# 通用矩阵乘法及Cpu优化

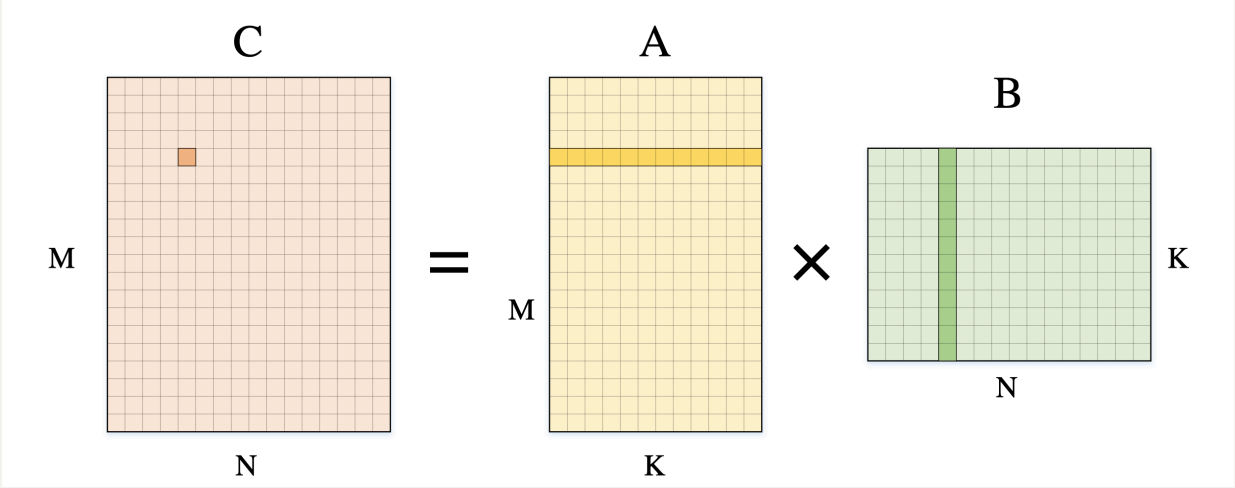
通用矩阵乘法(GEMM)是一个学习计算机体系结构和优化程序性能很好的一个例子。关于如何优化GEMM我最近也看了很多相关的文章，学到了很多关于计算机体系结构的知识，所以决定把最近学习到的相关知识，包括Cpu性能分析和计算机体系结构以及如何在cpu端优化GEMM总结一下。后续还会有针对GPU端GEMM的总结。

## 定义

再介绍GEMM(General )的定义之前，先介绍一下基础线性代数程序集BLAS(Basic Linear Algebra Subprograms)。BLAS是一个应用程序接口(API)标准，用以规范发布基础线性代数操作的数值库（如矢量或矩阵乘法）。其中GEMM的定义根据矩阵中元素的数据类型分为四种cgemm(单精度复数)，sgemm(单精度浮点数)，dgemm(双精度浮点数)，zgemm(双精度复数)。本文选取sgemm为例,

- sgemm C := alpha *  A x B  + beta * C

```
void sgemm (
    const char transa,    // 矩阵A是否需要转置
    const char transb,
    const int m,          // 矩阵A的行
    const int n,          // 矩阵A的列 矩阵B的行
    const int k,          // 矩阵B的列
    const float alpha,
    const float *a,
    const int lda,        // 如果A不转置，则lda=max(1, m),否则
lda=max(1, k)
    const float *b,
    const int ldb,        // 如果B不转置，则ldb=max(1, k),否则
lda=max(1, n)
    const float beta,
    float *c,
    const int ldc         // ldc=max(1, m)
) ;
```




## 优化

不经过任何优化的GEMM的源代码如下所示，一共三层循环，从外到内依次为n,m,k

```
int i, j, p;
 for (j = 0; j < n; j ++) {
     for (i = 0; i < m; i ++) {
         C(i, j) *= beta;
         for (p = 0; p < k; p ++) {
             C(i, j) += alpha * A(i, p) * B(p, j);
         }
     }
 }
```

- Optimization1 Cache访存优化
- Optimization2 SIMD指令集优化
- Optimization3 Blocked矩阵分块优化

## 参考


- Basic Linear Algebra Subprograms
- 如何加速矩阵乘法——优化GEMM (CPU单线程篇)
- how-to-optimize-gemm

- 通用矩阵乘（GEMM）优化与卷积计算