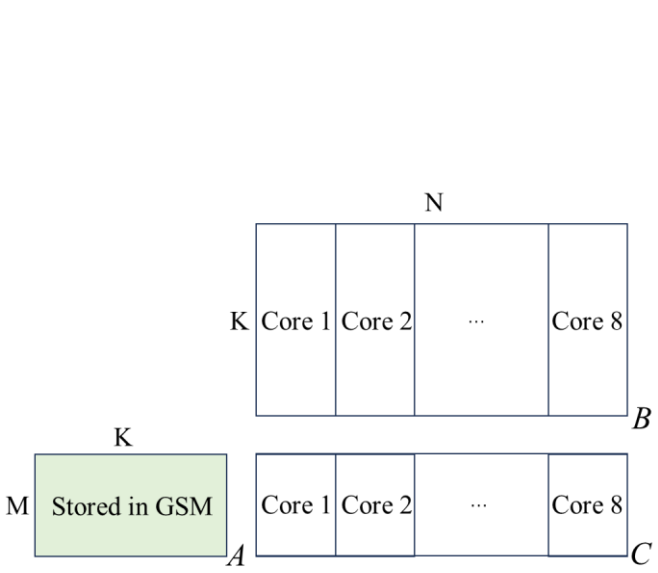
不规则矩阵乘算法设计和性能优化

1. 研究目标

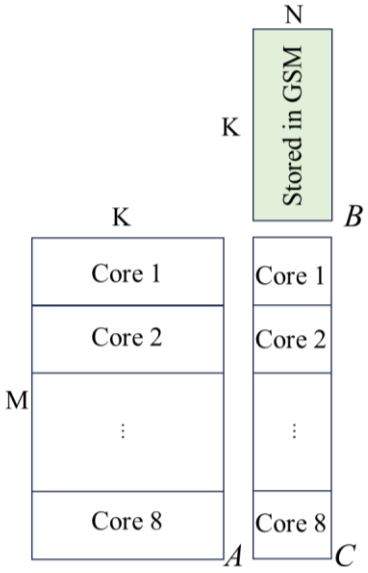
为了解决常规 GEMM 在处理不规则矩阵乘时暴露的性能瓶颈，将在现有 GEMM 方法的基础之上，通过算法的改进和分块策略的调整解决不规则矩阵乘的访存受限问题，实现更好的多核扩展性。

本研究面向的不规则矩阵乘分为以下两类，分别是 A 矩阵很矮的情况(shortA 类型，M<200)和 B 矩阵很瘦的情况(thinB 类型，N<200)。面向 MT3000 的 GEMM 算法通常采用一维核间分块方式。对于 M 很小的情况，应当沿着 N 维进行核间分块，矩阵 A 因此被多核共享，会被读入 GSM。对于 N 很小的情况，应当沿着 M 维进行核间分块，矩阵 B 因此被多核共享，会被读入 GSM。

由于计算规则 GEMM 的方法采用 N 维核间分块，这一点上与 shortA 情况一致，因此我们首先针对 shortA 情况的不规则矩阵乘进行分析和优化。



shortA 情况



thinB 情况

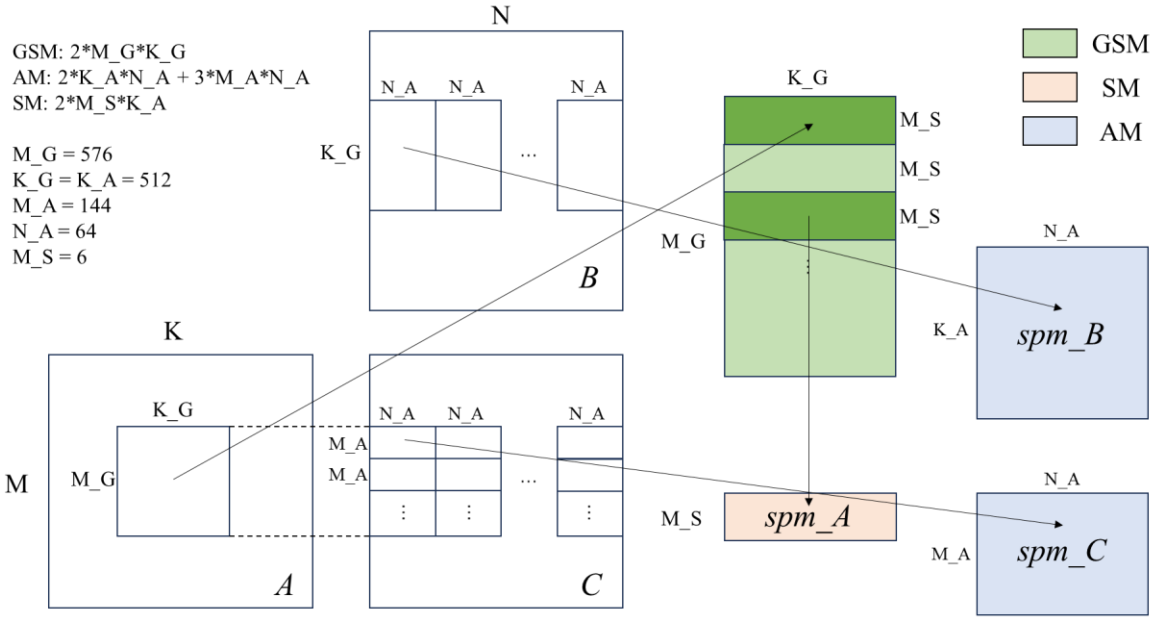
2. 研究过程

2.1 已有的 GEMM 算法

首先对现有 GEMM 方法做一个回顾。该方法沿 N 维度进行核间分块，每个核分得属于 B、C 矩阵的不同数据。A 矩阵的数据则被每个核共享，因此会从 DDR 预先加载到 GSM，再分发到每个核所属的片上缓存。单核的外层算法如下图所示，采用四级数据预取来重叠微内核的计算开销，每一层循环分别对应一级访存操作。最外层遍历的是 A 矩阵的分块，将 M_G×K_G 大小的数据块预取到 GSM；然后遍历 N 维度，将 K_A×N_A 大小的 B 分块从 DDR 预取到 AM；再往内是遍历 M_G 维度，将大小为 M_A×K_A 的 C 分块从 DDR 预取到 AM；最内层是进一步遍历 M_A 维度，将 M_S×K_A 大小的数据块从 GSM 预取到 SM，然后调用微内核计算。

```
for m_o = 0:M_G:M
  for k_o = 0:K_G:K
    load A[M_G×K_G] from DDR to GSM (double buffering)
    for n_i = 0:N_A:N
      load B[K_A×N_A] from DDR to AM (double buffering)
      for m_i = 0:M_A:M_G
        load C[M_A×N_A] from DDR to AM (triple buffering)
        for m_s = 0:M_S:M_A
          load A[M_S×K_A] from GSM to SM (double buffering)
          micro_kernel(sm_A, am_B, am_C);
        end m_s
        store C[M_A×N_A] from AM to DDR
      end m_i
    end n_i
  end k_o
end m_o
```

下图是 GEMM 算法对应的分块策略示意图，给出了分块的嵌套关系，数据块的大小，数据的流向，片上缓存的用途以及被占用的缓存大小。



经过分析，DDR 总访存数据量为：

$$Access_{DDR} = \frac{M \times K}{M\_G \times K\_G} \times \left(M\_G \times K\_G + N \times (K\_G + 2 \times M\_G)\right)$$

$$= M \times K + M \times K \times N \times \left(\frac{1}{M\_G} + \frac{2}{K\_G}\right)$$

可见 DDR 总访存数据量与第一层分块大小(A[M_G×K_G])直接相关。另外值得注意的是，现有算法在第一层分块之后虽然对 M_G 进一步做了步长为 M_A 的再

分块，但是没有对 K_G 进行再分块(即 K_G=K_A)，这使得一个 B[K_A×N_A]分块得以被若干个 C[M_A×N_A]分块重用，不需要重复加载，因此 K_A 与 M_A 两个参数并不影响 DDR 总访存数据量。

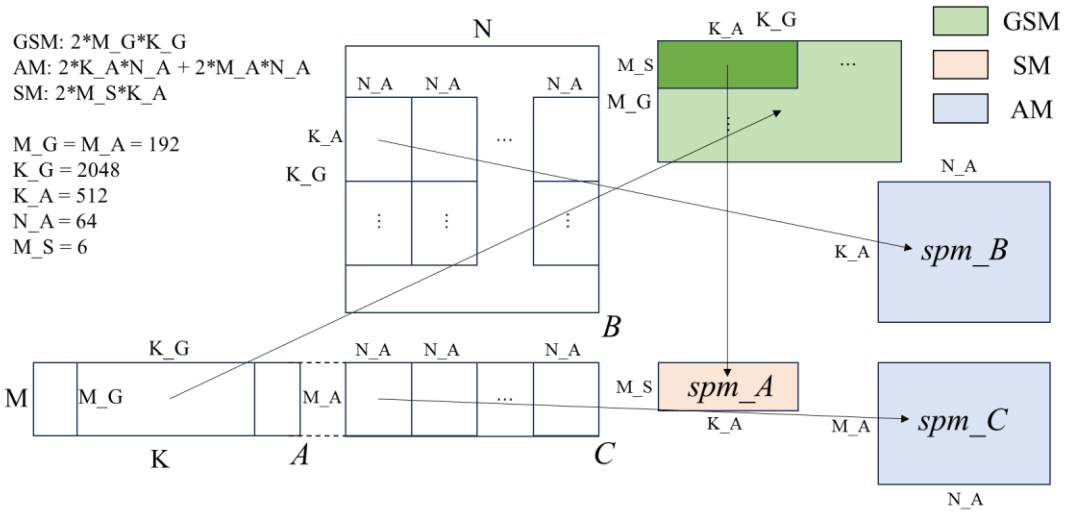
2.2 用于 shortA 情况的 GEMM 算法

M 很小的情况下，第一层分块(A[M_G×K_G])将从二维遍历退化为 K 方向的一维遍历，并且分块高度 M_G 也会被压缩至 200 以内，若沿用现有算法将显著增加 DDR 总访存数据量。为此，一个可行的解决方案是调整 K_G 的大小并加入一层新的循环。既然 M_G 的减小降低了 GSM 缓冲区的利用率，那么正好可以利用空闲出的 GSM 空间，增大 K_G，从而降低 DDR 总访存数据量。受限于 AM 的存储空间，K_A 并不能与 K_G 一同变大，二者不再相等，因此需要引入新的循环，以 K_A 为步长遍历 K_G。还可注意到，M_G 被压缩后与 M_A 相差无几(<200)，为了在引入 K_G-K_A 循环后不失去数据重用性，可以合并 M_G 与 M_A，并把对 N_A 的遍历移到 K_A 的遍历之前。这样一来，每个 C[M_A×N_A]将只需要被加载一次，供若干个 B[K_A×N_A]复用，DDR 总访存数据量同样不会受到 K_A 与 M_A 的影响。

改进后的算法如下图所示，可以总结为改变了 A[M_G×K_G]分块的长宽比，引入了一层新循环，拿掉了一层旧循环，并调整了循环的层次。

```
for m_o = 0:M_G:M
  for k_o = 0:K_G:K
    load A[M_G×K_G] from DDR to GSM (double buffering)
    for n_i = 0:N_A:N
      load C[M_A×N_A] from DDR to AM (double buffering)
      for k_i = 0:K_A:K_G
        load B[K_A×N_A] from DDR to AM (double buffering)
        for m_s = 0:M_S:M_A
          load A[M_S×K_A] from GSM to SM (double buffering)
          micro_kernel(sm_A, am_B, am_C)
        end m_s
      end k_i
      store C[M_A×N_A] from AM to DDR
    end n_i
  end k_o
end m_o
```

下图是改进后算法对应的分块策略示意图，给出了分块的嵌套关系，数据块的大小，数据的流向，片上缓存的用途以及被占用的缓存大小。



对改进前后的算子分别进行性能测试，输入矩阵的形状取作 M=144, K=4096, N=12288，沿 N 维进行核间分块，使用 8 个核并行计算，结果如下，可以看出改进后的算法性能有大幅提升。

方法	平均单核性能	计算峰值百分比
输入数据：M=144, K=4096, N=12288　8 核并行		
已有的 GEMM (M_G=144, 三缓冲)	98 GFlops	56.7% peak
改进后的 GEMM (M_G=144, 三缓冲)	140 GFlops	81.0% peak
改进后的 GEMM (M_G=144, 双缓冲)	141 GFlops	81.6% peak
输入数据：M=192, K=4096, N=12288　8 核并行		
改进后的 GEMM (M_G=192, 双缓冲)	158 GFlops	91.4% peak

另一点值得注意的是，已有 GEMM 方法为 C[M_A×N_A]分块开辟了 3 个缓冲区，用于同时进行 load、compute 和 store 操作，对于计算密集型问题有助于充分隐藏访存开销。但由于并行不规则矩阵乘在 MT3000 平台上通常为访存受限，总访存开销大于计算开销，无法再以重叠的方式完全实现隐藏，因此三缓冲策略有失效的可能。经过测试，果然，相比双缓冲方法，使用三缓冲并不能让改进后的 GEMM 算子有更好的性能。而换回双缓冲的另一个好处是，每个缓冲区的容量可以设得更大，体现在参数上，M_G 可以取更大的值，从而扩大了输入形状的支持范围(从 M<=144 变成 M<=192)。顺便，这里也测试了 M=192 时算子的性能，发现超过了峰值性能的 90%，这是因为 M_G 的增大显著改善了计算访存比。

为了支持 shortA 情况下的不同转置模式，可以效仿常规 GEMM 方法在不同的缓存层次对数据块进行转置操作。在片上缓存中实现转置可以消除额外的 DDR 访存开销，使转置操作退化为与微内核类似的计算环节。在访存受限的算子中，这对总体性能影响甚微。

2.3 处理 thinB 情况的一次失败尝试

在 N 维度很短的情况下，核间分块方式需要相应改变，从沿着 N 维度切分改为沿着更长的 M 维度切分。于是，每个核将分得 A、C 矩阵的不同部分，并共享 B 矩阵中的相同数据。在决定将 B 分块暂存在 GSM 的同时，还需考虑到，由于数据布局没有改变(都是行主序)，为了利用长向量的特点，微内核仍需采用之前的设计方案，默认 A 分块位于 SM，B、C 分块位于 AM，基于外积算法逐个广播 A 中的标量，与 B 中的行向量相乘，结果累加给 C 中对应位置的行向量。由于 SM 的存储空间有限，在双缓冲策略下一次只能加载 6×512 大小的数据，如果每次只从 DDR 加载如此少量的数据到片上，那么延迟将挤占相当一部分访存时长，导致真正留给数据传输的时间所剩无几，从而极大地降低了带宽利用率。

为了降低延迟带来的总开销，应该单次从 DDR 加载尽量多的数据。一个可行的做法是先从 A 矩阵的内存中加载较大的分块到 GSM，再将数据以更小的分块分多次加载到 SM。由于每个核读取 A 矩阵的不同部分，所以这种场景下 GSM 将失去多核共享的作用，而仅仅被用作数据从 DDR 向片上传输的集散点。

综上，相比已有的方法，这种针对 thinB 情况的算法采用相同的微内核，不同在于 GSM 既用于暂存核间共享的 B 矩阵分块，也用于暂存核间不共享的 A 矩阵分块。因此在使用双缓冲的情况下，GSM 需要具有以下大小的存储空间，

$$buffer_{GSM} = 2 \times (K\_G \times N\_G) + 2 \times (M\_A \times K\_A) \times n\_cores$$

这种算法的一个弊端在于，当并行核数较多，例如$n_cores = 8$时，A 分块将占据 GSM 的大部分存储空间，因此 B 分块的尺寸(K_G 和 N_G)将因此受到限制，从而导致访存开销的进一步增加。类似于 shortA 情况，可以计算 thinB 情况下 DDR 总访存数据量为

$$\begin{aligned} Access_{DDR} &= \frac{N \times K}{N\_G \times K\_G} \times \left(N\_G \times K\_G + M \times (K\_G + 2 \times N\_G)\right) \\ &= N \times K + N \times K \times M \times \left(\frac{1}{N\_G} + \frac{2}{K\_G}\right) \end{aligned}$$

基于这种算法可以得到三种具体的实现方案。

方案一设定 K_G=K_A, N_G=N_A，因为数据从 GSM 转移到 AM 的过程中没有在 K,N 两个维度上继续分块，所以外层循环只有 4 级数据预取。这一方案的缺点在于，AM 的存储空间限制了 K_A,N_A 两个分块参数的大小，也就导致无法进一步增大 K_G,N_G 来降低 DDR 总访存数据量。

方案二相比于方案一取消了 N_G=N_A 的约束，使 N_G 可以取得更大，从而降低了 DDR 总访存数据量，也增加了一层新的循环。

方案三相比于方案一取消了 K_G=K_A 的约束，使 K_G 可以更大，并同样新增了一层循环。

三种方案的算法流程和缓存分配情况如下。

```
GSM: 2*(K_G*N_G) + 2*(M_A*K_A)*n_cores
AM: K_A*N_A + 3*(M_C*N_A)
SM: 2*(M_S*K_A)

K_G = K_A = 512
N_G = N_A = 64
M_C = 288, M_A = 72, M_S = 6

for k_o = 0:K_G:K
  for n_o = 0:N_G:N
    load B[K_G×N_G] from DDR to GSM (double buffer)
    load B[K_A×N_A] from GSM to AM
    for m_c = 0:M_C:M
      load C[M_C×N_A] from DDR to AM (triple buffer)
      for m_a = 0:M_A:M_C
        load A[M_A×K_A] from DDR to GSM (double buffer)
        for m_i = 0:M_S:M_A
          load A[M_A×K_A] from GSM to SM (double buffer)
          micro_kernel(sm_A, am_B, am_C)
        end m_i
      end m_a
      store C[M_C*N_A] from AM to DDR
    end m_c
  end n_o
end k_o
```

方案一

```
GSM: 2*(K_G*N_G) + 2*(M_A*K_A)*n_cores
AM: 2*(K_A*N_A) + 3*(M_C*N_A)
SM: 2*(M_S*K_A)

K_G = K_A = 512
N_G = 192, N_A = 64
M_A = 72, M_C = 72, M_S = 6

for k_o = 0:K_G:K
  for n_o = 0:N_G:N
    load B[K_G×N_G] from DDR to GSM (double buffer)
    for m_a = 0:M_A:M
      load A[M_A×K_A] from DDR to GSM (double buffer)
      for n_i = 0:N_A:N_G
        load B[K_A×N_A] from GSM to AM (double buffer)
        for m_c = 0:M_C:M_A
          load C[M_C×N_A] from DDR to AM (triple buffer)
          for m_i = 0:M_S:M_A
            load A[M_S×K_A] from GSM to SM (double buffer)
            micro_kernel(sm_A, am_B, am_C)
          end m_i
          store C[M_C×N_A] from AM to DDR
        end m_c
      end n_i
    end m_a
  end n_o
end k_o
```

方案二

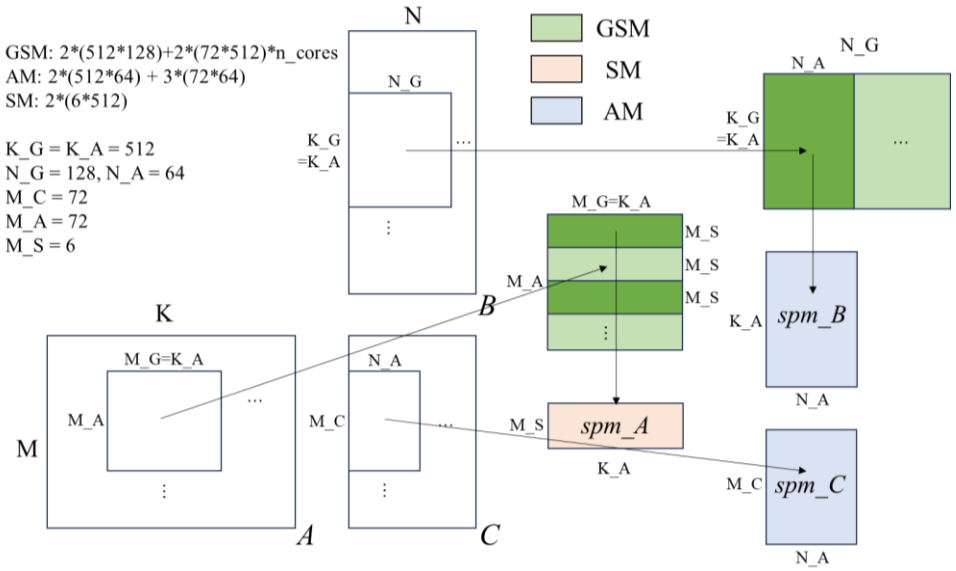
```
GSM: 2*(K_G*N_G) + 2*(M_A*K_A)*n_cores
AM: 2*(K_A*N_A) + 3*(M_C*N_A)
SM: 2*(M_S*K_A)

K_G = 1024, K_A = 512
N_G = N_A = 64
M_C = 144, M_A = 72, M_S = 6

for k_o = 0:K_G:K
  for n_o = 0:N_G:N
    load B[K_G×N_G] from DDR to GSM (double buffer)
    for m_c = 0:M_C:M
      load C[M_C×N_A] from DDR to AM (triple buffer)
      for k_i = 0:K_A:K_G
        load B[K_A×N_A] from GSM to AM (double buffer)
        for m_a = 0:M_A:M_C
          load A[M_A×K_A] from DDR to GSM (double buffer)
          for m_i = 0:M_S:M_A
            load A[M_S×K_A] from GSM to SM (double buffer)
            micro_kernel(sm_A, am_B, am_C)
          end m_i
        end m_a
      end k_a
      store C[M_C*N_G] from AM to DDR
    end m_c
  end n_o
end k_o
```

方案三

下图是方案二对应的分块示意图，给出了嵌套关系，数据块的大小和数据流向，方案一与方案三同理。



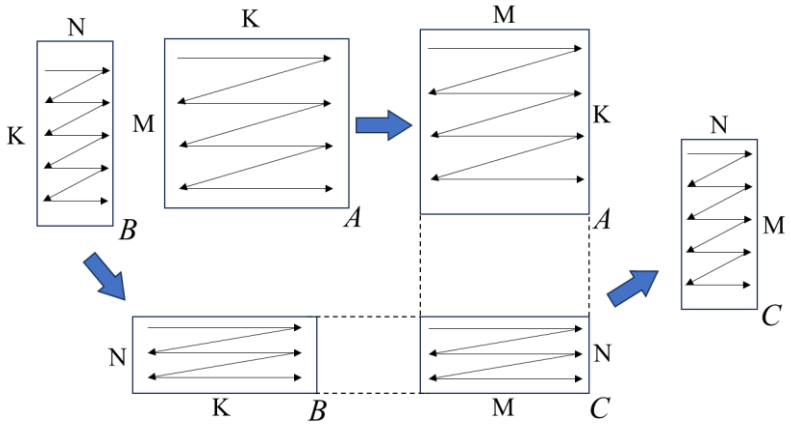
对方案 1-3 分别进行性能测试，输入矩阵的形状取作 M=13824， K=4096，N=192，沿 M 维进行核间分块，使用 8 个核并行计算，结果如下，可以看出即使是性能相对最好的方案二，也仅能达到峰值的 60%。

方法	平均单核性能	计算峰值百分比
输入数据：M= 13824， K=4096， N=192　　8 核并行		
方案一 (K_G=512， N_G=64)	61 GFlops	35.3% peak
方案二 (K_G=512， N_G=192)	104 GFlops	60.2% peak
方案三 (K_G=1024， N_G=64)	66 GFlops	38.1% peak

2.4 用 shortA 的方法处理 thinB 情况

鉴于直接计算 thinB 情况效果不理想，一个可行的替代方案是将 thinB 情况转化为 shortA 情况，然后沿用 shortA 的方法进行计算。

thinB 和 shortA 的共性是都有一个很短的维度，不同在于 shortA 情况下数据的存储方向与该维度垂直，而 thinB 情况下数据的存储方向与该维度平行。可以通过转置操作来消除这一差异，即先对输入矩阵 A,B 的分块进行转置，接着使用 shortA 的算法进行计算，最后再将计算结果 C 转置回行主序布局，如下图所示。尽管这种方法需要转置全部 3 个矩阵的分块，但由于操作是在片上完成的，不会引入额外的 DDR 访存开销，因此有理由猜测其对整体性能影响不大。



参考 shortA 的算法，并在对应的位置插入转置操作，得到适用于 thinB 情况的算法，完整流程以及分块参数的取值如下图所示。

GSM: 2*N_G*K_G
AM: 2*K_A*M_A + 2*N_A*M_A
SM: 2*N_S*K_A

N_G = N_A = 192
K_G = 2048, K_A = 512
M_A = 64
N_S = 6

```
for n_o = 0:N_G:N
  for k_o = 0:K_G:K
    load B[K_G×N_G] from DDR to GSM (double buffering)
    transpose B[K_G×N_G] to B[N_G×K_G]
    for m_i = 0:M_A:M
      load C[M_A×N_A] from DDR to AM (double buffering)
      for k_i = 0:K_A:K_G
        load A[M_A×K_A] from DDR to AM (double buffering)
        transpose A[M_A×K_A] to A[K_A×M_A]
        for n_s = 0:N_S:N_A
          load B[N_S×K_A] from GSM to SM (double buffering)
          micro_kernel(sm_B, am_A, am_C)
        end m_s
      end k_i
      C[M_A×N_A] += tranpose(am_C)
      store C[M_A×N_A] from AM to DDR
    end m_i
  end k_o
end n_o
```

对这种方法进行性能实测，输入矩阵的形状取作 M=12288, K=4096, N=192，沿 M 维进行核间分块，使用 8 个核并行计算，结果如下。可以看出多次转置操作果然对整体性能影响不大，性能远在前一种方法之上。

方法	平均单核性能	计算峰值百分比
输入数据：M= 13824, K=4096, N=192　8 核并行		
直接计算 thinB(方案二) (K_G=512, N_G=192)	104 GFlops	60.2% peak
输入数据：M= 12288, K=4096, N=192　8 核并行		
转置+shortA+转置 (K_G=2048, N_G=192)	147 GFlops	85.1% peak

3. 研究结果

总结以上过程。适用于 shortA 情况的算法改进自原有的 GEMM，通过调整循环嵌套的数量和层次改变了分块的长宽比，降低了 DDR 总访存数据量，取得了较好的多核并行性能。对于 thinB 情况，直接计算的方法为了单次从 DDR 加载尽量多的数据，不得不先将 A 分块暂存到 GSM，压缩了共享数据的可用空间，因此没能化解访存瓶颈。相比之下，经过转置操作转化为 shortA 情况求解的方法虽然引入了转置开销，但在访存瓶颈的主导下对整体性能影响较小，取得了与 shortA 情况非常接近的性能。

为了支持不同的转置模式，可以将转置操作融合进相应的缓存层次和程序位置，既不引入额外的访存开销，也不打断原有的数据预取流程。经过测试，shortA 情况下不同转置模式的性能如下表所示。

shorA 情况　输入数据：M=192, K=4096, N=12288　8 核并行			
转置模式	实际需要转置的分块	平均单核性能	计算峰值百分比
NN	无	158 GFlops	91.4%
NT	B	149 GFlops	86.2%
TN	A	157 GFlops	90.9%
TT	A,B	148 GFlops	85.6%

thinB 情况下不同转置模式的性能如下表所示。在 NN 模式中，为了将问题转化为 shortA 情况，每个矩阵都要经过转置，因此性能相对最低。而在非 NN 模式下，却恰好可以消除某项转置开销，因此性能有所上升。

thinB 情况　输入数据：M=12288, K=4096, N=192　8 核并行			
转置模式	实际需要转置的分块	平均单核性能	计算峰值百分比
NN	A,B,C	147 GFlops	85.1%
NT	A,C	148 GFlops	85.6%
TN	B,C	156 GFlops	90.2%
TT	C	157 GFlops	90.9%