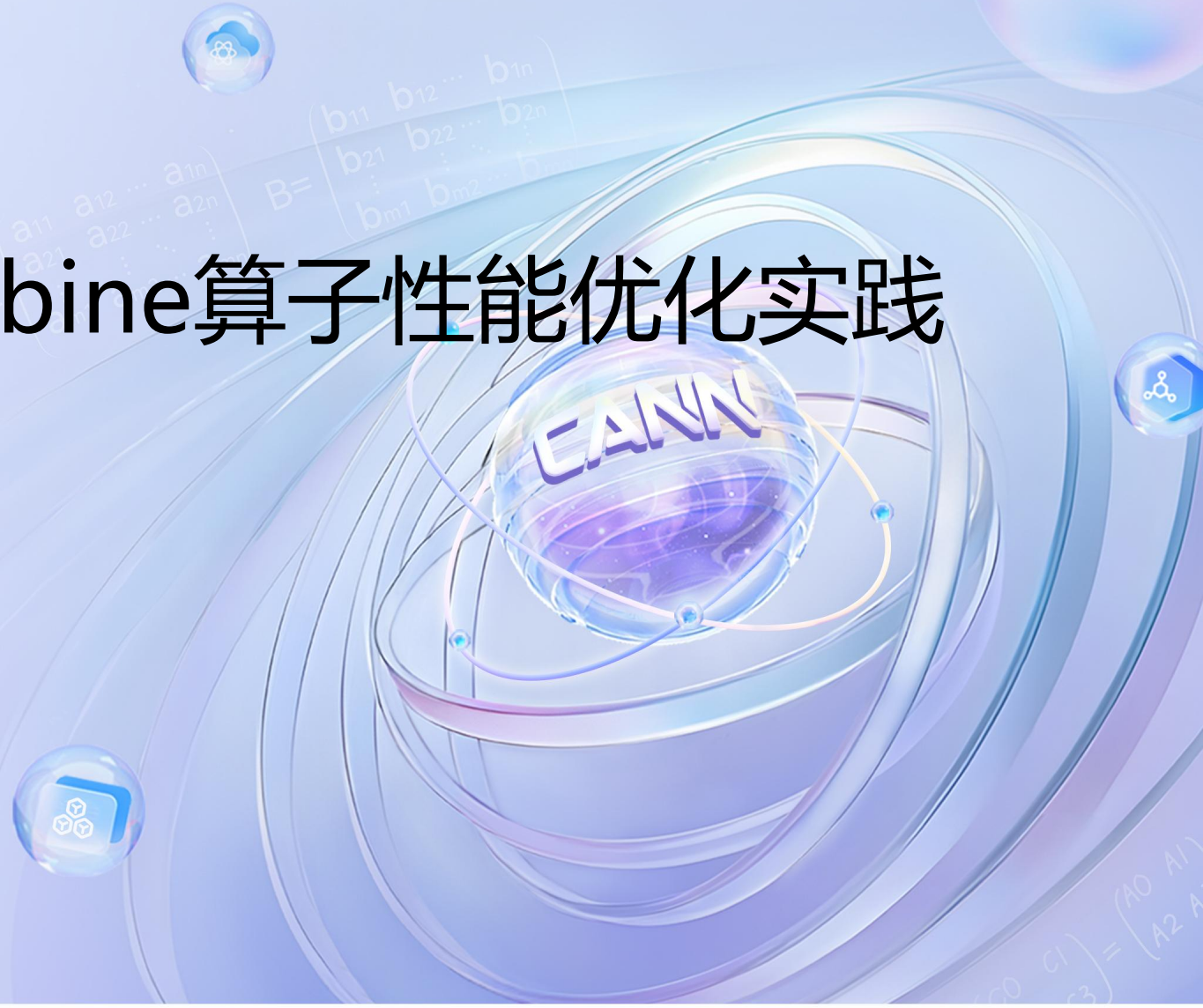
# A3 Dispatch&Combine算子性能优化实践






CANN

# 目录

# 背景

## 背景介绍

在大模型训练和推理领域，MOE架构凭借其动态专家激活机制带来的计算稀疏性优势，以及千亿参数规模下的高推理吞吐能力，已然成为超大规模模型的核心技术方案。该架构通过分发（Dispatch）和组合（Combine）这两个关键操作，实现了输入数据的动态分配与多专家输出的高效整合，从而，在维持海量参数规模的同时确保了高性能计算。然而，随着专家并行（EP）规模的不断扩大，专家间频繁交互带来的高额通信开销，逐渐演变成为制约大模型推理性能的关键瓶颈。

## 解决方案思考

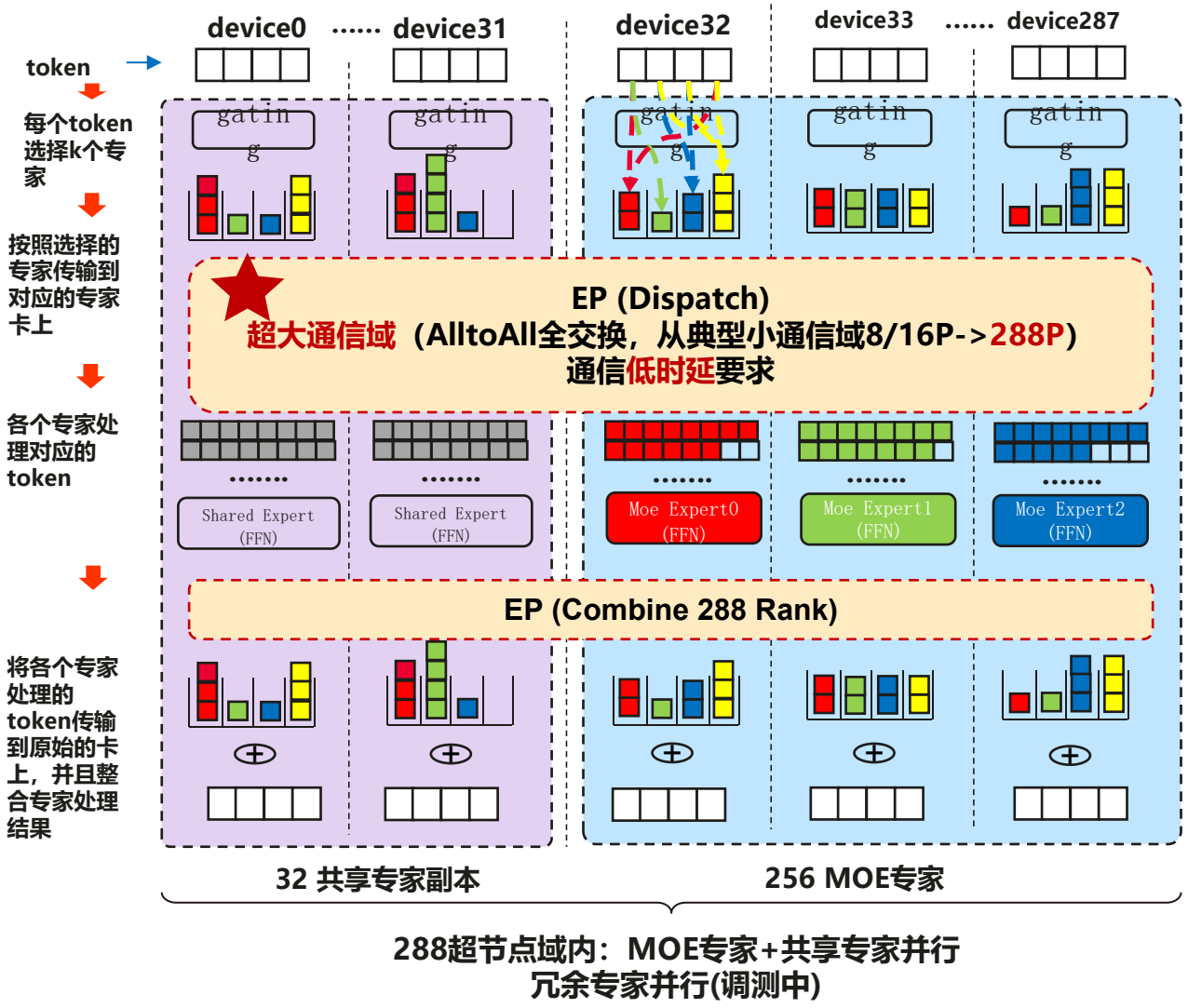
在整网推理时，每层的topk输入有变化，如果采用传统的alltoallv通信方式，那么我们需要先用alltoallv发送数据量，再通过sort把发给同一个专家的数据排在一起，最后通过alltoallv发送token数据，显然这种方式效率不高；

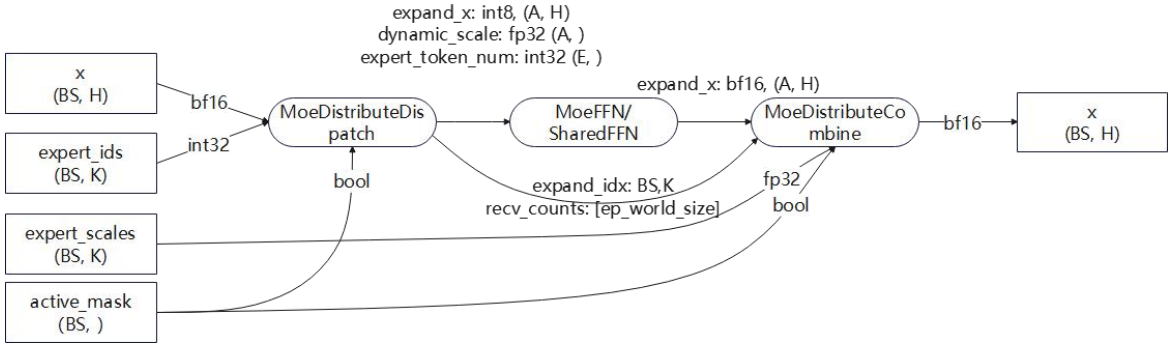
因此我们考虑把Dispatch&Combine发送数据的过程拆细，通过共享内存写的方式，逐个token的发送，形成流水并行来达到更好的性能。

CANN

# 目录

# Dispatch&Combine功能介绍




**Dispatch功能介绍**

- 完成token的分发，根据token选中的专家将token发送到对应的卡上。
- 为支持后续FFN计算的需要将各个卡发送过来的token连续的排好。
- 为支持后续的Combine处理，需要统计接收的token信息，支撑Combine将FFN计算完的token发送回到源端。
- 将量化提前到通信之前，将量化后int8的token和fp32的scale合并发送，在接收端重新排列分离。
- 支持active_mask：在固定batch size，模型进了pad数据时，为避免无效的数据传输和FFN计算，支持根据mask跳过pad的数据。

**Combine功能介绍**

- 将FFN计算完的token，发送回源端。
- 将选中的K(moe专家数)+1(shared专家数，也可能0个或多个)个FFN计算结果进行加权求和完成combine计算。

CANN

# 目录

Dispacth&Combine主要功能是通信，两者采用的基础通信方案是一致的。

# 基础通信方案介绍

**通信方式：**基于shared memory，使用AIV+UBmem进行通信；

**内存申请：**创建通信域时，每个卡申请同等大小的共享内存，并进行地址交换，从而获取到所有卡的共享内存地址（给远端发送数据时需要知道目的地址）；

**通信的4个步骤：**

1）发数据：使用DataCopy接口将数据**写到远端共享内存数据区**；

2）写状态：将Flag（数值1，按需一并发送数据量）写到远端共享内存状态区（这块内存在通信域初始化时会清0），**标识数据发送完成**；

3）等状态：**等待其他卡给本卡的数据发送完成。**从本地共享内存状态区读取Flag，直到所需数据对应的Flag全部变为1；

4）读数据：将本地共享内存中**收到的数据拷贝到output中**，并将Flag清0；

# Dispatch实现方案

**计算步骤：**

**Step1： 循环发送数据（datacopy->quant->datacopy流水并行）**

从hbm中搬运需要发送的数据到AIV核内的UBuffer上

对UBuffer上的token进行量化（并将量化后的int8 token和fp32scale拼接在一起）

将源端信息（rank_id, bs_id, k_offset，也叫三元组）拼接到量化后的UBuffer上

根据expert ids中的值查找对端rank的地址

将数据通过AIV +UBmem发送到对端（DataCopy）

**Step2： 发送flag标识和count到所有对端：**

根据expert id数据，计算发给每个expert的token数（分核处理）

计算后进行SyncAll多核同步，确保所有核的数据都发送完成后，将完成flag和count数通过AIV +UBmem发送到对端

**Step3： 等待所有对端向本地写数据完成**

分核读取状态区中的flag标识（直到读取到的flag均为1，即相应对端已经完成了数据发送）

读取状态区中的count数，计算各个rank数据搬运到output中的偏移

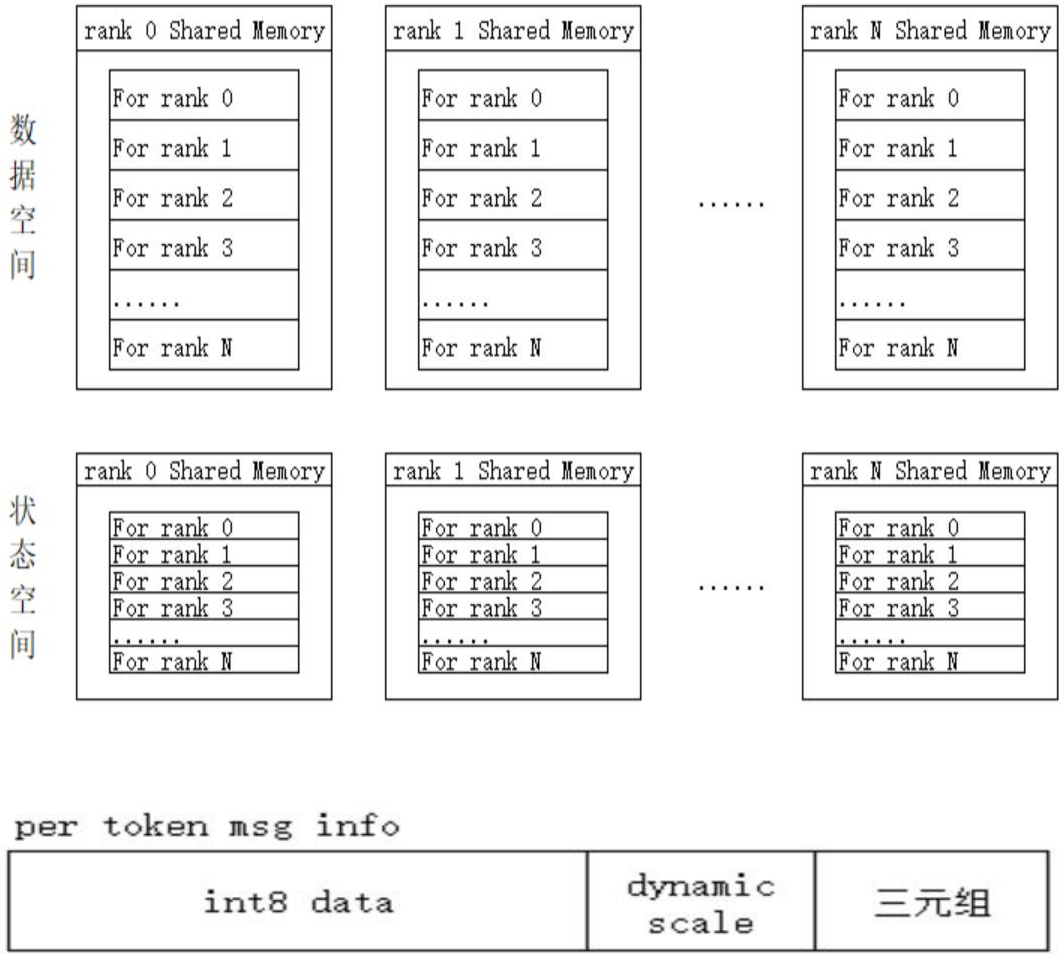
SyncAll多核同步

**Step4： local数据整理**

分核并行将通信Shared Memory中的数据整理到最终输出中（data，scale， expert token num）

输出各个expert的token num数量

**接收端空间排布**



# Combine实现方案

**计算步骤：**

**Step1：循环发送数据和发送Flag**（按照总token数量分核）

从recv count中读取总token数量，并平均分核

每个token都有对应的三元组信息（rank_id、bs_id、k_offset），发送数据时根据三元组信息计算对端地址偏移

使用AIV +UBmem将数据发送到对端

每个token发送完成后，按token发送Flag

**Step2：循环等待和计算combine**（按照batch size分核）

按每个batch循环处理

等待对应batch的状态（K+1个状态位，需全部为1）

等齐状态后，从通信shared memory中搬运各个ffn结果

从输入hbm搬运expert scale值

做combine计算，对应的moe ffn计算结果*expert scale, 对其求和，然后再加上shared ffn的计算结果

**接收端空间排布**

每个rank的Shared Memory都按此排布

数据空间

| | | | |
|---|---|---|---|
| For bs=0,k=0 | For bs=0,k=1 | | For bs=0,k=8 |
| For bs=1,k=0 | For bs=1,k=1 | | For bs=1,k=8 |
| | | | |
| For bs=n,k=0 | For bs=n,k=1 | | For bs=n,k=8 |

每个rank的Shared Memory都按此排布

状态空间

| | | | |
|---|---|---|---|
| For bs=0,k=0 | For bs=0,k=1 | | For bs=0,k=8 |
| For bs=1,k=0 | For bs=1,k=1 | | For bs=1,k=8 |
| | | | |
| For bs=n,k=0 | For bs=n,k=1 | | For bs=n,k=8 |

说明：k=8表示共享专家，可能有0个、1个、或多个；

CANN

# 目录

# 关键设计1：双分区

**思考点：**每张卡上的算子运行快慢不一致，可能存在本卡数据还未读取完成，其他卡就开始了下一轮的发送，导致数据踩踏；

**设计点：**

a）将shared memory平分成两块（分别记作0区、1区）进行交替使用，数据区与状态区都需要双分区；

b）考虑Dispatch&Combine在同一个通信域串行执行，为避免两个算子间数据的相互影响，Diapatch&Combine要分配不同的内存区域，数据区和状态区都需要分开；

c）数据区内存划分如下：

| shared memory 数据区内存 划分（默认200M，可通过HCCL_BUFFSIZE配置） | | | |
|---|---|---|---|
| 0区 | | 1区 | |
| Dispatch数据区 | Combine数据区 | Dispatch数据区 | Combine数据区 |

d）状态区内存划分如下：

| shared memory 状态内存区 划分（共1M） | | | | | |
|---|---|---|---|---|---|
| 0区 | | 1区 | | 0/1标识区 | |
| Dispatch状态区 | Combine状态区 | Dispatch状态区 | Combine状态区 | Dispatch 标识 | Combine 标识 |

# 关键设计2：Dispatch分核策略

**思考点：**收发数据时如何分核，才能达到效率最高

**设计点：**

1. 发送数据时分核：1）如果有共享专家卡（仅部署共享专家的卡），则按总数据量比例分部分核专用于给共享专家卡发送数据；2）按总数据量平均分核；
2. 写状态分核：按需要写的状态位数量平均分核；
3. 等状态分核：按需要等待的状态位数量平均分核，每个核判断一部分的状态，等到之后进行syncall全vector核同步
4. 接收数据分核：与等状态分核保持一致，接收所等待状态位对应的token；

CANN

# 关键设计3：Combine内存排布

**思考点：**Combine算子中所能接收的数据量和topk分布无关，仅和batch size、K、共享专家数量有关，为了便于组合，可以考虑把同一个batch的数据放到连续的内存中；

**设计点：**

数据区排布示例如下：

| | K | | | sharedExpertNum | | |
|---|---|---|---|---|---|---|
| BS | data | data | ... | data | data | ... | data |
| | data | data | ... | data | data | ... | data |
| | data | data | ... | data | data | ... | data |
| | ... | ... | ... | ... | ... | ... | ... |
| | data | data | ... | data | data | ... | data |
| | data | data | ... | data | data | ... | data |

说明：

a）data表示一个token的数据；

b）同一个batch先排moe专家发来的再排共享专家发来的；

c）所有data连续排列。

D）对于每个token的整合，只需要等待特定的状态位即可，减少等待时间；

CANN

# 关键设计4：Combine分核策略

**思考点：**收发数据时如何分核，才能达到效率最高。与Dispatch算子不同，Combine算子知道每个batch的数据是来源哪些卡，因此只需等待对应的状态即可，减少等待时间；

**设计点：**

1. 数据发送&写状态分核：按需发送的总token数量平均分核，发送完成后按token写状态位（发1个token写1个状态）；
2. 接收数据&等状态分核：按batch size进行分核，对于单个batch的处理，等状态时仅需等待k+1个状态位（1表示共享专家数量，可能为0或多个），等到状态全部为1之后即可进行combine处理；

CANN

# 其他设计点

除了内存排布和分核之外，还有一些其他设计点可以提升性能

**量化处理**

1、Dispatch：把量化挪到通信之前，减少通信数据量，通信时把动态量化参数一并带上；

2、Combine：当数据量比较大时，也可以量化减少数据量；先将数据用per block量化为int8，减少通信量，收到后反量化（会有一定的精度损失，需要从整网评估能否接受）；

**Double buffer处理**

在kernel实现中，我们可以使用Double buffer，做到流水并行，包括：量化计算、数据拷入、数据拷出；

CANN

# 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

https://gitcode.com/cann