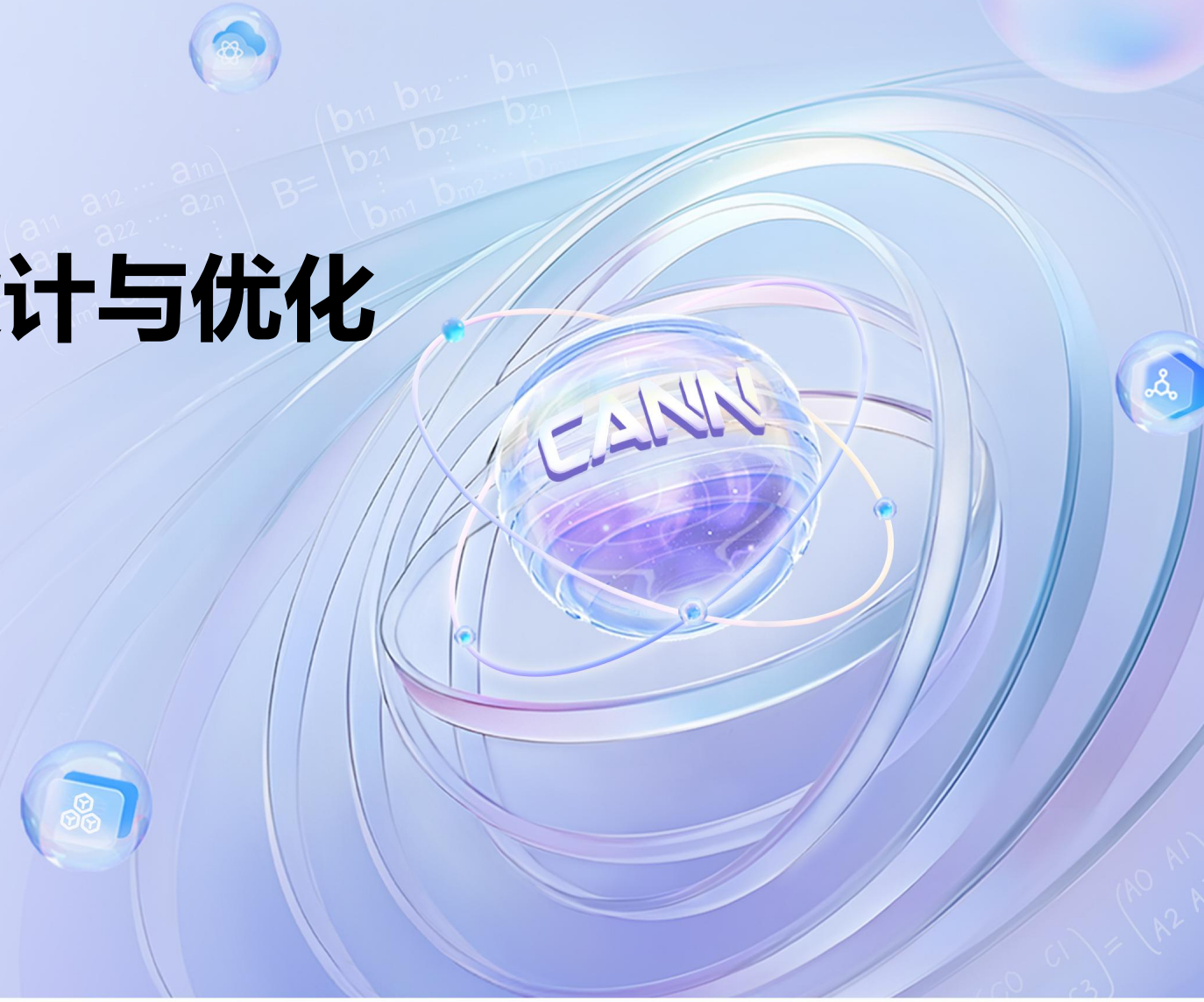
低精度GMM算子设计与优化

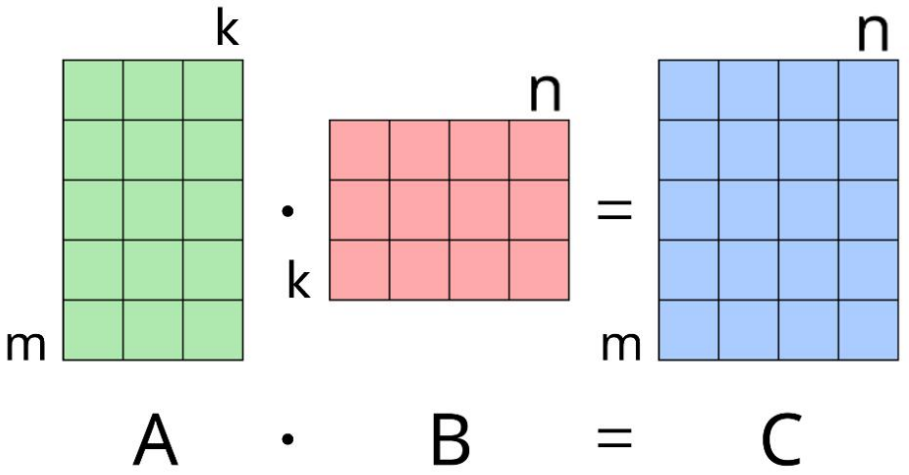


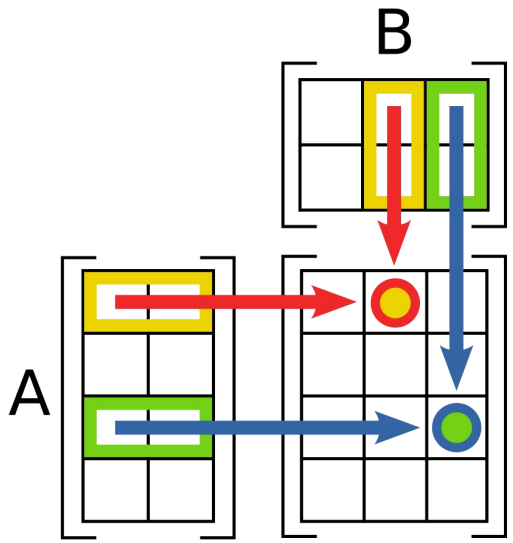
CANN

目录

GMM算子简介——前置基础知识Matmul

- 矩阵乘法是一种二元运算，它从两个矩阵生成一个矩阵。
- 对于矩阵乘法来说，第一个矩阵的列数必须等于第二个矩阵的行数。
- 生成的矩阵，称为矩阵乘积，具有第一个矩阵的行数和第二个矩阵的列数





设$A=\begin{bmatrix}1 & -2\\ -1 & 0\end{bmatrix}$, $B=\begin{bmatrix}2 & 0\\ 3 & 1\end{bmatrix}$, 计算AB:

$$AB=\begin{pmatrix}1 & -2\\ -1 & 0\end{pmatrix}\cdot\begin{pmatrix}2 & 0\\ 3 & 1\end{pmatrix}$$

$$=\begin{pmatrix}1\times 2+(-2)\times 3 & 1\times 0+(-2)\times 1\\ (-1)\times 2+0\times 3 & (-1)\times 0+0\times 1\end{pmatrix}$$

$$=\begin{pmatrix}-4 & -2\\ -2 & 0\end{pmatrix}$$

在某些场景下：
同时做多个矩阵乘时，可汇成一个算子
降低启动开销，优化负载均衡

多个同shapeMM->BMM
多个不同shapeMM->GMM

CANN

GMM算子简介——什么是GMM

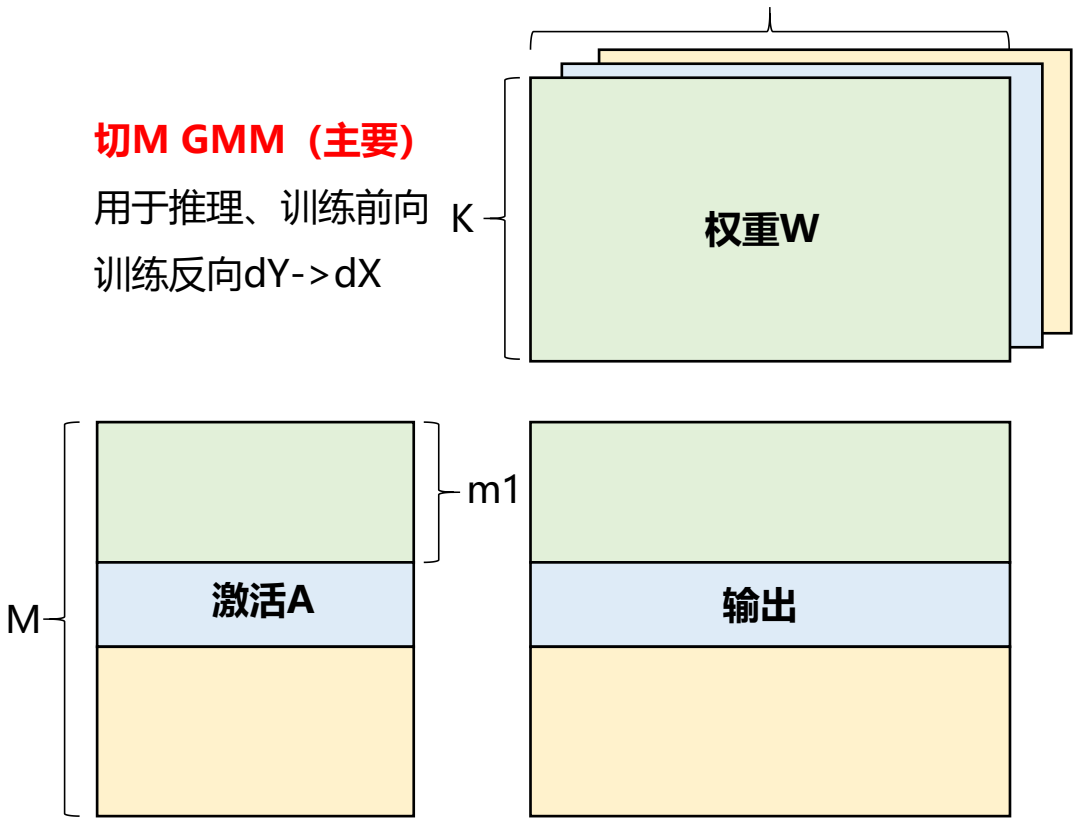
GMM（GroupedMatmul），指的是将多个独立、但大小可能不同的矩阵乘在一个kernel内计算的算子。

依据使用场景，划分为切M、切K两类。

切M GMM（主要）

用于推理、训练前向

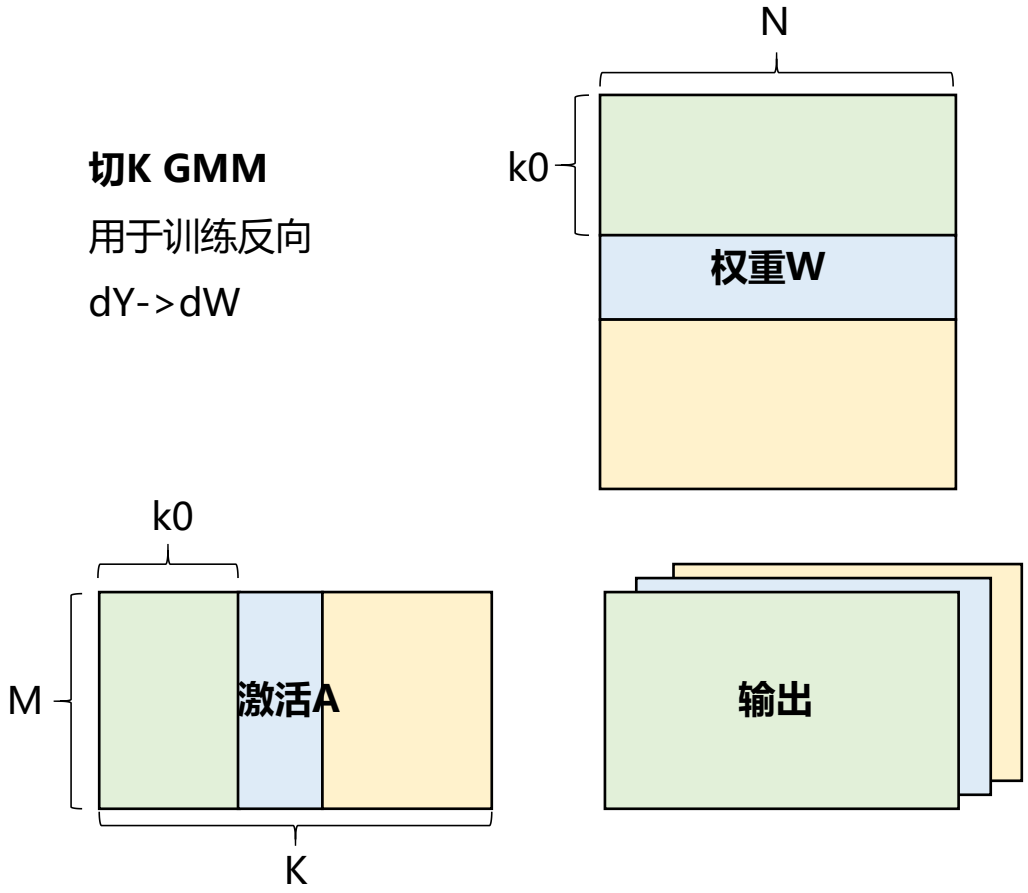
训练反向dY->dX



切K GMM

用于训练反向

dY->dW



GMM算子简介——GMM基本算子方案

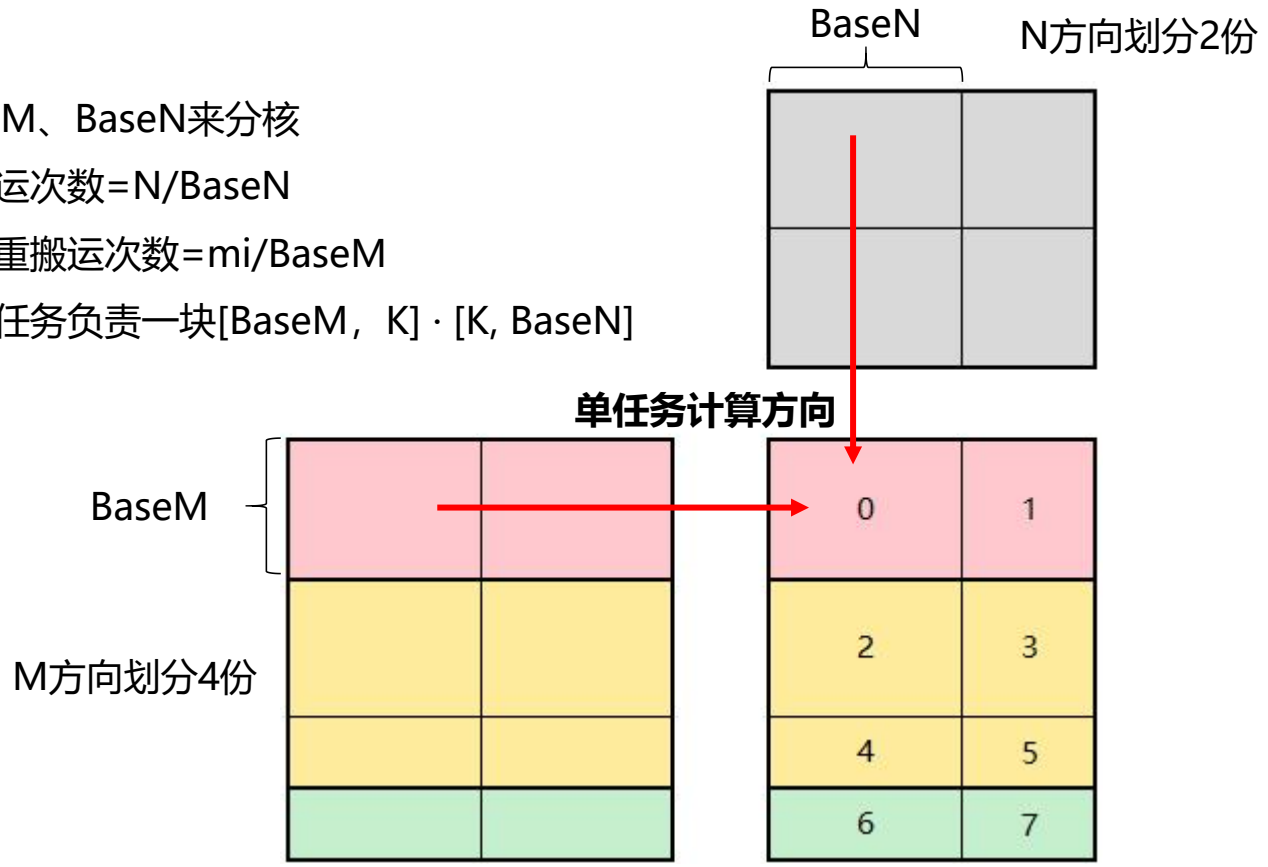
GMM的计算流程可看做多个Matmul，Matmul上的优化点均可应用于GMM

以BaseM、BaseN来分核

激活搬运次数=N/BaseN

每个权重搬运次数=mi/BaseM

每个子任务负责一块[BaseM，K] · [K, BaseN]



共划分8个子任务

无K轴分核，避免确定性问题

算子内：

分块优化算法√

K轴全载优化√

对角线分核优化√

权重使能私有格式NZ√

低精度算法√

算子外：

Decoding L2Cache预热优化√

CV双流并发√

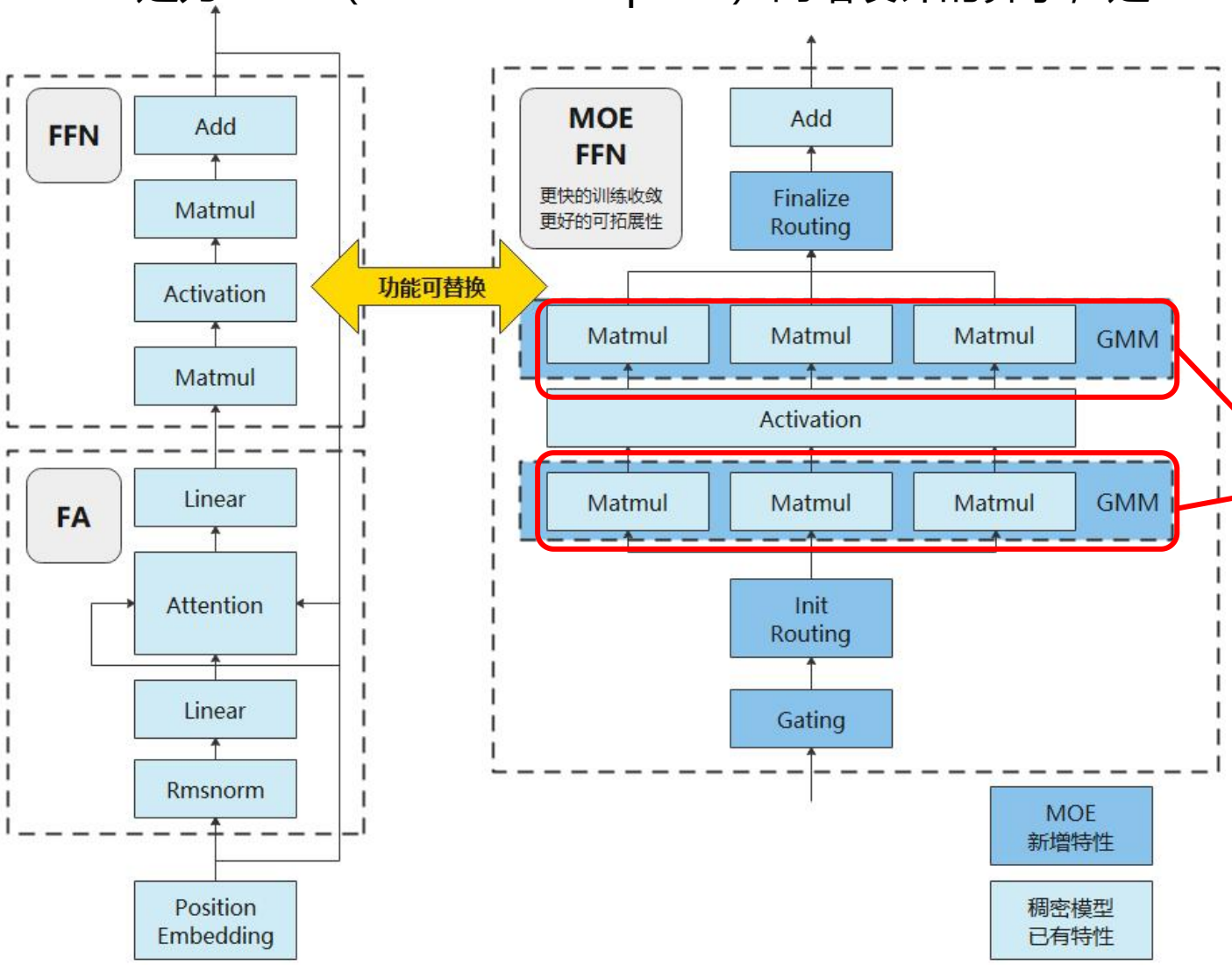
算子间：

GMM1+SwigluQuant融合√

GMM2+FinalizeRouting融合√

GMM算子简介——GMM位于哪里

GMM是为MOE（Mixture of Experts）网络设计的算子，是MOEFFN结构中的核心计算部分。



	A	B	C	D	E	F	G	H
1	OP_Type	Core Type	Count	Total Tim	Min Time(	Avg Time(	Max Time(	Ratio(%)
2	GroupedMa	MIX_AIC	1664	4784243	1879.878	2875.146	4246.705	31.199
3	MatMulV3	AI_CORE	1440	4180083	2022.841	2902.835	4189.304	27.259
4	MatMulV3	MIX_AIC	224	1210151	5372.407	5402.459	5657.773	7.892
5	Add	AI_VECTOR	7318	1159974	1.1	158.51	4612.332	7.564
6	MatMulV2	AI_CORE	3328	733294.8	15.12	220.341	477.589	4.782
7	FlashAtte	MIX_AIC	224	434584.7	1890.778	1940.11	1986.98	2.834

MOE网络中，GMM热点常为20%~35%

GMM在这

由于技术趋于稳定
当下的主要提升空间：

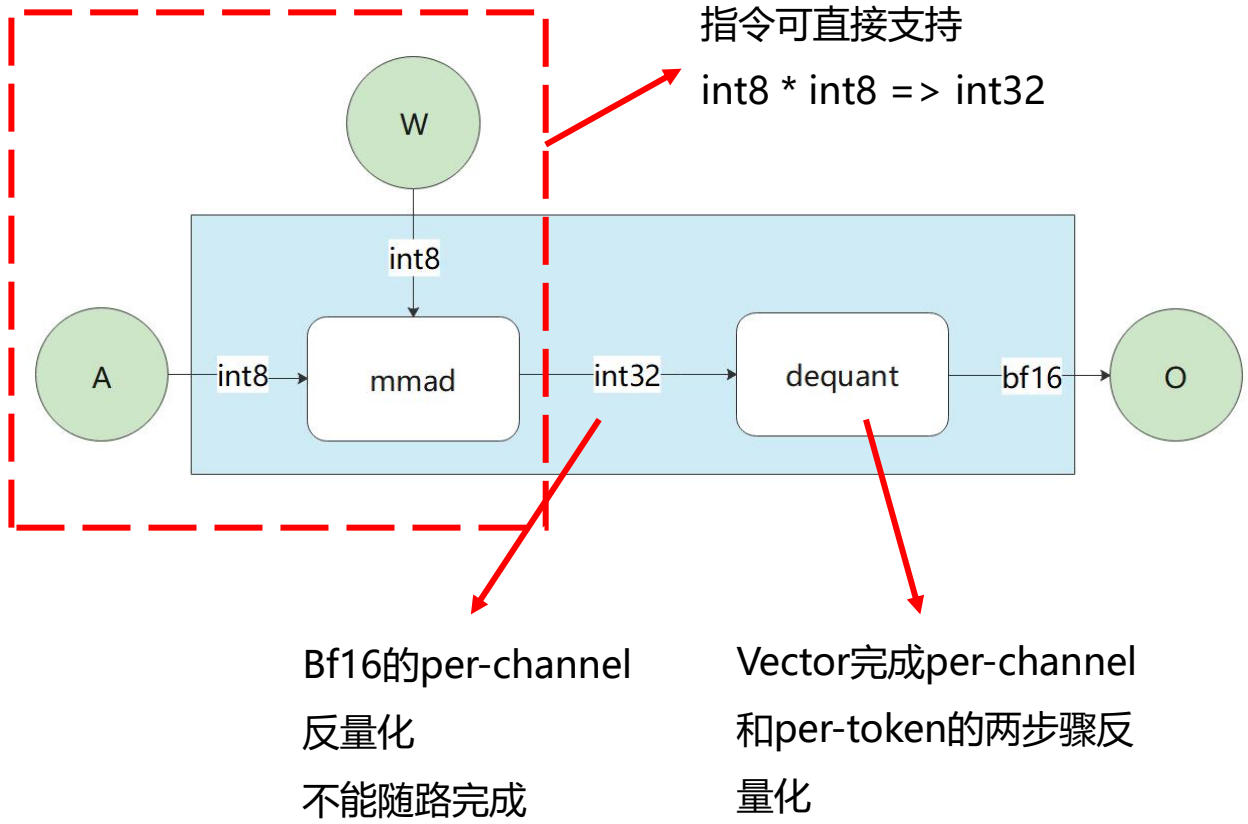
1、更低的量化精度

2、和前后结构更好的融合

目录

GMM算子低精度量化——量化算法简介

A2/A3硬件支持int8/int4数据类型矩阵乘，以最常见的A8W8 per-token/per-channel为例：



量化分类（以量化算法）

激活量化

Per-tensor、**Per-token（主要）**

权重量化

Per-channel（主要）、Per-group

量化分类（以数据类型）

全量化：硬件指令直接可支持，能完整发挥算力

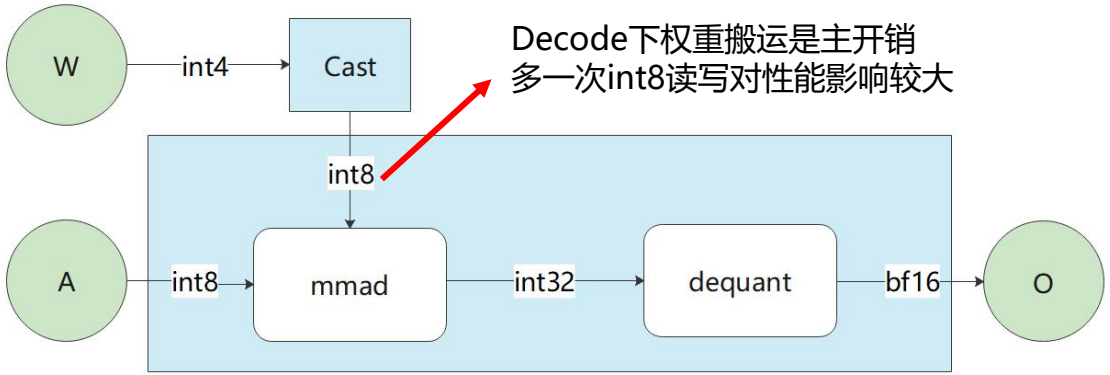
A8W8（主要）、A4W4

伪量化：A2/A3硬件指令不支持，需额外设计

A16W4、A16W8、**A8W4（主要）**

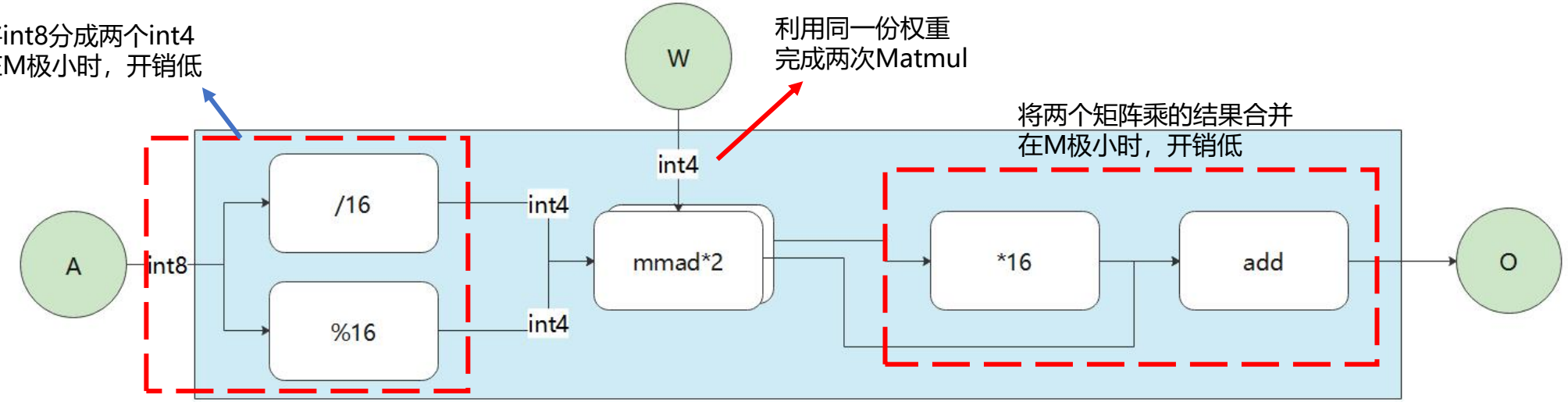
GMM算子低精度优化——伪量化方案与优化

A2/A3硬件不支持int8 * int4的矩阵乘，常见方案为对权重做int4->int8的cast后再进入GMM



	性能上限	优势场景
常规伪量化	0.8xA8W8	Prefill
MSD	1.7xA8W8	Decode

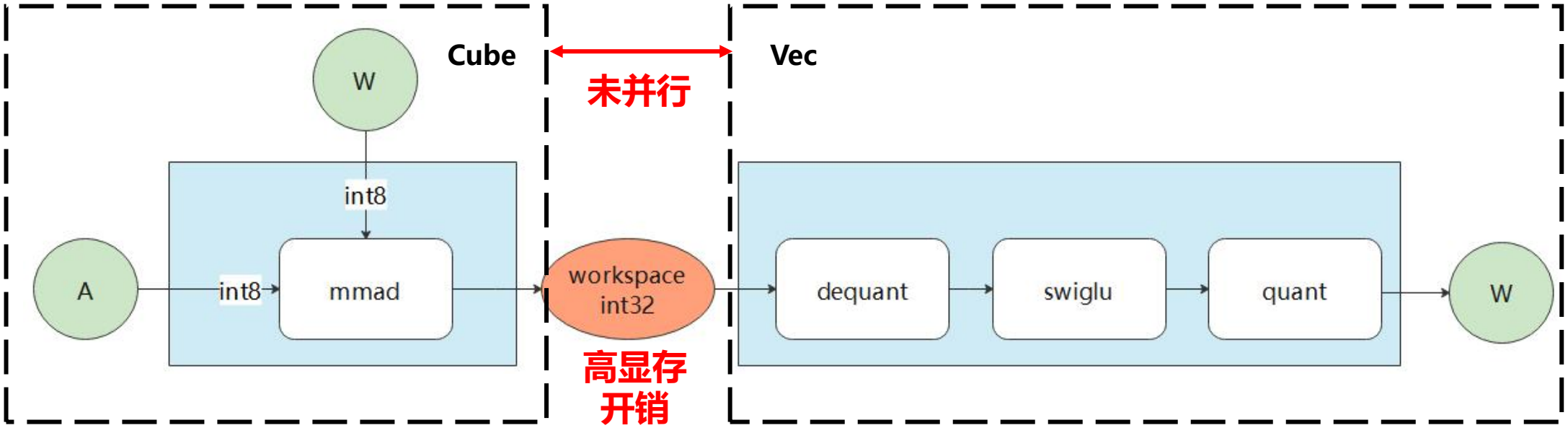
Decoding阶段，已知M相比N/K极小，可以有如下优化方案



目录

GMM算子融合优化——与SwigluQuant融合

由于MOEFFN阶段组网非常稳定，均为GMM1+SwigluQuant+GMM2的组合



Decoding阶段，此结构下存在两个痛点：

1、【性能不足】CV部分未实现深融合，若CV双流收益不够大，则CV并行度较差

2、【显存浪费】CV交互部分的int32数据类型会占据较大显存

（在较大EP时，GMM虽然激活仍较小，但M极大，可能浪费2~3G）

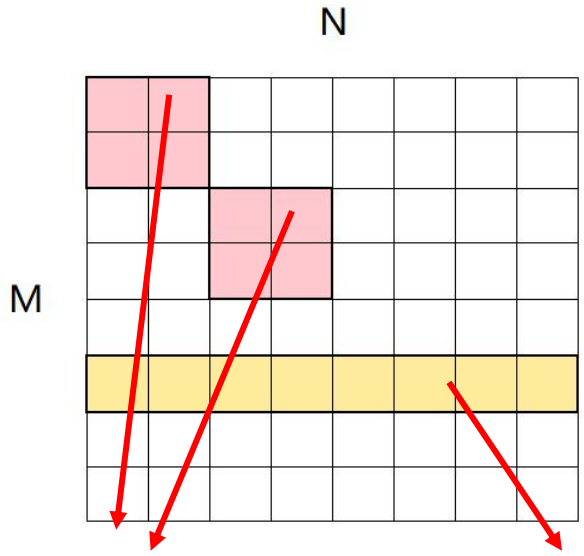
GMM算子融合优化——SwigluQuant融合优化（1）

为应对前面提及的两个痛点，对GMM+SwigluQuant做融合，融合优化分三个阶段

阶段1，浅融合，仅放在同一个kernel内，对于Decoding阶段有降低头开销和host开销的效果

阶段2，部分深融合降显存，CV分块不同导致难以直接深融合，采用大块workspace做流水掩盖

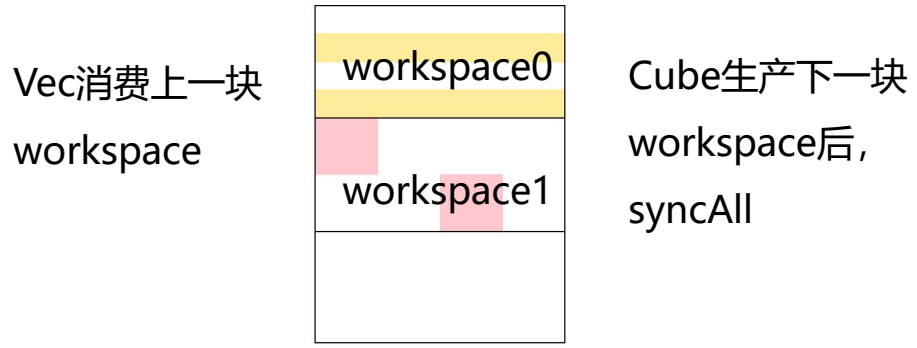
痛点问题



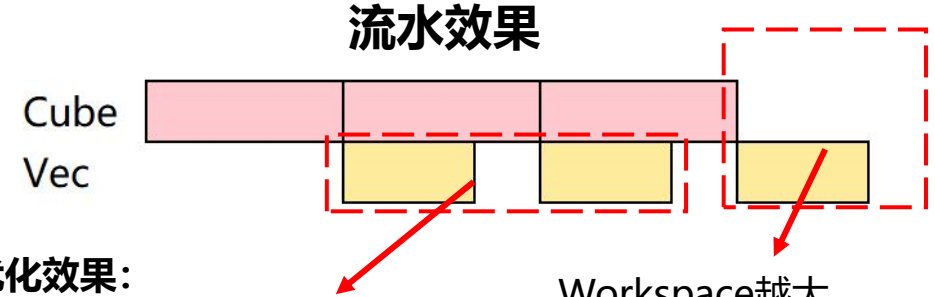
Cube以[BaseM, BaseN]
块颗粒度生产

Vec以[1, N]
条颗粒度消费

解决方案



流水效果



优化效果：

SwigluQuant被掩盖60%+

整体耗时降低30%+

Workspace越大
流水拖尾越长，**仍有提升空间**

GMM算子融合优化——SwigluQuant融合优化（2）

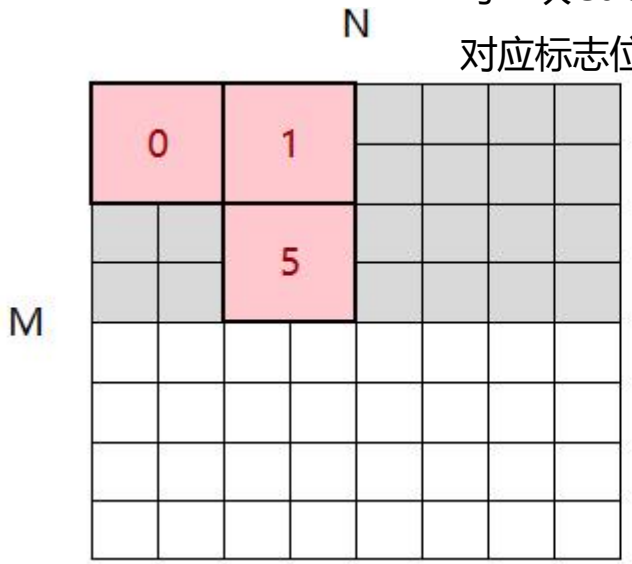
经过阶段1-2，大workspace方案实现了深融合，但是拖尾仍然很大，为此优化进入到下个阶段

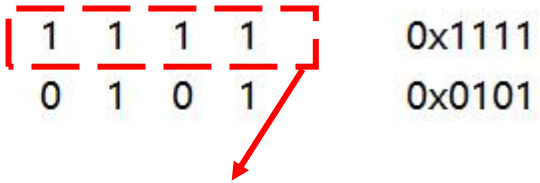
阶段3，全局深融合，降低拖尾；利用软同步实现，通过小技巧降低标量流控开销

基础方案

每一块Cube完成写入

对应标志位置1





Vector轮询标志位

全1即可读取

从而**最小依赖启动**

进阶方案

状态空间由bit表示，降低轮询成本

一次GetValue一个int32，即可检查32个flag

1 1 0 0 0x1100

0 1 0 0 0x0100

Atomic Add写入，避免读写冲突

0 1 0 0 0x0100

第四位完成写入后，atomic add写入0x0001

0 1 0 1 0x0101

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