件
    // …
  return 0;
```

```
}
```

选手主要实现下述函数。

```
auto softmaxFunc = [inputPtr, outputPtr, shapePtr, shapeSize, dataSize,
                    dim](sycl::group<1> myGroup) {
  // 选手在这里完成算子实现
};
```

其中“[]”中为算子捕获到的参数，即算子入参。不允许选手进行调整，所有实现均在内部完成。所有输入参数均合法，不需要额外校验。各参数说明如下：

输入参数:

- 参数 inputPtr：表示用于输入的昇腾设备侧 Global Memory 指针。
- 参数 outputPtr：表示用于输出的昇腾设备侧 Global Memory 指针。
- 参数 shapePtr：表示用于输出的昇腾设备侧 Global Memory 指针，它是一个大小为 shapeSize 的数组，表示 N 维中各个维度的尺寸大小，shape 数组中各个数据的乘积等于 inputPtr 和 outputPtr 指向 GM 地址空间的大小。
- 参数 shapeSize: 表示数据的维度大小。
- 参数 dataSize: 表示数据的总大小。
- 参数 dim：指定维度值。

输出参数:

- 参数 outputPtr：表示用于输出的昇腾设备侧 Global Memory 指针。

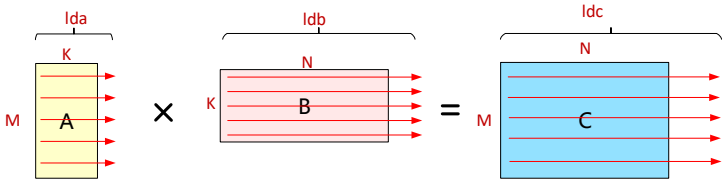
注意事项：由于按指定维度有指数求和，须处理在该计算过程中累加溢出的问题。

算子 2（GEMM）：

GEMM 是线性代数库中最基本的操作之一，使用频率高、计算密集。本赛题算子 2 要求使用毕昇 C++开发高性能 GEMM 算子，重点关注半精度浮点数在昇腾计算平台上的计算性能。

要求实现的 GEMM 为简化版本，输入矩阵（A、B）非转置，且 A、B、C 矩阵均为**行优先**，数据类型为半精度浮点数。GEMM 的数学公式如下：

$$\mathbf{C} = \text{alpha} * \mathbf{A} * \mathbf{B} + \text{beta} * \mathbf{C}$$



图中箭头表示 A、B、C 三个矩阵均为行优先排布。

接口说明：

GEMM 算子的开发与优化需要基于如下模板进行实现。选手完成**gemmFunc()**内部实现即可，其他代码不需要修改。

```
void gemmWrapper(sycl::queue &myQueue, size_t numCores, global_ptr<sycl::half> gmA,
                 global_ptr<sycl::half> gmB, global_ptr<sycl::half> gmC,
                 size_t M, size_t N, size_t K,
                 size_t lda, size_t ldb, size_t ldc,
                 sycl::half alpha, sycl::half beta) {
  auto gemmFunc = [gmA, gmB, gmC, M, N, K, lda, ldb, ldc, alpha, beta] (group<1> myGroup) {
    // 选手在这里完成算子实现
  };
  myQueue.launch<class gemm_ascend>(numCores, gemmFunc);
}

int main(int argc, char *argv[]) {
  // step1: 解析程序参数 m n k lda ldb ldc alpha beta
  // ...
  auto *matArowMajor = new sycl::half[lda * K];
  auto *matBrowMajor = new sycl::half[ldb * N];
  auto *matCrowMajor = new sycl::half[ldc * N];
  auto *MatCrstRowMajor = new sycl::half[ldc * N]; // 比对数据
  // step2：从数据文本中读取数据
  // ...
  // step3: 分配 device 上地址空间，并将 host 上输入数据拷贝到昇腾设备 Global Memory 上
  sycl::queue myQueue(sycl::ascend_selector{});
  auto *gmA = sycl::malloc_device<sycl::half>(lda * K, myQueue);
  auto *gmB = sycl::malloc_device<sycl::half>(ldb * N, myQueue);
  auto *gmC = sycl::malloc_device<sycl::half>(ldc * N, myQueue);
  myQueue.memcpy(gmA, matArowMajor, lda * K * sizeof(sycl::half));
  myQueue.memcpy(gmB, matBrowMajor, ldb * N * sizeof(sycl::half));
  myQueue.memcpy(gmC, matCrowMajor, ldc * N * sizeof(sycl::half));
  myQueue.wait();
  // step4: 调用算子计算
  constexpr size_t numCores = 32;
  gemmWrapper(myQueue, numCores, global_ptr<sycl::half>(&gmA[0]),
              global_ptr<sycl::half>(&gmB[0]),
              global_ptr<sycl::half>(&gmC[0]),
              M, N, K, lda, ldb, ldc, alpha, beta);
  myQueue.wait();
  // step5: 结果数据从昇腾 device 拷贝回 host，并做正确性校验
  myQueue.memcpy(matCrowMajor, gmC, ldc * N * sizeof(sycl::half));
  // ...
  // step6: 释放 host 或昇腾 device 上分配的空间
  // ...
  return 0;
}
```

选手主要实现下述函数。

```
auto gemmFunc = [gmA, gmB, gmC, M, N, K, lda, ldb, ldc, alpha, beta] (group<1> myGroup) {
   // 选手在这里完成算子实现
};
```

其中“[]”中为算子捕获到的参数，即算子入参。不允许选手进行调整，所有实现均在内部完成。所有输入参数均合法，不需要额外校验。各参数说明如下：

1) 输入参数：

- 参数 A: 表示矩阵 A（左矩阵）在昇腾设备侧 Global Memory 地址指针，矩阵 A 为半精度类型。
- 参数 B: 表示矩阵 B（右矩阵）在昇腾设备侧 Global Memory 地址指针，矩阵 B 为半精度类型。
- 参数 C：表示矩阵 C 在昇腾设备侧 Global Memory 上的地址指针，矩阵 C 为半精度类型。
- 参数 M：表示 A 或 C 矩阵的行数，范围: [16,131072]。
- 参数 N：表示 B 或 C 矩阵的列数，范围: [16,131072]。
- 参数 K：表示 A 的列数或 B 的行数（A 的列数 = B 的行数），范围: [1,8192]。
- 参数 alpha 和 beta：计算公式$\mathbf{C} = \text{alpha} * \mathbf{A} * \mathbf{B} + \text{beta} * \mathbf{C}$中的两个参数值，为半精度标量值。
- 参数 lda：表示矩阵 A 的 leading dimension，范围：[K, 8192]。
- 参数 ldb：表示矩阵 B 的 leading dimension，范围：[N, 131072]。
- 参数 ldc：表示矩阵 C 的 leading dimension，范围：[N, 131072]。

2) 输出参数：

- 参数 C：表示矩阵 C 在昇腾设备侧 Global Memory 上的地址指针，矩阵 C 为半精度类型。注意 C 矩阵既是输入也是输出。

注意事项：选手自测时需要注意昇腾设备上空间容量的限制，A、B、C 矩阵总大小不超过 16GB。出题组在设计基准测试用例时，会遵从相同的空间容量限制。