矩阵乘法 cannon 算法

11091222 余锋伟

1 设计方法和模式

Cannon 的算法基本思路是把矩阵分块，把 A、B、C 矩阵分成 P*P 份，每次 Aik*Bkj 加到 Cij 上。不断轮转，使得最终所有的 Cij 能得到应有的 AB 分块相乘的值。

算法的精髓在于，第一步把 AB 矩阵分完块之后，每次运算完只需要把 A 矩阵循环左移，B 矩阵循环上移，所有的 AB 矩阵块又能再次进行乘法运算。因此只需要经过 P-1 次交换数据+P 次分块乘法，就能得到最终的 C 矩阵。

我把 cannon 算法分为以下几个步骤：

1. 计算当然把矩阵分块的 P 是多少，把节点个数标号为 Nodeij，其中 0=<i,j<P。
2. 其中节点 Nodeij 得到分块后的 Ai,j+i，Bi+j,j。
3. 节点 Nodeij 中的 AB 分块矩阵相乘，加到 Cij 中
4. 节点 Nodeij 中的 A 矩阵往 Nodeij-1 发送，B 矩阵往 Nodei-1,j 发送。重复 3 步骤直到矩阵乘完 P 次。
5. 把结果输出到结果文件

1.1 矩阵分发

矩阵分发我认为有两种方式，一种是主从模式，也就是说节点 0 读取矩阵的所有数据，然后分发分块矩阵到各自的节点。

另一种方式（也是我采用的）是让每个节点自行从文件系统里读取自己需要的矩阵块 AijBij。这种方法的好处之一是由于集群使用的是并行文件系统，读取数据不会存在竞争关系。二是上一中方式需要 0 开辟空间存储矩阵的所有数据，这使得内存有可能成为瓶颈。当每个节点只各自读自己的数据，那么即使矩阵再大，只要节点数足够多，能装得下 N/(P*P)的数据，就能进行计算。使得数据也并行存储。

1.2 分块矩阵相乘

单节点矩阵乘法很容易想到使用多线程加速，在程序时间中，先获取当然计算机所能支持的并发数量，以此构造线程。

假设计算机有 nump 个可并行线程，对于第 i 个线程，它将计算第 I,i+nump,i+2nump…行 A 与 B 的乘法，并加到 C 的 i,i+nump,i+2nump…上。由于每个线程之间不存在相互影响，因此多线程矩阵相乘能达到最的并行效率。

另外，矩阵乘法中 ijk 的枚举顺序能影响到效率，这里采用的是 ikj 循环。原因在《深入理解计算机系统》中有分析，主要是 ikj 顺序能有效降低 CPU 的 cache 缺失率。

1.3 AB 块的分发

由于每次矩阵做完乘法之后，都需要把 A 矩阵循环左移，B 矩阵循环右移。因此如何分发数据对于程序的整体性能有较大影响。

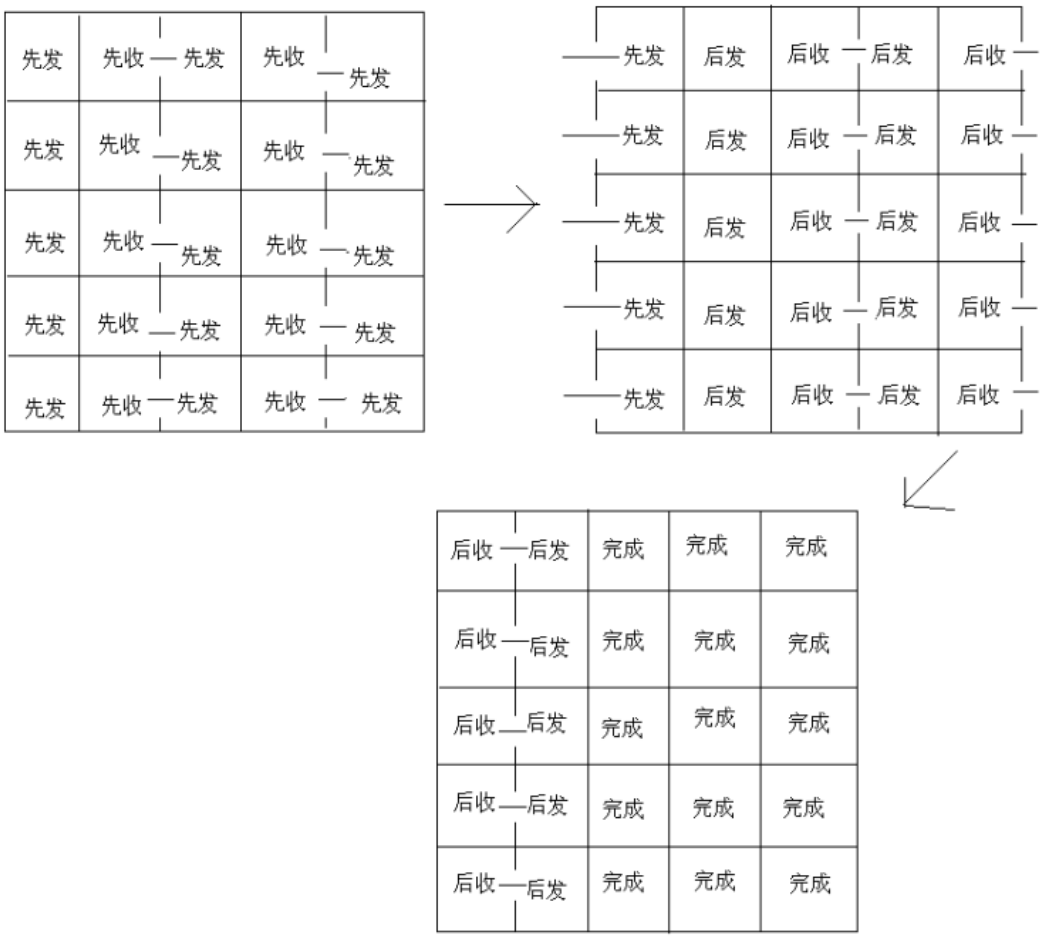
我一开始想采用的是所有的节点都先使用 MPI_Send 发送数据，再使用 MPI_Recv 接收

数据。这个时候发现程序死锁了，原因在于 MPI_Send 是阻塞发送，所有的节点都卡在 Send 阶段了。因此这个时候需要错开每个节点的 Send 和 Recv。我终于想到了一个好方法。

以 A 块的循环左传为例子，P=5 时，使用

```
If (j % 2 == 0)
{
    MPI_Send
    MPI_Recv
}
Else
{
    MPI_Recv
    MPI_Send
}
```

表示奇数列先发再收，偶数行先收再发。于是通讯情况如下。

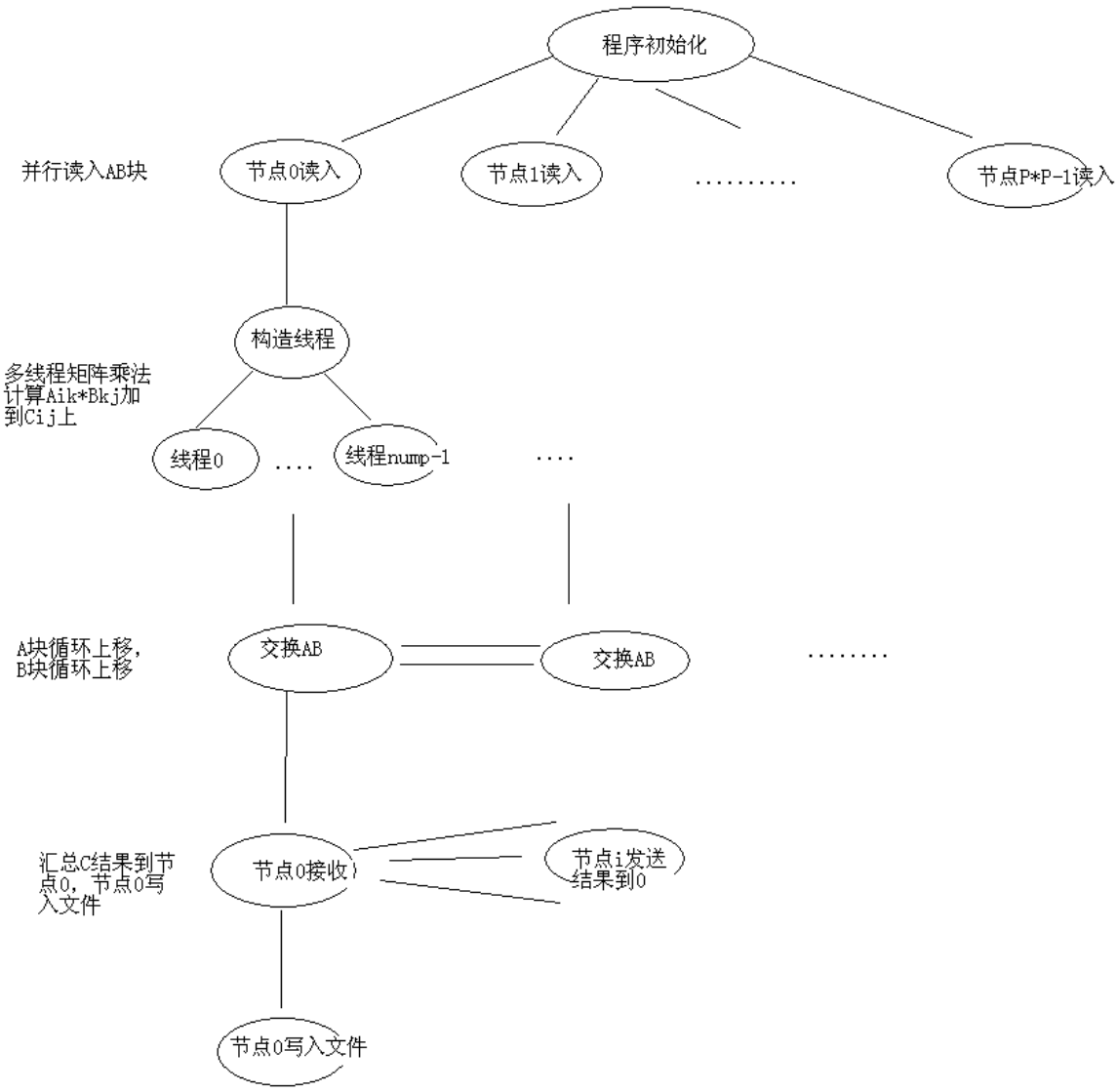


一共需要 3 次并行通信，原因是 P=5 为奇数。而当 P 为偶数时，只需要 2 次并行通信。并且该通信次数并不随着 P 的增大而增大。B 块同理。

1.4 输出结果

由于并行文件系统对于并行写支持的很不好，因此对于文件结果写入，采用主从式。即所有节点把它对于 C 的运算结果发送到 0 号节点，由 0 号节点写入文件。

2 流程图



3 复杂度

从流程图可以很容易分析出算法的复杂度。

3.1 空间复杂度

假设矩阵一共有 A=N1*N2,B=N2*N3,C=N1*N3 个数据，有 $P*P$ 个节点，那么每个节点需要开辟 $\frac{N1*N2+N2*N3+N1*N3}{P*P}$ 空间去存储 A，B，C 的分块数据。有

$P*P$ 个节点，所以，总的空间复杂度为 $N1*N2+N2*N3+N1*N3$ ，达到了理论上的最低空间复杂度。

3.2 时间复杂度

- 每个矩阵块读入的时间是 $Tread=\frac{(N1*N2+N2*N3)}{P*P}$ ，并行效率较高，
- 多线程矩阵乘法时间是 $\frac{(N1/P)*(N2/P)*(N3/P)}{nump}$ ，其中 $nump$ 是线程个数，说明多线程矩阵乘法能够达到理论最高的加速比。
- 假设一次数据交换(一块 A 或者一块 B)的时间是 $Tswap$ ，那么最坏情况下，需要并行 3 次交换，因此一次数据交换时间的上界是 $3*Tswap$ 。
- 0 号节点接受所有 $P*P-1$ 个节点发回的数据需要耗时 $(P*P-1)*Tswap$ ，需要串行写入 $N1*N3$ 个数据。

总时间复杂度：

$$O(Tread)+O(Tcalc)+O(Twrite)$$

其中

$$O(Tread)=\frac{N1*N2+N2*N3}{P*P}$$

$$O(Tcalc)=O\left(\frac{N1*N2*N3}{nump*P^3}*p+(p-1)*3*Tswap\right)$$

$$O(Twrite)=O((P*P-1)*Tswap+N1*N3)$$

4 结果分析和总结

本次作业所有代码都有自己一个人设计编写而成，一共 300 行左右，感觉很有成就感。在本次实验中，尽量设计出让每一步都高度并行的程序，经过测试。在 1000×1000 的数据中，普通乘法需要耗费 12s 时间，而使用我的方法进行计算，在开启 4×4=16 个节点计算时，耗费时间大约为 0.9s，达到了超过 12 的加速比，十分接近理论上 16 的加速比。

这次是最后一次作业，感觉学到了很多东西，尤其是如何优化 OI 操作，如何设计更好的并行流程，以及如何优化通信复杂度。

感谢这一学期来老师和助教的辛苦教学！