高性能计算导论实验1

通用矩阵乘法基于串行编程的实现及优化

秋季2022

**1.通用矩阵乘法**

数学上，一个m × n的矩阵是一个由m行n 列元素排列成的矩形阵列。矩阵是高等代数中常见的数学工具，也常见于统计分析等应用数学学科中。矩阵运算是数值分析领域中的重要问题。

通用矩阵乘法（GEMM）通常定义为：

C = AB

请根据定义用C语言实现一个矩阵乘法：

题目：用C语言实现通用矩阵乘法

输入：M , N, K 三个整数（512 ~2048）

问题描述：随机生成 M\*N 和N\*K 的两个矩阵 A,B,对这两个矩阵做乘法得到矩阵 C.

输出：A,B,C 三个矩阵以及矩阵计算的时间。

主函数：

int main(int argc,char\*\* argv)

{

    int m=atoi(argv[1]);

    int n=atoi(argv[2]);

    int k=atoi(argv[3]);

    vector< vector<int> > A(m,vector<int>(n));

    vector< vector<int> > B(n,vector<int>(k));

    vector< vector<int> > C(m,vector<int>(k));

    init\_matrix(A);

    init\_matrix(B);

    jser js;

    js.begin();

    universal(A,B,C);

    double cost=js.end();

    printf("m=%d n=%d k=%d cost:%.10fs\n",m,n,k,cost);

    printf("A\n");

    print\_matrix(A);

    printf("B\n");

    print\_matrix(B);

    printf("C\n");

    print\_matrix(C);

}

初始化矩阵：

void init\_matrix(vector<vector<int>> &A)

{

    srand(time(0));

    int m=A.size();

    int n=A[0].size();

    for(int i=0;i<m;i++)

    {

        for(int j=0;j<n;j++)

        {

            A[i][j]=rand()%(1<<16);

        }

    }

}

普通计算：

void universal(vector<vector<int>> &A,vector<vector<int>> &B,

               vector<vector<int>> &C)

{

    int m=A.size();

    int n=B.size();

    int k=C[0].size();

    for(int i=0;i<m;i++)

    {

        for(int j=0;j<k;j++)

        {

            C[i][j]=0;

            for(int s=0;s<n;s++)

            {

                C[i][j]+=A[i][s]+B[s][j];

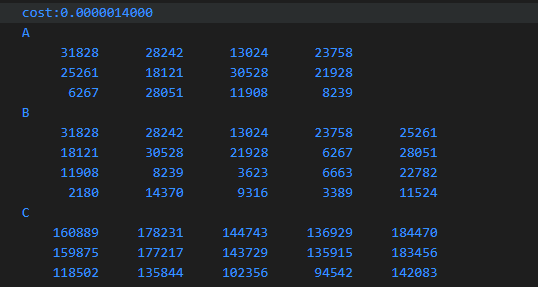
            }

        }

    }

}

当m=3,n=4,k=5时运行结果，用时0.0000014s：



当m=100 n=100 k=100 用时cost:0.0127663000s

**2.通用矩阵乘法优化**

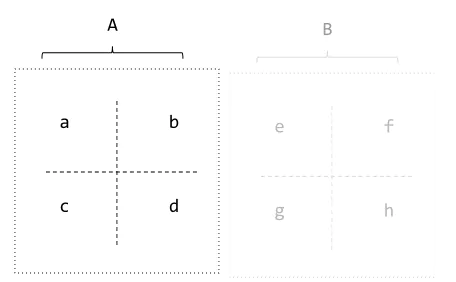
对上述的矩阵乘法进行优化，优化方法可以分为以下两类：

1. 基于算法分析的方法对矩阵乘法进行优化，典型的算法包括 [Strassen 算法](https://en.wikipedia.org/wiki/Strassen_algorithm)和 [Coppersmith–Winograd 算法](https://en.wikipedia.org/wiki/Coppersmith%E2%80%93Winograd_algorithm).
2. 基于软件优化的方法对矩阵乘法进行优化，如循环拆分向量化和内存重排

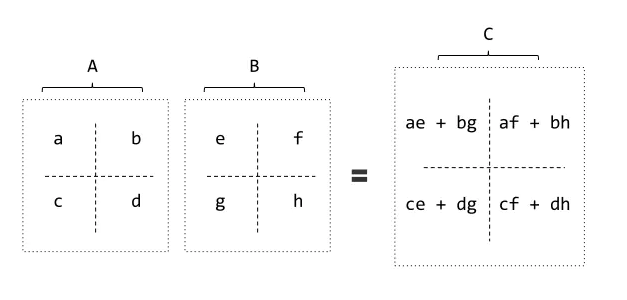
**实验要求**：对优化方法进行详细描述，并提供优化后的源代码，以及与GEMM 的计算时间对比

**Strassen:**

一般说来，当数据量较大时，我们往往会把大的数据分割成小的数据，各个分别处理。遵此思路，如果给我们一个很大的两个矩阵呢，是否可以考虑分治的方法循序渐进处理各个小矩阵的相乘，因为我们知道一个矩阵是可以分成更多小的矩阵的。  
如下图，当给定一个两个二维矩阵A B时：



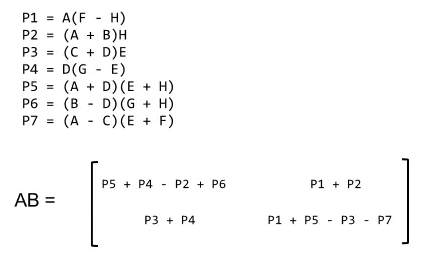
这两个矩阵A B相乘时，我们发现在相乘的过程中，有8次乘法运算，4次加法运算：



矩阵乘法的复杂度主要就是体现在相乘上，而多一两次的加法并不会让复杂度上升太多。

可以让矩阵乘法的运算过程中乘法的运算次数减少，从而达到降低矩阵乘法的复杂度。

用上文A B两个矩阵相乘的例子演示Strassen，定义了7个变量



这样计算只需要计算7次规模减半的乘法和若干次加法。时间复杂为

T(n/2)+O(n2) 展开为T(n2) 。

算法如下：

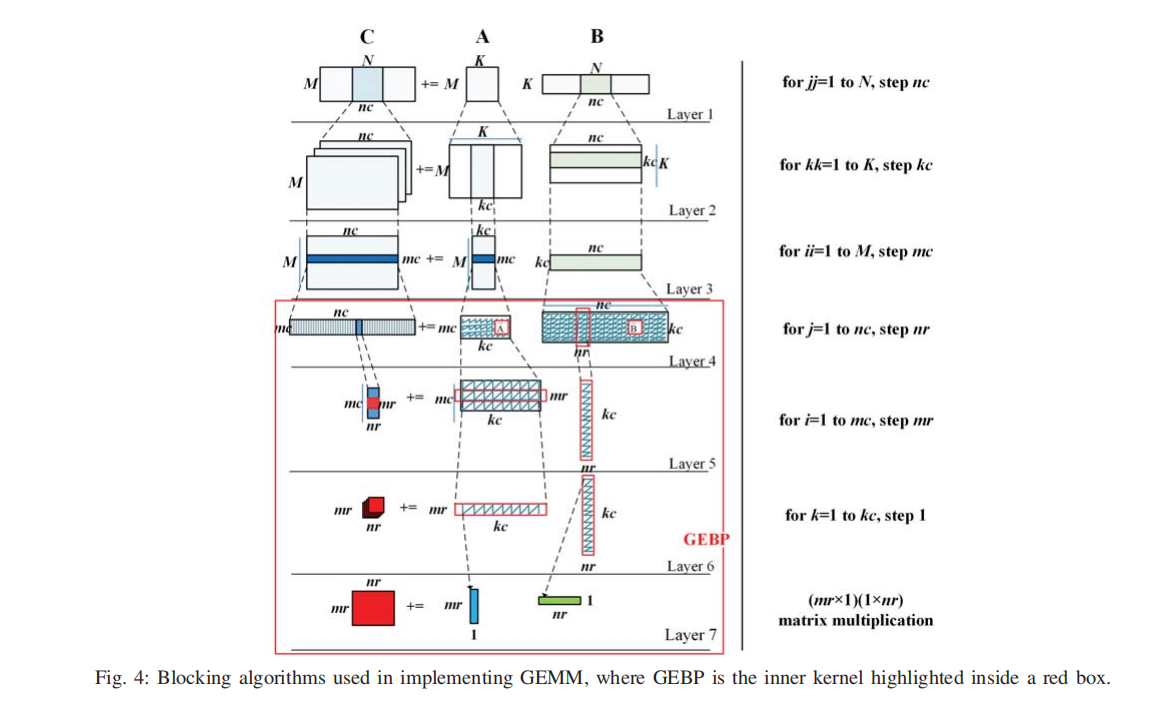
**3.进阶：大规模矩阵计算优化**

进阶问题描述：如何让程序支持大规模矩阵乘法？ 考虑两个优化方向

1. 性能，提高大规模稀疏矩阵乘法性能；
2. 可靠性，在内存有限的情况下，如何保证大规模矩阵乘法计算完成（M, N, K >> 100000），不触发内存溢出异常。

对优化方法及思想进行详细描述，提供大规模矩阵计算优化代码可加分。

**4.进阶2：通过block对矩阵进行计算优化**

优化思路：通过分块，增加cache的命中率

**References:**

1. {GEMM 优化}

<https://jackwish.net/2019/gemm-optimization.html>

1. {矩阵说明}

<https://zh.wikipedia.org/wiki/%E7%9F%A9%E9%98%B5>

1. {矩阵分块（block）}

Yang W, Fang J, Dong D. Characterizing Small-Scale Matrix Multiplications on ARMv8-based Many-Core Architectures[C]//2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2021: 101-110.