# ops-transformer仓&通算融合算子介绍

CANN Tech Connect 2025

CANN开发者MeetUp

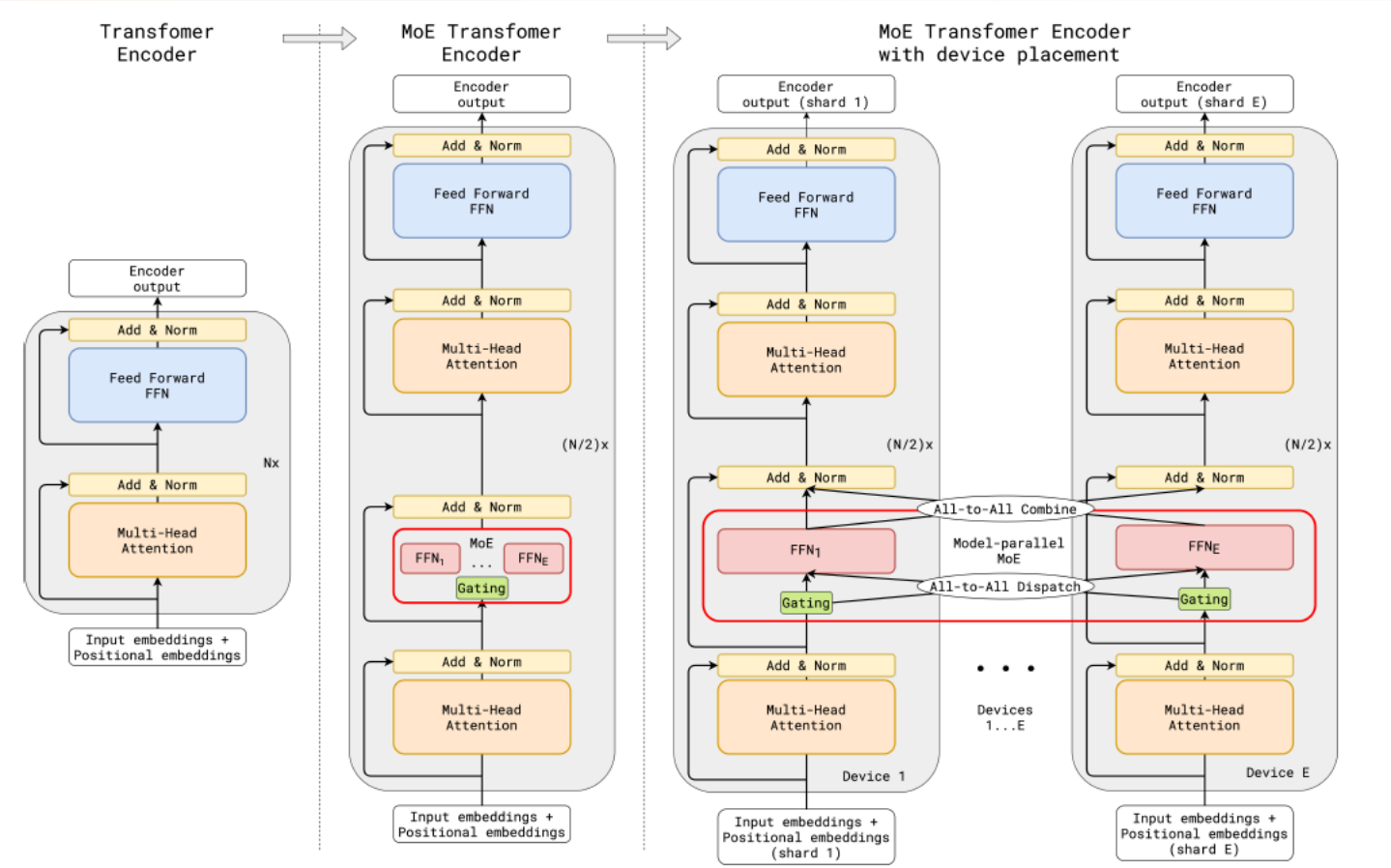
作者：陈建军

2025.12.6　中国·深圳

- 一、ops-transformer仓介绍

# ops-transformer仓基础介绍(一)

ops-transformer是CANN（Compute Architecture for Neural Networks）算子库中提供transformer类大模型计算的进阶算子库。




- 生态优先，兼容主流生态使用习惯
- 高内聚，低耦合原则
- 隔离原则等

https://gitcode.com/cann

# ops-transformer仓基础介绍（二）

本仓包含六大类算子，Attention（注意力类算子）、 MC2（通信和计算融合类算子） 、GMM（Grouped Matmul相关算子）、FFN 类算子、MOE 类算子和Posembedding (位置编码类算子)。


Transformer仓

- **Attention**：FlashAttention类一系列衍生融合算子，如PFA、IFA、FIA和FAG等;
- **MC2类**：通算融合类算子，包含一系列通信和计算融合类算子，如all_gather_matmul，moe_distribute_dispatch和moe_distribute_combine等算子;
- **GroupMatmul类**：实现分组矩阵乘计算及相关融合算子
- **FFN类**：提供FFN和MoeFFN的计算算子;
- **MOE类**：提供MOE计算中，包括moe_init_routing、_moe_gating_top_k、moe_finalize_routing等算子

算子列表

# ops-transformer 仓目录结构介绍

```
├── build.sh                    # 项目工程编译脚本
├── cmake                       # 项目工程编译目录
├── common                      # 项目公共头文件和公共源码
├── attention                   # attention类算子
├── docs                        # 项目文档介绍
├── example                     # 使用通用算子开发和调用示例
├── ...
├── moe                         # moe类算子
├── posembedding                # posembedding类算子
├── mc2                         # mc2类算子
│   ├── all_gather_matmul       # all_gather_matmul算子所有交付件，如Tiling、Kernel等
│   │   ├── CMakeLists.txt      # 算子编译配置文件
│   │   ├── docs                # 算子说明文档
│   │   ├── examples            # 算子使用示例
│   │   ├── op_graph            # 算子构图相关目录
│   │   ├── op_host             # 算子信息库、Tiling、InferShape相关实现目录
│   │   │   └── op_api          # 算子aclnn接口实现目录
│   │   ├── op_kernel           # 算子kernel目录
│   │   └── README.md           # 算子说明文档
│   ├── ...
│   └── CMakeLists.txt          # 算子编译配置文件
├── scripts                     # 脚本目录，包含自定义算子、kernel构建相关配置文件
├── tests                       # 测试工程目录
├── CMakeLists.txt
├── README.md
├── build.sh                    # 项目工程编译脚本
├── install_deps.sh             # 安装依赖包脚本
└── requirements.txt            # 本项目需要的第三方依赖包
```



- **Docs**：包含aclnn接口介绍等md文档；
- **Examples**：算子调用示例；
- **op_graph**：算子图模式场景交付件，例如InferDatatype等；
- **op_host**：算子基础host侧交付件，例如Tiling、aclnn接口等；
- **op_kernel**：算子device侧交付件，包含算子核心实现。

# 算子基本开发流程介绍

基于开源仓开发新算子，核心交付件包含Tilling与Kernel两部分内容


1. 前提条件：

   ① 环境部署：开发算子前，请确保基础环境已安装，例如依赖的驱动、固件、CANN软件包等。

   ② 算子设计：分析实际业务诉求，合理设计算子规格，包括算子输入、输出、属性的数据类型、shape等。

2. 工程创建：开发算子前，需按要求创建算子目录，方便后续算子的编译和部署。

3. Tiling实现：实现Host侧算子Tiling函数。

4. Kernel实现：实现Device侧算子核函数。

5. aclnn适配：自定义算子推荐aclnn接口调用，需完成二进制发布。如需入图，请参考附录。

6. 编译部署：通过工程编译脚本完成自定义算子的编译和安装。

7. 算子验证：通过常见算子调用方式，验证自定义算子功能。

详细开发指导

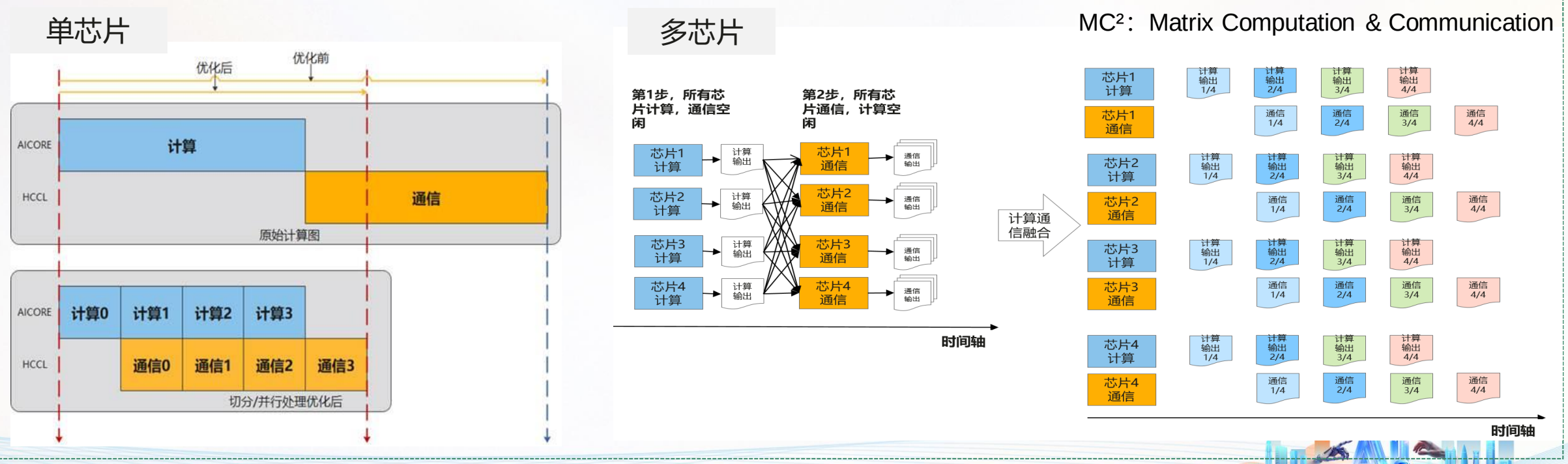
- **Host侧Tiling实现：**
  由于NPU中AI Core内部存储无法完全容纳算子输入输出的所有数据，需要将数据分批次进行处理。切分数据的算法称为Tiling算法或者Tiling策略。
- **Device侧Kernel实现：**
  Kernel实现即算子核函数实现，通过调用计算、数据搬运、内存管理、任务同步API，实现算子逻辑。其核心逻辑基本上都为计算密集型任务，需要在NPU上执行。

- 二、通算融合算子介绍

# 通算融合：MC²融合算子，实现通信与计算协同并行

**问题：**随着模型变大，数据增多，单卡的显存空间及算力已经无法满足训练/推理需求，**多卡并行**已成为趋势；多卡并行则会带来大量卡间通信，导致整网运行中**通信耗时占比高**（大部分20%以上，有些高达40%），模型执行效率下降。

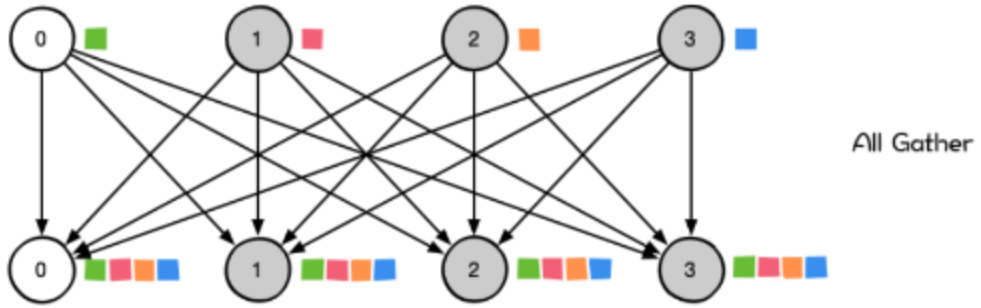
**方案：**实现芯片间**通信计算并行**，算子执行过程中，计算和通信相互掩盖执行时长。基于AscendC 提供通信计算融合算子MC² 。通信流水能力以高阶API的形式在Ascend C编程框架中提供，支持更多通信计算融合算子的开发；

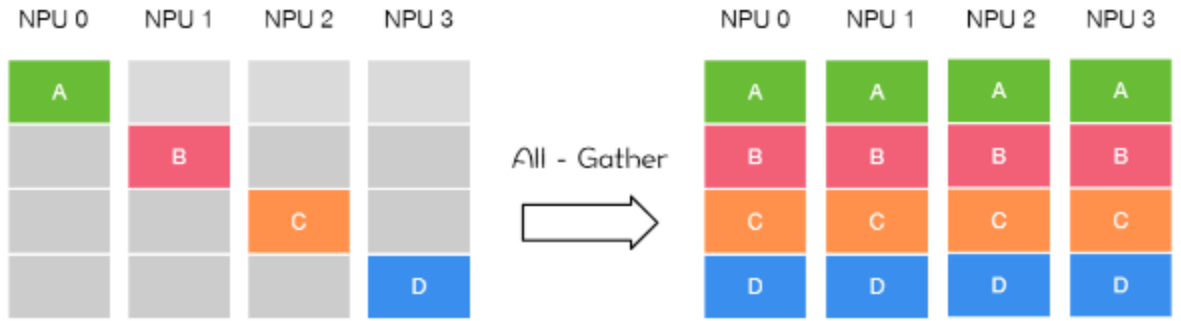


# 集合通信介绍—AllGather通信

All Gather会收集所有数据到所有节点上。 All Gather = Gather + Broadcast。

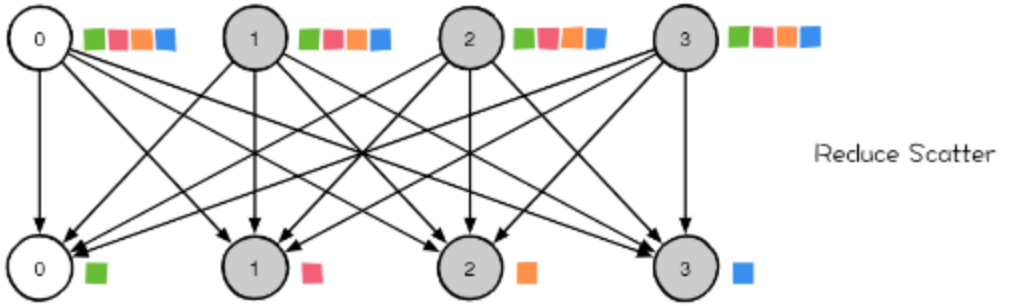
发送多个元素到多个节点很有用，即在多对多通信模式的场景。

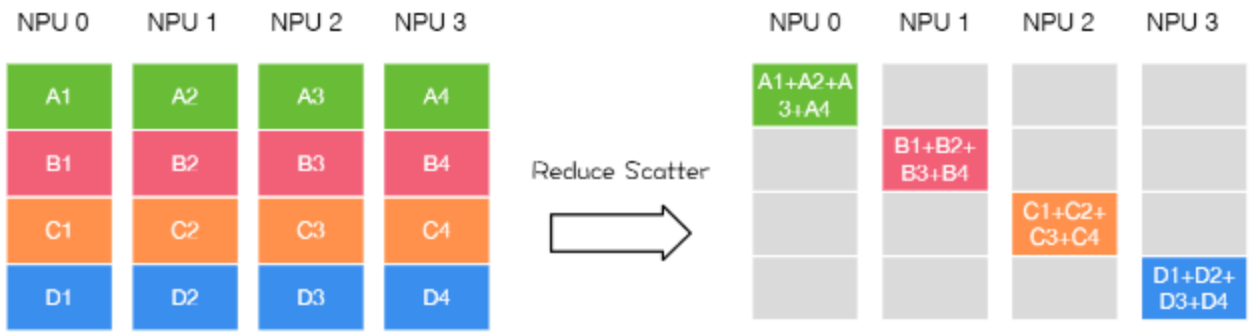


# 集合通信介绍—ReduceScatter通信

Reduce Scatter操作会将个节点的输入先进行求和，然后在第0维度按卡数切分，将数据分发到对应的卡上。

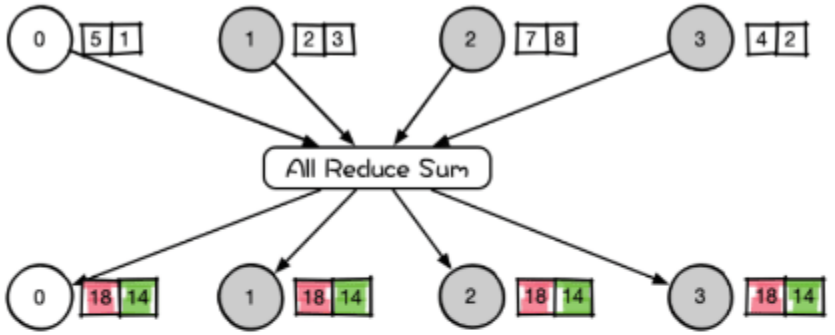


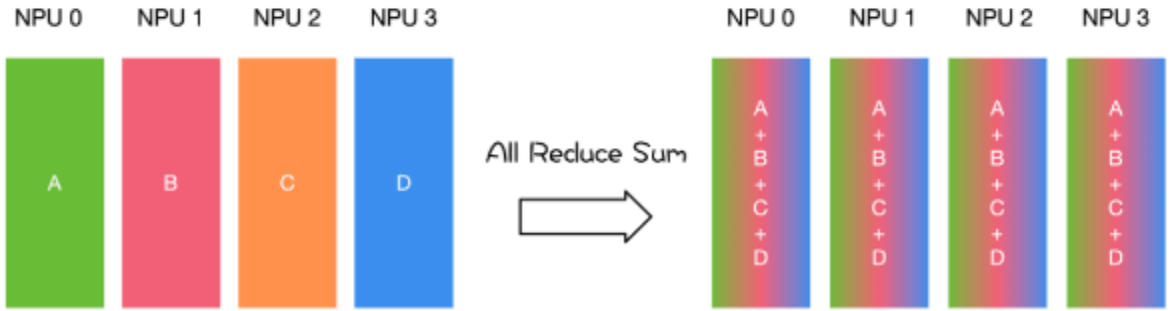



# 集合通信介绍—AllReduce通信

All Reduce则是在所有的节点上都应用同样的Reduce操作。

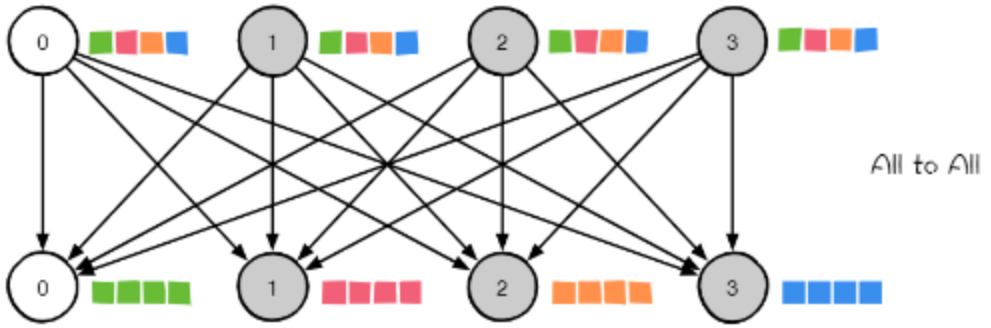
All Reduce操作可通过Reduce + Broadcast 或者 Reduce-Scatter + All-Gather 操作完成。

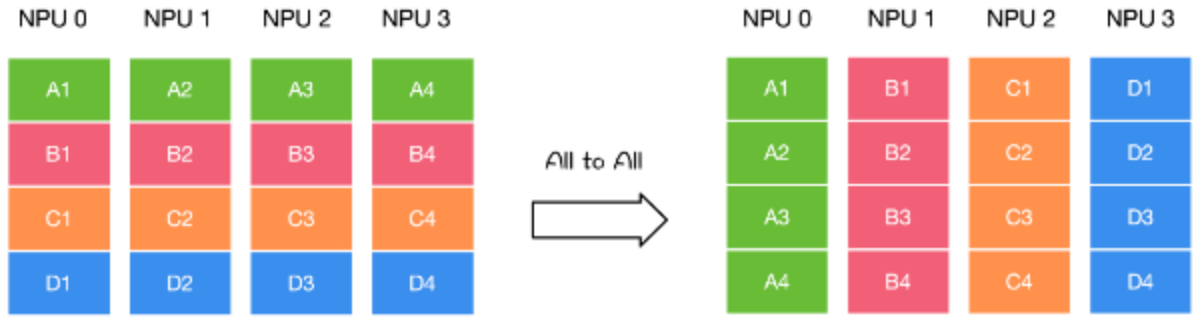


# 集合通信介绍—AlltoAll通信

将节点i的发送缓冲区中的第j块数据发送给节点j，节点j将接收到的来自节点i的数据块放在自身接收缓冲区的第i块位置。



# 计算介绍—Matmul计算

**MC2中的计算主要指Matmul计算，其他也有Add、Quant、激活等计算**

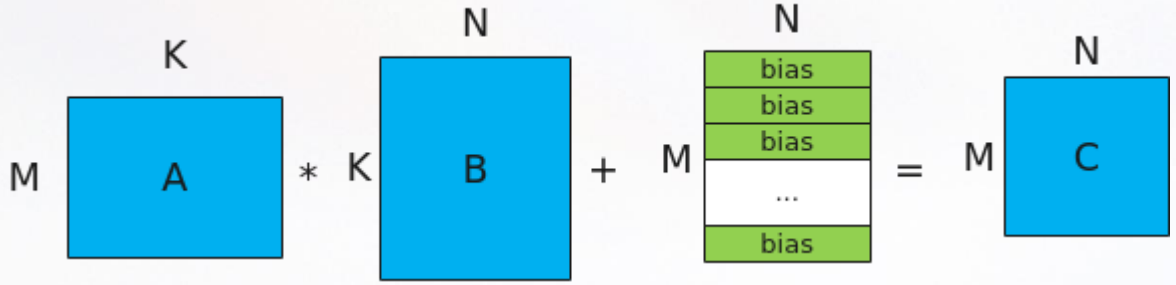
Matmul的计算公式为：C = A * B + Bias，其示意图如下。

•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。

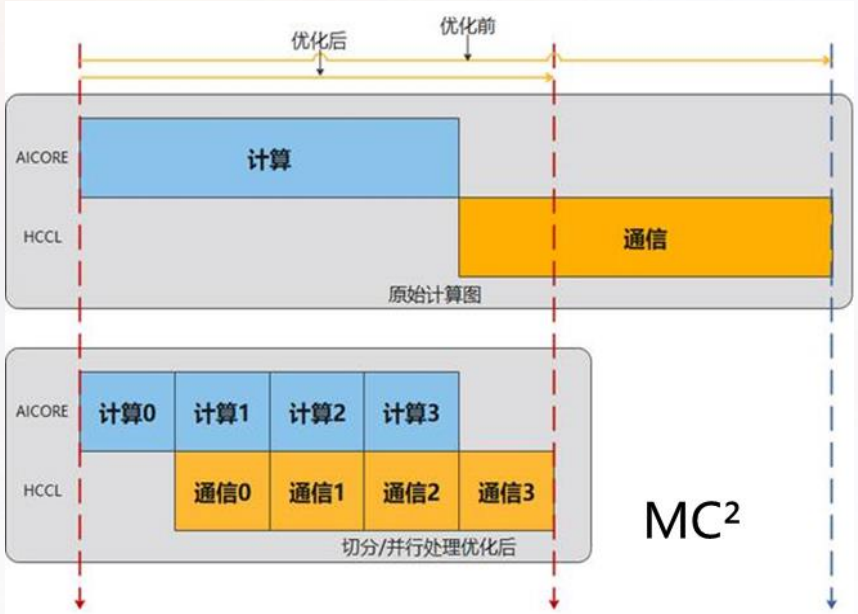
Matmul矩阵乘示意图：




**MC2涉及的matmul计算有：**

**MatMul　BatchMatMul　QuantBatchMatMul　GroupedMatMul**

# Tiling切分介绍

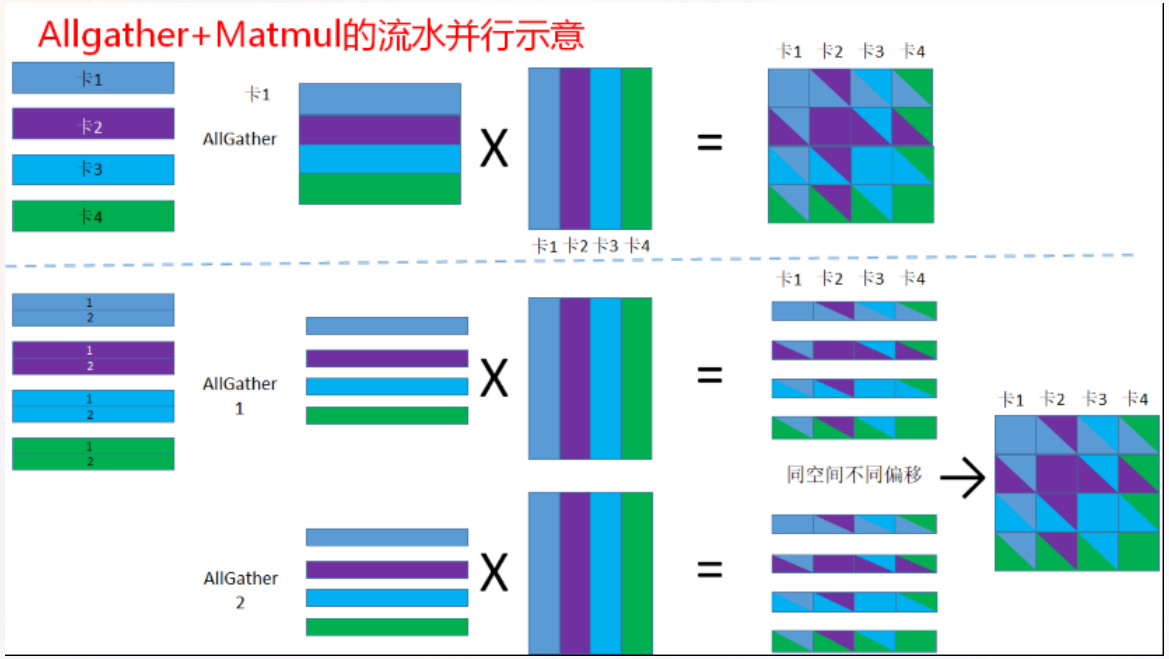



**tiling切分目标：找到最合适的切分，使得性能最优；**

- **主要按M轴进行切分**
- **尽量按MatMul计算的base块的整数倍切分**
- **从计算耗时图中可以看到切分后，小块的耗时总和>大块的耗时。会有通信膨胀，不易切分过细。**
- **拖尾部分“通信3”应尽量小。**

| M | K | N | QuantBMM时间us |
|---|---|---|---|
| 256 | 8192 | 1280 | 46.3 |
| 512 | 8192 | 1280 | 53.0 |
| 1792 | 8192 | 1280 | 112.6 |
| 3584 | 8192 | 1280 | 179.1 |
| 256 | 8192 | 7168 | 143.2 |
| 512 | 8192 | 7168 | 216.6 |
| 1792 | 8192 | 7168 | 529.2 |
| 3584 | 8192 | 7168 | 966.2 |

# AllGather+Matmul 通算并行介绍




由于AllGather和Matmul耗时存在差异，在做通算融合之后，
**耗时下降10%~30%**

## Allgather+Matmul的流水并行原理

- 中规中矩的流水并行方法是：
  - Allgather阶段将参与Matmul计算的数据按照行做Allgather，每次Allgather流水输出的数据不是连续的，而是一个“明暗条纹”，Matmul在根据“明暗条纹”做计算，输出“明暗条纹”，将每轮输出的明暗条纹拼接到一起，得到完整的Allgather+Matmul的结果
    - Matmul选择Allgather算法模式并在RPC Task中携带，HCCL_RPC根据算法模式选择正确的算法，按照算法模式的指示输出本轮Allgather的结果，模式指示至少包括：内存起始位置（对应Rank0）、Rank间隔。内存起始位置有两种表述方法：A、绝对起始位置+本轮轮次+每轮通信长度；B、本轮起始位置；Matmul计算端负责确定算法模式，HCCL_RPC负责按照模式通信；
    - 在典型的Transformer网络中，Allgather+Matmul对应Sequence并行，每个TP节点负责一段连续Sequence的计算，当采用Allgather+Matmul流水并行时，按照Sequence（外轴）做切分，每次Allgather的input是完整的
    - 每次AllGather，所有的Rank都参与，针对910B的Mesh结构，能够最大限度的利用链路带宽，缺点是每次通信，所有的Rank都参与，同步次数多，同步时间长，并且存在首数据开销：先通信再计算

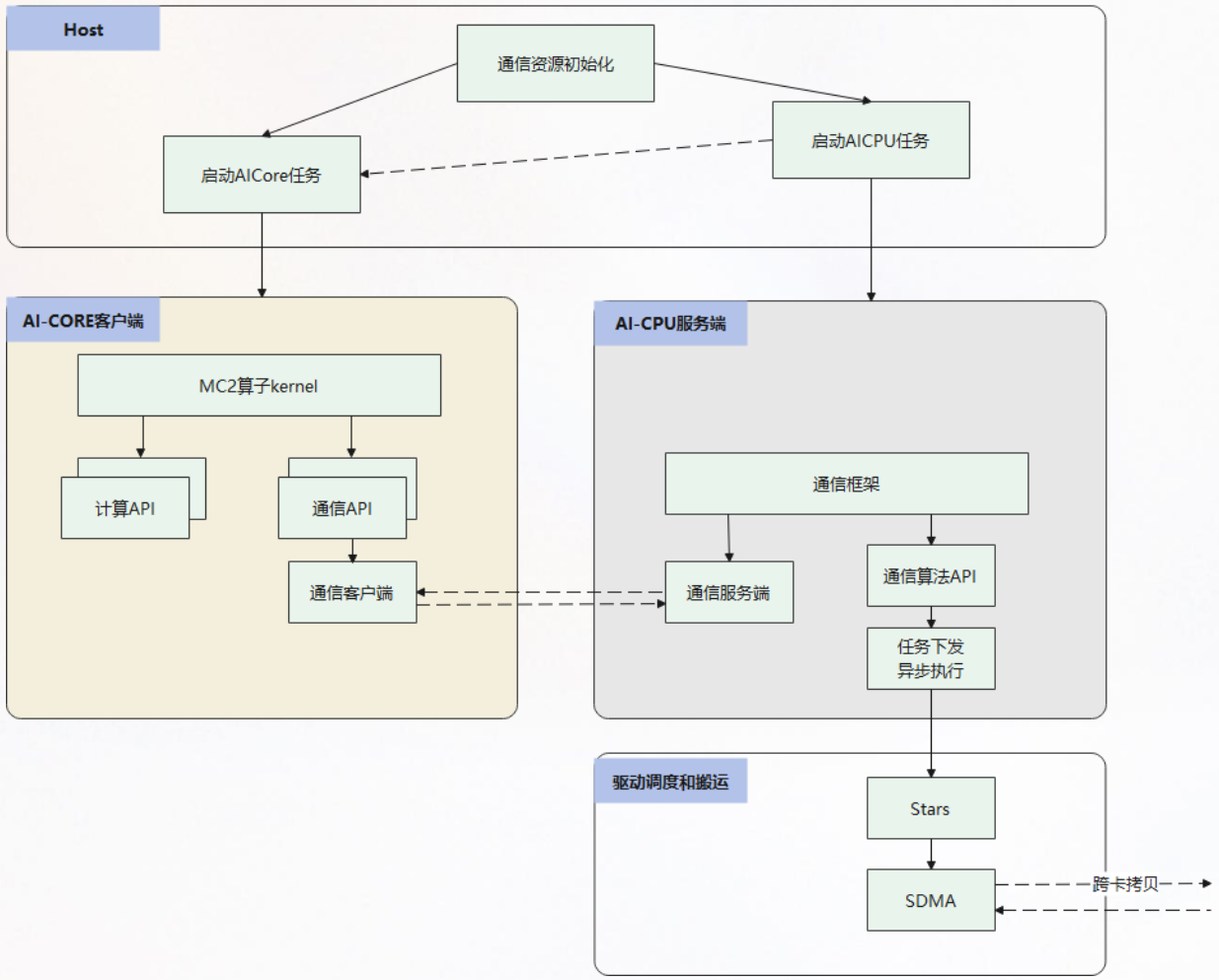
## Allgather+Matmul的流水并行原理

- 先通信后计算+P2P通信：
  - 对于Sequence并行，实际上每个节点在初始时，已经有了自己所负责部分的完整片段，可以先计算，再拉取其他节点的数据（Allgather）：$Matmul(Local)+[Allgather+Matmul(Remote)]_n$，这样就有希望将通信完全藏到计算中
  - 在流水Allgather时，通过Ring算法，每轮只获取上游数据，只向下游传递数据，减少同步次数，即整个Matmul计算周期只完成一次完整的Allgather Ring算法，一次Ring同步，而不用每次流水计算，一次完整的Ring同步
    - 但是这种方法破坏了对称性
      - 不同节点的Local对应Sequence不同位置，Matmul_TP需要在Tilling时感知RankID/RankTable，计算正确的偏移
      - 采用Ring算法，不同节点由于RankID位置的不同，每次Gather获得的Sequence数据位置也不同，需要Matmul_TP Tilling感知，并映射到每次流水计算的不同偏移上
      - 由于采用Ring算法，对910B的Mesh结构不友好，通信效率大幅度降低，适用于910C
      - Matmul_TP填写的RPC_Task中携带此种通信模式，非标

## Allgather+Matmul的流水并行原理

- 先计算后通信+Mesh通信：
  - 针对Sequence并行，计算时本地已经持有1/n数据，依然采用先计算后通信的策略，区别是：
    - 本地数据计算阶段，连续Sequence计算
    - 远端数据计算阶段，由于采用Mesh通信，一次性读到多个Sequence数据，不连续，需要Matmul_TP支持不连续计算
      - 麻烦的是，由于不同节点的RankID不同，因此远端数据阶段阶段，不同节点的数据离散特征是不同的，以TP=4为例，所有节点本轮都是处理3个Seq，数据跨度为4个Seq，Rank0在0号位轮空，Rank1在1号位轮空，Rank2在2号位轮空，Rank3在3号位轮空，Matmul_TP的执行流程需要感知这种数据特性，并正确处理偏移
      - 更麻烦的是，由于本地数据是单独计算的，本次通信拿到的是n-1个Seq数据，典型的TP=8时，n-1=7，不能整除24/48，不适合做核分割
      - Mesh方案最大的优点是针对910B，可以充分利用通信链路，另外，当通信Window足够大时，可以利用Allgather的特征（数据已经准备好）减少通信的次数
    - Matmul_TP在RPC Task中设置Mesh通信模型（即每次通信都从整个通信域Gather数据）

# MC²的算子编程框架




**通信API:**

- AllReduce
- AllGather
- ReduceScatter
- AlltoAll
- AlltoAllV
- Send
- Recv
- batchSendRecv

4 初始化hccl对象并下发AllGather通信任务。

```
Hccl hccl;
GM_ADDR contextGM = GetHcclContext<HCCL_GROUP_ID_0>();
hccl.InitV2(contextGM, &tilingData);
hccl.SetCcTilingV2(offsetof(AllGatherMatmulCustomTilingData, mc2CcTiling));
auto handleId = hccl.AllGather<true>(aGM, gatherOutGM, aTileEleCnt, HcclDataType::HCCL_DATA_TYPE_FP16, aRankEl
auto tailHandleId = hccl.AllGather<true>(aGM + tileNum * aTileSize, gatherOutGM + tileNum * aTileSize, aTailEl
                                         HcclDataType::HCCL_DATA_TYPE_FP16, aRankEleCnt, tailNum);
```

5 初始化Matmul对象，对本卡数据进行Matmul计算。

```
Matmul<MATMUL_TYPE, MATMUL_TYPE, MATMUL_TYPE> mm;
REGIST_MATMUL_OBJ(GetTPipePtr(), GetSysWorkSpacePtr(), mm);
mm.Init(&localTiling);
MatmulKernel(aGM, bGM, cGM + hccl.GetRankId() * cRankSize, localTiling, mm);
```

6 逐轮等待主块的通信完成，并对其进行Matmul计算。

```
auto aAddr = gatherOutGM;
auto cAddr = cGM;
mm.Init(&tileTiling);
for (uint32_t i = 0; i < tileNum; i++) {
    hccl.Wait(handleId);
    for (uint32_t rankId = 0; rankId < hccl.GetRankDim(); rankId++) {
        if (rankId == hccl.GetRankId())
            continue;
        MatmulKernel(aAddr + rankId * aRankSize, bGM, cAddr + rankId * cRankSize, tileTiling, mm);
    }
    aAddr += aTileSize;
    cAddr += cTileSize;
}
```

社区通算融合算子开发指导:
https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/850alpha001/opdevg/Ascendcopdevg/atlas_ascendc_10_10034.html

# MC²算子性能进阶

- **减少拷贝**，计算得到的数据直接放到数据发送区域（win区），减少拷贝；
- **细粒度通信**，每收到一块数据就进行计算，减少计算等待时间。
- **低bit通信**，在通信耗时占比高的场景下，例如对于fp16/bf16等类型量化为int8，通信后反量化回fp16/bf16类型；
- **本地数据块先进行计算。**
- **MTE通信**，提升通信效率；

在通算融合算子中，还存在很多优化的可能性，欢迎大家在开源社区贡献！

社区网站：https://gitcode.com/cann/ops-transformer/tree/master/mc2

# Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯