# 矩阵乘法优化

## **实验工具**

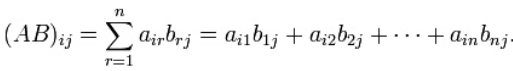
|  |  |
| --- | --- |
| 工具清单 | |
| 虚拟机 | VMware Workstation 14.1.1 build-7528167 |
| 系统镜像 | ubuntu-16.04.3-desktop-amd64.iso |
| 内核版本 | Linux ubuntu 4.13.0-41-generic |
| gcc版本 | version 5.4.0 20160609 |
| 处理器内核总数 | 4 |
| 运行内存 | 4GB |

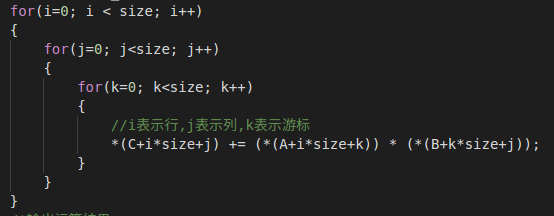
## **专题：矩阵乘法优化**

在数学中，矩阵（Matrix）是指纵横排列的二维数据表格，最早来自于方程组的系数及常数所构成的方阵，这一概念由19世纪英国数学家凯利首先提出，现如今已成为高等代数学中的常见工具，并广泛用于统计分析等应用数学学科中，阵乘法同时也是并行计算领域常常被用来作为范例的一个话题。它的特点是首先计算量可能相当大，适合利用并行实现来提高效率。其次，它所使用的各种数据之间没有相互依赖性，可以充分使用并行处理的计算资源。

### 2.1普通版本

矩阵相乘最重要的方法是一般矩阵乘积。它只有在第一个矩阵的栏数（column）和第二个矩阵的列数（row）相同时才有定义。一般单指矩阵乘积时，指的便是一般矩阵乘积。若A为m×n矩阵，B为n×p矩阵，则他们的乘积AB会是一个m×p矩阵。其乘积矩阵的元素如下面式子得出：

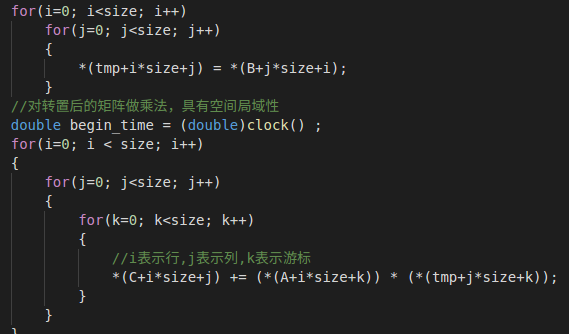




由于该算法很基础，显然运行速度比较慢，时间复杂度为O(n3)。

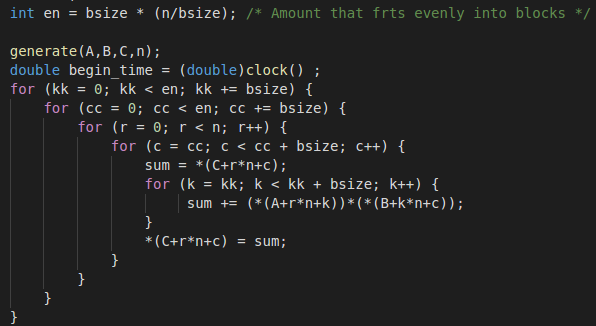
### 2.2矩阵转置[1]

在C语言中，数组的存储按照行优先规则，为进一步利用cache，对矩阵乘法中的第二个矩阵进行装置，提高命中率，进一步提高速度。Tmp作为中间数组保存矩阵B装置后的结果，运行时间在下文统一展示。



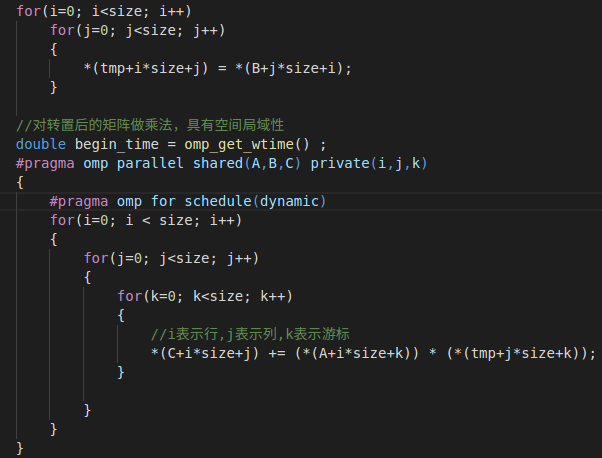
### 2.3分块优化[2]

核心是交叉乘加遍历，当数组大小增加时，时间局部性会明显降低，高速缓存中不命中数目增加。当使用分块技术时，时间局部性由分块大小来决定，而不是数组总大小来决定。本问题中选取块大小为bsize=4的方阵。

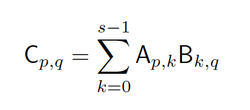


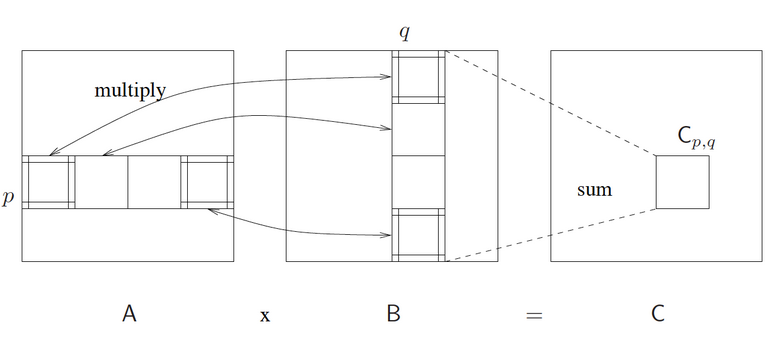
### 2.4矩阵转置+openmp优化

为进一步利用系统资源，在矩阵转置的基础上，进行openmp的初步优化。代码呈现如下：



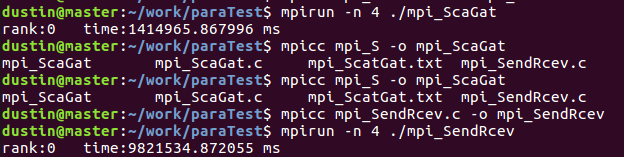
### 2.5 MPI并行

本门课程中仅剩mpi的并行方法未讲授，通过查阅资料[3]并自学部分内容，对该问题尝试进行mpi并行化。Mpi并行适用用数据规模较大，依赖性较弱的问题，如果采用更多的处理器，将会显著提高计算效率。实际中，通常并不可能有像矩阵元素那么多的处理器资源。这时我们该怎么做。对于一个大小为n × n 的大矩阵A，我们其实可以把它切分成s^2个子矩阵Ap,q，每个子矩阵的大小为 m × m，其中 m = n / s，即0 <= p, q < s。对于两个大矩阵A和B，现在我们有： 用图象形象表示如下：



* 使用较简单的MPI\_Send和MPI\_Recv实现
* 使用较高级的MPI\_Scatter和MPI\_Gather实现

具体代码详见c文件，文件中已加很多必要的注释，报告中不再赘述。





## **三.** **实验结果与总结**

### 3.1时间测量结果

下列四个版本整合在一个c文件中，编译命令为：

**gcc -mavx2 -msse2 -O3 -fopenmp matrix.c -o test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **矩阵A\*B=C，均为方阵，A=B=，其中n=1024** | | | | | |
| **（单位：ms）** | 1 | 2 | 3 | 4 | 平均时间 |
| **普通版本** | 6602757.0 | 6284082.0 | 6060442.0 | 6298664.0 | 6311486.3 |
| **矩阵转置** | 914533.0 | 873473.0 | 931000.0 | 1010096.0 | 932275.5 |
| **分 块** | 6349623.0 | 5452277.0 | 6361220.0 | 6153948.0 | 6079267.0 |
| **矩阵转置+openmp优化** | 724606.5 | 796924.2 | 995741.2 | 809043.0 | 831578.7 |

对于Mpi程序，进程数分别设置为1-8，可发现当设置为4时，整体运行效率较高，符合实验环境的特点。

### 3.2总结

通过本学期对并行计算理论与实践的结合学习，使得对计算机多个层次的并行方法的理解进一步加深，但实践方面很清醒得意识到要学习的东西还有很多。

## 参考资料

[1] cache测试及其矩阵优化

https://blog.csdn.net/GVFDBDF/article/details/49817979?utm\_source=blogxgwz4

[2] 矩阵乘法优化之分块矩阵

https://blog.csdn.net/u013471946/article/details/43957423

[3] 矩阵乘法的并行化算法讨论

https://blog.csdn.net/baimafujinji/article/details/48751037