1.1 行列划分矩阵乘法

一个 $m\times n$ 阶矩阵 $\boldsymbol{A}=[a_{ij}]$乘以一个 $n\times k$ 的矩阵 $\boldsymbol{B}=[b_{ij}]$就可以得到一个 $m\times k$ 的矩阵 $\boldsymbol{C}=[c_{ij}]$，它的元素 c_{ij} 为 $\boldsymbol{A}$ 的第 i 行向量与 $\boldsymbol{B}$ 的第 j 列向量的内积。矩阵相乘的关键是相乘的两个元素的下标要满足一定的要求(即对准)。为此常采用适当旋转矩阵元素的方法(如后面将要阐述的 Cannon 乘法)，或采用适当复制矩阵元素的办法(如 DNS 算法)，或采用流水线的办法使元素下标对准，后两种方法读者可参见文献[1]。

1.1.1 矩阵相乘及其串行算法

由矩阵乘法定义容易给出其串行算法 18.5，若一次乘法和加法运算时间为一个单位时间，则显然其时间复杂度为 O(mnk) 。

算法 18.5　单处理器上矩阵相乘算法

输入：$A_{m\times n}$，$B_{n\times k}$

输出：$C_{m\times k}$

```
Begin
    for i=0 to m-1 do
        for j=0 to k-1 do
            c[i,j]=0
            for r=0 to n-1 do
                c[i,j]= c[i,j]+a[i,r]*b[r,j]
            end for
        end for
    end for
End
```

1.1.2 简单的矩阵并行分块乘法算法

矩阵乘法也可以用分块的思想实现并行，即**分块矩阵乘法**(Block Matrix Multiplication)，将矩阵 $\boldsymbol{A}$ 按行划分为 p 块(p 为处理器个数)，设 $u=\lceil m/p\rceil$，每块含有连续的 u 行向量，这些行块依次记为 $\boldsymbol{A}_0,\boldsymbol{A}_1,\cdots,\boldsymbol{A}_{p-1}$，分别存放在标号为 0,1,…, p-1 的处理器中。对矩阵 $\boldsymbol{B}$ 按列划分为 P 块，记 $v=\lceil k/p\rceil$，每块含有连续的 v 列向量，这些列块依次记为 $\boldsymbol{B}_0,\boldsymbol{B}_1,\cdots,\boldsymbol{B}_{p-1}$，分别存放在标号 0,1, …, p-1 为的处理器中。将结果矩阵 $\boldsymbol{C}$ 也相应地同时进行行、列划分，得到 $p\times p$ 个大小为 $u\times v$ 的子矩阵，记第 i 行第 j 列的子矩阵为 $\boldsymbol{C}_{ij}$，显然有 $\boldsymbol{C}_{ij}=\boldsymbol{A}_i\times\boldsymbol{B}_j$，其中，$\boldsymbol{A}_i$ 大小为 $u\times n$，$\boldsymbol{B}_j$ 大小为 $n\times v$。

处理器编号	存储内容	
0	A_0	B_0
1	A_1	B_1
2	A_2	B_2
•••	•••	•••
P-1	A_{p-1}	B_{p-1}

(a)

处理器编号	存储内容	
0	A_0	B_1
1	A_1	B_2
2	A_2	B_3
•••	•••	•••
P-1	A_{p-1}	B_0

(b)

图.1 矩阵相乘并行算法中的数据交换

开始，各处理器的存储内容如图.1 (a)所示。此时各处理器并行计算 $\boldsymbol{C}_{ii}=\boldsymbol{A}_i\times\boldsymbol{B}_j$ 其中 i=0,1,…,p-1，此后第 i 号处理器将其所存储的 B 的列块送至第 i-1 号处理器（第 0 号处理器将 B 的列块送至第 p-1 号处理器中，形成循环传送），各处理器中的存储内容如图.1 (b)所示。它们再次并行计算 $\boldsymbol{C}_{ij}=\boldsymbol{A}_i\times\boldsymbol{B}_j$，这里 $j=(i+1)\bmod p$。B 的列块在各处理器中以这样的方式循环传送 p-1 次并做 p 次子矩阵相乘运算，就生成了矩阵 $\boldsymbol{C}$ 的所有子矩阵。编号为 i 的处理器的内部存储器存有子矩阵 $\boldsymbol{C}_{i0},\boldsymbol{C}_{i1},\ldots,\boldsymbol{C}_{i(p-1)}$。为了避免在通信过程中发生死锁，奇数号及偶数号处理器的收发顺序被错开，使偶数号处理器先发送后接收；而奇数号处理器先将 B 的列块存于缓冲区 buffer 中，然后接收编号在其后面的处理器所发送的 B 的列块，最后再将缓冲区中原矩阵 B 的列块发送给编号在其前面的处理器，具体并行算法框架描述如下：

算法 18.6　矩阵并行分块乘法算法

输入： $A_{m\times n}$, $B_{n\times k}$,

输出： $C_{m\times k}$

Begin

对所有处理器 my_rank(my_rank=0,…,p-1)同时执行如下的算法:

(1)目前计算 C 的子块号 $l=(i+\text{my\_rank}) \bmod p$

(2)**for** z=0 **to** u-1 **do**

for j=0 **to** v-1 **do**

$c[l,z,j]$=0

for s=0 **to** n-1 **do**

$c[l,z,j]=c[l,z,j]+a[z,s]*b[s,j]$

end for

end for

end for

(3)计算左邻处理器的标号 $mm1=(p+\text{my\_rank}-1) \bmod p$

计算右邻处理器的标号 $mp1=(\text{my\_rank}+1) \bmod p$

(4)**if** ($i\neq p$-1) **then**

(4.1)**if** (my_rank mod 2= 0) **then**　　/*编号为偶数的处理器*/

(i)将所存的 B 的子块发送到其左邻处理器中

(ii)接收其右邻处理器中发来的 B 的子块

end if

(4.2)**if** (my_rank mod 2≠ 0) **then**　　/*编号为奇数的处理器*/

(i)将所存的 B 子块在缓冲区 buffer 中做备份

(ii)接收其右邻处理器中发来的 B 的子块

(iii)将 buffer 中所存的 B 的子块发送到其左邻处理器中

end if

end if

End

设一次乘法和加法运算时间为一个单位时间，由于每个处理器计算 p 个 $u\times n$ 与 $n\times v$ 阶的子矩阵相乘，因此计算时间为 $u*v*n*p$；所有处理器交换数据 p-1 次，每次的通信量为 $v*n$，通信过程中为了避免死锁，错开奇数号及偶数号处理器的收发顺序，通信时间为 $2(p-1)(t_s+nv*t_w)=O(nk)$，所以并行计算时间 $T_p=uvnp+2(p-1)(t_s+nvt_w)=mnk/p+2(p-1)(t_s+nvt_w)$。

MPI 源程序请参见所附光盘。

1.2 Cannon 乘法

1.2.1 Cannon 乘法的原理

Cannon 算法是一种存储有效的算法。为了使两矩阵下标满足相乘的要求，它和上一节的并行分块乘法不同，不是仅仅让 B 矩阵的各列块循环移动，而是有目的地让 A 的各行块以及 B 的各列块皆施行循环移位，从而实现对 C 的子块的计算。将矩阵 A 和 B 分成 p 个方块 A_{ij} 和 B_{ij}，$(0 \le i, j \le \sqrt{p}-1)$，每块大小为$\lceil n/\sqrt{p} \rceil \times \lceil n/\sqrt{p} \rceil$，并将它们分配给 $\sqrt{p} \times \sqrt{p}$ 个处理器 $(P_{00}, P_{01}, ..., P_{\sqrt{p}-1\sqrt{p}-1})$。开始时处理器 P_{ij} 存放块 A_{ij} 和 B_{ij}，并负责计算块 C_{ij}，然后算法开始执行：

⑴将块 A_{ij} 向左循环移动 i 步；将块 B_{ij} 向上循环移动 j 步；

⑵P_{ij} 执行乘加运算后将块 A_{ij} 向左循环移动 1 步，块 B_{ij} 向上循环移动 1 步；

⑶重复第⑵步，总共执行 $\sqrt{p}$ 次乘加运算和 $\sqrt{p}$ 次块 A_{ij} 和 B_{ij} 的循环单步移位。

1.2.2 Cannon 乘法的并行算法

图 2 示例了在 16 个处理器上，用 Cannon 算法执行 $A_{4\times4} \times B_{4\times4}$ 的过程。其中(a)和(b)对应于上述算法的第⑴步；(c)、(d)、(e)、(f)对应于上述算法的第⑵和第⑶步。在算法第⑴步时，A 矩阵的第 0 列不移位，第 1 行循环左移 1 位，第 2 行循环左移 2 位，第 3 行循环左移 3 位；类似地，B 矩阵的第 0 行不移位，第 1 列循环上移 1 位，第 2 列循环上移 2 列，第 3 列循环上移 3 列。这样 Cannon 算法具体描述如下：

算法 18.7　Cannon 乘法算法

输入：$A_{n\times n}$，$B_{n\times n}$

输出：$C_{n\times n}$

Begin

对所有处理器 my_rank(my_rank=0,…,p-1)同时执行如下的算法:

(1)计算子块的行号 i=my_rank/sqrt(p)

计算子块的列号 j=my_rank mod sqrt(p)

(2)**for** k=0 **to** $\sqrt{p}$ -1 **do**

if (i>k) **then** Leftmoveonestep(a) **end if**　/* a 循环左移至同行相邻处理器中*/

if (j>k) **then** Upmoveonestep(b) **end if**　/* b 循环上移至同列相邻处理器中*/

end for

(3)**for** i=0 **to** m-1 **do**

for j=0 **to** m-1 **do**

c[i,j]=0

end for

end for

(4)**for** k=0 **to** $\sqrt{p}$ -1 **do**

for i=0 **to** m-1 **do**

```
            for j=0 to m-1 do
                for k1=0 to m-1 do
                    c[i,j]= c[i,j]+ a[i,k1]* b[k1,j]
                end for
            end for
        end for
        Leftmoveonestep(a) /*子块 a 循环左移至同行相邻的处理器中*/
        Upmoveonestep(b) /*子块 b 循环上移至同列相邻的处理器中*/
    end for
End
```

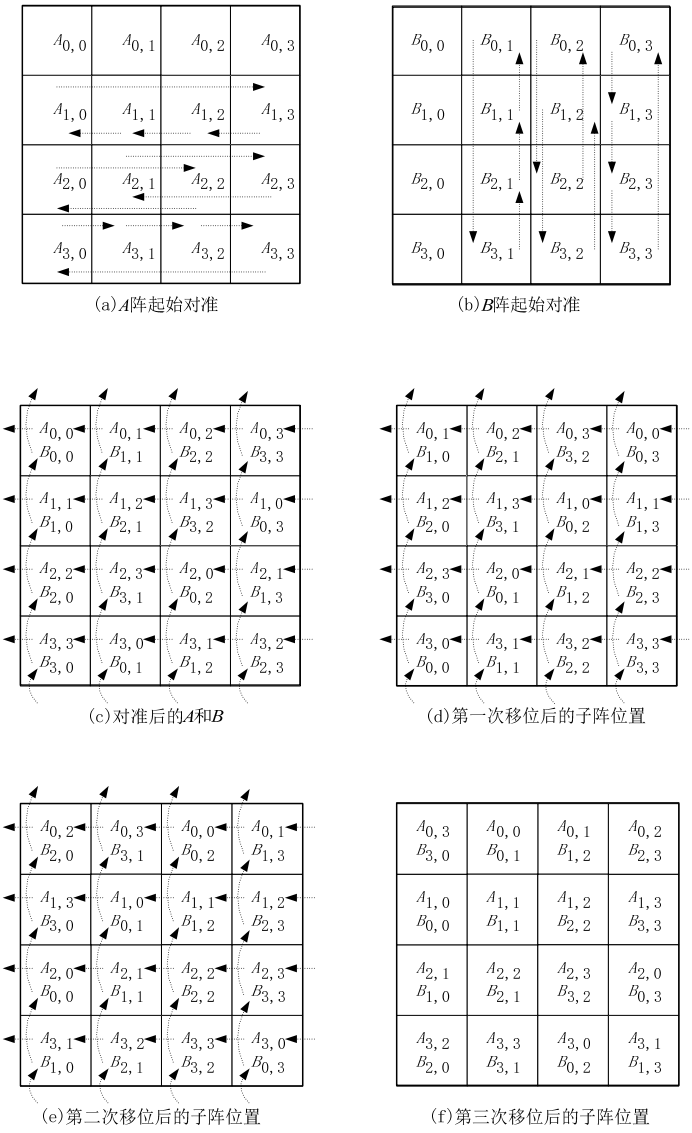

图 2　16 个处理器上的 Cannon 乘法过程

这里函数 Leftmoveonestep(*a*)表示子块 *a* 在编号处于同行的处理器之间以循环左移的方式移至相邻的处理器中；函数 Upmoveonestep(*b*)表示子块 *b* 在编号处于同列的处理器之间以

循环上移的方式移至相邻的处理器中。这里我们以函数 Leftmoveonestep(*a*)为例，给出处理器间交换数据的过程：

算法 18.8　函数 Leftmoveonestep(*a*)的基本算法

Begin

(1)**if** (*j*=0) **then**　/*最左端的子块*/

(1.1)将所存的 A 的子块发送到同行最右端子块所在的处理器中

(1.2)接收其右邻处理器中发来的 A 的子块

end if

(2)**if** ((*j* = sqrt(*p*)-1) and (*j* mod 2 = 0)) **then**　/*最右端子块处理器且块列号为偶数*/

(2.1)将所存的 A 的子块发送到其左邻处理器中

(2.2)接收其同行最左端子块所在的处理器发来的 A 的子块

end if

(3)**if** ((*j* = sqrt(*p*)-1) and (*j* mod 2 ≠ 0)) **then**　/*最右端子块处理器且块列号为奇数*/

(3.1)将所存的 A 的子块在缓冲区 buffer 中做备份

(3.2)接收其同行最左端子块所在的处理器发来的 A 的子块

(3.3)将在缓冲区 buffer 中所存的 A 的子块发送到其左邻处理器中

end if

(4)**if** ((*j* ≠ sqrt(*p*)-1) and (*j* mod 2 = 0) and (*j* ≠ 0)) **then**　/*其余的偶数号处理器*/

(4.1)将所存的 A 的子块发送到其左邻处理器中

(4.2)接收其右邻处理器中发来的 A 的子块

end if

(5)**if** ((*j* ≠ sqrt(*p*)-1) and (*j* mod 2 = 1) and (*j* ≠ 0)) **then**　/*其余的奇数号处理器*/

(5.1)将所存的 A 的子块在缓冲区 buffer 中做备份

(5.2)接收其右邻处理器中发来的 A 的子块

(5.3)将在缓冲区 buffer 中所存的 A 的子块发送到其左邻处理器中

end if

End

当算法执行在 $\sqrt{p}\times\sqrt{p}$ 的二维网孔上时,若使用切通 CT 选路法,算法 18.7 第(2)步的循环移位时间为 $2(t_s+t_w\frac{n^2}{p})\sqrt{p}$,第(4)步的单步移位时间为 $2(t_s+t_w\frac{n^2}{p})\sqrt{p}$ 、运算时间为 n^3/p 。所以在二维网孔上 Cannon 乘法的并行运行时间为 $T_p=4(t_s+t_w\frac{n^2}{p})\sqrt{p}+n^3/p$ 。

MPI 源程序请参见章末附录。

1.3 LU 分 解

从本小节起我们将对 LU 分解等矩阵分解的并行计算做一些简单讨论。在许多应用问题的科学计算中，矩阵的 LU 分解是基本、常用的一种矩阵运算，它是求解线性方程组的基础，尤其在解多个同系数阵的线性方程组时特别有用。

1.3.1 矩阵的 LU 分解及其串行算法

对于一个 n 阶非奇异方阵 $A=[a_{ij}]$，对 A 进行 LU 分解是求一个主对角元素全为 1 的下三角方阵 $L=[l_{ij}]$与上三角方阵 $U=[u_{ij}]$，使 A=LU。设 A 的各阶主子行列式皆非零，U 和 L 的元素可由下面的递推式求出：

$$a_{ij}^{(1)}=a_{ij}$$

$$a_{ij}^{(k+1)}=a_{ij}^{(k)}-l_{ik}u_{kj}$$

$$l_{ik}=\begin{cases}0 & i<k\\ 1 & i=k\\ a_{ik}^{(k)}u_{kk}^{-1} & i>k\end{cases}$$

$$u_{kj}=\begin{cases}0 & k>j\\ a_{kj}^{(k)} & k\le j\end{cases}$$

在计算过程中，首先计算出 U 的第一行元素，然后算出 L 的第一列元素，修改相应 A 的元素；再算出 U 的第二行，L 的第二列…，直至算出 u_{nn} 为止。若一次乘法和加法运算或一次除法运算时间为一个单位时间，则下述 LU 分解的串行算法 18.9 时间复杂度为 $\sum_{i=1}^{n}(i-1)i=(n^3-n)/3=O(n^3)$。

算法 18.9　单处理器上矩阵 LU 分解串行算法

输入：矩阵 $\boldsymbol{A}_{n\times n}$

输出：下三角矩阵 $\boldsymbol{L}_{n\times n}$,上三角矩阵 $\boldsymbol{U}_{n\times n}$

```
Begin
    (1)for k=1 to n do
        (1.1)for i=k+1 to n do
                a[i,k]=a[i,k]/a[k,k]
            end for
        (1.2)for i=k+1 to n do
                for j=k+1 to n do
                    a[i,j]=a[i,j]-a[i,k]*a[k,j]
                end for
            end for
      end for
    (2)for i=1 to n do
        (2.1)for j=1 to n do
                if  (j<i)  then
                    l[i,j]=a[i,j]
                else
                    u[i,j]=a[i,j]
                end if
            end for
        (2.2)l[i,i]=1
      end for
End
```

1.3.2 矩阵 LU 分解的并行算法

在 *LU* 分解的过程中，主要的计算是利用主行 *i* 对其余各行 *j*，(*j*>*i*)作初等行变换，各行计算之间没有数据相关关系，因此可以对矩阵 *A* 按行划分来实现并行计算。考虑到在计算过程中处理器之间的负载均衡，对 *A* 采用行交叉划分：设处理器个数为 *p*，矩阵 *A* 的阶数为 *n*，$m=\lceil n/p \rceil$，对矩阵 *A* 行交叉划分后，编号为 *i*(*i*=0,1,⋯,*p*-1)的处理器存有 *A* 的第 *i*, *i*+*p*,⋯, *i*+(*m*-1)*p* 行。然后依次以第 0,1,⋯,*n*-1 行作为主行，将其广播给所有处理器，各处理器利用主行对其部分行向量做行变换，这实际上是各处理器轮流选出主行并广播。若以编号为 my_rank 的处理器的第 *i* 行元素作为主行，并将它广播给所有处理器，则编号大于等于 my_rank 的处理器利用主行元素对其第 *i*+1,⋯,*m*-1 行数据做行变换，其它处理器利用主行元素对其第 *i*,⋯,*m*-1 行数据做行变换。具体并行算法框架描述如下：

算法 18.10　矩阵 LU 分解的并行算法

输入：矩阵 $A_{n\times n}$

输出：下三角矩阵 $\boldsymbol{L}_{n\times n}$,上三角矩阵 $\boldsymbol{U}_{n\times n}$

```
Begin
    对所有处理器 my_rank(my_rank=0,⋯,p-1)同时执行如下的算法:
    for i=0 to m-1 do
        for j=0 to p-1 do
            (1)if (my_rank=j) then   /*当前处理的主行在本处理器*/
                (1.1) v=i*p+j        /* v 为当前处理的主行号*/
                (1.2)for k=v to n do
                        f[k]=a[i,k]   /*  主行元素存到数组 f 中*/
                    end for
                (1.3)向其它所有处理器广播主行元素
              else   /*当前处理的主行不在本处理器*/
                (1.4)v=i*p+j
                (1.5)接收主行所在处理器广播来的主行元素
              end if
            (2)if (my_rank ≤ j) then
                (2.1)for k=i+1 to m-1 do
                        (i)a[k,v]=a[k,v]/f[v]
                        (ii)for w=v+1 to n-1 do
                            a[k,w]=a[k,w]-f[w]*a[k,v]
                          end for
                    end for
              else
                (2.2)for k=i to m-1 do
                    (i)a[k,v]=a[k,v]/f[v];
                    (ii)for w=v+1 to n-1 do
                        a[k,w]=a[k,w]-f[w]*a[k,v]
                      end for
                    end for
              end if
    end for
```

end for

End

计算完成后，编号为 0 的处理器收集各处理器中的计算结果，并从经过初等行变换的矩阵 A 中分离出下三角矩阵 L 和上三角矩阵 U。若一次乘法和加法运算或一次除法运算时间为一个单位时间，则计算时间为$(n^3-n)/3p$；又 n-1 行数据依次作为主行被广播，通信时间为$(n\text{-}1)(t_s+nt_w)\log p=O(n\log p)$，因此并行计算时间 $T_p=(n^3-n)/3p+(n\text{-}1)(t_s+nt_w)\log p$。

MPI 源程序请参见章末附录。

1.4 QR 分 解

$A=[a_{ij}]$ 为一个 n 阶实矩阵，对 A 进行 QR 分解，就是求一个**非奇异**(Nonsingular)方阵 Q 与上三角方阵 R，使得 $A=QR$。其中方阵 Q 满足：$Q^{\mathrm{T}}=Q^{-1}$，称为**正交矩阵**(Orthogonal Matrix)，因此 QR 分解又称为正交三角分解。

1.4.1 矩阵 QR 分解的串行算法

对 A 进行正交三角分解，可用一系列平面旋转矩阵左乘 A，以使 A 的下三角元素逐个地被消为 0。设要消去的下三角元素为 $a_{ij}(i>j)$，则所用的平面旋转方阵为：

$$Q_{ij}=\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & c & s & & \\ & & -s & c & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}$$

其中，除了$(i,i)(j,j)$元素等于 c，(i,j)元素与(j,i)元素分别等于$-s$ 与 s 以外，其余元素皆与单位方阵 I 的对应元素相等，而 c,s 由下式计算：

$$c=\cos\theta=a_{jj}/\sqrt{a_{jj}^2+a_{ij}^2}$$

$$s=\sin\theta=a_{ij}/\sqrt{a_{jj}^2+a_{ij}^2}$$

其中，θ 为旋转角度。这样，在旋转后所得到的方阵 A' 中，元素 a_{ij}' 为 0，A' 与 A 相比仅在第 i 行、第 j 行上的元素不同：

$$a'_{jk}=c\times a_{jk}+s\times a_{ik}$$

$$a'_{ik}=-s\times a_{jk}+c\times a_{ik} \qquad (k=1,2,...,n)$$

消去 A 的 $r=n(n\text{-}1)/2$ 个下三角元素后得到上三角阵 R，实际上是用 r 个旋转方阵 $Q_1,Q_2,\cdots,Q_r$ 左乘 A，即：

$$R=Q_r\,Q_{r\text{-}1}\cdots Q_1A$$

设 $Q_0=Q_r\,Q_{r\text{-}1}\cdots Q_1$，易知 Q_0 为一正交方阵，则有：

$$R=Q_0A$$

即 $$A = Q_0^{-1}\mathrm{R} = Q_0^{\mathrm{T}}\mathrm{R} = Q\mathrm{R}$$

其中 $Q = Q_0^{-1} = Q_0^T$ 为一正交方阵，而 Q_0 可以通过对单位阵 I 进行同样的变换而得到, 这样可得到 A 的正交三角分解。QR 分解串行算法如下：

算法 18.11　单处理器上矩阵的 QR 分解串行算法算法

输入：矩阵 $A_{n\times n}$，单位矩阵 Q

输出：矩阵 $Q_{n\times n}$，矩阵 $R_{n\times n}$

```
Begin
    (1)for j=1 to n do
        for i=j+1 to n do
            (i)sq=sqrt(a[j,j]*a[j,j]+a[i,j]*a[i,j])
                c= a[j,j]/sq
                s= a[i,j]/sq
            (ii)for k=1 to n do
                aj[k]= c*a[j,k] + s*a[i,k]
                qj[k]=c*q[j,k] + s*q[i,k]
                ai[k]= −s*a[j,k] + c*a[i,k]
                qi[k]= −s*q[j,k] + c*q[i,k]
              end for
            (iii)for k=1 to n do
                a[j,k]=aj[k]
                q[j,k]=qj[k]
                a[i,k]=ai[k]
                q[i,k]=qi[k]
              end for
        end for
     end for
    (2)R=A
    (3)MATRIX_TRANSPOSITION(Q)     /* 对 Q 实施算法 18.1 的矩阵转置*/
End
```

算法 18.11 要对 $n(n\text{-}1)/2$ 个下三角元素进行消去，每消去一个元素要对两行元素进行旋转变换，而对一个元素进行旋转变换又需要 4 次乘法和 2 次加法，所以若取一次乘法或一次加法运算时间为一个单位时间，则该算法的计算时间为 $n(n\text{-}1)/2*2n*6=6n^2(n\text{-}1)=\mathrm{O}(n^3)$。

1.4.2 矩阵 QR 分解的并行算法

由于 QR 分解中消去 a_{ij} 时，同时要改变第 i 行及第 j 行两行的元素，而在 LU 分解中，仅利用主行 $i(i<j)$变更第 j 行的元素。因此 QR 分解并行计算中对数据的划分与分布与 LU 分解就不一样。设处理器个数为 p，对矩阵 A 按行划分为 p 块，每块含有连续的 m 行向量，$m = \lceil n/p \rceil$，这些行块依次记为 $A_0, A_1, \cdots, A_{p\text{-}1}$，分别存放在标号为 0,1,⋯,$p$-1 的处理器中。

在 0 号处理器中，计算开始时，以第 0 行数据作为主行，依次和第 0,1,⋯,m-1 行数据做旋转变换，计算完毕将第 0 行数据发送给 1 号处理器，以使 1 号处理器将收到的第 0 行数据作为主行和自己的 m 行数据依次做旋转变换；在此同时，0 号处理器进一步以第 1 行数据作为主行，依次和第 2,3,⋯,m-1 行数据做旋转变换，计算完成后将第 1 行数据发送给 1 号处理

器；如此循环下去。一直到以第 m-1 行数据作为主行参与旋转变换为止。

在 1 号处理器中，首先以收到的 0 号处理器的第 0 行数据作为主行和自己的 m 行数据做旋转变换，计算完将该主行数据发送给 2 号处理器；然后以收到的 0 号处理器的第 1 行数据作为主行和自己的 m 行数据做旋转变换，再将该主行数据发送给 2 号处理器；如此循环下去，一直到以收到的 0 号处理器的第 m-1 行数据作为主行和自己的 m 行数据做旋转变换并将第 m-1 行数据发送给 2 号处理器为止。然后，1 号处理器以自己的第 0 行数据作为主行，依次和第 1, 2,…,m-1 行数据做旋转变换，计算完毕将第 0 行数据发送给 2 号处理器；接着以第 1 行数据作为主行，依次和第 2,3,…,m-1 行数据做旋转变换，计算完毕将第 1 行数据发送给 2 号处理器；如此循环下去。一直到以第 m-1 行数据作为主行参与旋转变换为止。除了 p-1 号处理器以外的所有处理器都按此规律先顺序接收前一个处理器发来的数据，并作为主行和自己的 m 行数据作旋转变换，计算完毕将该主行数据发送给后一个处理器。然后依次以自己的 m 行数据作主行对主行后面的数据做旋转变换，计算完毕将主行数据发给后一个处理器。

在 p-1 号处理器中，先以接收到的数据作为主行参与旋转变换，然后依次以自己的 m 行数据作为主行参与旋转变换。所不同的是，p-1 号处理器作为最后一个处理器，负责对经过计算的全部数据做汇总。具体并行算法框架描述如下：

算法 18.12　矩阵 QR 分解并行算法

输入：矩阵 $A_{n\times n}$，单位矩阵 Q

输出：矩阵 $Q_{n\times n}$，矩阵 $R_{n\times n}$

Begin

对所有处理器 my_rank(my_rank=0,…, p-1)同时执行如下的算法:

(1)**if** (my_rank=0) **then**　/*0 号处理器*/

(1.1)**for** j=0 **to** m-2 **do**

(i)**for** i=j+1 **to** m-1 **do**

Turnningtransform( )　/*旋转变换*/

end for

(ii)将旋转变换后的 A 和 Q 的第 j 行传送到第 1 号处理器

end for

(1.2)将旋转变换后的 A 和 Q 的第 m-1 行传送到第 1 号处理器

end if

(2)**if** ((my_rank>0) and (my_rank<(p-1))) **then**　/*中间处理器*/

(2.1) **for** j=0 **to** my_rank*m-1 **do**

(i)接收左邻处理器传送来的 A 和 Q 子块中的第 j 行

(ii)**for** i=0 **to** m-1 **do**

Turnningtransform( )　/*旋转变换*/

end for

(iii)将旋转变换后的 A 和 Q 子块中的第 j 行传送到右邻处理器

end for

(2.2) **for** j=0 **to** m-2 **do**

(i)z=my_rank*m

(ii)**for** i=j+1 **to** m-1 **do**

Turnningtransform( )　/*旋转变换*/

end for

(iii)将旋转变换后的 A 和 Q 子块中的第 j 行传送到右邻处理器

end for

(2.3)将旋转变换后的 A 和 Q 子块中的第 *m*-1 行传送到右邻处理器

end if

(3)**if** (my_rank= (*p*-1)) **then**　/*p-1 号处理器*/

(3.1) **for** *j*=0 **to** my_rank**m*-1 **do**

(i)接收左邻处理器传送来的 A 和 Q 子块中的第 *j* 行

(ii)**for** *i*=0 **to** *m*-1 **do**

Turnningtransform( )　/*旋转变换*/

end for

end for

(3.2)**for** *j*=0 **to** *m*-2 **do**

for *i*=*j*+1 **to** *m*-1 **do** *f*

Turnningtransform( )　/*旋转变换*/

end for

end for

end if

End

当所有的计算完成后，编号为 *p*-1 的处理器负责收集各处理器中的计算结果，*R* 矩阵为经旋转变换的 *A* 矩阵，*Q* 矩阵为经旋转变换的 *Q* 的转置。QR 分解并行计算的时间复杂度分析因每个处理器的开始计算时间不同而不同于其它算法，每个处理器都要顺序对其局部存储器中的 *m* 行向量做自身的旋转变换，其时间用 $T_{computer}$ 表示。以 16 阶矩阵为例，4 个处理器对其进行 QR 分解的时间分布图如图.3 所示，这里 $i^{(j)}$表示以接收到的第 *j* 号处理器的第 *i* 行数据作为主行和自己的 *m* 行数据做旋转变换。Δ*t* 表示第 *p*-1 号处理器与第 0 号处理器开始计算时间的间隔。

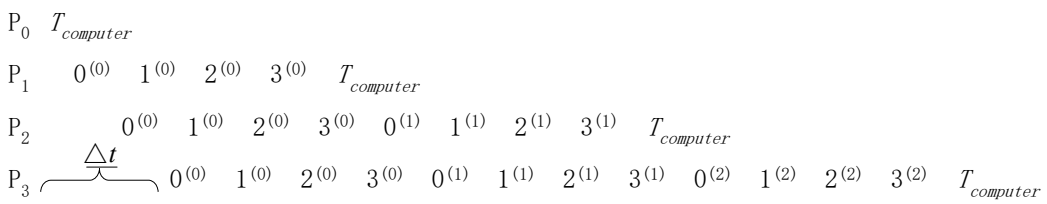

图.3　QR 分解并行计算的时间分布图

此处不妨以第 *p*-1 号处理器为对象分析并行计算时间，若取一次乘法或一次加法运算时间为一个单位时间，由于对一个元素进行旋转变换需要 4 次乘法和 2 次加法，因而需要六个单位时间。要对 *m*(*m*-1)/2 个下三角元素进行消去，每消去一个元素要对两行元素进行旋转变换，故自身的旋转变换时间 $T_{computer}$=6*mn*(*m*-1)；第 *p*-1 号处理器依次接收前 *n*-*m* 行数据作为主行参与旋转变换，其计算时间为(*n*-*m*)**m***2*n**6=12*nm*(*n*-*m*)，通信时间为 2(*n*-*m*)(t_s+ nt_w)；时间间隔Δ*t* 即第 *p*-1 号处理器等待第 0 号处理器第 0 行数据到达的时间，第 0 行数据在第 0 号处理器中参与(*m*-1)行旋转变换，然后被发送给第 1 号处理器，这段计算时间为(*m*-1)*2*n**6，通信时间为 2(t_s+ nt_w)；接着第 0 行数据经过中间(*p*-2)个处理器的旋转变换被发送给第 *p*-1 号处理器，这段计算时间为(*p*-2)*m**2*n**6，通信时间为 4(*p*-2)(t_s+nt_w)，所以Δ*t*=12*n*(*mp*-*m*-1)+(4*p*-6)(t_s+nt_w), 并行计算时间为 T_p=12n^2-18*mn*-6 *nm*^{2}+12n^2*m*-12*n*+(2*n*-2*m*+4*p*-6) (t_s+ nt_w)。

MPI 源程序请参见所附光盘。