Broadcast 算子优化——任务书

文档版本 V0.1

发布日期 2025-09-19

华为技术有限公司

1 修订记录

版本	修改说明	发布时间
V0.1		2025-09-19

2 赛题简介

在人工智能的时代浪潮中，各类大模型的发展突飞猛进，加速各行业应用创新。进入 2025 年，大语言模型（LLMs）快速迭代与进化，特别是在推理能力和运行效率方面有了显著提升。推理能力上，以 OpenAI o1 系列和 DeepSeek R1 为代表的推理模型，带来了 scaling law 范式的转移（inference-time scaling），非常擅长解决如解谜、高级数学和编码等复杂任务，将大模型推向更广泛的应用场景；运行效率上，以量化、MOE、知识蒸馏为代表的模型优化技术，使得大模型可以部署到更轻量化的设备上，给端侧 AI 带来更大的想象空间。在参数量、算力需求指数倍增长情况下，如何充分释放 AI 芯片算力、提升大模型性价比也成为 AI 领域的关键技术，其中高性能算子的实现与优化是达成此目标的基础。

因此，本期比赛由选手通过对 Ascend C BroadCast 算子进行优化。期待您的精彩解决方案!

1．本次比赛基于昇腾 AI 云服务，要求参赛团队使用 AI 芯片 Ascend-snt9b、AI 异构计算框架 CANN、AI 应用开发平台 ModelArts 等华为全栈 AI 技术，增强模型推理能力并提升性能。

2．参赛团队需要掌握华为昇腾 AI 处理器架构、CANN 软件栈与 ModelArts 开发环境，实现昇腾算子开发，助力参赛团队学习相关技术，了解实践操作。2．参赛团队需要掌握华为昇腾 AI 处理器架构、CANN 软件栈与 ModelArts 开发环境，实现昇腾算子开发，助力参赛团队学习相关技术，了解实践操作。

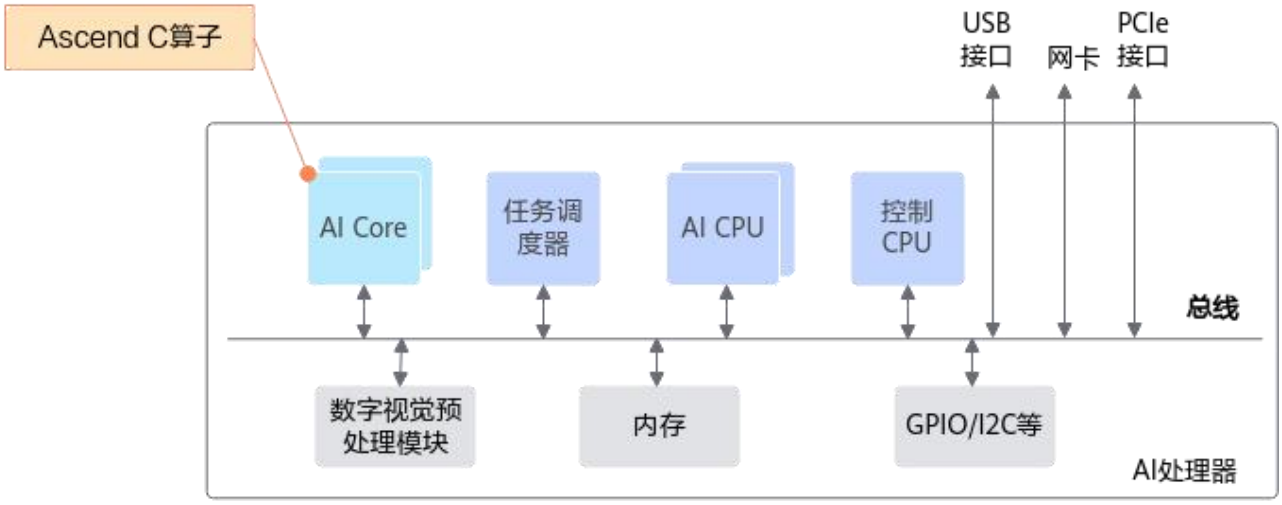
3 题目背景说明

3.1 题目信息

3.1.1 问题抽象

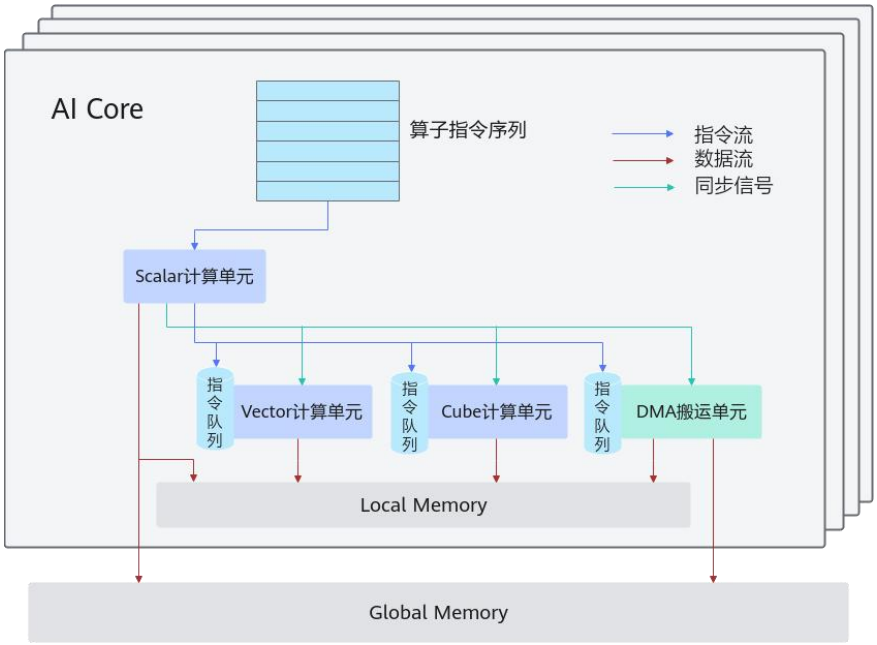
Ascend C 是 CANN 针对算子开发场景推出的编程语言，原生支持 C 和 C++标准规范，兼具开发效率和运行性能。基于 Ascend C 编写的算子程序，通过编译器编译和运行时调度，运行在昇腾 AI 处理器上。使用 Ascend C，开发者可以基于昇腾 AI 硬件，高效的实现自定义的创新算法。

如下图所示：AI Core 负责执行标量、向量和张量相关的计算密集型算子，包括三种基础**计算单元**：Cube（矩阵）计算单元、Vector（向量）计算单元和 Scalar（标量）计算单元，同时还包含**存储单元**（包括硬件存储和用于数据搬运的搬运单元）和**控制单元**。硬件架构根据 Cube 计算单元和 Vector 计算单元是否同核部署分为**耦合架构**和**分离架构**两种。Ascend C 基于硬件抽象架构进行编程， 从而屏蔽不同硬件之间的差异。



AI Core（如下图）中包含**计算单元、存储单元、搬运单元**等核心组件。

- 计算单元包括了三种基础计算资源：Cube 计算单元、Vector 计算单元和 Scalar 计算单元。
- 存储单元包括内部存储和外部存储：
  - AI Core 的内部存储，统称为 Local Memory，对应的数据类型为 LocalTensor。由于不同芯片间硬件资源不固定，可以为 UB、L1、L0A、L0B 等。
  - AI Core 能够访问的外部存储称之为 Global Memory，对应的数据类型为 GlobalTensor。
  - DMA（Direct Memory Access）搬运单元：负责数据搬运，包括 Global Memory 和 Local Memory 之间的数据搬运，以及不同层级 Local Memory 之间的数据搬运。



AI Core 内部核心组件及组件功能详细说明如下表：

组件分类	组件名称	组件功能
计算单元	Scalar	执行地址计算、循环控制等标量计算工作，并把向量计算、矩阵计算、数据搬运、同步指令发射给对应单元执行。
	Vector	负责执行向量运算。
	Cube	负责执行矩阵运算。
存储单元	Local Memory	AI Core的内部存储。
搬运单元	DMA（Direct Memory Access）	负责数据搬运，包括Global Memory和Local Memory之间的数据搬运以及不同层级Local Memory之间的数据搬运，包含搬运单元MTE2（Memory Transfer Engine，数据搬入单元），MTE3（数据搬出单元）等。

3.1.2 Broadcast 算子简介和描述

在深度学习模型中，Broadcast 算子广泛应用于张量形状扩展（如加法、乘法等元素级运算前）。然而，在高维 Shape 场景下，当前 CANN 中的 Broadcast 算子存在性能瓶颈，计算耗时长，影响整体模型训练和推理效率。

假设有两个张量：

- A 的 shape 是 [3, 1]
- B 的 shape 是 [1, 4]

它们能直接进行加法运算，因为维度不匹配。但通过 Broadcast 机制，可以将它们扩展成相同形状 [3, 4] 后再相加：

```
A = [[1],[2],[3]]

B = [[10, 20, 30, 40]]

Broadcast 后：
A' = [
     [1, 1, 1, 1],
     [2, 2, 2, 2],
     [3, 3, 3, 3]
]

B' = [
      [10, 20, 30, 40],
      [10, 20, 30, 40],
      [10, 20, 30, 40]
]

A' + B' = [
          [11, 21, 31, 41],
          [12, 22, 32, 42],
          [13, 23, 33, 43]
]
```

Broadcast 遵循以下规则：

1. **对齐维度**：从尾部（最内层）开始对齐两个张量的 shape。
2. **维度扩展**：若某维大小为 1，可以扩展为任意大小。
3. **自动填充**：若某一侧维度缺失，自动填充为 1。

例如：

- [3, 1] + [1, 4] → [3, 4] ☑
- [2, 3, 4] + [3, 4] → [2, 3, 4] ☑
- [2, 3] + [3, 2] → ✗ 不匹配

在深度学习中的典型应用场景

1. **激活函数偏置加法**
   - 输入张量：[batch, channel, height, width]
   - 偏置张量：[channel]
   - 通过 Broadcast 将偏置扩展至每个空间位置。
2. **BatchNorm 中的缩放与偏移**
   - γ（scale）和 β（bias）为[channel]，需广播至输入张量。
3. **Attention 中的 Masking**
   - Mask 张量通常为[batch, 1, 1, seq_len]，需广播至[batch, heads, seq_len, seq_len]。

3.2 算子工程介绍

算子工程目录 Broadcast 算子的实现文件，

```
├── BroadcastCustom          // Broadcast自定义算子工程
│   ├── op_host              // host侧实现文件
│   └── op_kernel            // kernel侧实现文件
```

CANN 软件包中提供了工程创建工具 msOpGen，BroadcastCustom 算子工程可通过 BroadcastCustom.json 自动创建，自定义算子工程具体请参考 Ascend C 算子开发>工程化算子开发>创建算子工程章节。

创建完自定义算子工程后，开发者重点需要完成算子 host 和 kernel 文件的功能开发。为简化样例运行流程，本样例已在 BroadcastCustom 目录中准备好了必要的算子实现，install.sh 脚本会创建一个 CustomOp 目录，并将算子实现文件复制到对应目录下，再编译算子。

备注：CustomOp 目录为生成目录，每次执行 install.sh 脚本都会删除该目录并重新生成，切勿在该目录下编码算子，会存在丢失风险。

4 任务说明

4.1 任务要求

Broadcast 算子是深度学习中的基础算子，用于将低维张量扩展至高维以匹配运算维度（如加法、乘法前的形状对齐）。当前在高维 Shape（如[batch, 1024, 1024, 256]）场景下，算子存在显著性能瓶颈，表现为计算耗时长、内存占用高，影响模型训练/推理效率。

4.2 关键技术方向（**参考建议**）

1. 动态 Shape 处理优化
   - **问题**：运行时动态 Shape 分支判断导致额外开销。
   - **要求**：
     - 基于 range 参数预分配资源，减少分支跳转。
     - 实现自适应 Tiling 策略，支持超大规模张量（如 shape=-1 场景）。
2. 内存访问优化
   - **问题**：高维下内存非连续访问、对齐效率低。
   - **要求**：
     - 优化数据分块（Blocking）策略，提升局部性。
     - 复用统一缓冲区（UB），减少全局内存访问次数。
     - 确保内存访问对齐至硬件位宽（如 256B）。
3. 计算流程重构
   - **问题**：串行复制操作未充分利用多核并行。
   - **要求**：
     - 设计多核并行广播策略（如分块并行+核内流水）。
     - 应用向量化指令（SIMD）加速数据复制。
     - 探索异步计算掩盖内存延迟。

4.3 交付要求

1. **代码交付**
   - 算子实现（≤2000 行），禁用黑盒 API；
   - 提供 APROF 性能分析报告及瓶颈定位说明。
2. **验证标准**
   - 测试用例：覆盖 Shape=[2048,2048,2048]、[4096,1024,4096] 等典型高维场景；
   -
3. **文档输出**
   - 优化技术说明（含 Roofline 模型瓶颈分析）；

5 评分规则

维度	权重	细则说明
精度正确性	30%	所有测试 Shape 下结果必须与参考实现（如 PyTorch/CANN）对齐，误差控制在 IEEE FP16/BF16 标准内 1。
性能表现	50%	在 5 个 Shape 中，分别进行性能打分。采用相对评分法，对所有精度正确的选手按执行时间从快到慢排名
泛化能力	10%	泛化场景中（未公开 Shape）性能是否稳定，是否能应对动态 Shape 输入。
代码规范性	5%	包括代码可读性、注释完整性、是否符合 Ascend 开发规范（TBE/CCE）等。
文档完整性	5%	提交说明文档是否完整，是否包含优化思路、性能瓶颈分析、Roofline 模型、调优过程等。

测试 Shape 说明（示例）

Shape 编号	输入 Shape (M, K, N)	类型	是否公开
S1	(2048, 2048, 2048)	高维典型	❌ 否
S2	(4096, 1024, 4096)	高维典型	❌ 否
S3	(1024, 512, 1024)	中维	❌ 否
S4	(512, 128, 512)	低维	❌ 否
S5~S8	(动态生成)	泛化场景	❌ 否

注：每个 Shape 需在 Ascend NPU（如 910B）上运行，使用 msprof 或 APROF 工具记录端到端耗时。

5.1.1.1 三、性能评分方式详解

1. **精度验证**（30 分）
   - 所有测试 Shape 需通过精度校验（逐元素误差 ≤ 1e-3）；
   - 若任一 Shape 精度失败，则该选手**性能部分得分为 0。**
2. **性能排名打分**（50 分）
   - 每个 Shape 按执行时间从低到高排序，设共有 M 名精度合格选手：
     - 第 1 名得 50 分；
     - 第 n 名得分 = 50 × (M - n + 1) / M；
   - 所有 Shape 得分加权平均，得出最终性能得分。
3. **泛化能力评分**（10 分）
   - 泛化 Shape 性能得分 = 泛化场景平均性能排名得分 × 10%；
   - 若未支持动态 Shape 输入，此项得分**减半。**

6 注意事项

1. 判题程序会从选手程序标准输出读取分配方案，计算调度总得分并记录程序运行时间（单位为 ms），总得分高的方案胜出。
2. 如果不同选手的输出方案的总得分相同，则先提交代码者胜出。
3. 判题采用多组数据，得分依据多组结果求和后进行排名。
4. **禁止在代码中执行 shell 命令、使用多线程**等影响判题机器运行与公平性的行为。此行为在赛后的最终测评阶段也将无法得分！
5. 比赛结束后将进行代码查验，如发现代码重复或违规等情况，将取消该团队参赛资格和现有成绩。
6. 总时间不超过 1 分钟，超时判题将失败。
7. 所有选手提交代码后，统一在指定设备上运行精度校验，精度通过者进入性能测试，记录各 Shape 执行时间，按上述规则汇总得分，结合加分/扣分项给出最终评分，对前 10%选手进行人工复核，确保评分客观性。
8. 对于开源代码，可以借鉴参考，但必须有相应的原创、创新。比赛结束后将进行代码查验和复测，如发现代码重复、搬运、抄袭、重复率过高的作品，经查实后我们将取消获奖资格

⚠ 注意

因为进程调用存在一定的时间开销，用时统计在判题程序侧和选手程序侧可能存在细微差异。建议选手控制算法用时的时候要留有一定的冗余
