用部分选主元的高斯消去法并行求解线性方程组

刘向娇，刘佳梅
（宁夏大学数学计算机学院，银川 750021）

摘要：高斯消去法，又称高斯消元法，实际上就是我们俗称的加减消元法。数学上，高斯消去法或称高斯－约当消去法，由高斯和约当得名(很多人将高斯消去作为完整的高斯－约当消去的前半部分)，它是线性代数中的一个算法，用于决定线性方程组的解，决定矩阵的秩，以及决定可逆方矩阵的逆。当用于一个矩阵时，高斯消去产生“行消去梯形形式”。用高斯消去法求解线性方程组的解是一种比较常见的解线性方程组的方法，这种方法尤其在利用计算机求解线性方程组时是更是常用。但大多数情况下都是用串行的算法来解方程组，本文介绍了利用高斯消去法并行求解线性方程组的方法。

关键词：高斯消去法；求解线性方程组；并行

中图分类号：TP3

Portion of pivoting in parallel Gaussian elimination method for solving linear equations

Liu Xiangjiao, Liu Jiamei
(Ningxia University,YinChuan 750021)

Abstract: Gaussian elimination, also calls Gauss the elimination, in fact is the elimination by addn which we are named. In mathematics, the gaussian elimination or called Gauss - approximately works as elimination, by the Gauss peace treaty, when acquires fame (many people to eliminate Gauss takes complete Gauss - when approximately eliminates the first half part), it is in a linear algebra algorithm, uses in deciding that system of linear equations's solution, decides the matrix the order, as well as decision reversible side matrix going against. When uses in a matrix, Gauss eliminates the production “the line to eliminate the trapezoidal form”. With the gaussian elimination solution system of linear equations's solution is one quite common solution system of linear equations's method, this method when solves the system of linear equations especially using the computer is commonly used. But in the majority situations was solves the equation set with the serial algorithm, this article introduced used the gaussian elimination parallel solution system of linear equations's method.

Keywords: Gaussian elimination; system of linear equations; parallel

1 高斯消去法

在求解线性方程组的算法中，有两类最基本的算法：直接法和迭代法。在直接法中最主要的就是高斯消去法，它可以分为消元和回代两个过程，消元过程是将方程组转换为一个等价的三角方程组，而回代过程则是逆反求解这个三角方程组[1][2]。

2 部分选主元的高斯消去思想

前述的消去过程中，未知量是按其出现于方程组中的自然顺序消去的，所以又叫顺序消去法。实际上已经发现顺序消去法有很大的缺点。设用作除数的$a_{kk}^{(k-1)}$为主元素，首先，消元过程中可能出现$a_{kk}^{(k-1)}$为零的情况，此时消元过程亦无法进行下去；其次如果主元素$a_{kk}^{(k-1)}$

作者简介：刘向娇（1984-），女，硕士研究生，主要研究方向：并行处理与高性能计算. E-mail: liuxiangjiao042@163.com

很小，由于舍入误差和有效位数消失等因素，其本身常常有较大的相对误差，用其作除数[3]，会导致其它元素数量级的严重增长和舍入误差的扩散，使得所求的解误差过大，以致失真。

我们来看一个例子：

例

$$\begin{cases}0.0001x_1+1.00x_2=1.00\\1.00x_1+1.00x_2=2.00\end{cases}$$

它的精确解为：

$$x_1=\frac{10000}{9999}\approx 1.00010$$

$$x_2=\frac{9998}{9999}\approx 0.99990$$

用顺序消去法，第一步以 0.0001 为主元，从第二个方程中消 x_1 后可得：

$-10000\,x_2=-10000$　　$x_2=1.00$

回代可得　　$x_1=0.00$

显然，这不是解。

造成这个现象的原因是：第一步主元素太小，使得消元后所得的三角形方程组很不准确所致。

如果我们选第二个方程中 x_1 的系数 1.00 为主元素来消去第一个方程中的 x_1，则得出如下方程式：

$1.00\,x_1=1.00$　　$x_1=1.00$

这是真解的三位正确舍入值。

从上述例子中可以看出，在消元过程中适当选取主元素是十分必要的。误差分析的理论和计算实践均表明：顺序消元法在系数矩阵 A 为对称正定时，可以保证此过程对舍入误差的数值稳定性，对一般的矩阵则必须引入选取主元素的技巧，方能得到满意的结果。

高斯消去法的一个简单变形——部分选主元高斯消去法，可以产生可靠的结果。在部分选主元的高斯消去法的第 i 步，我们在第 i 行到第 n-1 行中寻找第 i 列元素的绝对值最大的行并将这一行与第 i 行交换（变成主元）[1]。

在部分主元消去法中，未知数仍然是顺序地消去的，只是选各方程中要消去的那个未知数的系数的绝对值最大的作为主元素，然后用顺序消去法的公式求解。

例：用部分选主元高斯消去法求解方程

$$\begin{cases}2x_1-x_2+3x_3=1\\4x_1+2x_2+5x_3=4\\x_1+2x_2=7\end{cases}$$

由于解方程组取决于它的系数，因此可用这些系数（包括右端项）所构成的“增广矩阵”作为方程组的一种简化形式。对这种增广矩阵施行消元手续：

$$\begin{pmatrix}2 & -1 & 3 & 1\\ 4^{*} & 2 & 5 & 4\\ 1 & 2 & 0 & 7\end{pmatrix}$$

第一步将 4 选为主元素，并把主元素所在的行定为主元行，然后将主元行换到第一行得到

$$\begin{pmatrix} 4 & 2 & 5 & 4 \\ 2 & -1 & 3 & 1 \\ 1 & 2 & 0 & 7 \end{pmatrix} \xrightarrow{\text{第一步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & -2^{*} & 0.5 & -1 \\ 0 & 1.5 & -1.25 & 6 \end{pmatrix}$$

$$\xrightarrow{\text{第二步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & 1 & -0.25 & 0.5 \\ 0 & 0 & -0.875 & 5.25 \end{pmatrix} \xrightarrow{\text{第三步消元}} \begin{pmatrix} 1 & 0.5 & 1.25 & 1 \\ 0 & 1 & -0.25 & 0.5 \\ 0 & 0 & 1 & -6 \end{pmatrix}$$

消元过程的结果归结到下列三角形方程组：

$$\begin{cases} x_1 + 0.5x_2 + 1.25x_3 = 1 \\ x_2 - 0.25x_3 = 0.5 \\ x_3 = -6 \end{cases}$$

回代，得

$$\begin{cases} x_1 = 9 \\ x_2 = -1 \\ x_{-3} = -6 \end{cases}$$

3 串行部分选主元高斯消去算法

先部分选主元然后回代的串行高斯消去算法如下所示：

```
for i←0 to n-1
loc[i] ←i
endfor
for i ←0 to n-1
{找到主元行 picked}
magnitude ←0
for j ←i to n-1
if|a[loc[j]，i]|>magnitude
magnitude ← |a[loc[j]，i]|
picked ←j
endif
endfor
tmp ←loc[i]
loc[i] ←loc[picked]
loc[picked] ←tmp
{将第 i 列中位于未标记行中的元素消为 0}
for j ←i+1 to n-1
t ←a[loc[j]，i]/a[loc[i]，i]
for k ←i+1 to n+1
```

a[loc[j]，k] ←a[loc[j]，k]-a[loc[i]，k] ×t
endfor
endfor
endfor
{回代}
for i←n-1 down to 0
x[i] ←a[loc[i]，n]/a[loc[i]，i]
for j←0 to i-1 do
a[loc[j]，n] ←a[loc[j]，n]-x[i] ×a[loc[j]，i]
endfor
endfor

该算法有两个值得注意的特性：

1．没有独立的数组来存储向量 b，而是用 b 邻接 A 创建一个 n 行 n+1 列的增广矩阵。因此，在这个算法中 a 代表这个增广矩阵。

2．算法每次迭代中间接地访问主元行，而不是直接地将主元行和第 i 行相交换。数组元素 loc[i]记录了第 i 次迭代中所用的主元行的下标[4]。

4 并行部分选主元高斯消去算法

4.1 算法思想

在上面的串行算法中位于最内层下标为 k 的 for 循环和位于中间层的下标为 j 的 for 循环都可以拿来并行执行，换句话说，一旦找到主元行，对所有没被标记的行的修改就可以同时进行。在每一行内，一旦系数 a[loc[j]，i]/a[loc[i]，i]被计算出来了，那么这一行内从位置 i+1 开始的 n-1 个元素的修改工作也可以同时进行。

在并行化的过程中，由于高斯消去法是利用主元行 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。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)的处理器含有 A 的第 i，i+p，…，i+(m-1)p 行和向量 B 的第 i，i+p，…，i+(m-1)p 一共 m 个元素。

矩阵划分好以后，接下来的主要工作是寻找主元行。我们发现为了决定主元行，需要对分布在不同处理器中的第 i 列的各个元素进行规约。我们可以把这一过程称为巡回赛，因为与主元行的第 i 列中存储的具体数值(胜利者的得分)相比，我们对主元行的位置(胜利者的身份)更感兴趣。

在第 i 次迭代进行的过程中，主元行的确定需要两个步骤。首先，每个进程在它所负责的未标记的行中寻找在第 i 列数值最大的那一行。第二，进程加入寻找主元行的巡回赛[5]。

在每次迭代过程中，这一步都是首先要完成的，之后还有一个涉及通信的任务，如图 1 所示。为了计算 a[j，k]的新值，其对应的任务需要访问 a[j，i]、a[picked，i]和 a[picked，k]的值。每个任务至少分配到 A 的一行，所以这个任务既然控制了 a[j，k]，那么也就控制着 a[j，i]。但是 a[picked，i]和 a[picked，k]的值可能为另外一个任务所控制。因此，我们还需要做一次广播[3]。

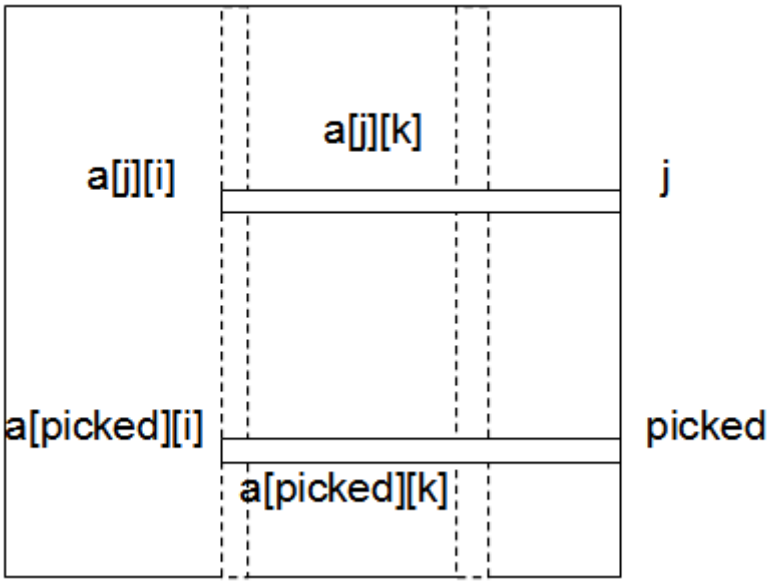

图 1 迭代时通信任务图

4.2 算法实现

输入：系数矩阵 $\mathbf{A}_{n\times n}$，常数向量 $\mathbf{b}_{n\times 1}$
输出：解向量 $\mathbf{x}_{n\times 1}$
Begin
对所有处理器 my_rank(my_rank=0，…，p-1)同时执行如下的算法：
/*消去过程*/
(1) **for** i=0 **to** m-1 **do**
for j=0 **to** p-1 **do**
(1.1) **if**(my_rank<j) **then**
(i) v=i*p+j
(ii) lmax[0]=a[i+1，v]
(iii) **for** k=i +1 **to** m-1 **do**
for t=v **to** n-1 **do**
if(|a[k，t]|)>lmax[0]) **then**
lmax[0]=a[k，t]，lmax[1]=k，
lmax[2]=t，lmax[3]=my_rank
end if
end for
end for
end if
(1.2) **if** (my_rank≥j) **then**
（i） v=i*p+j
(ii) lmax[0]=a[i，v]
(iii) **for** k=i **to** m-1 **do**
for t=v **to** n-1 **do**
if(|a[k，t]|)>lmax[0]) **then**
lmax[0]=a[k，t]，lmax[1]=k，
lmax[2]=t，lmax[3]=my_rank
end if
end for
end for
end if
(1.3) 将每一个处理器中的 lmax 元素广播到其他所有处理器中
(1.4) maxvalve=getpivort(max)，maxrow=getrow(max)，
maxrank=getrank(max)
(1.5) **if**(my_rank=j) **then**
(i) **if**([maxrank=j] and [i≠maxrow]) **then**
innerexchangerow()
end if

```
                    (ii) if (maxrank≠j) then
                            outerexchangerow()
                         end if
                      end if
                   end if
          (1.6)    if(my_rank=j) then
                    (i) for k=v+1 to n-1 do
                          a[i，k]=a[i，k]/a[i，v]
                        end for
                    (ii) b[i]=b[i]/a[i，v]，a[i，v]=1
                    (iii) for k=v+1 to n-1 do
                            f[k]=a[i，k]
                          end for
                    (iv) f[n]=b[i]
                    (v) 广播主行到所有处理器
                   else 接收广播来的主行元素存与数组 f 中
                   end if
          (1.7)    if(my_rank≤j) then
                  for k=i+1 to m-1 do
                    (i) for w=v+1 to n-1 do
                          a[k，w]=a[k，w]-f[w]*a[k，v]
                        end for
                    (ii) b[k]=b[k]-f[n]*a[k，v]
                  end for
                end if
          (1.8)    if(my_rank>j) then
                  for k=i to m-1 do
                    (i) for w=v+1 to n-1 do
                          a[k，w]=a[k，w]-f[w]*a[k，v]
                        end for
                    (ii) b[k]=b[k]-f[n]*a[k，v]
                  end for
                  end if
            end for
          end for
    /*回代过程*/
      (2)  for i=0 to m-1 do
             sum[i]=0.0
           end for
      (3)  for i=m-1 downto 0 do
             for j=p-1 downto 0 do
               if(my_rank=j) then
                 (i) x[i*p+j]=(b[i]-sum[i])/a[i，i*p+j]
                 (ii) 将 x[i*p+j]广播到所有处理器中
                 (iii) for k=0 to i-1 do
                         sum[k]=sum[k]+a[k，i*p+j]* x[i*p+j]
                       end for
                else   /*非主行所在的处理器*/
                  (iv) 接收广播来的 x[i*p+j]的值
                  (v) if(my_rank>j) then
                        for k=0 to i-1 do
                          sum[k]=sum[k]+a[k，i*p+j]* x[i*p+j]
                        end for
                      end if
                   (vi) if(my_rank<j) then
                        for k=0 to i do
                          sum[k]=sum[k]+a[k，i*p+j]* x[i*p+j]
                        end for
                      end if
               end if
             end for
          end for
End
```

4.3 算法分析

在第 i 次迭代主元行确定的两个步骤中，每个进程在它所负责的未标记的行中寻找在第 i 列中最大的那一行的时间复杂度是 O(n/p)，而进程加入寻找主元行的巡回赛的时间复杂度是 O(logp)。

综合考虑通信的总开销，我们发现面向行的部分选主元的高斯消去算法的总消息延迟是 O(nlogp)，而总的消息传输时间是 $O(n^2logp)$[6]。

[参考文献] (References)

[1] 陈国良 安虹 陈峻等.并行算法实践[M].高等教育出版社.2004 年 1 月.第九章 P504-511
[2] 胡哲光.在 C++中实现带主元选择的高斯消去法求解线性方程[J].大众科技.2006 年第八期
[3] 陈文光译.MPI 与 Open MP 并行程序设计(C 语言版)[M].清华大学出版社.2004 年 10 月
[4] 陈国良等.并行算法实践[M].高等教育出版社.2004 年 1 月
[5] 陆鑫达等译.并行程序设计(第 2 版)[M].机械工业出版社.2005 年 5 月
[6] 张云泉 陈英译.并行算法导论[M].机械工业出版社.2004 年 2 月