**Abstract**

1. Compute-intensive Operators Fusion 在NN中出现次数很多，但是现在fusion做的不好（存在功能性约束，和有的shape下性能不如unfused kernel）

2. Hopper Arch 作为新的GPU arch，带来了新的hardware feature，包括不限于XX/XX/XX（你用上了什么提什么），这些feature目前被现有的自动优化和编译框架所忽略

随着硬件的专门化，计算核心的增幅远高于内存，一些计算密集型的操作也受到内存延迟的影响，再结合上其前后的其他内存密集型操作，多个计算密集型算子之间的融合内核变的受关注起来。Cutlass-b2b和flash attention是两种GEMM融合方案，但是都严格限制了GEMM的尺寸。近些年来的新趋势：芯片互联技术，可以有效缓解对GEMM尺寸的限制。例如hopper的DSMEM，允许每个核心直接访问其他核心的scratch pad。然而当前的DL编译器和自动优化代码库没有能很好的利用这个性能。

在本文中，我们提出不同的计算密集型算子的融合策略XXX，一个利用计算核心间互联特性，并针对不同场景选择合适融合策略的优化框架。提出了分布式数据流动抽象loc-dim-exchange(LDE)，LDE通过将每个block的任务位置，发生数据交换的维度映射到每个核心上，实现了核间通信开销，片上内存开销，计算通信重叠，多操作符可融合性的多因素权衡。

**Introduction**

典型例子就是flash attention。单独看矩阵乘法是计算受限的，而其特殊的尺寸和其中的softmax操作，使得GEMM的融合变得相当有价值。然而这种实现却没有被推广到更多GEMM的场景下，如QKV-生成，attention后的concat，以及FFN。针对两个GEMM的融合而言，cutlass的b2b方案约束MNKT的N和T都要小于256，而FA则是限制T要小于256。这使得很多场景无法实现GEMM融合。之前没有实现的主要困难在于不同block间的数据交换只能通过global memory，而DSM的出现能够很好的缓解这个问题。此外，Block可以互通的场景也进一步扩展了WASP的概念，以往的WASP是不同warp不同的工作，而当不同block做不同工作时，数据只能通过global交换，这限制了可能的收益。

**Design**

* 四个权衡是什么，怎样选择？

核间通信开销（如果cluster很大，开销很大。），片上内存消耗（使用核间通信则需要消耗DSM，否则就是走global，但是occupancy可以提高。或者不占用DSM，而直接用寄存器的ld指令移动，速度慢一些，但是可以提高occupancy），计算通信重叠（pipeline的重叠更好，因为可以选择GEMM0和GEMM1分配的CTA数量。有可能更细粒度的调整。策略1在传输时没法计算，好吧也可以用tensorcore做累加，然后边传边算，策略2在传输的时候可以选择同时计算。），中间结果交换方式是否有利于后续计算（stationary的不同。）

* LDE是什么，三种策略下怎么表示？

数据依赖要考虑不同维度。比如在GEMM0的M维度上交换，那就是上下两个矩阵交换B矩阵的内容。在GEMM1的N维度交换，那就是不同block间做reduce。这种数据交换的抽象表达，有助于统一表示不同的融合策略。GEMM0的N维度和GEMM1的N维度相关联。（或者叫GEMM1的K维度），如果在GEMM0的N维度上发生数据交换，则GEMM1的N维度也要发生交换。MNK中，K是累加维，而MN是平等的。对于grid特别小，可以用DSM-reduce来做K的block交互。T10的场景就仅考虑M或者N。而我要考虑的就加入了GEMM0-N-GEMM1。我觉得应该以block负责的位置来表示。结合上哪个维度的数据交换。

比如策略1就是，cluster负责GEMM0的一行，和GEMM1的一块，然后MNKT中N发生了数据交换。（也可以增加M）。但是考虑到我们不希望在GEMM1的一行里安排多个block，所以T维度就不发生数据交换。N维度在GEMM0计算的部分也有数据交换。（对B矩阵的读取上）----->考虑到这个，那么表示就不能混合为MNKT。应该写成：

GEMM0: MNK，M可以有，N有，K无。一个cluster算一行

GEMM1:MTN，M可以有，T无，N有。一个cluster算一块

策略2就是，cluster负责GEMM0的一行，和GEMM1的一行。然后MNKT中N发生了数据交换，T也发生了数据交换。

GEMM0: MNK，M可以有，N有，K无。一个cluster算一行

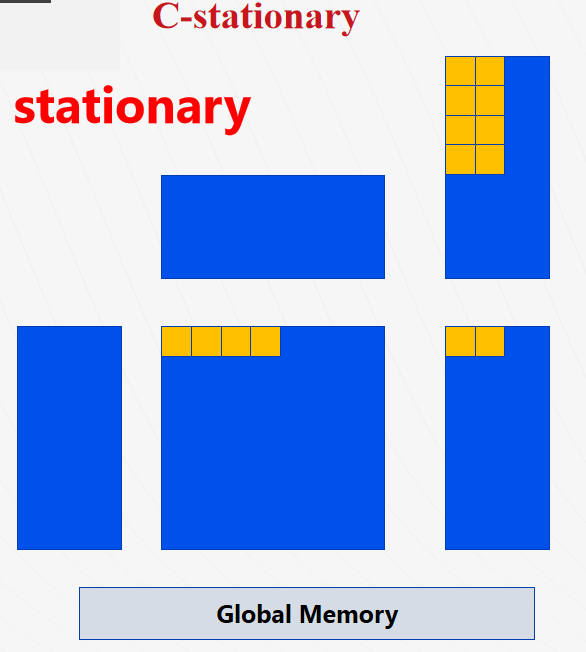
GEMM1:MTN，M可以有，T有，N无。一个cluster算一行

策略3就是，cluster部分负责GEMM0的一行，部分GEMM1的一行。如果GEMM1分配了多个block，也可以在T上发生数据交换。

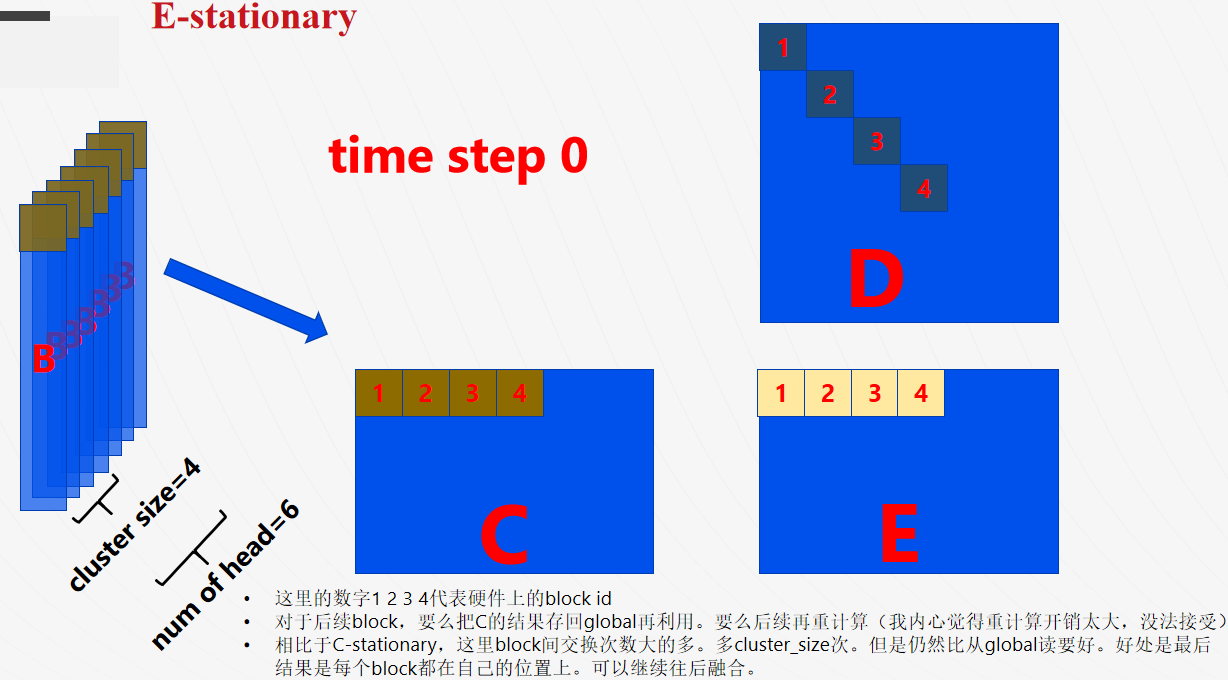
GEMM0: MNK，M可以有，N有，K无。一个cluster的部分算一行

GEMM1:MTN，M可以有，T可以有，N有。一个cluster的另一部分算一行

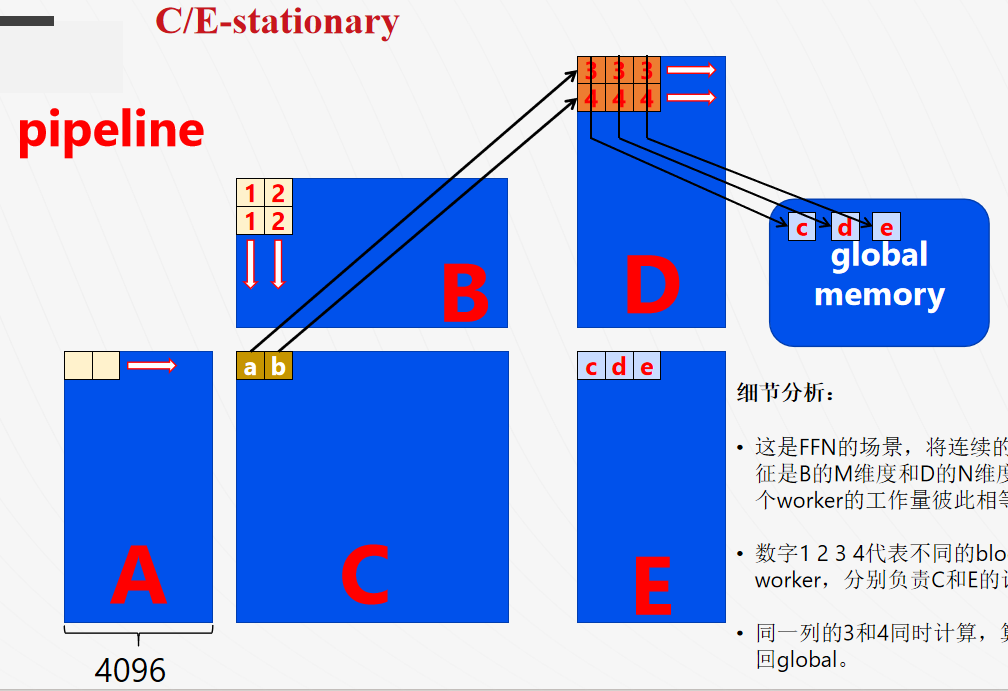
* 三种融合的策略



策略1，一个cluster先负责C矩阵的一行，然后借助本地的SMEM的C的结果，计算E的一个块。多个block计算完结果后执行reduce，然后存回global。接着再利用相同的C的结果，计算下一个E的块。这样的好处是能够充分利用C的中间结果，但是对E的后续的列需要串行，（反之则是不同block原子累加，更慢）。这使得其仅在C的block数目足以填满SM的情况下是较好的策略。此外，计算结束后，每个block并不持有E矩阵的一整行的不同block，这对后续融合造成了困难。注意，计算C的时候，存在N维度的数据交换。



策略2，一个cluster先负责C的一行，然后计算E的一行。每次计算都需要对block间交换C的值。此时数据交换发生在T维度，而N维度上不再需要跨block累加，直接在本block内就足够了。这种计算方式在每轮计算时都要交换所有C的值。虽然最后计算得到的E也更多，不过这可能会引入更大的cluster.sync的开销。但是这种方式有利于后续进一步的融合，因为每个block都持有自己unique的值。



策略3，一个cluster的一部分block先算C的一行，然后另一部分block去计算E，如上图，可以给C分配2个block，给E分配6个block，这6个block既需要左右交换来自C的值，也需要上下做reduce来加回global。那么对GEMM1就存在T维度，N维度的数据交换。对GEMM0则是有N维度的交换。这样的好处是能调整C和E的block的数目，能更精细的调整数据传输和计算的重叠，类似与WASP，不过扩展了概念到block间。要是以后能够支持不同block的寄存器/SMEM/线程资源不同就更好了。

所以上述讨论论证了，我们可以通过确定两个GEMM的各个维度是否有数据交换，以及cluster负责执行哪部分任务来确定具体的算法。或者说cluster具体执行哪部分任务也是根据数据交换与否来相应确定下来的。因为我们知道，在cluster的情况下，N和K至少有一个维度需要有数据交换。（所以上述三种就表示完全了所有可能性。）

**Comparison**

和T10相比，它关注的是单个算子内数据是如何旋转交换的。但是这一点也可以用寄存器直接ld，或者包括TMA-reduce/multicast也可以执行ring通信所需的加速（还没人测过这三种哪个更快）。本文关注的点是多算子场景下的数据交换需求，这自然会需要用到单个维度下数据如何交换，可以用T10的方法，不过也可以用刚才提的其他方法。

我的优化是基于GEMM的，这样就有MNKL四个维度。子汉他们是GEMV，这样就没有M维度。我可以不强调专门针对于大模型，我甚至可以做做conv，splitK。子汉他们可以更针对推理场景一些。