



第六讲

矩阵计算并行算法

主要内容

- 并行算法基础知识
- 矩阵向量乘积的并行算法
- 矩阵矩阵乘积的并行算法
- 矩阵的 LU 分解并行算法
- 下三角线性方程组的并行算法

并行算法基础知识

■ 一些基本概念

- 加速比

$$S_p(q) = \frac{T_s}{T_p(q)}$$

其中 T_s 串行程序运行时间, $T_p(q)$ 为 q 个进程的运行时间

- 并行效率

$$E_p(q) = \frac{S_p(q)}{q}$$

程序性能优化

■ 串行程序性能优化 —— 并行程序性能优化的基础

- 调用高性能库。如：BLAS、LAPACK、FFTW
- 选择编译器优化选项：-O2、-O3
- 合理定义数组维数
- 注意嵌套循环次数：数据访问的空间局部性和时间局部性
- 循环展开 例：ex4performance.c
- 数据分块

程序性能优化

■ 并行程序性能优化

- 设计好的并行算法和通信模式
- 减少通信次数、提高通信粒度
- 多进程通信时尽量使用高效率的聚合通信算法
- 负载均衡
- 减少进程的空闲时间
- 通信与计算的重叠
- 通过引入重复计算来减少通信

矩阵并行算法

■ 一些记号和假定

- 假设有 p 个处理器，每个处理器上运行一个进程
- P_j 表示第 j 个处理器， P_{myid} 表示当前的处理器
- $\text{send}(x; j)$ 表示在 P_{myid} 中把数据块 x 发送给 P_j 进程
- $\text{recv}(x; j)$ 表示从 P_j 进程接收数据块 x
- $i \bmod p$ 表示 i 对 p 取模运算

程序设计与机器实现是密不可分的，计算结果的好坏与编程技术有很大的关系，尤其是在并行计算机环境下，开发高质量的程序对发挥计算机的性能起着至关重要的作用

主要内容

- 并行算法基础知识
- 矩阵向量乘积的并行算法
- 矩阵矩阵乘积的并行算法
- 矩阵的 LU 分解并行算法
- 下三角线性方程组的并行算法

矩阵向量乘积

$$y = Ax$$

$$A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^n$$

■ 串行算法

● 实现方法一：i-j 循环

```
for i=1 to m
  y(i)=0.0
  for j=1 to n
    y(i)=y(i)+A(i,j)*x(j)
  end for
end for
```


矩阵向量乘积

■ 串行算法

● 实现方法二：j-i 循环

```
y=0 % 先赋初值
for j=1 to n
    for i=1 to m
        y(i)=y(i)+A(i,j)*x(j)
    end for
end for
```

例：ex4matvec.f

矩阵向量乘积

■ 并行算法一

- 矩阵的划分方法：按行划分和按列划分

- 按行划分并行算法

将矩阵 A 按行划分成如下的行块子矩阵

$$A = [A_0^T, A_1^T, \dots, A_{p-1}^T]^T$$

则 $Ax = [A_0^T, A_1^T, \dots, A_{p-1}^T]^T x$
 $= [(A_0x)^T, (A_1x)^T, \dots, (A_{p-1}x)^T]$

将 A_i 存放在结点 P_i 中，每个结点计算 $A_i x$ ，最后调用 **MPI_GATHER** 或 **MPI_GATHERV** 即可

矩阵向量乘积

■ 并行算法二

● 按列划分并行算法

将矩阵 A 按列划分，并对 x 也做相应的划分

$$A = [A_0, A_1, \dots, A_{p-1}], \quad x = (x_0^T, x_1^T, \dots, x_{p-1}^T)^T$$

其中 x_i 的长度与 A_i 的列数相同，则有

$$Ax = \sum_{i=0}^{p-1} A_i x_i^T$$

将 A_i 和 x_i 存放在结点 P_i 中，每个结点计算 $A_i x_i^T$ ，
最后调用 **MPI_REDUCE** 或 **MPI_ALLREDUCE** 即可

矩阵向量乘积示例

■ 例：按列划分，用 p 个进程并行计算矩阵向量乘积，其中

$$A = (a_{ij}) \in \mathbb{R}^{n \times n}, \quad x = [1, 1, \dots, 1]^T \in \mathbb{R}^n,$$
$$a_{ij} = \frac{1}{i + j - 1}$$

示例程序：ex4matvec.f

上机作业

● 上机作业：

- 1、编写按行划分计算矩阵向量乘积的通用并行程序
- 2、按列划分，编写通用并行程序计算上面的乘积

主要内容

- 并行算法基础知识
- 矩阵向量乘积的并行算法
- 矩阵矩阵乘积的并行算法
- 矩阵的 LU 分解并行算法
- 下三角线性方程组的并行算法

矩阵矩阵乘积

$$C = A * B$$

$$A \in \mathbb{R}^{m \times l}, B \in \mathbb{R}^{l \times n}$$

■ 串行算法一：i-j-k 循环

```
for i=1 to m
  for j=1 to l
    C(i,j)=0
    for k=1 to n
      C(i,j)=C(i,j)+A(i,k)*B(k,j)
    end for
  end for
end for
```

矩阵矩阵乘积

$$C = A * B$$

$$A \in \mathbb{R}^{m \times l}, B \in \mathbb{R}^{l \times n}$$

■ 串行算法二：j-k-i 循环

```
C=0
for j=1 to l
  for k=1 to n
    for i=1 to m
      C(i,j)=C(i,j)+A(i,k)*B(k,j)
    end for
  end for
end for
```


并行矩阵乘积

$$C = A * B$$

$$A \in \mathbb{R}^{m \times l}, B \in \mathbb{R}^{l \times n}$$

■ 假定：

m, l, n 均能被 p 整除，其中 p 为进程个数

■ 基于 A 、 B 的不同划分，矩阵乘积的并行算法可分为

- 行列划分
- 行行划分
- 列列划分
- 列行划分

行列划分

■ 行列划分：A 按行划分、B 按列划分

$$AB = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{p-1} \end{bmatrix} [B_0 \ B_1 \ \cdots \ B_{p-1}] = \begin{bmatrix} A_0 B_0 & A_0 B_1 & \cdots & A_0 B_{p-1} \\ A_1 B_0 & A_1 B_1 & \cdots & A_1 B_{p-1} \\ \vdots & & \ddots & \\ A_{p-1} B_0 & A_{p-1} B_1 & \cdots & A_{p-1} B_{p-1} \end{bmatrix}$$

- 令 $C = (C_{ij})$ ，其中 $C_{ij} = A_i B_j$
- 将 A_i, B_j 和 C_{ij} ($j = 0, 1, \dots, p-1$) 存放在第 i 个处理器中
(这样的存储方式使得数据在处理器中不重复)
- P_i 负责计算 C_{ij} ($j = 0, 1, \dots, p-1$)
- 由于使用 p 个处理器，每次每个处理器只计算一个 C_{ij} ，
故计算出整个 C 需要 p 次完成
- C_{ij} 的计算是按对角线进行的

行列划分

■ 并行算法一：行列划分

```
for i=0 to p-1
  j=(i+myid) mod p
  Cj=A*B
  src = (myid+1) mod p
  dest = (myid-1+p) mod p
  if (i!=p-1)
    send(B,dest)
    recv(B,src)
  end if
end for
```

- 本算法中， $C_j = C_{myid,j}$, $A = A_{myid}$, B 在处理器中每次循环向前移动一个处理器，即每次交换一个子矩阵数据块，共交换 $p-1$ 次

行列划分程序示例

■ 例：按**行列**划分并行计算矩阵乘积，其中

$$A = (a_{ij}) \in \mathbb{R}^{n \times n}, \quad a_{ij} = \frac{1}{i + j - 1}$$

$$B = (b_{ij}) \in \mathbb{R}^{n \times n}, \quad b_{ij} = i + j - 1$$

示例程序：ex4matmul01.f

行行划分

■ 行行划分：A 按行划分、B 按行划分

将矩阵 A , B 分别划分成如下的行块子矩阵

$$A = [A_0^T, A_1^T, \dots, A_{p-1}^T]^T, \quad B = [B_0^T, B_1^T, \dots, B_{p-1}^T].$$

这时, 需要将 A 的子块 A_i 进一步按列分块, 且与 B 的行分块相对应:

$$A_i = [A_{i0}, A_{i1}, \dots, A_{i,p-1}], \quad i = 0, 1, \dots, p-1.$$

设 C_i 为与 A_i 相应的 C 的第 i 个行分块, 则有

$$C_i = A_i * B = \sum_{j=0}^{p-1} A_{ij} B_j.$$

将 A_i, B_i 存放在第 i 个处理器 P_i 中, 处理器 P_i 负责计算 C_i 。整个计算过程也需要 p 次来完成。

行行划分

算法 2 行行划分矩阵乘积并行算法

```
for ( $i = 0$  to  $p - 1$ ) do  
     $j = (i + myid) \bmod p$   
     $C = C + A_j * B$   
     $source = (myid + 1) \bmod p$   
     $dest = (myid - 1) \bmod p$   
    if ( $i \neq p - 1$ ),  $send(B, dest), recv(B, source)$   
end\{for\}
```

- 在本算法中, $C = C_{myid}$, $A_j = A_{myid, j}$, B 在处理器中每次循环向前移动一个处理器。本算法中的数据交换量和计算量均与算法 1 相同, 不同的只是计算 C 的方式。

列列划分

■ 列列划分：A 按列划分、B 按列划分

将矩阵 A , B 分别划分成如下的列块子矩阵

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

这时, 需要将 B 的子块 B_j 进一步按行分块, 且满足与 A 的列分块相对应:

$$B_j = [B_{0j}^T, B_{1j}^T, \dots, B_{p-1,j}^T]^T, \quad j = 0, 1, \dots, p-1.$$

此时, 设 C 的划分与 B 的划分相对应, 即按列分块。同样将 A_j, B_j 存放在处理器 P_j 中, C_j 也存放处理器 P_j 中。有

$$C_j = A * B_j = \sum_{i=0}^{p-1} A_i B_{ij}.$$

注意此时计算过程中需交换的数据矩阵是 A , 而不是 B 。

列列划分

算法 3 列列划分矩阵乘积并行算法

```
for ( $i = 0$  to  $p - 1$ ) do  
     $k = (i + myid) \bmod p$   
     $C = C + A * B_k$   
     $source = (myid + 1) \bmod p$   
     $dest = (myid - 1) \bmod p$   
    if ( $i \neq p - 1$ ),  $send(A, dest), recv(A, source)$   
end\{for\}
```

- 在本算法中, $C = C_{myid}$, $B_k = B_{k, myid}$, A 在处理器中每次循环向前移动一个处理器。本算法的计算量与算法 1 和算法 2 的是相同的, 当 $m \neq n$ 时, 通信量略有不同, 所以在具体应用中可以按通信量大小选择算法就能得到好的并行效率。

列行划分

■ 列行划分：A 按列划分、B 按行划分

将矩阵 A , B 分别划分成如下的行块子矩阵和列块子矩阵

$$A = [A_0, A_1, \dots, A_{p-1}], \quad B = [B_0^T, B_1^T, \dots, B_{p-1}^T]^T,$$

则

$$C = A * B = \sum_{i=0}^{p-1} A_i B_i.$$

此时 C 的计算是通过计算 p 个规模和 C 相同的矩阵之和得到的。从对问题的划分可以看出，并行算法的关键是计算矩阵的和，设计有效地计算矩阵和的算法，对发挥分布式并行系统的效率起着重要作用。假设结果矩阵 C 也是按列分块存放在处理器中的，记 $B_i = [B_{i0}, B_{i1}, \dots, B_{i,p-1}]$ ，则

$$C_j = \sum_{i=0}^{p-1} A_i B_{ij}.$$

列行划分

算法 4 列行划分矩阵乘积并行算法

$$C = A * B_{myid}$$

for ($i = 1$ *to* $p - 1$) *do*

$$j \equiv (i + myid) \bmod p, k \equiv (myid - i) \bmod p$$

$$T = A * B_j$$

send(T, j), *recv*(T, k)

$$C = C + T$$

end{for}

- 如果采用按行分块方式计算 C ，算法 4 也同样适合，且通信量不变，因此选择何种方式计算 C 可根据需要而定。

Cannon 算法

假设矩阵 A , B 和 C 都可以分成 $m \times m$ 块矩阵, 即: $A = (A_{ij})_{m \times m}$, $B = (B_{ij})_{m \times m}$ 和 $C = (C_{ij})_{m \times m}$, 其中 A_{ij} , B_{ij} 和 C_{ij} 是 $n \times n$ 矩阵, 进一步假定有 $p = m \times m$ 个处理器。为了讨论 Cannon 算法, 引入块置换矩阵 $Q = (Q_{ij})$ 使得

$$Q = \begin{pmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I \\ I & 0 & 0 & \cdots & 0 \end{pmatrix}, \quad \text{即 } Q_{ij} = \begin{cases} I_n, & j \equiv (i + 1) \pmod{m} \\ 0_n, & \text{other case} \end{cases}$$

则 QA 就是将 A 的所有行向上移动一个位置, 即第 m 行移到第 $m - 1$ 行, 第 $m - 1$ 行移到第 $m - 2$ 行, 以此类推, 其中第 1 行移到第 m 行。而 AQ 则是将 A 的所有列向右移动一个位置。

Cannon 算法

- 定义块对角矩阵 $D_A^{(l)} = \text{diag}(D_i^{(l)}) = \text{diag}(A_{i,i+l \bmod m})$, 容易证明 $A = \sum_{l=0}^{m-1} D_A^{(l)} Q^l$, 于是

$$\begin{aligned} C = AB &= \sum_{l=0}^{m-1} D_A^{(l)} Q^l B \\ &= D_A^{(0)} B^{(0)} + D_A^{(1)} B^{(1)} + \dots + D_A^{(m-1)} B^{(m-1)}, \end{aligned}$$

其中 $B^{(l)} = Q^l B = Q B^{(l-1)}$, $l = 0, 1, \dots, m-1$ 。

- 利用这个递推关系式, 并把处理器结点编号从一维映射到二维, 即有 $P_{myid} = P_{myrow, mycol}$, 数据 A_{ij} , B_{ij} 和 C_{ij} 存放在 P_{ij} 中, 容易得到下面的在处理器 P_{myid} 结点上的算法。

算法 5 矩阵乘积 Cannon 并行算法

$C = 0$

for ($i = 0$ to $m - 1$) *do* % 第 i 步计算的是第 $i + 1$ 项

$k = (\text{myrow} + i) \bmod m$

$\text{mp1} = (\text{mycol} + 1) \bmod m$

$\text{mm1} = (\text{mycol} - 1) \bmod m$

if ($\text{mycol} = k$) *then*

 % 将 $A_{\text{myrow},k}$ 广播给 $P_{\text{myrow},j}$, $j = 0, \dots, m - 1, j \neq k$

$\text{send}(A, (\text{myrow}, \text{mp1})); \text{copy}(A, \text{tmpA})$

else

$\text{recv}(\text{tmpA}, (\text{myrow}, \text{mm1}))$

if ($k \neq \text{mp1}$), $\text{send}(\text{tmpA}, (\text{myrow}, \text{mp1}))$

end\{if\}

$C = C + \text{tmpA} * B$

$\text{mp1} = \text{myrow} + 1 \bmod m$

$\text{mm1} = \text{myrow} - 1 \bmod m$

if ($i \neq m - 1$) *then* % 在同列中滚动 B

$\text{send}(B, (\text{mm1}, \text{mycol})); \text{recv}(B, (\text{mp1}, \text{mycol}))$

end\{if\}

end\{for\}

Cannon 算法示例

■ 以 3×3 分块为例：9 个进程，进行三轮计算

● A 、 B 的起始存放位置：

A_{00}	A_{01}	A_{02}
A_{10}	A_{11}	A_{12}
A_{20}	A_{21}	A_{22}

B_{00}	B_{01}	B_{02}
B_{10}	B_{11}	B_{12}
B_{20}	B_{21}	B_{22}

● 第一轮：计算 $D_A^{(0)} B^{(0)}$

A_{00}	A_{00}	A_{00}
A_{11}	A_{11}	A_{11}
A_{22}	A_{22}	A_{22}

B_{00}	B_{01}	B_{02}
B_{10}	B_{11}	B_{12}
B_{20}	B_{21}	B_{22}

Cannon 算法示例

- 第二轮：计算 $D_A^{(1)} B^{(1)}$

A_{01}	A_{01}	A_{01}
A_{12}	A_{12}	A_{12}
A_{20}	A_{20}	A_{20}

B_{10}	B_{11}	B_{12}
B_{20}	B_{21}	B_{22}
B_{00}	B_{01}	B_{02}

- 第三轮：计算 $D_A^{(2)} B^{(2)}$

A_{02}	A_{02}	A_{02}
A_{10}	A_{10}	A_{10}
A_{21}	A_{21}	A_{21}

B_{20}	B_{21}	B_{22}
B_{00}	B_{01}	B_{02}
B_{10}	B_{11}	B_{12}

Cannon 算法

- 该算法具有很好的负载平衡，其特点是在同一行中广播 A ，计算出 C 的部分值之后，在同列中滚动 B 。
- 本算法也可以通过在水平方向（向左）滚动 A ，在垂直方向（向上）滚动 B 来实现。
- 由于计算量对每个处理器来说是相同的，因此在选择算法时只需考虑通信量。对于方阵的乘积，所给出的五个计算矩阵乘积的并行算法中，当 $p \geq 4$ 时，Cannon 算法具有优越性。

Cannon 算法

本算法也可以通过在水平方向（向左）滚动 A ，在垂直方向（向上）滚动 B 来实现。但此时 A, B 在处理器中存放位置如下：

A_{00}	A_{01}	A_{02}
A_{11}	A_{12}	A_{10}
A_{22}	A_{20}	A_{21}

B_{00}	B_{11}	B_{22}
B_{10}	B_{21}	B_{02}
B_{20}	B_{01}	B_{12}

即： A 的第 i 行向左移 i 格，而 B 的第 j 列向上移 j 格。在计算时，每计算完一轮后，沿水平方向向左滚动 A ，并同时沿垂直方向向上滚动 B 。

- 起始存放位置:

A_{00}	A_{01}	A_{02}	B_{00}	B_{11}	B_{22}
A_{11}	A_{12}	A_{10}	B_{10}	B_{21}	B_{02}
A_{22}	A_{20}	A_{21}	B_{20}	B_{01}	B_{12}

- 第一轮计算:

A_{00}	A_{01}	A_{02}	B_{00}	B_{11}	B_{22}
A_{11}	A_{12}	A_{10}	B_{10}	B_{21}	B_{02}
A_{22}	A_{20}	A_{21}	B_{20}	B_{01}	B_{12}

- 第二轮计算:

A_{01}	A_{02}	A_{00}	B_{10}	B_{21}	B_{02}
A_{12}	A_{10}	A_{11}	B_{20}	B_{01}	B_{12}
A_{20}	A_{21}	A_{22}	B_{00}	B_{11}	B_{22}

- 第三轮计算:

A_{02}	A_{00}	A_{01}	B_{20}	B_{01}	B_{12}
A_{10}	A_{11}	A_{12}	B_{00}	B_{11}	B_{22}
A_{21}	A_{22}	A_{20}	B_{10}	B_{21}	B_{02}

上机作业

- 按行行划分并行计算矩阵乘积，其中

$$A = (a_{ij}) \in \mathbb{R}^{n \times n}, \quad a_{ij} = \frac{1}{i + j - 1}$$

$$B = (b_{ij}) \in \mathbb{R}^{n \times n}, \quad b_{ij} = i + j - 1$$

- 编写用第二种方式实现上述矩阵乘积的 Cannon 并行算法

主要内容

- 并行算法基础知识
- 矩阵向量乘积的并行算法
- 矩阵矩阵乘积的并行算法
- 矩阵的 LU 分解并行算法
- 下三角线性方程组的并行算法

线性方程组直接解法

- 线性方程组的并行直接求解算法

线性方程组是许多重要问题的核心，因此有效地求解线性方程组在科学与工程计算中是非常重要的。并行计算机的问世，使求解问题的速度和规模大幅度地提高，同时也使计算方法产生了变化。在传统的串行机上，Linpac是求解线性方程组的有效软件包。然而在并行机上求解线性方程组时，就需要设计出适合于并行计算机的并行算法，算法的优劣会对并行计算机的效率产生很大的影响。

线性方程组直接解法

我们考虑的问题是用直接法（Gauss 消元法）求解下面的线性方程组

$$Ax = b.$$

这里的任务可以分为两部分，一是并行计算矩阵 A 的 LU 分解，其中 L 是单位下三角矩阵， U 是上三角矩阵，也即存在一置换矩阵 P ，使得 $A = PLU$ 。另一部分是并行求解三角形方程组： $Ly = b$ 和 $Ux = y$ 。下面我们给出相关的算法。

矩阵 LU 分解

- *LU* 分解串行算法

采用部分选主元的 Gauss 消元法进行列消去, 使得 L 是单位下三角矩阵。在算法中 A_k 表示矩阵 A 的第 k 行。

算法 1 矩阵 *LU* 分解的串行算法

```
for ( $j = 0$  to  $n - 2$ ) do  
    find  $l$  :  $|a_{lj}| = \max\{|a_{ij}|, i = j, j + 1, \dots, n - 1\}$   
    if ( $l \neq j$ ), swap  $A_j$  and  $A_l$   
    if ( $a_{jj} = 0$ ,  $A$  is singular and return  
     $a_{ij} = a_{ij}/a_{jj}, i = j + 1, \dots, n - 1$   
    for  $k = j + 1$  to  $n - 1$  do  
         $a_{ik} = a_{ik} - a_{ij} * a_{jk}, i = j + 1, \dots, n - 1$   
    end{for}  
end{for}
```

矩阵 LU 分解

在 LU 分解中，主要的计算工作量是修正矩阵 A ，即做 $a_{ik} = a_{ik} - a_{ij}a_{jk}$ ，因此并行计算的主要任务就是在多处理器上同时对矩阵 A 的不同部分做修正。在多处理器上 LU 分解的重要工作是使载荷尽可能的平衡，我们采用卷帘（wrap）存储方式在各处理器上分配矩阵 A ，即把矩阵 A 的第 i 列存放在 $P_{i \bmod p}$ 中。

假设 $n = p \times m$ ，在算法中， A 的第 i 列为原来 A 的第 $i \times p + myid$ 列，即矩阵在处理器中的存放方式为：

$$\underbrace{A_0, A_p, A_{2p}, \dots}_{P_0} \left| \underbrace{A_1, A_{p+1}, A_{2p+1}, \dots}_{P_1} \right| \dots \dots$$

算法：LU分解并行算法

```
icol=0
for j=0 to n-2 do
  if myid=j mod p then
    find l :  $|a_{l,icol}| = \max\{a_{i,icol}, i=j, \dots, n-1\}$ 
    if  $l \neq j$ , swap  $a_{j,icol}$  and  $a_{l,icol}$ 
    if  $a_{j,icol} = 0$ , A is singular and kill all processes
     $a_{i,icol} = a_{i,icol} / a_{j,icol}, i=j+1, \dots, n-1$ 
     $f_{i-j-1} = a_{i,icol}, i=j+1, \dots, n-1$ 
    send(l, myid+1 mod p) and send (f, myid+1 mod p)
    swap  $a_{j,icol}$  and  $a_{l,icol}$ , icol=icol+1
  else
    recv(l, myid-1 mod p) and recv(f, myid-1 mod p)
    if myid+1  $\neq j$  mod p,
      send(l, myid+1 mod p) and send (f, myid+1 mod p)
  endif
  if  $l \neq j$ , swap  $A_j$  and  $A_l$ 
  for k=icol to m-1 do
     $a_{ik} = a_{ik} - f_{i-j-1} * a_{jk}, i=j+1, \dots, n-1$ 
  endfor
endfor
```

矩阵 LU 分解

- 上述算法是在分布式并行计算器上做 LU 分解的有效方法之一。如果采用分块卷帘方式存储，增大了算法的粒度，效果会更好。

上机作业

- 编写LU分解的并程序，其中

$$A = (a_{ij}) \in \mathbb{R}^{n \times n}, \quad a_{ij} = i + j - 1$$

主要内容

- 并行算法基础知识
- 矩阵向量乘积的并行算法
- 矩阵矩阵乘积的并行算法
- 矩阵的 LU 分解并行算法
- 下三角线性方程组的并行算法

三角方程并行求解

- 三角方程组的并行解法

这里考仅讨论求解下三角方程组 $Lx = b$ 。

$$\begin{pmatrix} l_{00} & & & \\ l_{10} & l_{11} & & \\ \vdots & & \ddots & \\ l_{n-1,0} & l_{n-1,1} & \cdots & l_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

我们用向前递推法求解，即原方程的解为：

$$x_0 = b_0 / l_{00}$$

$$x_1 = (b_1 - l_{10}x_0) / l_{11}$$

$$x_2 = (b_2 - l_{20}x_0 - l_{21}x_1) / l_{22}$$

$$x_3 = (b_3 - l_{30}x_0 - l_{31}x_1 - l_{32}x_2) / l_{33}$$

$$x_4 = (b_4 - l_{40}x_0 - l_{41}x_1 - l_{42}x_2 - l_{43}x_3) / l_{44}$$

$$\vdots$$

$$x_i = (b_i - l_{i0}x_0 - l_{i1}x_1 - l_{i2}x_2 - \cdots - l_{i,i-1}x_{i-1}) / l_{i,i}$$

$$\vdots$$

三角方程并行求解

算法 2 下三角方程组 $Lx = b$ 的串行算法

```
for  $i = 0$  to  $n - 1$  do  
     $x_i = b_i / l_{ii}$   
    for  $j = i + 1$  to  $n - 1$  do  
         $b_j = b_j - l_{ji} * x_i$   
    end{for}  
end{for}
```

- 在这个算法中每次对 b 进行修正时用到 L 的一列，如果按这种方式并行修正 b ，则称之为**列扫描算法**。对于列扫描算法，原始数据 L 适合于按行存放，当修正 b 的值时，就可以并行计算。同时为使每个处理器的工作量尽可能均衡，要采取**卷帘方式**存放数据。

三角方程并行求解

- 正如算法所描述的，为了实现并行计算，需要将每步计算出来的解的一个分量传送到所有其它处理器中，其通信次数是很多的，这对于消息传递型并行系统是不太适合的。但是对于有共享存储的系统是可以采用这种计算方案。
- 下面介绍一种在分布式并行机上的下三角方程组的求解方法，该方法采用按**列卷帘方式**存放数据，每次传递的是部分修正的右端项，而不是新求出的解，通过叠加的方式计算下次的新解。

算法 3 使用 p 个处理器求解下三角方程组的算法

$k = 0$

if myid = 0 then

$u_i = b_i, i = 0, 1, \dots, n - 1$

$v_i = 0, i = 0, 1, \dots, p - 2$

else

$u_i = 0, i = 0, 1, \dots, n - 1$

end{if}

for (i = myid to n - 1 with stepsize p) do

if i > 0, recv(v, i - 1 mod p)

$x_k = (u_i + v_0) / l_{ik}$

$v_j = v_{j+1} + u_{i+1+j} - l_{i+1+j,k} * x_k, j = 0, 1, \dots, p - 3$

$v_{p-2} = u_{i+p-1} - l_{i+p-1,k} * x_k$

send(v, i + 1 mod p)

$u_j = u_j - l_{jk} * x_k, j = i + p, \dots, n - 1$

$k + 1 \rightarrow k$

end{for}

表 0.1 $p = 3$ 时各个处理器的运算状态

P_0	P_1	P_2
$k = 0, u = b, v = 0, i = 0$ $\tilde{x}_0 = u_0/\tilde{l}_{00} = b_0/l_{00} = x_0$ $v_0 = u_1 - \tilde{l}_{10}\tilde{x}_0 = b_1 - l_{10}x_0$ $v_1 = u_2 - \tilde{l}_{20}\tilde{x}_0 = b_2 - l_{20}x_0$	$k = 0, u = 0, i = 1$	$k = 0, u = 0, i = 2$
$send(v, P_1)$ $u_j = u_j - \tilde{l}_{j0}\tilde{x}_0 = b_j - l_{j0}x_0$ $j = 3, 4, \dots, n-1$	$recv(v, P_0)$ $\tilde{x}_0 = (u_0 + v_0)/\tilde{l}_{10}$ $= (b_1 - l_{10}x_0)/l_{11} = x_1$ $v_0 = v_1 + u_2 - \tilde{l}_{20}\tilde{x}_0$ $= b_2 - l_{20}x_0 - l_{21}x_1$ $v_1 = u_3 - \tilde{l}_{30}\tilde{x}_0 = -l_{31}x_1$	
$k = k + 1 = 1$ $i = i + p = 3$	$send(v, P_2)$ $u_j = u_j - \tilde{l}_{j0}\tilde{x}_0 = -l_{j1}x_1$ $j = 4, 5, \dots, n-1$	$recv(v, P_1)$ $\tilde{x}_0 = (u_0 + v_0)/\tilde{l}_{20}$ $= (b_2 - l_{20}x_0 - l_{21}x_1)/l_{22} = x_2$ $v_0 = v_1 + u_3 - \tilde{l}_{30}\tilde{x}_0$ $= -l_{31}x_1 - l_{32}x_2$ $v_1 = u_3 - \tilde{l}_{40}\tilde{x}_0 = -l_{42}x_2$
$recv(v, P_2)$ $\tilde{x}_1 = (u_3 + v_0)/\tilde{l}_{31}$ $= (b_3 - l_{30}x_0 - l_{31}x_0$ $= -l_{32}x_2)/l_{33} = x_3$ $v_0 = v_1 + u_4 - \tilde{l}_{41}\tilde{x}_1$ $= b_4 - l_{40}x_0 - l_{42}x_2 - l_{43}x_3$ $v_1 = u_5 - \tilde{l}_{41}\tilde{x}_1$ $= b_5 - l_{50}x_0 - l_{54}x_4$	$k = k + 1 = 1$ $i = i + p = 4$	$send(v, P_0)$ $u_j = u_j - \tilde{l}_{j0}\tilde{x}_0 = -l_{j2}x_2$ $j = 5, 6, \dots, n-1$
$send(v, P_1)$ \vdots \vdots	$recv(v, P_0)$ \vdots \vdots	\vdots \vdots

三角方程并行求解

- 这个算法在分布式系统上被广泛应用，是非常有效的并行算法。注：这个算法在支持非阻塞通信的并行机上使用时，只有一开始有等待，之后便可以并行计算，因此在使用时尽量使用非阻塞型通信。

上机作业

- 用算法 3 编写并行程序求解下三角方程组

$$Ax = b$$

其中

$$A = (a_{ij}) \in \mathbb{R}^{n \times n}, \quad a_{ij} = \begin{cases} \frac{1}{i+j-1}, & i \geq j \\ 0, & i < j \end{cases}$$

$$b = [1, 1, \dots, 1]^T \in \mathbb{R}^n$$