

昇腾推理技术的优化实践

演讲人：张君

华为 / 高级开发工程师

AiCon

全球人工智能开发与应用大会

目录

01

大模型推理的现状与挑战

02

昇腾硬件亲和的FA融合算子性能优化实践

03

基于Ascend C的MC²通算融合算子性能优化

04

总结与展望

📍 北京

QCon

全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

📍 北京

AiCon

全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

📍 上海

QCon

全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月

5月

6月

8月

10月

12月

📍 上海

AiCon

全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

📍 深圳

AiCon

全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

📍 北京

AiCon

全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播



参会咨询

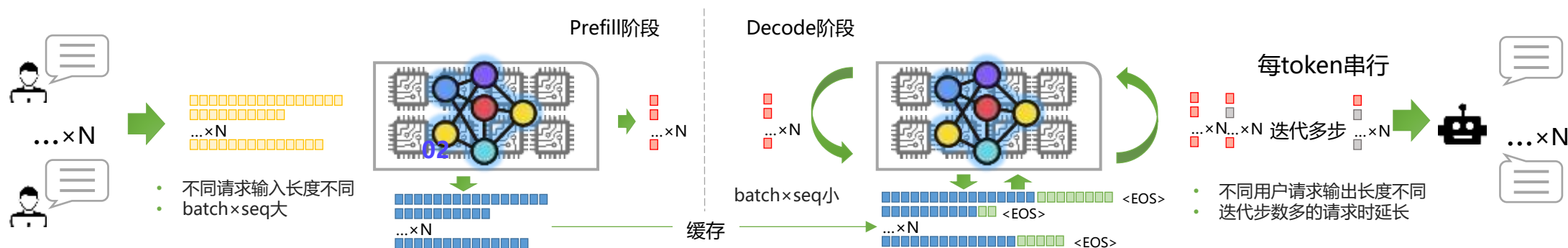


查看会议

01

大模型推理的现状以及挑战

现状1：模型规模增大及自回归解码带来访存及算力利用率压力



时延决定用户体验，吞吐衡量系统成本

模型规模大，内存容量和访存是瓶颈

内存容量不足导致单卡无法推理

- 模型参数：百亿/千亿模型参数超过单卡内存容量
- KVCache：随着batchsize和序列长度增长，占用更多内存；

访存带宽制约50ms/token推理时延达成

- 多路并发缓和带宽压力，提升吞吐但增加时延
- 随着序列增长，KVCache访存成为瓶颈，推理时延成倍增长

自回归算力利用率低，低时延高吞吐难以兼顾

Prefill和Decode两阶段推理差异大，难以充分利用算力资源

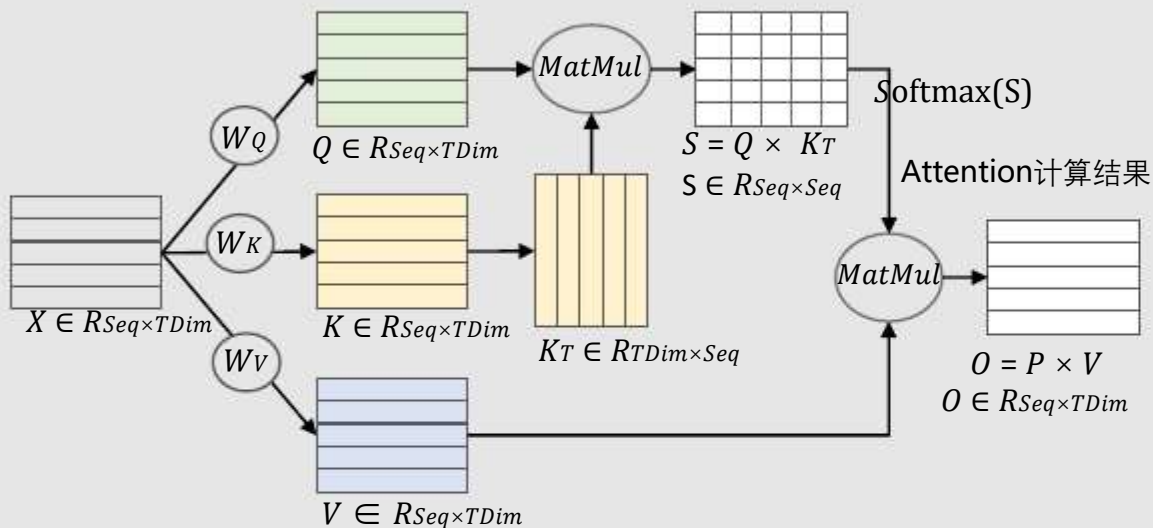
- 基于request的调度导致大量算力空闲：同batch序列长度差异大
- 不同阶段的请求难以batch：prefill/decode输入、kvcache维度差别大

Decode阶段每token串行解码算力利用率低

- 串行解码以GEMV为主，计算访存比低
- KVCache访存量随序列长度增长，Attention占比增加

■ 现状2: KV Cache导致“内存墙” 瓶颈进一步加剧

不采用KV Cache, 全量计算硬推, 计算量
随序列长度增加指数级增长



原理: 矩阵 × 矩阵, 瓶颈: 算力

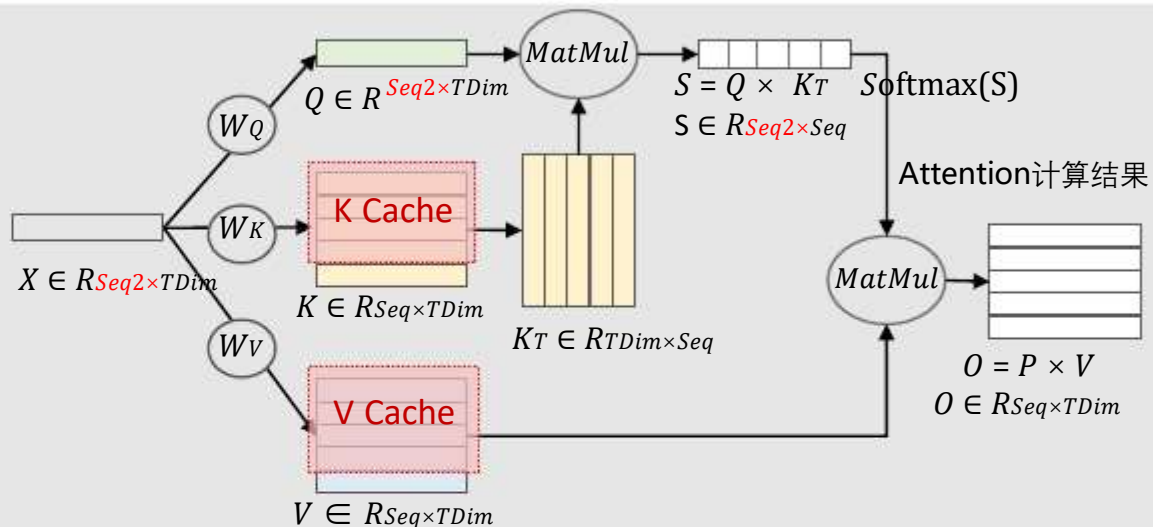
推理计算量 = $Layer * (4 * Seq^2 * TDim + 24 * Seq * TDim^2)$

Attention计算量与序列长度成平方关系, 线性层计算量与序列长度成正比

示例: 以Llama2 70B为例, 1M序列长度推理时延52秒, 算力利用率50%, 需要消耗216NPU卡

(备注: Layer网络层数, TDim特征维度, Seq序列长度, KVB为精度)

采用KV Cache, 推理内存开销数线性增长,
形成“内存墙”和“带宽墙”



原理: 向量 × 矩阵, 瓶颈: HBM或者显存开销

KVCache内存占用 = $2 * KVB * Seq * Layer * TDim * \frac{KVNum}{QNum}$

示例: 以Llama2 70B为例, 1M序列长度推理时延52秒, HBM利用率50%, 需要消耗18 NPU卡

内存墙: KV Cache内存占用随序列长度、Batch size增加, KV cache显存开销成倍上升, 采用HBM存储KV Cache成本依然很高。

■ 大模型推理常用加速技术

01

算子层优化

- 算子融合：QKV融合大算子，提升增量响应。
- 高性能加速库：cuBLAS，FasterTransformer，ATB等
- ...

发力点

02

算法层优化

- 模型分片：分片策略优化，增量推理
- 投机推理：自投机增量推理
- 模型量化：8Bit量化，4Bit量化
- Attention计算优化
- ...


03

框架层优化


- Continuous Batching
- PageAttention
- PD分离部署
- ...

昇腾硬件亲和的FA融合算子 性能优化实践


挑战1：vector能力不足，如何用cube补齐




◆ FA / PA算法的重点就是使用onlineSoftmax(分块、动态)的方法小步快跑，逐步对kv cache进行计算，提升计算效率，减少内存占用



◆ onlineSoftmax算法一般涉及exp、sub、Mul、Add等步骤，由于它们都是向量操作，所以一般放在Vec进行更新计算，这部分约占算子执行总时间的15%



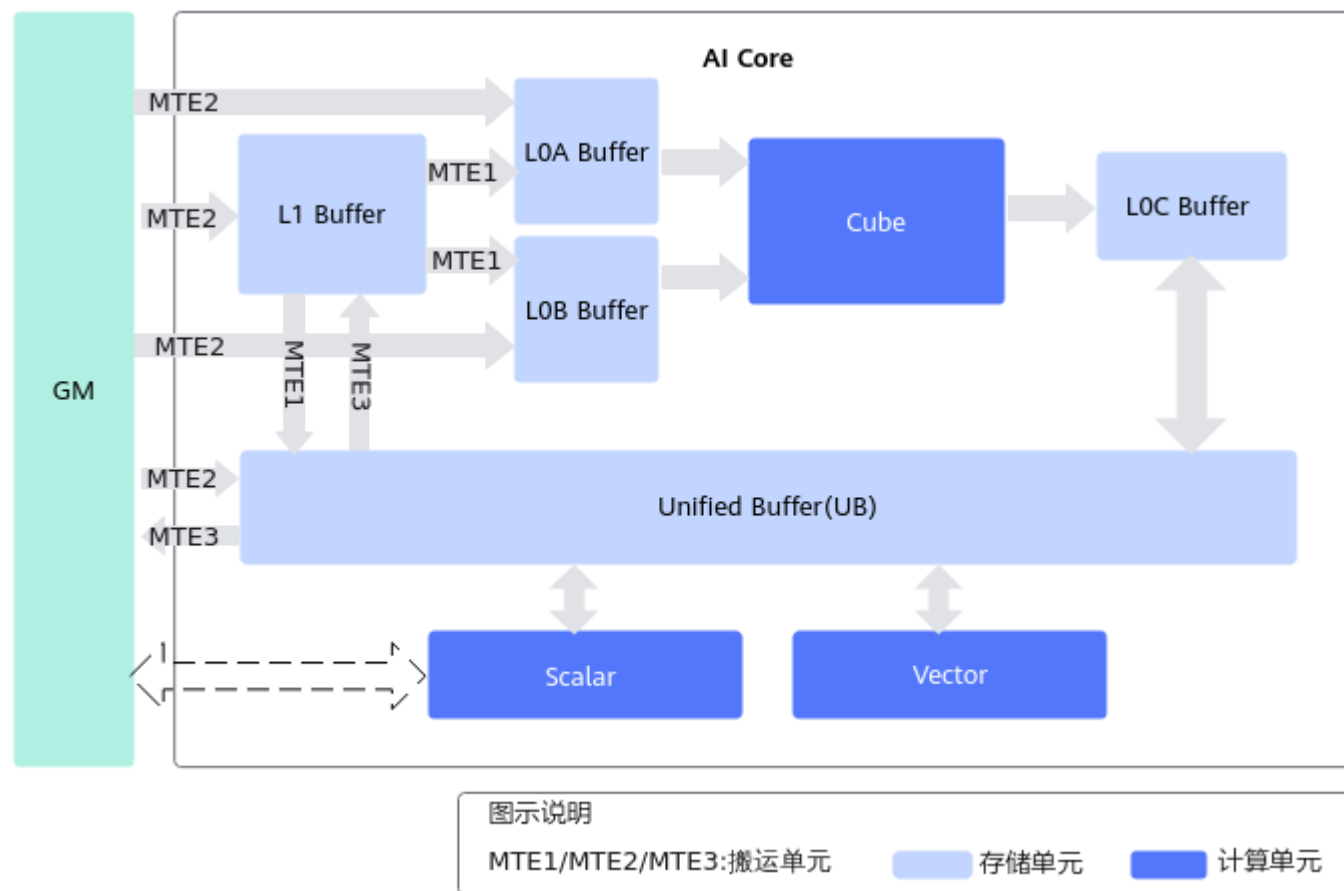
◆ 受限于昇腾Atlas 300I上Vec算力相较于Cube低很多，FA(FlashAttention)算子的性能分析结果呈现明显的VecBound(Vector是性能瓶颈，约占总执行时间的90%)



◆ 但Mul和Add操作是非常亲和矩阵乘计算的，所以现考虑能否把这两个计算步骤改到Cube进行计算，提升性能

	Task Duration	AI Core Time	Cube Time	Vec Time	MTE1	MTE2 Time	MTE3	OnlinSoftmax
耗时 (us)	2601.484	2412.06	263.347	2362.267	237.497	240.989	59.014	390.051
占比 (%)	-	92.719%	10.123%	90.805%	9.129%	9.264%	2.268%	14.993%

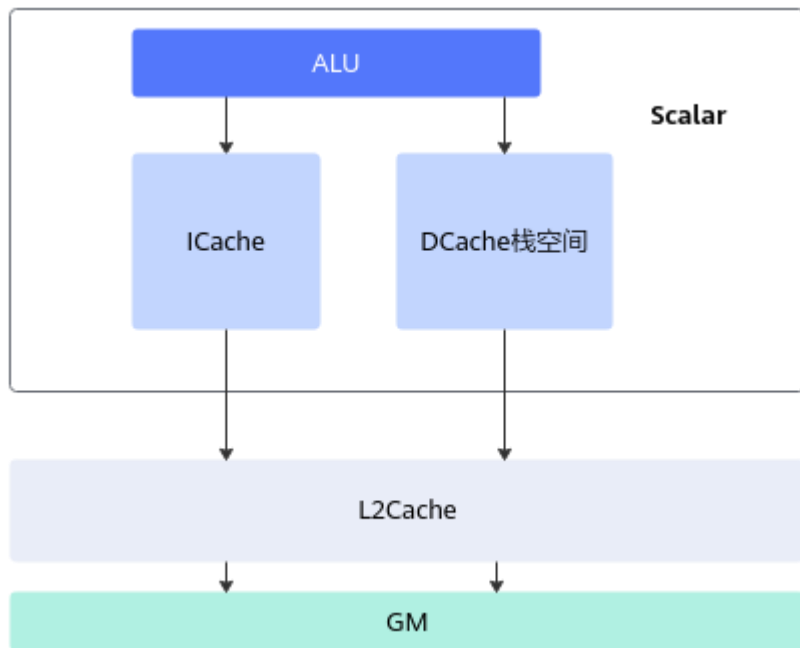
AI Core的耦合架构



- ◆ Cube计算单元和Vector计算单元同核部署，Cube计算单元和Vector计算单元共享同一个Scalar单元，统一加载所有的代码段。
- ◆ 列出了计算架构中的存储单元和计算单元，箭头表示数据处理流向，MTE1/MTE2/MTE3代表搬运单元。

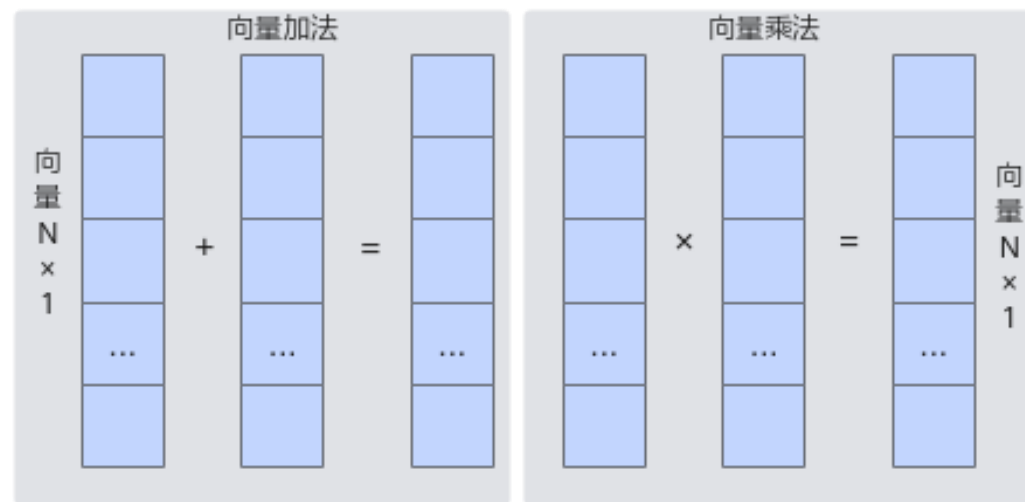
Scalar, Vector计算单元

Scalar



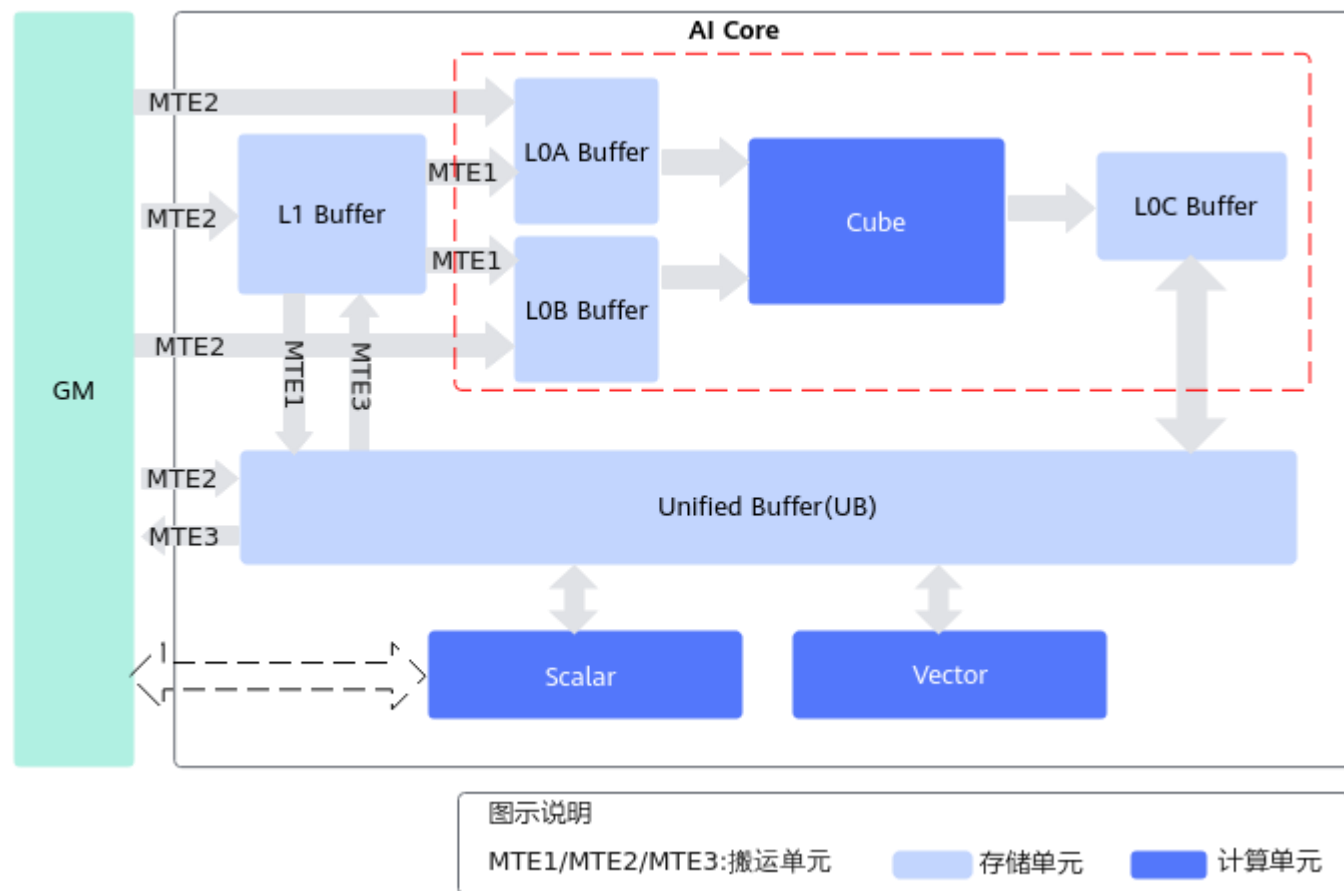
Scalar负责各类型的标量数据运算和程序的流程控制。功能上可以看做一个小CPU，完成整个程序的循环控制、分支判断。

Vector



Vector负责执行向量运算。向量计算单元可以快速完成两个FP16类型的向量相加或者相乘，向量指令支持多次迭代执行。

Cube计算单元



- ◆ Cube计算单元负责执行矩阵运算，一次执行即可完成A矩阵 ($M * K$) 与B矩阵 ($K * N$) 的矩阵乘。
- ◆ 如左图所示红色虚线框划出了Cube计算单元及其访问的存储单元，其中LOA存储左矩阵，LOB存储右矩阵，LOC存储矩阵乘的结果和中间结果。

■ 构造对角矩阵，使用Cube代替Vector

FlashAttention算法中onlineSoftmax部分的更新公式为：

$$O_i = e^{m_{j-1}-m_j} \cdot O_{i-1} + P_i \cdot V_i$$

其中，计算 $e^{m_{j-1}-m_j} \cdot O_{i-1}$ 时，由于 e 是一维向量， O 是二维矩阵，维度不同，所以一般在Vec中需要先对 e 做广播和扩展得到 e_{brd} ，然后再拿 e_{brd} 和 O 做逐元素的相乘。

那么可以考虑把向量 $e^{m_{j-1}-m_j}$ 转化为对角矩阵 $diag(e^{m_{j-1}-m_j})$ ，再与 O_{i-1} 做矩阵乘。那么就可以把这部分Vector的耗时挪到cube去。即使Cube的运行耗时会因此增加，但由于在Atlas 300I上这个算子是VectorBound，只要Vector耗时减少，算子总体耗时必然也会减少。

■ 遭遇 “假” VecBound性能瓶颈

待根据优化思路实现完一版后，测试性能却发现效果与预期相差较远：

	Task Duration	AI Core Time	Cube Time	Vec Time	MTE1	MTE2 Time	MTE3
优化前 (us)	2601.484	2412.06	263.347	2362.267	237.497	240.989	59.014
优化后 (us)	2891.383	2681.1	363.272	2418.704	307.946	252.969	122.162
优化效果 (%)	-11.14%	-11.15%	-37.94%	-2.39%	-29.66%	-4.97%	-107.01%

Vec耗时不降反增

- 原Vec中onlineSoftmax的部分耗时确实已经消减(300us左右)，但构造对角矩阵引入了额外的成本(370us左右)，此消彼长，最终导致Vec耗时增加了60us左右。
- 因此首先要做的，就是优化对角矩阵的构造逻辑，使Vec耗时有明显下降。这是整个方案效果的基石。

总耗时的增加远大于Vec耗时的增加

- 另外分析总耗时发现，它增加了约290us：
 $290(\text{总耗时}) = 60(\text{vector}) + 100(\text{cube}) + 70(\text{mte1}) + 70(\text{mte3})$
这个数据也非常不合理，因为Vec耗时只增加了60us左右。
- 无论是Cube, MTE1, MTE3都没有被Vec很好地掩盖，所以他们耗时的增加也导致了总耗时的增加。
- 除优化对角阵构造外，另一个需要关注的点就是新方案下的流水调整和掩盖，目标是无论cube、MTE1或MTE3，它们的流水能最大程度地被Vec掩盖。

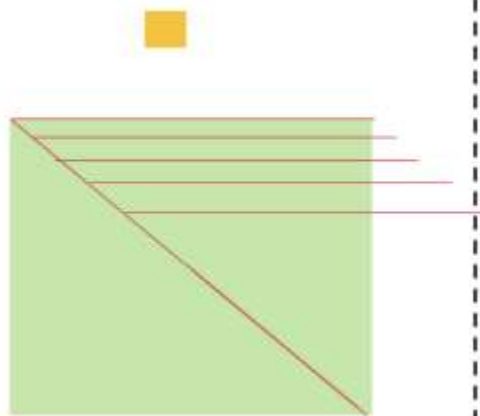
■ 对角阵构造方案

方案一：Scalar直接赋值



- **使用Scalar指令完成：**通过 GetValue()、SetValue()方法，直接将目标值赋到 $m \times m$ 矩阵的对角线上

方案二：Vector: Vadd+Mask



- **使用Vector操作，按照行进行遍历，**通过设置 `vector_mask=1`，利用 `AscendC:add_v()`方法每次在对角线上赋值一个元素。

方案三：Vector: 广播+Vmul指令

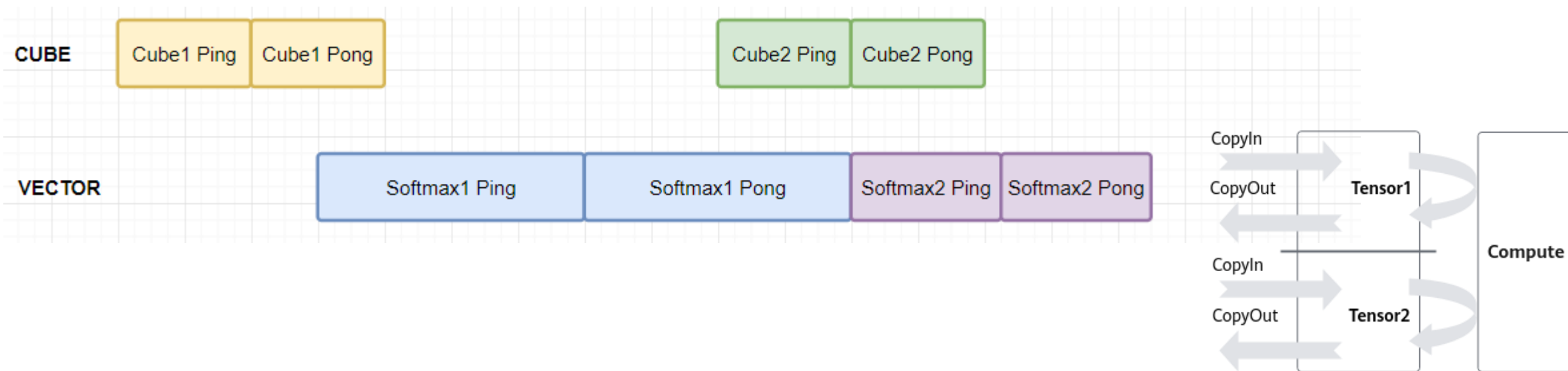
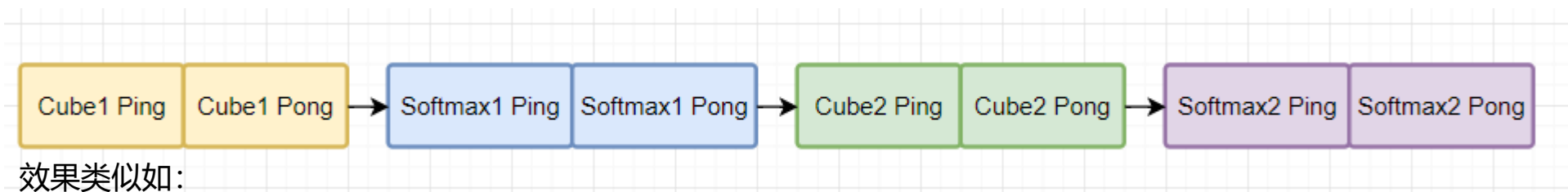


- 使用Vector操作，但是利用 **16*16的单位矩阵和原向量做乘法**来一次生成16行的结果，并将结果直接放到 $m \times m$ 的对角矩阵中。

方案三：
耗时最低，循环
次数减少了16
倍,vector下降10%

进一步优化：流水调整和掩盖

通过分析代码和流水发现，原来的代码结构在开启double-buffer(双缓冲区)的情况下，是执行完第一个步骤的Ping-Pong再去执行下一个步骤的Ping-Pong：



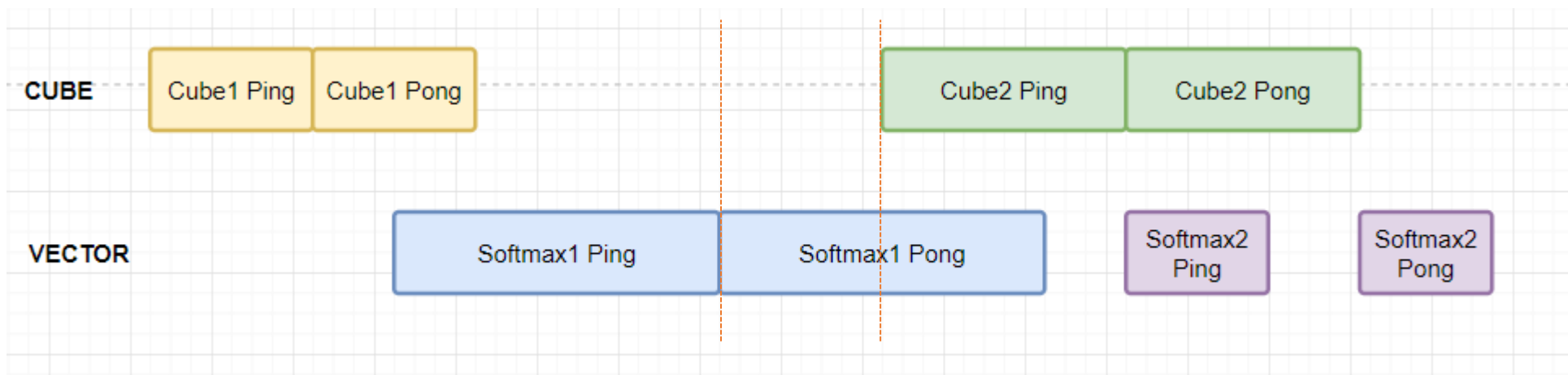
PingPong Buffer通过将数据流分为两部分（例如Tensor1和Tensor2），使得在计算Tensor1的同时可以进行Tensor2的数据搬运。

进一步优化：流水调整和掩盖

在原先的代码流程里，得益于softmax1/2执行时间都较长，所以cube计算都比较好地做了掩盖，这样的计算排布方式没有任何问题。但是将onlineSoftmax计算移到cube之后，流水排布中产生了两大变化：

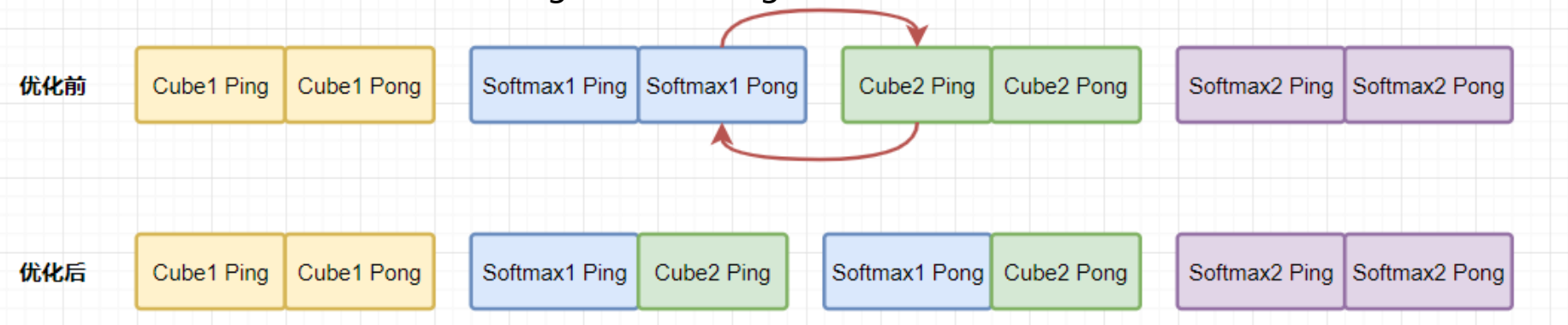
- Cube2中增加了onlineSoftmax步骤，耗时变长；
- Softmax2中减少了onlineSoftmax步骤，耗时变短；

于是流水图出现了下面的变化：

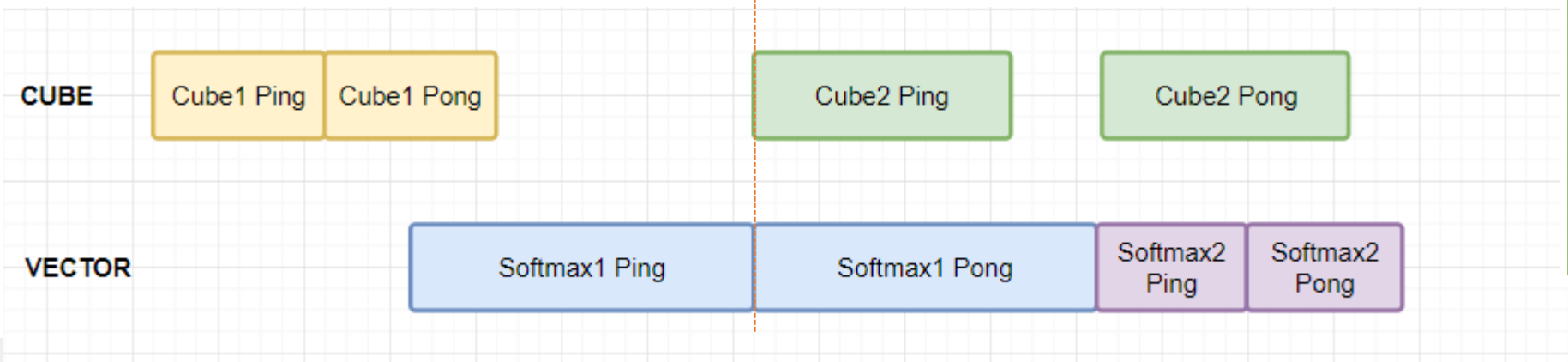


进一步优化：流水调整和掩盖

经过分析和测试，交换Softmax1-Pong和Cube2-Ping的执行顺序，类似下图：



- 首先，这不会影响计算结果，因为Cube2-Ping只依赖于Softmax1-Ping的计算完成，不依赖Softmax1-Pong的计算结果。
- 其次，这样做的好处是使Cube2-Ping提前执行，那么它就能更早地被Softmax1-Pong所掩盖。而由于它被提前执行结束了，依赖于它的Softmax2-Ping也能提前执行，那么Vec之间的间隔也会减少，效果如下图：



效果：算子总耗时下降了5%左右，整体性能提升来到了**8%**左右(1k序列场景)

思考



- ◆ 考虑Pipe之间（不同计算单元、模块）的掩盖，如果没有的话，即便某个pipe耗时只占1%，它的性能波动仍然会同等程度地影响算子的总耗时。



- ◆ 性能优化要有整体视角。需要从计算、内存搬移等多个角度考虑。



- ◆ 性能优化中很重要的一部分内容就是**流水间的相互掩盖**。
- ◆ 可以通过分析流水图等方式，提升对流水掩盖的直观理解，完成流水优化的最重要一公里。

03 基于Ascend C的MC²通算融合 算子性能优化

■ 挑战2：计算效率与通信开销的平衡？ MC²算子优化

- 模型规模的指数级增长使得单设备训练在计算能力、存储容量和能效等方面面临根本性瓶颈，这使得分布式并行推理/训练从可选方案转变为必选技术路径。以1750亿参数的GPT-3模型为例，其训练至少需要数百GB的内存资源。
- 传统的单维度并行策略（如数据并行）已无法满足需求，而混合并行（Hybrid Parallelism）通过同时切分计算图（模型并行）、数据批次（数据并行）以及流水线阶段（流水线并行），成为当前主流解决方案。
- 昇腾CANN通过将需要切分并行的计算和通信算子进行融合（又称：通算融合），实现了计算和通信的流水并行，从而提升性能。融合后的算子简称为MC²通算融合算子（Matrix Computation & Communication）。

在分布式推理/训练框架中，
MC²性能优化成为关键挑战

MC²通算融合算子的性能收益

MC²通算融合算子的性能收益主要来自于通信、计算的并行执行。



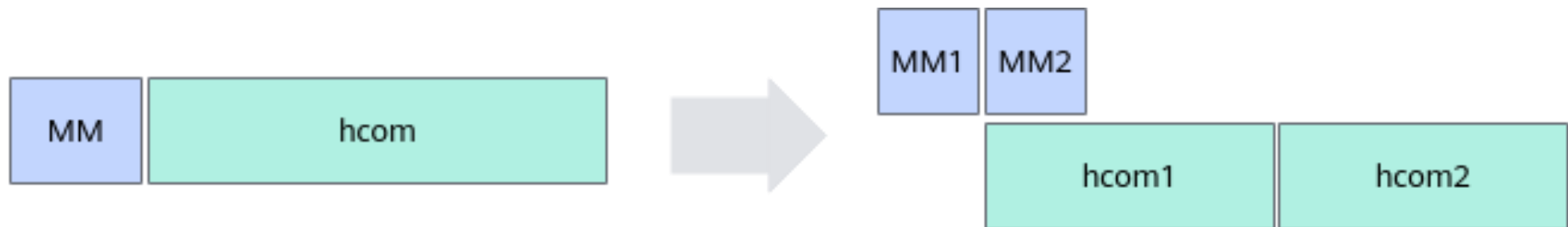
MM代表Matmul计算，hcom代表通信任务。

对Matmul计算的矩阵进行切分，通过下一块数据的Matmul计算与当前数据块的通信任务并行执行，从而达到隐藏计算/通信时间的目的。

上图中，进行Matmul计算的矩阵沿M轴（行）被切分为两块，则第二块数据的Matmul计算和第一块数据的通信可以并行执行，从而通过隐藏计算时间提高算子性能。

■ 瓶颈点1：通算时间差异大

- 若计算和通信任务的执行时间**相差不大**，则融合后，MC2算子的计算和通信并行执行，能得到较好的流水掩盖，**性能收益较大**。
- 若计算和通信任务的执行时间**差异较大**，则融合后，MC2算子内计算和通信并行执行，能够掩盖的时间较少，算子整体执行耗时与未切分串行时的算子执行耗时接近，此时**无法获得较大性能收益**。

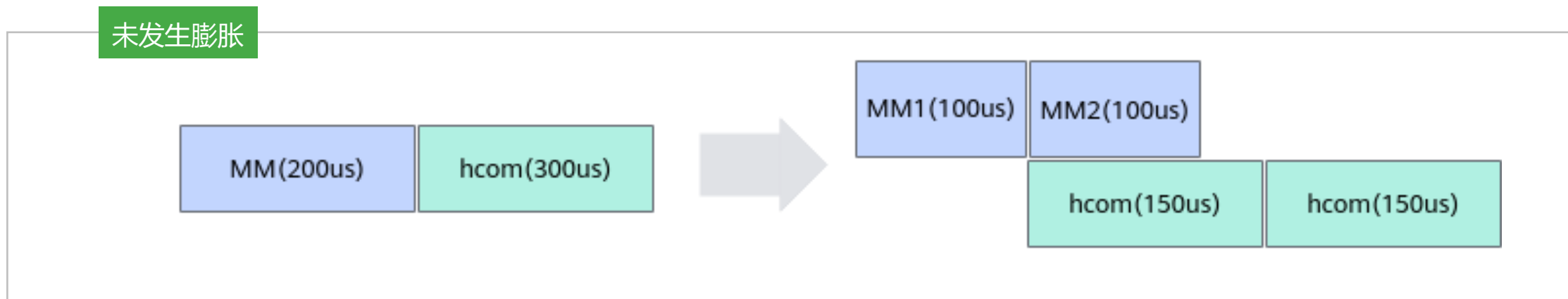


■ 瓶颈点2：数据切分导致的计算或通信的执行时间膨胀

原本的整块数据被切分成若干小数据块，对若干小数据块分别做Matmul计算或者执行通信任务，此时相比切分前，计算或者通信任务的执行时间可能发生膨胀（执行时间变长）。

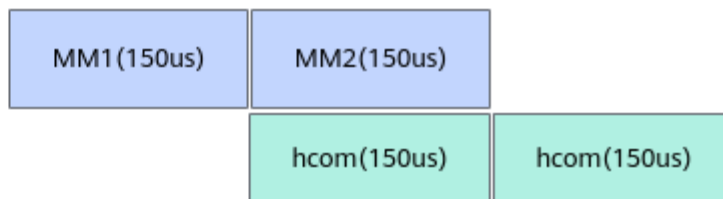
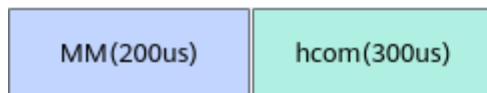
膨胀产生的原因包括：

- 切分后的数据块过小导致计算或通信的效率降低；
- 切分的数据块过多导致增加额外的调度开销；
- 并行执行后计算和通信对L2Cache（L2Cache用于缓存访问GM的数据（包括代码段、数据段））或device存储内存的访问冲突等。



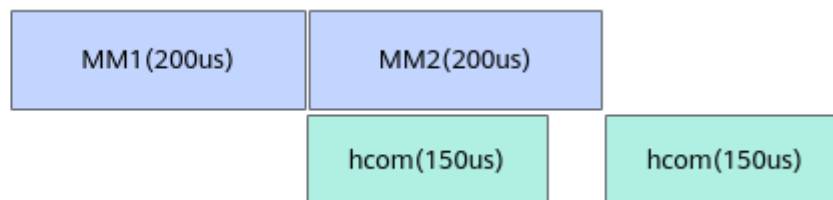
■ 瓶颈点2：数据切分导致的计算或通信的执行时间膨胀

一般膨胀



数据切分前，Matmul执行时间为200us，将Matmul的输入均匀切分为两块，假设切分后，每块数据的Matmul执行时间都是150us，通过计算的并行执行，上图实际性能收益为50us。

严重膨胀



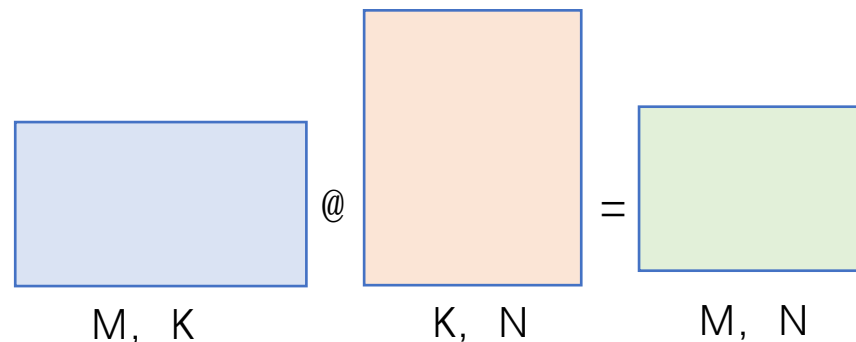
数据切分前，Matmul执行时间为200us，将Matmul的输入均匀切分为两块，假设切分后，每块数据的Matmul执行时间都是200us，通过计算的并行执行，上图实际性能收益为劣化50us。

- 计算和通信执行时间较均衡的场景，有更好的流水掩盖和性能收益；
- 同时，性能收益也受到数据切分导致的执行时间膨胀的影响。
- 需要制定并行切分策略，以达到最佳流水掩盖效果。

MC²优化方案设计

以MatmulAllreduce算子为例，该算子中计算执行在前，通信任务执行在后。

算子中的通信对象为Matmul的输出矩阵，则通信任务的输入shape为[M, N]。



方案设计步骤：

1. 判定bound场景。在制定数据切分策略前，对原始矩阵分别执行计算和通信任务，根据两个任务的执行时间判定bound场景。

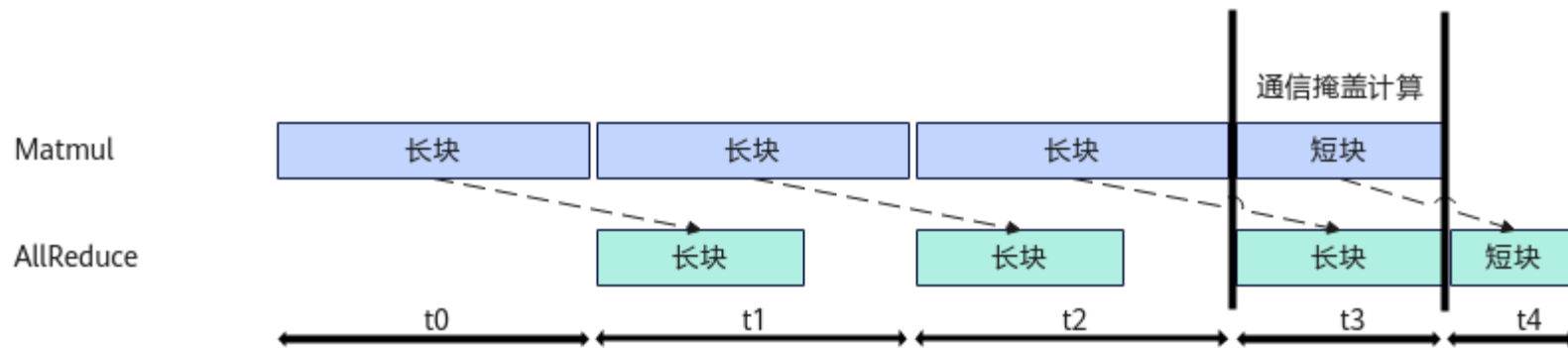
- 当K轴较大时，计算量大，计算执行时间大于通信执行时间，此时计算为算子的瓶颈（bound）；
- 反之，当K轴较小时，计算量小，计算执行时间小于通信执行时间，此时通信为算子的瓶颈。

2. 设定数据切分策略。

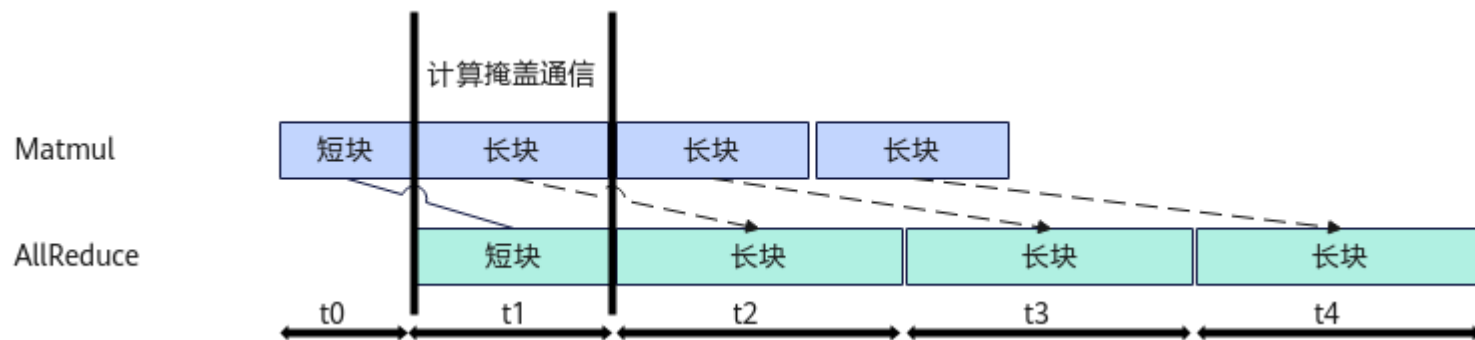
- 只切分M轴(行)。按M轴切分，则每行数据都是内存连续的，满足通信要求；若按N轴（列）切分，则每行数据都被切断，导致通信数据的内存不连续，不满足通信要求。
- 若A表示长块、B表示短块，只能切出A或B连续排布的形式，方便任务做掩盖。例如AAAB、BAAA等情况。

数据切分目标：尽可能多的流水掩盖

计算bound：对于同一个切分后的数据块，**计算执行耗时大于通信执行耗时**。此时，计算连续，且通信的尾块要短，如下图所示。



通信bound：对于同一个切分后的数据块，**通信执行耗时大于计算执行耗时**，此时，通信连续，且计算的头快要短，如下图所示。



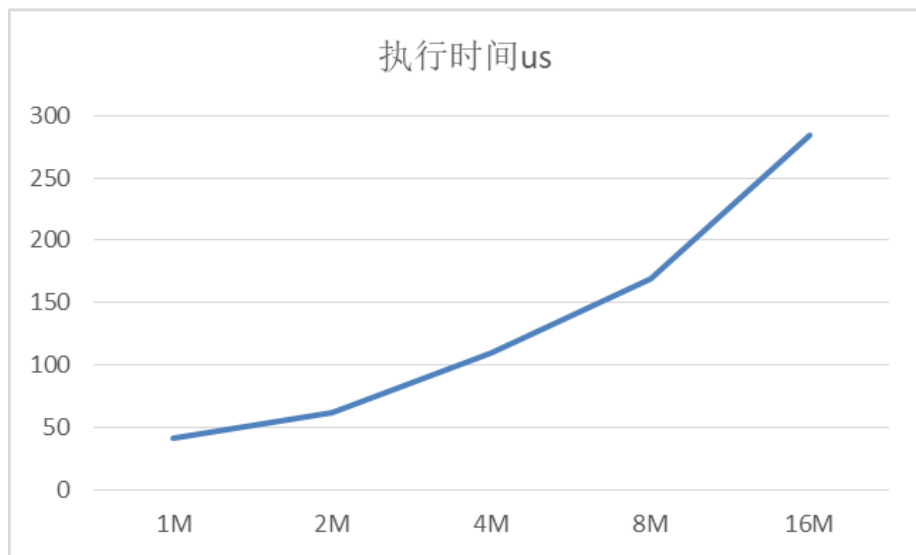
■ MC²优化方案：前置分析

前置分析工作有：

- **确定bound场景**。将输入数据分别单独执行Matmul计算和AllReduce通信任务，利用msProf工具（性能分析工具）分别采集执行时间，判定时间较长的任务为对应的bound场景。
- **对Matmul计算和AllReduce通信的数据量与执行时间关系做公式拟合**。将输入数据按M轴切分，分别切成M轴为256、512、768、1024、2048的若干数据块，这里可以根据实际情况调整切块大小。得到的数据块分别做AllReduce通信，利用msProf工具采集执行时间，得到每个数据块的执行时间 t_1, t_2, \dots, t_n 。

MC²优化方案：前置分析

作图分析数据量与对应的执行时间的关系，拟合得到公式 $t = \text{CostComm}(m)$ ，其中 m 表示数据块的M轴长度， t 表示数据块的通信执行时间。该映射关系一般为线性，若不满足线性关系，可以采用分段拟合的方式。示例如下：



m值	数据量	执行时间us
1	1M	42
2	2M	62.1
4	4M	109.8
8	8M	168.8
16	16M	284.8

数据量 $x = m * N * \text{sizeof}(\text{dataType})$ ，单位是Bytes。该拟合公式表示为：

- x 小于8MB: $t = -0.9698202 * x * x + 27.0622573 * x + 14.769$ ，单位是us。
- x 大于等于8MB: $t = 13.58491263 * x + 61.508333$ ，单位是us。

MC²优化方案：方案验证

本MatmulAllreduce案例中，给定的输入矩阵Shape为M=4096，K=3072，N=8192，数据类型为half，分核数为8，通过融合前msProf工具采集，得到该输入的Matmul计算执行时间为803us，AllReduce通信执行时间为1071us，总耗时1874us，属于通信bound场景。按照上述的切分算法，本案例的具体切分情况如下：

1. 根据经验值选定断块（bound场景中断块为头块，即切分的第一个数据块）M方向长度m0为384，则通信数据量x为： $384 * 8192 * 2 / 1024 / 1024 = 6\text{MB}$ 。按通信拟合公式估算通信的执行时间为143us。考虑可能会发生内存带宽冲突，因此再乘以1.15的系数，得出通信执行时间164us。

短块经验值公式，a、b、c表示根据经验值给出的短块长度的备选。

- $a * K * N \geq 4 * 1024 * 1024 * 1024$ ，a取不等式的最小值；
- $b * K * N / 1024 + b * N \geq 6 * 1024 * 1024$ ，b取不等式的最小值；
- $c \geq 3 * 128$ ，c取不等式的最小值
- $m0 = \min(a, b, c)$

MC²优化方案：方案验证

2. 计算M方向长块长度m1。根据短块通信时间，配平长块的计算执行时间同样为164us，即将t1作为长块的计算执行时间，带入 $t1 = \text{CostMM}(m1)$ 公式，按计算拟合公式估算出该长度m1为768。
3. 根据M=4096、m0=384、m1=768，计算长块个数： $(4096 - 384) / 768 = 4.83$ ，向下取整为4。
4. 根据短块长度m0=384，长块个数4，调整长块m1长度： $(4096 - 384) / 4 = 928$ ，向下按128对齐，调整m1为896。
5. 根据长块长度m1=896，长块个数4，调整短块m0长度： $4096 - 896 * 4 = 512$ 。

最终得到将原始输入矩阵切分为5个数据块，长度分别为： $\{512, 896, 896, 896, 896\}$ 。

效果：在算子的Tiling代码中设置制定好的切分策略。按该切分策略测试，融合后该算子的执行时间为1262us，则融合算子的性能收益为 $(1874 - 1262) / 1874 = 32.7\%$ 。

思考总结



◆ 是否能做计算与通信并行。

MC2算子通过数据切分后计算和通信的并行执行，获得性能收益，但受数据切分后执行时间膨胀的影响。



◆ 确定理论切分策略。

对MC2算子进行性能调优的主要方式是制定数据切分策略，开发人员需要根据理论推导找到理想切分策略，然后根据实测结果调整，最终找到最优切分策略。



◆ 动态调整切分策略。

切分数据会引起计算或通信执行时间的膨胀，使实测结果与理论值有偏差。比如切块数量较多时，执行时间的膨胀对性能影响较大，可能导致性能收益变小或者出现性能劣化，因此最终需要根据上述理论切分策略，结合实测，对切分策略做调整。

04 总结与展望

总结

- 大模型推理加速是一个系统工程，涉及算子、算法、框架、资源调度、底层芯片等全栈综合能力，需要不断探索与实践。
- 大模型推理加速涉的技术，核心是提升硬件资源利用率、减少计算量、减少通信开销（分布式），最终提高端到端的性能，降低推理成本。

展望

1. **专用硬件加速器的发展**：软硬协同设计方法。例如，存储单元更加接近计算单元的存算一体芯片、计算型存储盘，针对LLM算法数据流优化芯片架构等。
2. **长上下文/序列场景优化**：随着应用场景变得更加复杂，处理更长的上下文或序列的需求不断增长。服务长序列负载的LLM需要解决算法和系统两方面的挑战。在算法方面，依然面临长度泛化失效问题。目前的解法主要是通过召回增强、序列压缩和缓存来尽可能缩短序列长度并保存相关信息。

📍 北京

QCon

全球软件开发大会

会议时间：4月10-12日

- 大模型赋能 AIOps
- 越挫越勇的大前端
- 云上业务架构演进
- 多模态大模型及应用
- 海外 AI 应用创新实践

📍 北京

AiCon

全球人工智能开发与应用大会

会议时间：6月27-28日

- 端侧智能
- AI Agent
- 多模态大模型
- 金融+大模型

📍 上海

QCon

全球软件开发大会

会议时间：10月23-25日

- AI Agent
- 大模型训练推理
- 端侧 AI
- 搜推深度融合
- 数智金融

4月

5月

6月

8月

10月

12月

📍 上海

AiCon

全球人工智能开发与应用大会

会议时间：5月23-24日

- 通用大模型
- AI Agent
- 垂直领域应用
- 模型可解释性

📍 深圳

AiCon

全球人工智能开发与应用大会

会议时间：8月22-23日

- 模型效率与部署
- 多模态大模型
- 大模型安全
- 智能硬件

📍 北京

AiCon

全球人工智能开发与应用大会

会议时间：12月19-20日

- 通用大模型
- 智能硬件
- LMOps
- 具身智能

极客邦科技 2025 年会议规划

促进软件开发及相关领域知识与创新的传播



参会咨询



查看会议

THANKS

探索 AI 应用边界

Explore the limits of AI applications

AiCon
全球人工智能开发与应用大会