大型矩阵乘法的FOX算法MPI流水并行

--并行应用实例

*工学院 洪瑶 1801111621*

**目录**

[一、应用问题描述和应用价值分析 2](#_Toc533421230)

[1.1问题背景 2](#_Toc533421231)

[1.2数学描述 2](#_Toc533421232)

[二、应用问题的计算特征分析、影响数据处理效率的关键因素 3](#_Toc533421233)

[2.1计算特征分析 3](#_Toc533421234)

[2.2影响数据处理效率的关键因素 3](#_Toc533421235)

[三、任务分解方案、数据访问冲突分析 4](#_Toc533421236)

[3.1任务分解方案 4](#_Toc533421237)

[3.2数据访问冲突分析 5](#_Toc533421238)

[四、目标并行计算机体系结构概述 6](#_Toc533421239)

[五、计算划分方案、负载均衡分析、同步开销分析 6](#_Toc533421240)

[5.1计算划分方案 6](#_Toc533421241)

[5.2计算负载均衡性分析 8](#_Toc533421242)

[5.3同步开销分析 9](#_Toc533421243)

[六、实现设计 10](#_Toc533421244)

[七、实验评估与实验结果分析 12](#_Toc533421245)

[7.1计算量与通信量分析 12](#_Toc533421246)

[7.2实验结果分析 14](#_Toc533421247)

# 一、应用问题描述和应用价值分析

## 1.1问题背景

稠密线性代数运算对模式识别和生物信息学等领域有着许多实际应用都十分重要，而矩阵的乘法就是十分重要的一项运算，矩阵乘法在数值预报中就必须要用到，因此研究在大型数值预报时就要用到大型矩阵乘法。

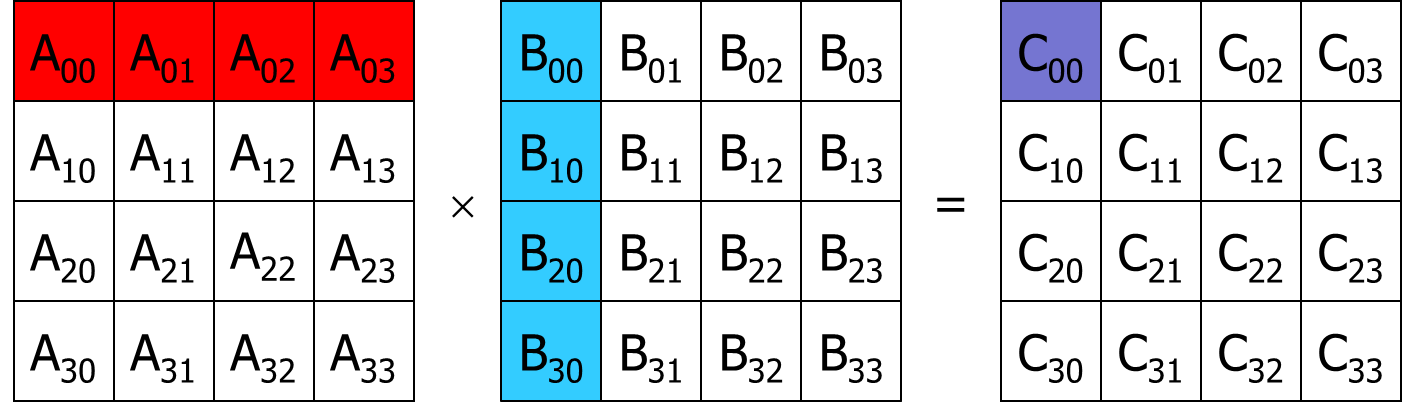
我们知道大型矩阵乘法的计算量十分庞大，如果没有任何优化，两个N\*N的矩阵相乘的 标量运算复杂性为O(N3)。20世纪60年代末期，strassen提出了一个新的快速算法，其标量运算只需要O(Nlog27)。但是当矩阵规模十分大时，对计算器而言仍然是个十分巨大的挑战，因此考虑采用分布式并行计算是十分必要的。

## 1.2数学描述

设有M\*K个数据aij的二阶矩阵***A***，有K\*N个数据的bij二阶矩阵***B***，计算矩阵***C:***

***C***=***AB***

其中cij计算原则为:

******

# 二、应用问题的计算特征分析、影响数据处理效率的关键因素

## 2.1计算特征分析

该问题最大的特征就是每个元素C­ij都需要A矩阵的第i行和B矩阵第j列的数据，因此若考虑并行计算数据划分是最大的困难，尤其对于大型稠密矩阵而言，全部数据存储在每个进程中将导致储存空间不足的问题，并起不到并行的目的。

## 2.2影响数据处理效率的关键因素

本问题的数据依赖性特别高，因此在进行计算和规约时，如果不行合理的算法，将导致R-R冲突和R-W冲突大大降低计算效率，因此必须设计合理的流水并行算法，减少数据的R-R冲突。

# 三、任务分解方案、数据访问冲突分析

## 3.1任务分解方案

利用高等代数的分块矩阵的方法，将A、B、C矩阵分别进行分块划分，划分方案如下（这里尤其要注意分块子矩阵的下标界限）：

а 
021 
а12 
аи 
к 
12 
11 
1q 
21 
q1 
мк 

11 
21 
12 
22 
q 
21 
ql 
12 
Bll 
Iq 

C11 
С21 
С12 
C22 
12 
С 11 
1q 
21 
ММ 

将矩阵进行划分以后，可以按照高等代数里的知识：

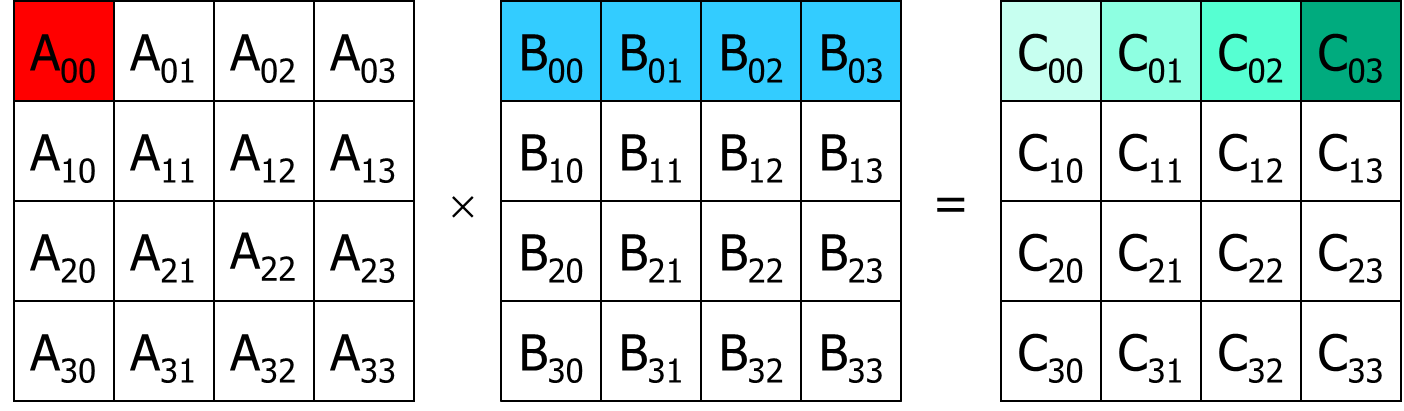
Cij 
Ai0 + Ail BI j + . . . + Aii 
I 13 
i,q 1 q—l,j 

注意，此时的Cij和cij的区别，Aij/Bij/Cij分别A/B/C的子矩阵。

到这里，可以根据C矩阵的划分原则，将其进行并行化。

## 3.2数据访问冲突分析

如下图所示，在进行第一次的计算时，如果按行优先的计算原则，那么A00必然会R-R冲突，而B矩阵的读取不会发生冲突，因此规避A子矩阵的冲突时必然要做的事情，于是可以考虑用行通信子将A00广播给每一行的C0j即可。



# 四、目标并行计算机体系结构概述

由于目前没有足够的资金购置多台计算机进行群配置，因此本次案例采用个人计算机进行并行计算，计算机参数为：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

所采用的MPI版本号为：

# 五、计算划分方案、负载均衡分析、同步开销分析

## 5.1计算划分方案

(1)矩阵A是M×K矩阵，矩阵B是K×N矩阵。

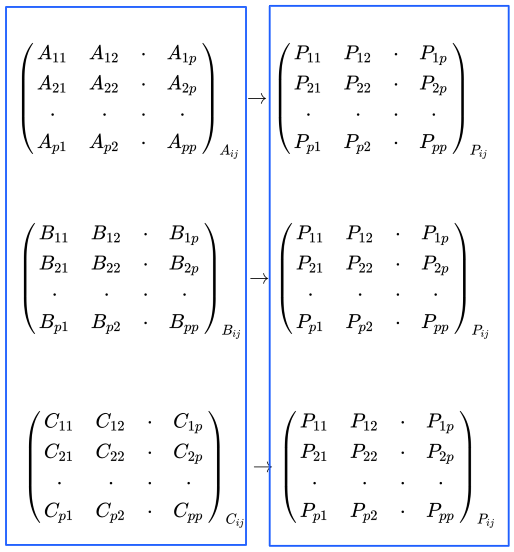
(2)并行计算矩阵***C=AB***。

(3)设*p*=num(处理器)是处理器的数目，q=是整数，使得它除以M和N。

(4)建立具有网格Pi j的笛卡尔拓扑进程，*i*=0~q-1，*j*=0~q-1。

(5)定义。

(6)在p进程上按块分配A和B，使得Aij是M×K块，Bij是K×N块，存储在进程Pij上。



Maping

上图为分块矩阵数据最初存储在进程中匹配情况，之后的数据交换仅仅为数据块的交换，对数据进行传输时，以打包的方式进行传输。

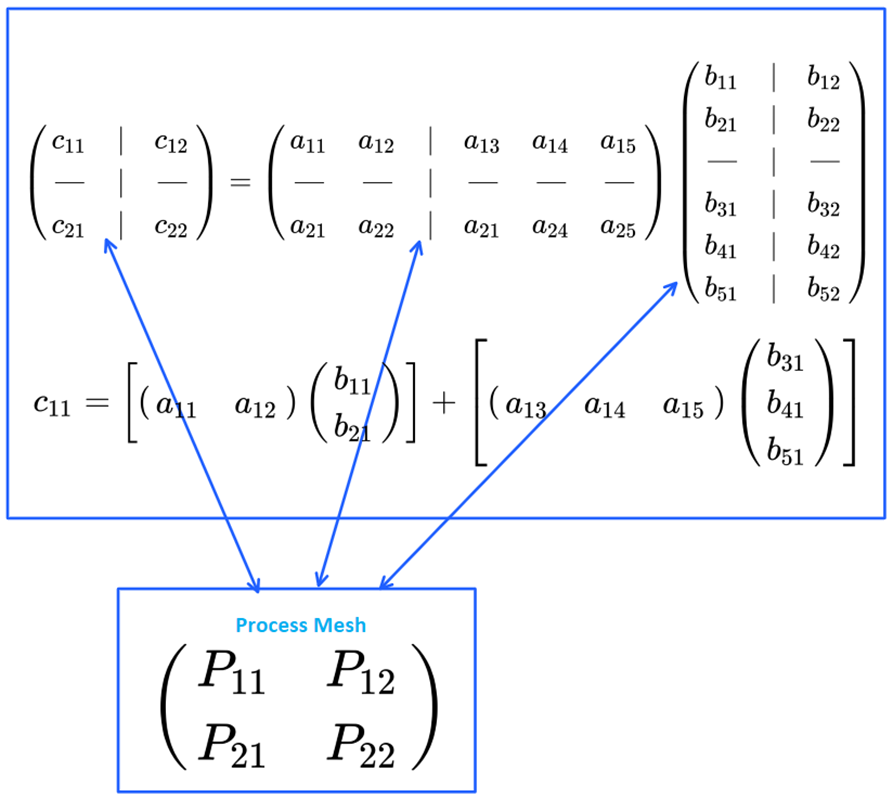
(7) 使得Aij是M×K块，Bij是K×N块，存储//在进程Pij上,进行一次FOX kernel的向量乘法计算，既在Pij计算AikBkj，其中k代表第k个超级计算步，每个计算步最后都得进行一次规约：

|  |  |
| --- | --- |
| **Super Step** | **Mathematical Operation** |
| **0** |  |
| **1** |  |
| **2** |  |
| **…** | **…** |
| **q-2-i** |  |
| **q-1-i** |  |
| **…** |  |
| **…** |  |
| **…** | **…** |
| **q-1** |  |

## 5.2计算负载均衡性分析

由于我们把行列都设为能被进程数的数量的开根号（既q）整除，于是在计算步的层面上，本问题不存在负载不均衡，也就意味着每一次同步，所有处理器上都进行了一次FOX kernel 运算并进行了规约。

但是从每个超级步之中的不同进程计算量来看，k的值没有规定能被q整除，因此其将导致负载的不均衡，如下图所示，当一个最简单的负载不均衡存在时，不同处理器的计算量不一样，将导致同步开销的增加。



## 5.3同步开销分析

同步开销体现在一个超级步上的时间不均衡性上面，这部分不均衡分为两个部分：

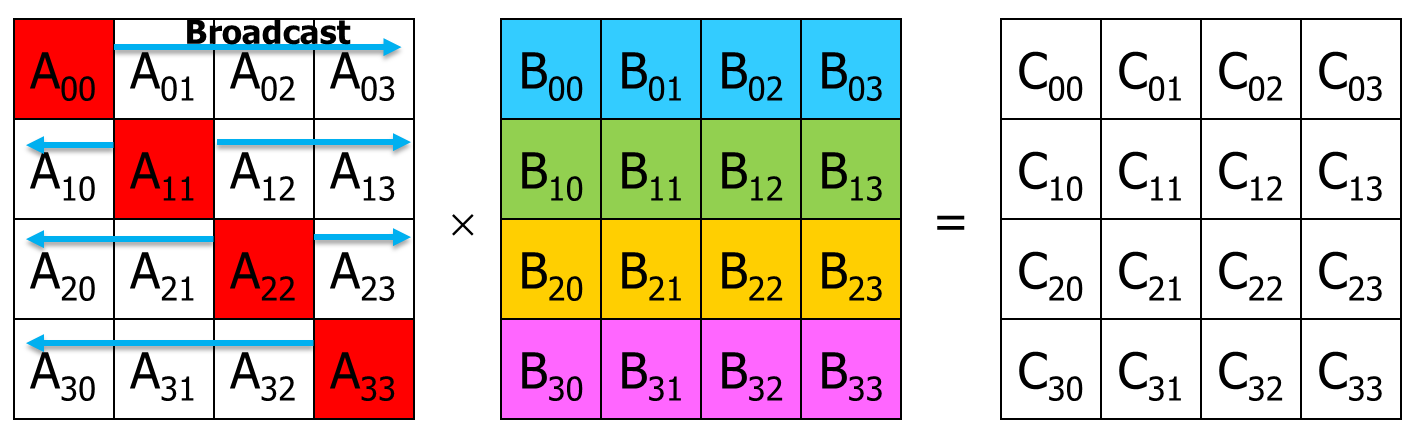
1. 从5.2中知道，由于负载不均衡，负载最重的处理器显然所需要的时间更长，同步的等待时间开销一部分来自这里。
2. 另一部分同步开销来自数据的通信导致，由于**拥有者原则**的存在，一部分进程不需要进行数据通信，直接进行计算，而另一部分进程则需要先通信数据再计算，这部分时间不均衡也导致同步开销增加。

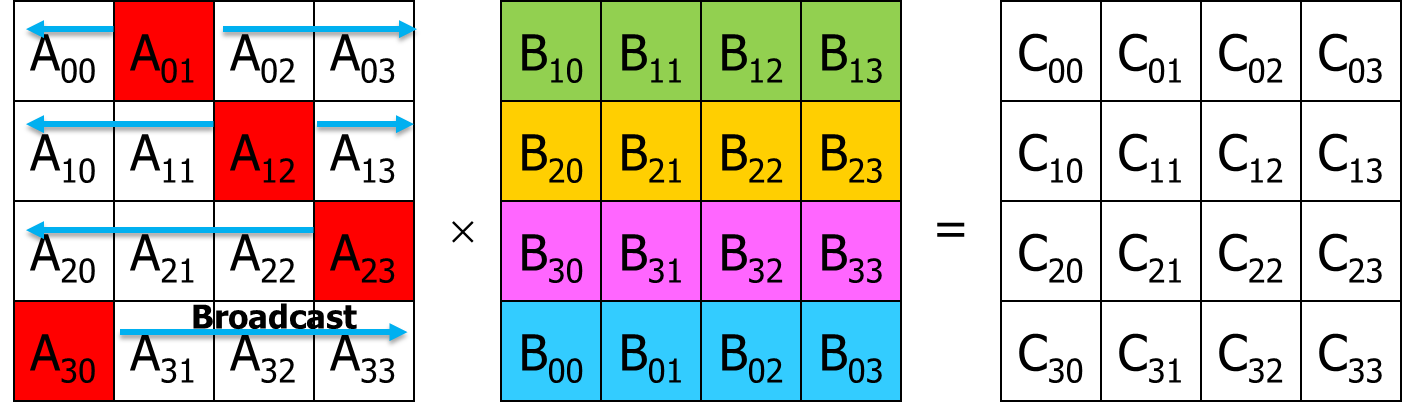
# 六、实现设计

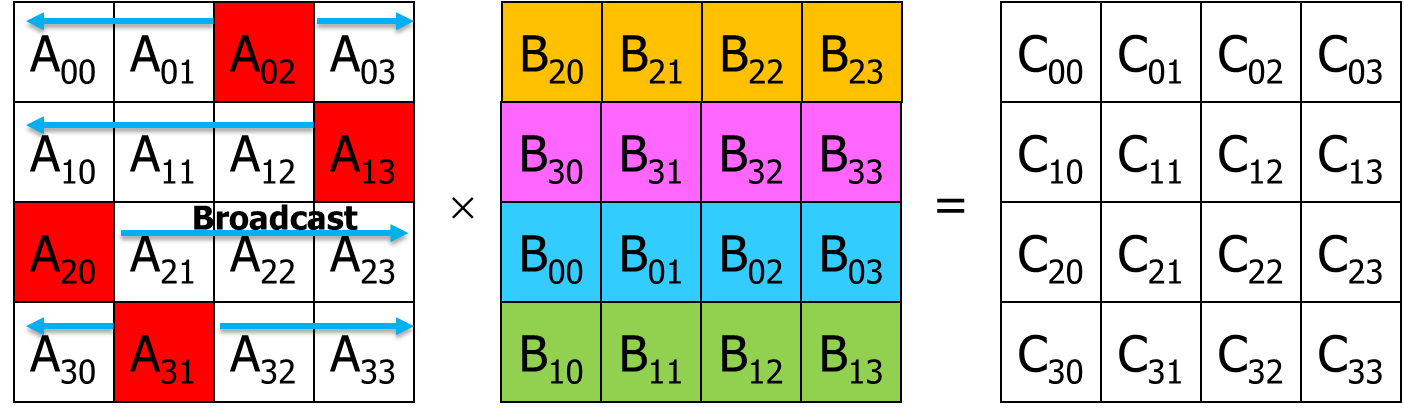
超级步中实现如下具体步骤：

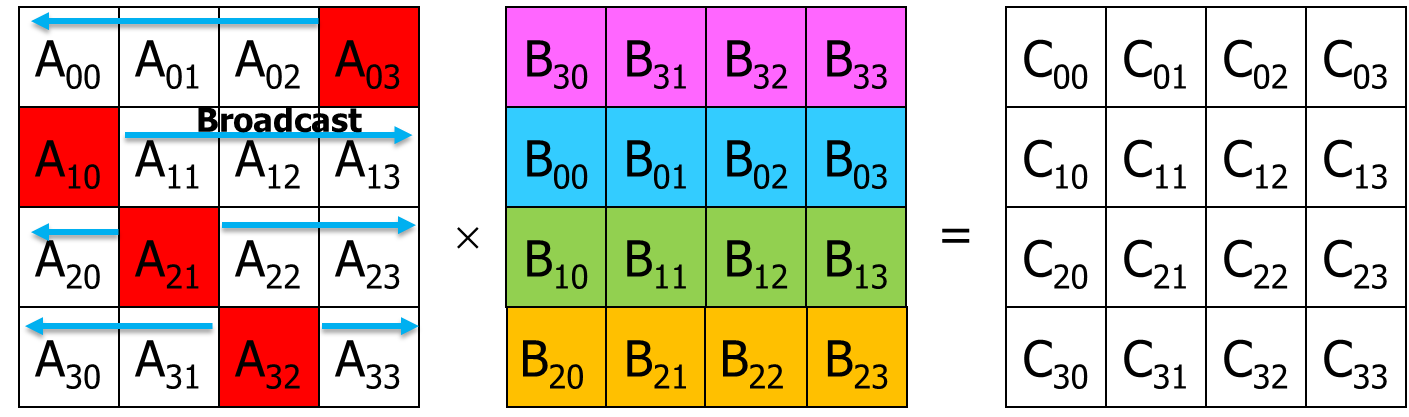
|  |  |
| --- | --- |
| **Stage** | **Algorithm Operation** |
| **0** | 到进程网格的第i行的所有进程中  *3.* 进程 *Pij*计算 |
| **1** | *1.*  1.1 将中的数据统一上翻一行 (行 转到第行)  1.2 broadcast 到第i行的所有进程  2. 进程 计算 |
| **2** | *1.*  1.1 将中的数据统一上翻一行 (第 𝟎行 转到第𝒒−𝟏行)  1.2 broadcast 到第i行的所有进程  2. 进程 计算 |
| **…** | … |
| **q-2-i** |  |
| **q-1-i** |  |
| **…** |  |
| **…** |  |
| **…** | … |
| **q-1** |  |

以处理器数量为16为例，可视化上述步骤如下：









在通过传播之后（相当于点乘了！）

# 七、实验评估与实验结果分析

## 7.1计算量与通信量分析

|  |  |
| --- | --- |
| **Stage** | 计算量和通信量分析 |
| **0** |  |
| **1** |  |
| **2** |  |
| **…** |  |
| **q-2-i** |  |
| **q-1-i** |  |
| **…** |  |
| **…** |  |
| **…** |  |
| **q-1** |  |

总通信量:

Comm 
Ⅱ (M × K + K × 3 × (q | 1 一 
K × N 
× 冖 q | 1 一 

总计算量:

Comput 
х 
х 
МхКхЛТ 

## 7.2实验结果分析