第九讲：矩阵计算并行算法

陈俊清

jqchen@math.tsinghua.edu.cn

清华大学数学科学系

May 17, 2013

Part I

矩阵并行计算

Outline

1. 稠密矩阵向量乘并行计算
2. 稠密矩阵乘并行计算
3. BLAS简介

前言

矩阵计算在数值代数中起着最基本的作用：

- 线性代数方程组：Ax＝b；
- 线性最小二乘问题：$\min_{x\in\mathcal{R}^n}\|Ax-b\|_2, b\in\mathcal{R}^m$;
- 矩阵特征值问题：$Ax=\lambda x$;
- 矩阵奇异值分解：$A=U\Sigma V^T$.

消息传递型并行系统的通信模式为：$T=\alpha+\beta N$,其中，α为启动时间，β为传输单位数据所需的时间，N是数据传输量。

Outline

稠密矩阵向量乘

给定稠密矩阵

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

与n维向量$x = [x_1, x_2, ..., x_n]$，计算A与x的乘积$y = [y_1, y_2, ..., y_n]$， 其串行算法如下：

算法9.1,矩阵向量乘串行算法

```
for i=1:n
    y(i)=0;
    for j=1:n
        y(i)= y(i) + a(i,j)*x(j);
    end
end
```

一维块划分下的并行算法

将矩阵A按行分成p块，假设n可被p整除（令$m = n/p$.若不能整除，可将某些块多一行），则第k($0 \le k \le p-1$)块为：

$$A_k = [A_{k,0}, A_{k,1}, \dots, A_{k,p-1}],$$

其中

$$A_{k,j} = \begin{pmatrix} a_{k\times m+1, j\times m+1} & a_{k\times m+1, j\times m+2} & \cdots & a_{k\times m+1, j\times m+m} \\ a_{k\times m+2, j\times m+1} & a_{k\times m+2, j\times m+2} & \cdots & a_{k\times m+2, j\times m+m} \\ \cdots & \cdots & \cdots & \cdots \\ a_{k\times m+m, j\times m+1} & a_{k\times m+m, j\times m+2} & \cdots & a_{k\times m+m, j\times m+m} \end{pmatrix}$$

对应的

$$x_k = [x_{k\times m+1}, x_{k\times m+2}, \dots, x_{k\times m+m}],$$
$$y_k = [y_{k\times m+1}, y_{k\times m+2}, \dots, y_{k\times m+m}],$$

一维块划分并行算法(续)

假设A_k, x_k, y_k都存在进程k上，对应的局部变量为A', x', y'。

$$y_k = A_k x = \sum_{j=0}^{p-1} A_{k,j} x_j = \sum_{j=0}^{p-1} A_{k,(k+j) mod p} x_{(k+j) mod p}$$

在给定进程k上，每步计算中需用的子矩阵A的行标号不变，即需用的A的块实际上都处在本进程上。而在第j步计算的时候需用到x的子向量，其标号比当前进程号大，

$$\{(k+j) mod p - k\} mod p = j.$$

故可以通过每计算一次将x的块在(同列)进程中循环上移一个位置的方式来实现。在计算到第j步的时候，由于x的块总共循环上移了j个位置，所以此时需要用到的x的块恰好已经移到当前进程上。

一维块划分并行算法（续）

记$A'_j = A_{myid,j}$.具体算法如下：

算法9.2 稠密矩阵向量乘一维行块划分并行算法

1. Set $w = x'; y' = A'_{myid} x'$;
2. for j = 1 to p-1 do
   Send w to $P_{(myid-1+p)modp}$;
   Recv w from $P_{(myid+1)modp}$;
   $y' = y' + A'_{(myid+j)modp} w$;
   endfor

注： 为简单起见，可以让每个进程对向量x作一到多广播（多对多广播），然后再计算。

一维块划分并行算法（续）

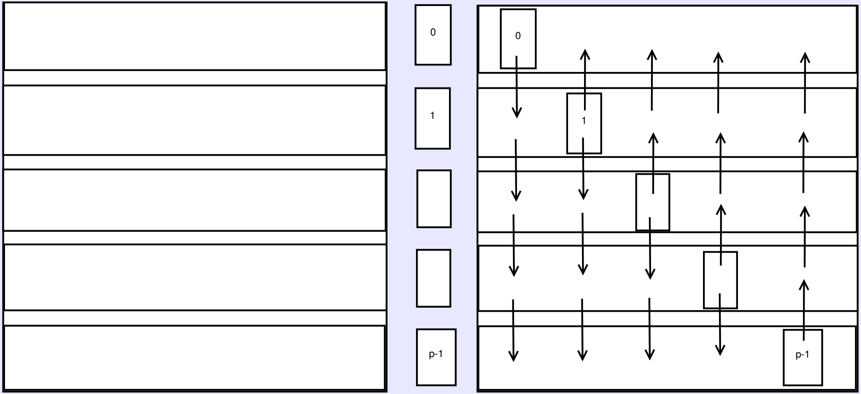


Figure: 矩阵划分与向量广播

一维块划分并行算法（续）

- 计算时间：每个进程都进行了一次$n/p \times n$矩阵与n维向量的乘积，设一个浮点操作的时间为c，则总计算时间为：

$$2cn^2/p$$

- 通信时间：需要进行$p-1$次通信，每次通信的数据量是n/p,故总通信时间：

$$(p-1)(\alpha+\beta n/p)$$

一维块划分并行算法（续）

- 总时间：

$$T_p = 2cn^2/p + (p-1)\alpha + (p-1)\beta n/p$$

- 串型计算的时间为：

$$2cn^2$$

- 并行效率：

$$E_p = \frac{T_1}{pT_p} = \frac{2cn^2}{2cn^2 + p(p-1)\alpha + (p-1)n\beta}$$

注：试利用上次课的方法分析其可扩展性，求其等效率函数。

二维块划分下的并行算法

设一共有$q=p^2$个进程，组成$p\times p$的进程网格，将矩阵A分成如下$p\times p$块：

$$A=\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,p-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,p-1} \\ \cdots & \cdots & \cdots & \cdots \\ A_{p-1,0} & A_{p-1,1} & \cdots & A_{p-1,p-1} \end{pmatrix}$$

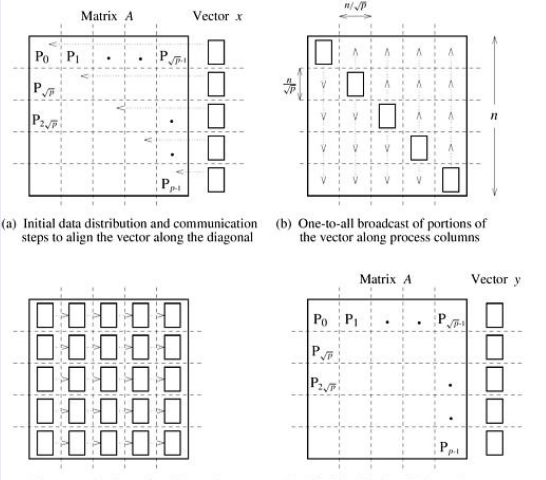
$\mathbf{x}_k,\mathbf{y}_k$都按一维的划分方式分成p个块。假设$A_{k,j}$存储在进程$P_{k,j}$上的局部变量$A'$中，$\mathbf{x}_k,\mathbf{y}_k$存储在进程$P_{k,0}$的局部变量$\mathbf{x}',\mathbf{y}'$中，由于在进程$P_{k,j}$上需要计算$\mathbf{y}_k$的一个部分和$A_{k,j}\mathbf{x}_j$,所以开始需要进行对齐的工作：
将$P_{k,0}$上的$\mathbf{x}_k$传到$P_{k,k}$上，然后在第k列进行广播$\mathbf{x}_k$。 各进程计算后，每行求和归约到第0列进程中得到计算结果。

二维块划分下的并行算法（续）

数据的初始存储位置:

$$
\begin{array}{cccc}
P_{0,0} : A_{0,0}, \mathbf{x}_0 & P_{0,1} : A_{0,1} & \cdots & P_{0,p-1} : A_{0,p-1} \\
P_{1,0} : A_{1,0}, \mathbf{x}_1 & P_{1,1} : A_{1,1} & \cdots & P_{1,p-1} : A_{1,p-1} \\
\cdots & \cdots & \cdots & \cdots \\
P_{p-1,0} : A_{p-1,0}, \mathbf{x}_{p-1} & P_{p-1,1} : A_{p-1,1} & \cdots & P_{p-1,p-1} : A_{p-1,p-1}
\end{array}
$$

二维块划分下的并行算法（续）



(a) Initial data distribution and communication steps to align the vector along the diagonal

(b) One-to-all broadcast of portions of the vector along process columns

(c) All-to-one reduction of partial results

(d) Final distribution of the result vector

二维块划分下的并行算法（续）

算法9.3 进行稠密矩阵向量乘$y = Ax$的二维块划分并行算法:

1. myrow = myrank mod p; mycol = myrank/p;
2. if mycol == 0 then
   Send $\mathbf{x}'$ to $P_{myrow,myrow}$;
   if myrow == mycol then
   Recv $\mathbf{x}'$ from $P_{myrow,0}$;
3. Broadcast $\mathbf{x}'$ from $P_{mycol,mycol}$ in comlumn;
   Compute $\mathbf{y}' = A'\mathbf{x}'$;
   Reduce in each row for $\mathbf{y}'$ to $P_{myrow,0}$。

二维块划分下的并行算法（续）

- 对齐通信时间： $\alpha + \beta n/p$
- 广播时间： $(\alpha + \beta n/p)\log p$
- 归约时间： $(\alpha + c + \beta n/p)\log p$
- 计算时间： $2n^2c/p^2$

总时间为

$$T_q = (\alpha + \beta n/p)(2\log p + 1) + c\log p + 2n^2c/p^2$$

串型计算的时间为：

$$T_1 = 2cn^2$$

同样，可以计算效率与等效率函数。试比较在同样多处理器下前述两种方法的快慢。

Outline

稠密矩阵乘的串行算法

给定两个n阶矩阵A, B,矩阵乘法指计算$C = A \times B$,当前已经存在许多矩阵乘的快速算法，其所需的浮点数少于$O(n^{log7})$.
下面以简单的$O(n^3)$矩阵乘串行算法为基础进行并行算法的介绍与分析。

算法9.4 稠密矩阵乘ijk形式的串行算法

```
for i=1:n
    for j=1:n
        c(i,j)=0;
        for k=1:n
            c(i,j)=c(i,j)+a(i,k)b(k,j);
        end
    end
end
```

分块矩阵相乘

一个$n \times n$的矩阵A可以视为$q \times q$的分块阵$A_{ij}, (0 \le i, j \le q-1)$， 每个子矩阵的大小为$n/q \times n/q$。 我们可以将前述串行算法写为：

算法9.4.1 稠密分块矩阵乘串行算法

```
for i=0:q-1
     for j=0:q-1
         Cij=0
         for k=0: q-1
              Cij=Cij+Aik * Bkj
         end
       end
end
```

基于行列划分的一维并行算法

假设一共有p个进程，将矩阵A按行分成p个块，B按列分成p个块：

$$A = \begin{pmatrix} A_0 \\ A_1 \\ \dots \\ A_{p-1} \end{pmatrix}, B = [B_0, B_1, ..., B_{p-1}],$$

每块包含若干行。为使负载平衡，应使每块中的行数尽量相等。A_k, B_k分别存储在进程P_k的A', B'中， 将C分成$p \times p$块，且将$C_{i,j}$存储在P_i的C'_j中。
将$C_{i,j} = A_i \times B_j$的计算放在P_i上进行，这样在初始数据分布时， 数据A_i已经在进程P_i上，但P_i上只有B_i,其它的B_j在其它进程上， 计算的时候可以通过循环左移的方式将需要的B_j移到当前进程。
数据分布：

$$P_0 : A_0, B_0;\ P_1 : A_1, B_1;\ \dots\ P_{p-1} : A_{p-1}, B_{p-1}.$$

基于行列划分的一维并行算法(续)

算法9.5 稠密矩阵乘$C = A \times B$的行列划分并行算法

1. for k=0:p-1

    Compute $C'_{(myrank+k) mod p} = A' \times B'$;

    Send B' to $P_{(myrank-1+p) mod p}$;

    Recv B' from $P_{(myrank+1) mod p}$;

    endfor

基于行列划分的一维并行算法（续）

算法的计算量与数据交换量：

- 数据交换量：每次交换数据为 $n \times n/p$,交换次数为 $p-1$次, 故总数据交换量为：

$$DTA = (p-1) \times n \times n/p = (p-1)n^2/p$$

- 计算量：每次计算一个 $n/p \times n$与$n \times n/p$阶矩阵的乘积，共需p次

$$CA = p \times n/p \times n \times n/p = n^3/p$$

运行时间(忽略最后一次数据对齐)：

$$T_p = 2n^3c/p + (p-1)(\alpha + \beta n^2/p)$$

基于行行划分的一维并行算法

将B也进行划分：

$$B = \begin{pmatrix} B_0 \\ B_1 \\ \dots \\ B_{p-1} \end{pmatrix}$$

并将A作进一步划分$A_i = [A_{i,0}, A_{i,1}, \dots, A_{i,p-1}]$,其中$A_{i,j}$的列数与$B_j$的行数对应相等。记$C_i = [C_{i,0}, C_{i,1}, \dots, C_{i,p-1}]$,将$A_{i,j}, B_i, C_i$分别存储在$P_i$的$A'_j, B', C'$中，计算过程可用如下公式表示：

$$C_i = \sum_{j=0}^{p-1} A_{i,j} B_j = \sum_{j=0}^{p-1} A_{i,(i+j) mod p} B_{(i+j) mod p},$$

基于行行划分的一维并行算法（续）

可以类似于行列划分的方法对B进行循环上移，使得每步需要用到B的块刚好移动到当前进程。具体算法如下：

算法9.6，行行划分的一维并行算法

1. Set $C' = 0$;
2. for j=0 to p-1 do
   Compute $C' = C' + A'_{(myrank+j) mod p} \times B'$;
   Send B' to $P_{(myrank-1+p) mod p}$;
   Recv B' from $P_{(myrank+1) mod p}$;
   endfor

基于列列划分的一维并行算法

将矩阵A，B分别按列划分为$A = [A_0, A_1, ..., A_{p-1}], B = [B_0, B_1, ..., B_{p-1}]$，并将$B_j$进一步按行划分为：

$$B_j = \begin{pmatrix} B_{0,j} \\ B_{1,j} \\ ...B_{p-1,j} \end{pmatrix}$$

其中A_i的列数等于$B_{i,j}$的行数。对C进行与B相对应的划分，得到$C = [C_0, C_1, ..., C_{p-1}]$,于是

$$C_j = A \times B_j = \sum_{i=0}^{p-1} A_i \times B_{i,j} = \sum_{i=0}^{p-1} A_{(i+j)modp} \times B_{(i+j)modp,j}$$

基于列列划分的一维并行算法（续）

算法9.7，稠密矩阵乘的列列划分并行算法

1. Set $C' = 0$;
2. for i=0 to p-1 do

   Compute $C' = C' + A' \times B'_{(myrank+i)modp}$;

   Send A' to $P_{(myrank-1+p)modp}$;

   Recv A' from $P_{(myrank+1)modp}$;

   endfor

基于列行划分的一维并行算法

将矩阵A按列划分为$A = [A_0, A_1, ..., A_{p-1}]$,将矩阵B按行划分为:

$$B = \begin{pmatrix} B_0 \\ B_1 \\ \cdots \\ B_{p-1} \end{pmatrix}$$

将B_i进一步划分为$B_i = [B_{i,0}, B_{i,1}, \ldots, B_{i,p-1}]$,将C按与$B_i$相对应的方式划分为$C = [C_0, C_1, ..., C_{p-1}]$,则:

$$C_j = \sum_{i=0}^{p-1} A_i \times B_{i,j} = \sum_{i=0}^{p-1} A_{(j-i) mod p} \times B_{(j-i) mod p, j}$$

基于列行划分的一维并行算法（续）

进行第i步计算时，需用到的$A_{(j-i)modp}$与$B_{(j-i)modp,j}$都处于进程$P_{(j-i)modp}$上， 计算得到的是C_j的一个部分和，而该部分和应存储在进程P_j上， 换言之，在第i步，进程P_j上计算得到了应该存储在进程$P_{(i+j)modp}$的一个部分和$A_j \times B_{(j+i)modp}$， 我们将该算法具体描述如下：

算法9.8，稠密矩阵乘的列行划分并行算法

1. Set $C' = A' \times B'$;
2. for i=1 to p-1 do
   Compute $W = A' \times B'_{(myrank+i)modp}$;
   Send W to $P_{(myrank+i)modp}$;
   Recv W from $P_{(myrank-i+p)modp}$;
   Compute $C' = C' + W$;

   endfor

基于多对多广播的二维并行算法

假设现在要在$q = p \times p$个进程上进行计算，且这q个进程组织为$p \times p$二维环状网格， 同时对矩阵A、B、C都进行二维块划分，各分成$p \times p$块。例如：

$P_{0,0} : A_{0,0}, B_{0,0}, C_{0,0}$	$P_{0,1} : A_{0,1}, B_{0,1}, C_{0,1}$	$P_{0,2} : A_{0,2}, B_{0,2}, C_{0,2}$
$P_{1,0} : A_{1,0}, B_{1,0}, C_{1,0}$	$P_{1,1} : A_{1,1}, B_{1,1}, C_{1,1}$	$P_{1,2} : A_{1,2}, B_{1,2}, C_{1,2}$
$P_{2,0} : A_{2,0}, B_{2,0}, C_{2,0}$	$P_{2,1} : A_{2,1}, B_{2,1}, C_{2,1}$	$P_{2,2} : A_{2,2}, B_{2,2}, C_{2,2}$

Table: 稠密矩阵二维块划分并行算法中数据存储位置

基于多对多广播的二维并行算法(续)

为计算进程$P_{i,j}$上的子矩阵$C_{i,j}$,最直接的方法是将所有的$A_{i,k}$与$B_{k,j}$都收集到进程$P_{i,j}$上，之后将所有$A_{i,k}B_{k,j}(k=0,...,p-1)$累加起来就可得到结果:

$$C_{i,j}=\sum_{k=0}^{p-1}A_{i,k}B_{k,j}$$

算法9.9，稠密矩阵乘多对多广播并行算法

1. Set $U = A'; V = B'; Z_{mycol} = A'; W_{myrow} = B'$;
2. for k=1 to p-1 do
   Send U to $P_{myrow,(mycol+k) mod p}$;
   Recv U from $P_{myrow,(mycol-k+p) mod p}$;
   Set $Z_{(mycol+k) mod p} = U$;
   endfor
3. for k=1 to p-1 do
   Send V to $P_{(myrow+k) mod p, mycol}$;
   Recv V from $P_{(myrow-k+p) mod p, mycol}$;
   Set $W_{(myrow+k) mod p} = V$;
   endfor
4. Set $C' = 0$;
5. for k=0 to p-1 do $C' = C' + Z_k \times W_k$.

基于多对多广播的二维并行算法（续）

- 通信时间：
$$2(\alpha + \beta n^2/p^2)(p-1)$$
- 计算时间：
$$2(n/p)^3 \times p \times c = 2cn^3/p^2$$
- 并行计算总时间
$$T_q = 2cn^3/p^2 + 2(\alpha + \beta n^2/p^2)(p-1)$$
- 串行时的时间：
$$T_1 = 2cn^3$$
- 并行效率：
$$E_q = \frac{T_1}{p^2 T_q}$$

正方形网格上的Fox算法

多对多广播并行算法的两个显著缺点：

- 每个进程上需要存储 p个A和p个B的块，这些块所需总存储量为$2n^2/p$;
- 算法中要先进行两次多对多广播后才能进行计算，必须严格遵守先后顺序，不利于计算与通信的重叠。

将计算公式写为：

$$C_{i,j} = \sum_{k=0}^{p-1} A_{i,k} B_{k,j} = \sum_{k=0}^{p-1} A_{i,(i+k) mod p} B_{(i+k) mod p,j}.$$

Fox算法基于上式中的后一个等式。

正方形网格上的Fox算法(续)

在Fox算法中，矩阵乘一共分为p步。

- 第k（$0 \leq k \leq p-1$）步上所有行号i相同的进程都要用到$A_{i,(i+k)modp}$，只与i、k有关，与j无关。 从而，第k步时先把$A_{i,(i+k)modp}$广播到同行上其它进程即可。
- 进程$P_{i,j}$上需用到的B的块的两个下标中，列标为j，所以这些块必定处于同列进程中。 同时行标号为$(i+k)modp$,这说明该B的块位于当前进程下方与$P_{i,j}$相距k的进程之上，这个距离与j无关， 可以通过每计算一次将B的块在同列进程中循环上移一个位置来实现。

正方形网格上的Fox算法(续)

k=0前	k=0后
$A_{0,0}, B_{0,0}$; $A_{0,1}, B_{0,1}$; $A_{0,2}, B_{0,2}$	$A_{0,0}, B_{1,0}$; $A_{0,1}, B_{1,1}$; $A_{0,2}, B_{1,2}$
$A_{1,0}, B_{1,0}$; $A_{1,1}, B_{1,1}$; $A_{1,2}, B_{1,2}$	$A_{1,0}, B_{2,0}$; $A_{1,1}, B_{2,1}$; $A_{1,2}, B_{2,2}$
$A_{2,0}, B_{2,0}$; $A_{2,1}, B_{2,1}$; $A_{2,2}, B_{2,2}$	$A_{2,0}, B_{0,0}$; $A_{2,1}, B_{0,1}$; $A_{2,2}, B_{0,2}$

k=1后	k=2 后
$A_{0,0}, B_{2,0}$; $A_{0,1}, B_{2,1}$; $A_{0,2}, B_{2,2}$	$A_{0,0}, B_{0,0}$; $A_{0,1}, B_{0,1}$; $A_{0,2}, B_{0,2}$
$A_{1,0}, B_{0,0}$; $A_{1,1}, B_{0,1}$; $A_{1,2}, B_{0,2}$	$A_{1,0}, B_{1,0}$; $A_{1,1}, B_{1,1}$; $A_{1,2}, B_{1,2}$
$A_{2,0}, B_{1,0}$; $A_{2,1}, B_{1,1}$; $A_{2,2}, B_{1,2}$	$A_{2,0}, B_{2,0}$; $A_{2,1}, B_{2,1}$; $A_{2,2}, B_{2,2}$

Table: Fox算法数据分布变化

正方形网格上的Fox算法(续)

算法9.10,Fox算法

1. Set $C' = 0$;
2. for k=0 to p-1 do

    if mycol == (myrow+k)mod p then Set $W = A'$;
    Broadcast W from $P_{myrow,(myrow+k)modp}$ in the same row;
    Compute $C' = C' + W \times B'$;
    Send B' to $P_{(myrow-1+p)modp,mycol}$;
    Recv B' from $P_{(myrow+1)modp,mycol}$;

    endfor

注：在$p-1$步之后，数据分布与初始分布是一致的，若只关心结果，最后一步的移动可以忽略。

正方形网格上的Cannon算法

Fox算法每一步都需要在行中进行广播，Cannon算法通过对A的块在行中循环移动来进一步减少由于广播引起的开销，同时也进一步增加了计算与通信的重叠程度。 该算法基于如下公式：

$$C_{i,j} = \sum_{k=0}^{p-1} A_{i,(i+j+k)modp} B_{(i+j+k)modp,j}$$

当k取遍0...$p-1$后，不难验证$(i+j+k)modp$也取遍$0,...,p-1$，从而说明了上述公式的正确性。

正方形网格上的Cannon算法（续）

在第$k=0$步，进程$P_{i,j}$上计算的是

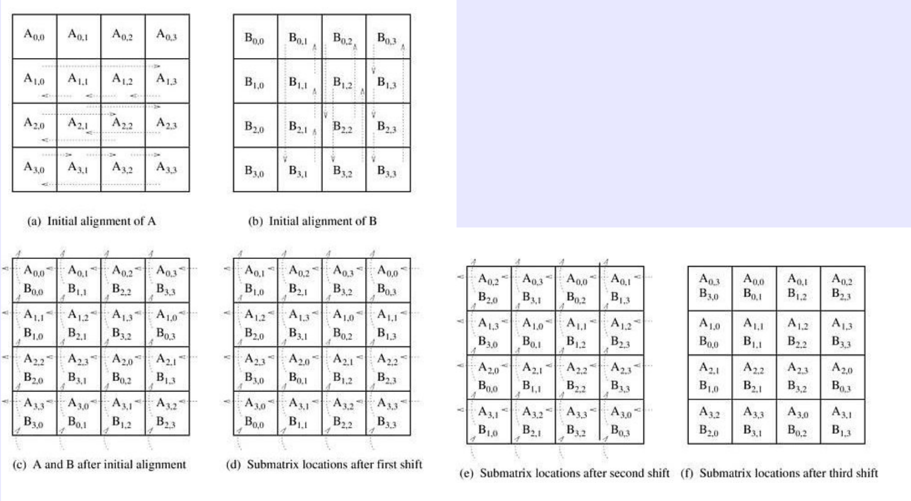
$$A_{i,(i+j)modp}B_{(i+j)modp,j},$$

而这两个块事先都不在当前进程上，那么在计算的时候要先将$A_{i,(i+j)modp}$从$P_{i,(i+j)modp}$传给$P_{i,j}$，$B_{(i+j)modp,j}$从$P_{(i+j)modp,j}$传给$P_{i,j}$。即计算前有如下的对齐过程：

- $A_{i,j}$循环左移i个位置；
- $B_{i,j}$循环上移j个位置。

$k=1,...,p-1$步的计算中，$A_{i,(i+j+k)modp}$行指标不变，列指标依次加1，$B_{(i+j+k)modp,j}$列指标不变，行指标依次加1，故可以通过行的循环左移与列的循环上移来实现。

正方形网格上的Cannon算法（续）



算法9.11 Cannon算法

1 Send A' to $P_{myrow,(mycol-myrow+p)modp}$;
Recv A' from $P_{myrow,(myrow+mycol)modp}$;
Send B' to $P_{(myrow-mycol+p)modp,mycol}$;
Recv B' from $P_{(myrow+mycol)modp,mycol}$;

2 Set $C'=0$;

3 for k=0 to p-1 do
  Compute $C'=C'+A'\times B'$;
  Send A' to $P_{myrow,(mycol-1+p)modp}$;
  Recv A' from $P_{myrow,(mycol+1)modp}$;
  Send B' to $P_{(myrow-1+p)modp,mycol}$;
  Recv B' from $P_{(myrow+1)modp,mycol}$;
endfor

正方形网格上的Cannon算法（续）

在第$p-1$步结束后，进程$P_{i,j}$上存储的数据恢复为$A_{i,(i+j)modp}, B_{(i+j)modp,j}$，在需要使得计算结束后$P_{i,j}$上存储$A_{i,j}, B_{i,j}$的情况下，还需要一个反对齐的过程：

4. Send A' to $P_{myrow,(myrow+mycol)modp}$;
   Recv A' from $P_{myrow,(mycol-myrow+p)modp}$;
   Send B' to $P_{(myrow+mycol)modp,mycol}$;
   Recv B' from $P_{(myrow-mycol+p)modp,mycol}$;

三维正方形网格上的DNS算法

假设现在有$q = p^3$个进程，组织成$p \times p \times p$三维网格，不妨设n能被p整除， 且A、B、C各按二维方式分成$p \times p$块，$A_{i,j}$、$B_{i,j}$分别存储在$P_{i,j,0}$上的A', B'中， 最后结果存在$P_{i,j,0}$上的C'中,对DNS算法，矩阵乘法在$P_{i,j,k}$上计算$A_{i,k}B_{k,j}$， 之后将$P_{i,j,k}$上的部分和归约到$P_{i,j,0}$上得以实现。
首先，由于初始数据$A_{i,k}$存储在$P_{i,k,0}$上，为使所有进程$P_{i,j,k}$上都拥有$A_{i,k}$，先将$A_{i,k}$传到$P_{i,k,k}$上， 然后再广播到所有$P_{i,j,k}$上。$B_{k,j}$存在$P_{k,j,0}$上，先将其传到$P_{k,j,k}$上，然后再广播到所有$P_{i,j,k}$上。

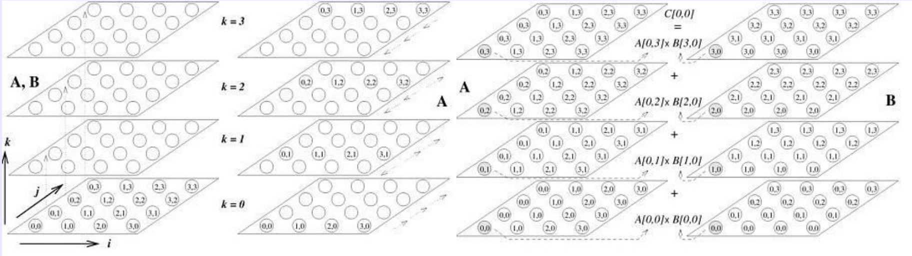
三维正方形网格上的DNS算法（续）

当前进程为$P_{i,j,k}$:

算法9.12，DNS算法

1. if k==0 then Send A' to $P_{i,j,j}$;
   if k==j then Recv A' from $P_{i,j,0}$;
   if k==0 then Send B' to $P_{i,j,i}$;
   if k==i then Recv B' from $P_{i,j,0}$;
2. Broadcast A' from $P_{i,j,j}$ to $P_{i,0:p-1,j}$;
   Broadcast B' from $P_{i,j,i}$ to $P_{0:p-1,j,i}$;
3. Compute $C' = A' \times B'$;
4. Reduce C' in $P_{i,j,0:p-1}$ to $P_{i,j,0}$;

DNS算法（续）



(a) Initial distribution of A and B (b) After moving $A[i,j]$ from $P_{i,j,0}$ to $P_{i,j,j}$ (c) After broadcasting $A[i,j]$ along j axis (d) Corresponding distribution of B

稠密矩阵的算法的进一步说明

基于一维划分的并行算法:

- 行列划分
- 行行划分
- 列行划分
- 列列划分

若考虑方阵乘积，它们具有相同的计算量和通信量，但是对一般矩阵乘来说，通信量各不相同，可以根据具体情况选择具体的算法（如$A = (a_{i,j})_{m\times n}, B = (b_{i,j})_{n\times k}$）。
二维、三维算法:

- Fox算法
- Cannon算法
- DNS算法

矩阵乘算法的实现

从多对多广播到Fox算法，Cannon算法，实现计算与通信的重叠，这也是优化一个算法的有效途径。

实现算法的要点：

- 利用MPI的进程拓扑结构来实现通信(1d,2d,3d笛卡尔拓扑)；
- 数据分配考虑负载平衡。

Outline

1. 稠密矩阵向量乘并行计算
2. 稠密矩阵乘并行计算
3. BLAS简介

概述

BLAS(basic Linear Algebra Subprograms)是一组基本线性代数子程序，用以实现基本的向量和矩阵运算，其下载地址：

`http://www.netlib.org/blas/`

也可以直接网络安装。
BLAS支持4种数据类型：单精度（real）、双精度（double）、单精度复数（real complex）、双精度复数（double complex），在 BLAS中用首字母表示，分别对应的首字母为S、D、C、Z。
按计算量的大小分为三个层次： Level 1 BLAS、Level 2 BLAS、Level 3 BLAS。

Level 1 BLAS

Level 1 BLAS包含向量和向量、向量和标量之间的运算。

- 向量内积与模:

$$\mathbf{x}^T\mathbf{y}, \mathbf{x}^H\mathbf{y}, \|\mathbf{x}\|_1, \|\mathbf{x}\|_2, \dots$$

  相关子程序有：DDOT，DDOTU，DDOTC，DNRM2，DASUM等;
- 向量、标量运算:

$$\mathbf{x} := \alpha\mathbf{x}, \mathbf{y} := \mathbf{x}, \mathbf{x}\text{与}\mathbf{y}\text{交换}, \mathbf{y} := \alpha\mathbf{x} + \mathbf{y}$$

  相关子程序有：DSCAL，DCOPY，DSWAP，DAXPY。
- 平面旋转变换， 相关子程序有DROT,DROTG,DROM,DROTMG.

Level 2 BLAS

Level 2 BLAS包含矩阵和向量之间的运算。

- 矩阵乘向量:

$$\mathbf{y} := \alpha A\mathbf{x} + \beta\mathbf{y}, \mathbf{y} := \alpha A^T\mathbf{x} + \beta\mathbf{y}, \mathbf{y} : \alpha\bar{A}^T\mathbf{x} + \beta\mathbf{y}$$

  $\mathbf{x}, \mathbf{y}$表示向量, α, β表示标量, A表示矩阵。
- 秩1、秩2修正:

$$A := \alpha\mathbf{x}\mathbf{y}^T + A, A := \alpha\mathbf{x}\bar{\mathbf{y}}^T + A$$
$$H := \alpha\mathbf{x}\bar{\mathbf{x}}^T + H, H := \alpha\mathbf{x}\bar{\mathbf{y}}^T + \bar{\alpha}\mathbf{y}\bar{\mathbf{x}}^T + H$$

  其中H表示Hermitian矩阵。
- 三角形方程组求解:

$$\mathbf{x} := T^{-1}\mathbf{x}, \mathbf{x} := T^{-T}\mathbf{x}, \mathbf{x} := \bar{T}^{-T}\mathbf{x}$$

Level 2 BLAS

Level 2 BLAS子程序名称的最后一个或两个字母表示运算类型;

- MV: 矩阵向量乘
- R:秩1修正
- R2:秩2修正
- SV:解线性方程组

中间两个字母表示矩阵类型：如GE与GB分别表示普通矩阵与普通带状矩阵， HE，HP，HB分别表示普通Hermite阵、压缩存储的Hermite阵和带状Hermite阵 SY，SP与SB分别表示对称阵，压缩对称阵和带状对称阵，TR、TP、TB表示三角阵、压缩三角阵和带状三角阵。
例如：DGEMV计算普通矩阵向量乘，DTRSV求解三角方程组。

Level 3 BLAS

Level 3 BLAS针对矩阵与矩阵之间的运算。

- 矩阵乘积

$$C := \alpha AB + \beta C, C := \alpha A^T B + \beta C;$$
$$C := \alpha AB^T + \beta C, C := \alpha A^T B^T + \beta C$$

- 对称矩阵秩k，秩2k修正

$$C := \alpha AA^T + \beta C, C := \alpha A^T A + \beta C,$$
$$C := \alpha AB^T + \alpha BA^T + \beta C, C := \alpha A^T B + \alpha B^T A + \beta C.$$

- 矩阵与三角形矩阵的乘积：

$$B := \alpha TB, B := \alpha T^T B, B := \alpha BT, B := \alpha BT^T$$

- 求解含有多个右端项的三角方程组：

$$B := \alpha T^{-1} B, B := \alpha T^{-T} B, B := \alpha BT^{-1}, B := \alpha BT^{-T}.$$

Level 3 BLAS

命名规则与Level 2 BLAS类似，最后的2个或3个字母表示运算类型

- MM:矩阵乘
- RK：秩k修正
- R2K：秩2k修正
- SM：三角矩阵方程求解

中间两个字母表示矩阵类型，GE、SY、HE、TR分别表示普通、对称、Hermite和三角阵。

例如DGEMM表示普通矩阵乘，而DSYMM表示对称矩阵乘等等。

使用BLAS库的原则：尽可能使用Level 3的BLAS，其次使用Level 2的BLAS。

作业

分析Cannon算法的效率。