第6卷第8期
1995年8月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 8
Aug. 1995

稀疏三角矩阵线性系统的基于树结构并行求解*

李 程 田新民 王鼎兴 郑纬民

(清华大学计算机科学与技术系,北京 100084)

摘要 本文分析了大型稀疏矩阵线性方程组直接法求解的回代过程.基于改进的树结构(M-tree),提出了一种新的面向分布存储多机系统的稀疏三角矩阵线性系统并行 Forward 求解算法 MPFS.文中讨论了 M-tree 的结构特征,并将所提出的并行求解算法与基于 Elimination-tree 求解算法进行了分析和比较.结果表明,MPFS 算法不仅适用于更多的稀疏矩阵系统,而且在求解过程中可以开发 Elimination-tree 算法不能开发的计算并行性,从而使求解性能得到显著改进.

关键词 分布存储多机系统,稀疏三角矩阵线性系统,Cholesky 分解,消元树.

线性方程组的数值解法有两类:直接法和迭代法.直接法将问题转化为三角矩阵系统的求解.本文着重研究稀疏三角矩阵线性方程组的并行解法.对任一正定矩阵 A,进行 Cholesky 分解可得 $A=L*L^T$,其中 L 和 L^T 分别为下、上三角矩阵.这样 $Ax=b$ 的求解就可通过 $Ly=b$ 和 $L^Tx=y$ 的求解完成,这两者分别被称为 Forward 和 Backward 求解过程.分析 Forward 过程 $Ly=b$,其基本解法有两种[1],一种为 inter product 算法,如 George 等[2]提出的算法,即 fan-in 类算法.另一种为 column sweep 算法,如 Li 和 Coleman 提出的 PCTS 算法(Parallel Column Triangular Solver)[3].消元树(Elimination-tree)在大型稀疏矩阵的分解中具有十分重要的作用,它提供了稀疏矩阵分解过程中的结构信息[4-6].文献[7]中,Kumar 等将 Li 和 Coleman 针对稠密矩阵提出的 PCTS 算法应用于稀疏矩阵,并利用矩阵的 Elimination-tree,给出了 Forward 和 Backward 回代算法.我们分析了 Elimination-tree 的结构,提出另一种改进的树结构 M-tree,设计了基于 M-tree 的有效的 Forward 算法 MPFS(M-tree based Parallel Forward Solver).理论分析表明,此算法能发掘基于 Elimination-tree 算法所不能开发的并行性,有效提高求解效率.该算法不仅对 Cholesky 分解后的 Forward 过程适用,对其它的下三角稀疏矩阵,如 LU 分解的结果,同样适用,从而扩大了适用范围.

* 本文 1993-12-26 收到,1994-04-21 定稿

本研究受到国家自然科学基金项目和国家 863 高技术项目资助.作者**李程**,1970 年生,博士生,主要研究领域为并行/分布处理.**田新民**,1964 年生,博士,主要研究领域为并行/分布计算机系统.**王鼎兴**,1937 年生,教授,主要研究领域为并行处理与智能计算机系统.**郑纬民**,1946 年生,教授,主要研究领域为并行计算机体系结构.

本文通讯联系人:李程,北京 100084,清华大学计算机科学与技术系

本文第1、2节给出基于M－tree的Forward算法并证明正确性.算法分析及与原有算法的比较在第3节给出.最后阐明结论和进一步的工作.

1 M－tree及其特性分析

三角矩阵$M=(m_{ij})_{n\times n}$,根据M可构造它的有向相关图$G(M)=(V(M),E(M))$,其中$V(M)$为顶点集,$V(M)=\{v_1,\cdots,v_n\}$表示矩阵的各列,边$(v_i,v_j)\in E(M)$当且仅当$m_{ij}\neq 0$.分别用PARTCOL(M,i)、PARTROW(M,i)表示矩阵M的第i列、第i行的下三角部分中的非零元素集$(1\leqslant i\leqslant n)$,即:

$$\text{PARTCOL}(M,i)=\{k\mid m_{ki}\neq 0 \text{ 且 } n\geqslant k>i\}$$

$$\text{PARTROW}(M,i)=\{k\mid m_{ik}\neq 0 \text{ 且 } 0<k<i\}$$

对$1\leqslant i<n$,定义函数parent(),$\text{parent}(i)=\min\{j\mid j\in\text{PARTCOL}(M,i)\}$

定义矩阵M的树结构$\text{Tree}(M)=(V',E')$,V'与$G(M)$的顶点集相同,边$(v_i,v_j)\in E'$当且仅当$j=\text{parent}(i)$.可见,Tree(M)是森林或树.对应矩阵M,构造$G(M)$的子图$M-tree(M)=(V,E)$,V与$G(M)$的顶点集相同,边$(v_i,v_j)\in E$当且仅当存在顶点序列$v_i=v_{t_1},v_{t_2},\cdots,v_{t_p}=v_j$,其中$p>1$,且$\text{parent}(t_{k-1})=t_k(1<k<p)$,$t_p=j\in\text{PARTCOL}(M,t_{p-1})$.与正定矩阵$A$的Elimination－tree相比,M－tree(L)一般不是树.稀疏矩阵系统中使用的图论概念详见文献[8].图1所示例1的矩阵A及L,其中对角线元素用图中的结点表示,非对角线非零元用"·"表示.$G(A)$、Elimination tree及M－tree如图2所示.

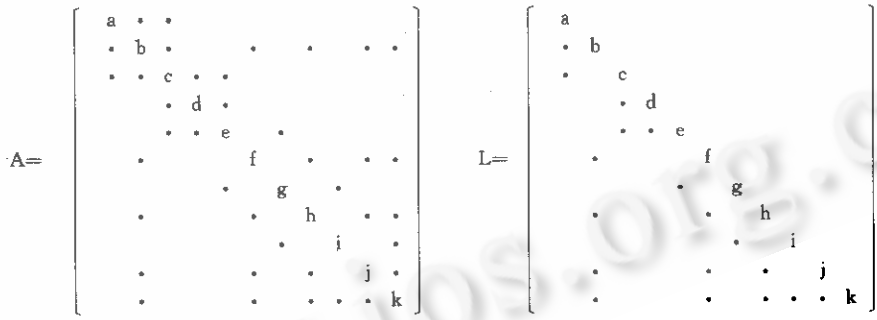

图1 例1的矩阵结构

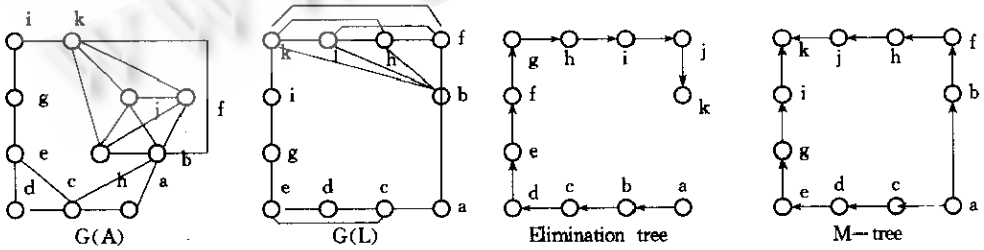

图2 例1的图结构

扩充定义树结构中的descend,ancestor,定义:

descendT$(M,k)=\{j\mid$Tree(M)中存在由v_j至v_k的路径$\}$

$k\in \mathrm{ancT}(M,j)$当且仅当$j\in \mathrm{descendT}(M,k)$

$$\mathrm{descendG}(M,k)=\bigcup_{(v_i,v_k)\in E}\mathrm{descendT}(M,i)$$

$k\in \mathrm{ancG}(M,j)$当且仅当$j\in \mathrm{descendG}(M,k)$

下面的定理刻划了M－tree的性质.

定理1. M－tree(M)中的边集E分成两个子集E_1、E_2,即$E=E_1\cup E_2$,$E_1\cap E_2=\phi$,其中边$(v_i,v_j)\in E_1$当且仅当$\mathrm{parent}(i)=j$,则$(v_i,v_j)\in E_2$当且仅当$j\notin \mathrm{ancT}(M,i)$且$j\in \mathrm{PARTCOL}(M,i)$,并有对任意$k\in \mathrm{ancT}(M,i)$,$j\notin \mathrm{PARTCOL}(M,k)$.

证明:⇒. 任一$(v_i,v_j)\in E_2$,则$j\in \mathrm{PARTCOL}(M,i)$,易见$j\notin \mathrm{ancT}(M,i)$.如果存在$k\in \mathrm{ancT}(M,i)$,$j\in \mathrm{PARTCOL}(M,k)$,由M－tree($M$)的定义可知$(v_i,v_j)\notin E$,与$(v_i,v_j)\in E_2$矛盾.

⇐. 如果$(v_i,v_j)\notin E_2$,则存在$k\in \mathrm{ancT}(M,i)$,且$j\in \mathrm{PARTCOL}(M,k)$,与已知矛盾,所以,$(v_i,v_j)\in E_2$. □

定理2. 下三角矩阵M,则M－tree(M)中$\mathrm{PARTROW}(M,i)\subseteq \mathrm{descendG}(M,i)$

证明:任一$j\in \mathrm{PARTROW}(M,i)$,$(v_j,v_i)$是$G(M)$的一条边.如果$j\in \mathrm{descendT}(M,i)$,则$j\in \mathrm{descendG}(M,i)$.否则,取$k=\max\{t\mid j\in \mathrm{descendT}(M,t)$且$i\in \mathrm{PARTCOL}(M,t)\}$,则任意$t\in \mathrm{descendT}(M,k)$,$i\notin \mathrm{PARTCOL}(M,t)$.由定理1,有$(v_k,v_i)\in E_2$,而$j\in \mathrm{descendT}(M,k)$,所以$j\in \mathrm{descendG}(M,i)$. □

2　基于M－tree的Forward算法MPFS

这部分首先给出Forward算法MPFS,然后证明算法的正确性.最后讨论分配函数map(·),并调整算法.

矩阵L是正定矩阵$A(n\times n)$的Cholesky因子.首先,假设处理器个数$P\geqslant n$.L的列i和b_i分配给处理器P_i.顺序Forward算法如下:

$$x_i=(b_i-\sum_{j\in \mathrm{PARTCOL}(L,i)}l_{ij}*x_j)/l_{ii}\qquad i=1,2,\cdots,n$$

计算中,每一处理器保存一更新向量U,大小为n.b_i也作为更新变元.childnumber(i)表示M－tree(L)中指向顶点v_i的边的个数.用Untree－out(M,i)表示M－tree(L)中v_i发出的,E_2中的边,即Untree－out$(L,i)=\{j\mid (v_i,v_j)$是M－tree(L)中的边,且$j\neq$parent$(i)\}$.首先自叶子结点开始,计算$x_i=b_i/l_{ii}$.修改更新向量U:任一$j\in \mathrm{PARTCOL}(M,i)$,$U(j)=U(j)+l_{ji}*x_i$,然后将更新向量中的PARTCOL($M,i$)部分传送给它的双亲处理器,并加上TREEFLAG标记.为减小信息长度,只发送非零元素.如果$(v_i,v_j)\in$Untree－out(L,i),则将j、$U(j)$加上UNTREEFLAG标记(标志非树边),送至处理器P_j.对于中间结点P_i,如收到带有TREEFLAG标志的信息,更新U,否则更新b_i.当所需的全部信息都收到后,计算$x_i=(b_i-U(i))/l_{ii}$.更新向量及数据传送方式与叶子结点的处理相同.当算法结束时,处理器P_i求得解x_i.若U'为向量,PART(U')表示其非零元素集.算法如图3所示:

```
procedure fwd－solver(L)
    U(*)=0;              /*U 为向量*/
    while (childnumber(k)>0)
```

```
{  receive(flag,U') from a child G of v_k;      /* flag 为消息标志 */
   if (flag==UNTREEFLAG)
      b_k=b_k-U'(k);
   else
      for j∈PART(U')) do U(j)=U(j)+U'(j);
   childnumber(k)=childnumber(k)-1;  }
x_k=(b_k-U(k))/l_kk;
for (j∈PARTCOL(L,k)) do U(j)=U(j)+l_jk * x_k;
send (TREEFLAG,U(j|j∈ancG(L,k)且 j∉Untree-out(L,k))) to P_parent(k);
for (j∈Untree-out(L,k)) do send (UNTREEFLAG, j, U(j)) to P_j;
end
```

图3 并行 Forward 回代算法 MPFS(处理器 P_k,$P=n$)

定理3. Forward 求解算法是正确的.

证明:处理器 P_i 求解 x_i,需要计算 $l_{ij}*x_j$,其中 $j\in$PARTROW(L,i). $l_{ij}*x_j$ 或按 parent$(\cdot)$传递,或按 Untree-out 传. 由定理2,$l_{ij}*x_j$ 一定能传至 P_i. 如果存在 $j\in$descendG(L,i),$j\notin$PARTROW(L,i),则 $l_{ij}=0$,对 x_i 的求解无影响. 由定义和树性质保证不会重复,所以算法的求解结果是正确的. □

考虑依据 map$(\cdot)$函数将各列派生至各处理器的情况,矩阵 L 的第 i 列与元素 b_i 被分配至处理器 $P_{map(i)}$. 对每个处理器,仍使用一个更新向量 U. 算法需要修改. 观察存在 $j\in$ancG(L,i),$j\neq$parent(i)且 $map(i)=$map$(j)=P_x$ 的情况. 如果 ancG$(L,i)\cap$ancG$(L,j)\neq\phi$,不妨设 $k\in$ancG$(L,i)\cap$ancG(L,j),设 map$(k)=P_y$. P_x 计算 x_i 后将 $U(k)$送至 P_y,P_x 计算 x_j 后将 $U(k)$送至 P_y($U(K)$的传递可能经过其它结点,且这两个 $U(k)$可能是不一致的). 这造成 P_y 接收一部分重复计算结果. 解决办法是 P_x 计算 x_i 后,更新向量 U,发送给 $P_{map(parent(i))}$ 的信息为 $U(l|l\in$ancG(L,i)且 $l\notin$ancG$(L,j))$. 这不仅减少信息长度,而且使得处理器 $map(parent(i))$发送信息前所必须作的向量 U 的更新操作次数减少.

对矩阵 L 的每一列,如列 i,定义 Same$(i)=\{j|j\in$ancG$(L,i)\wedge$map$(i)=$map$(j)\}$,

$$\text{定义 } S(i)=\begin{cases}n & \text{if Same}(i)=\phi\\ \min(\text{Same}(i)) & \text{otherwise}\end{cases}$$

改进的算法如图4所示:

```
procedure fwd-solver(L)
   U(*)=0;
   for (column k mapped to this processor) do
      if (childnumber(k)==0)
         {x_k=b_k/l_kk;Update-and-Send(U,x_k,k);}
   while (some x_k needed to be computed in this processor)
      { receive(flag,U') from some x_t for x_i;      /* message 中定义 */
        if (flag==TREEFLAG)                          /* receive (TREEFLAG) */
           for (j∈PART(U'))  U(j)=U(j)+U'(j);
        else b(i)=b(i)-U'(i);                  /* receive (UNTREEFLAG,t,U(t)) */
        childnumber(i)=childnumber(i)-1;
        if (childnumber(i)==0)
```

```
            {x_i=(b_i-U(i))/l_ii; Update-and-Send(U,x_i,i);}
        }
end

function Update-and-Send(U,x_k,k)
    for (j∈PARTCOL(L,k)/Untree-out(L,k))且 j∉ancG(L,S(k))) do U(j)=U(j)+l_jk*x_k;
    send(TREEFLAG, U(l|l∈ancG(L, k)且 l∉ancG(L,S(k))) to processor map(parent(k));
    for (j∈Untree-out(L,k))
        {U(j)=U(j)+l_jk*x_k;
        send(UNTREEFLAG,j,U(j)) to processor map(j);
        U(j)=0;}
    for (j∈PARTCOL(L, k)/Untree-out(L, k)且 j∉ancG(L,S(k))) do U(j)=0;
    for (j∈PARTCOL(L, k)/Untree-out(L,k)且 j∈ancG(L, S(k))) do U(j)=U(j)+l_jk*x_k;
endfunction
```

图4 并行 *Forward* 求解算法 *MPFS*(处理器 P_i)

3 Forward 算法分析比较

MPFS 算法与基于 Elimination－tree 的算法[7]均属于 column sweep 算法. 它们均利用树的特点. 树用以表示高斯消去过程中的矩阵各列之间的相关性. Elimination－tree 算法利用 Cholesky 分解的结果，将列之间的相关性映射成树结构的序，其目的是减少通信的次数. 这不可避免的造成一部分并行计算没有开发. 这可由下面引理看出：

引理1. 如果 $l_{ij}\neq 0$，则 Elimination－tree 中结点 v_i 是 v_j 的祖先[6]. □

矩阵 L 的 M－tree 中，$(v_i,v_j)\in E_2$，则 $l_{ji}\neq 0$. 由引理，L 的 Elimination－tree 中 j 是 i 祖先且 $parent(i)\neq j$. 所以 Elimination－tree 算法中子任务 $P_{parent(i)}$和 P_j 只能串行执行. 而 M－tree 不再是严格的树，为开发并行性，树(森林)中引入带有标记的边，即边集 E_2. 由于 E_2中边的引入，我们的算法中 P_j 和 $P_{parent(i)}$的计算可并发进行. 可见基于 M－tree 的算法发掘了 Elimination－tree 中不能开发的并行性. 可以证明当 M－tree(L)是树时，M－tree(L)即 L 的 Elimination tree. 此时 MPFS 算法与基于 Elimination tree 的算法是一致的.

我们研究的是大型稀疏矩阵，例1中的 Elimination－tree 可以是大矩阵的 Elimination－tree 的一枝. 在不影响其它部分的前提下，加快这一部分的求解，整个系统的求解效率就会提高. 这也是讨论这个例子的目的所在.

根据 Elimination－tree 的特点，基于 Elimination－tree 的算法中，串行求解，也就是只在一个处理器上求解速度最快. 由 M－tree 结构可见，可将(1,2,6,8,10)的计算分配给一个处理器，如 P_1. (3,4,5,7,9,11)的计算分配给另一处理器，如 P_2. Te 表示 Elimination－tree 算法所需时间，Tm 表示 M－tree 算法所需时间. 比较 Tm 和 Te. 用 $t_m(i,j)$表示 v_i 向 v_j 发送信息的延迟，$t_u(i,j)$表示 v_j 接收 v_i 发送的消息后进行的更新操作所需的时间. $t_{cu}(i)$表示计算 x_i 并进行相应更新操作所需时间，可得：

$$Te=\sum_{i=1}^{11}t_{cu}(i),\ Tm=t_{cu}(1)+\max(t_m(1,3)+t_u(1,3)+(t_{cu}(3)+t_{cu}(4)+t_{cu}(5)+t_{cu}(7)+t_{cu}(9)),(t_{cu}(2)+t_{cu}(6)+t_{cu}(7)+t_{cu}(8)+t_{cu}(10))+t_m(10,11))+t_u(10,11)+t_{cu}(11).$$

当 $t_m(1,3)+t_u(1,3)+t_u(10,11)\leqslant t_{cu}(2)+t_{cu}(6)+t_{cu}(8)+t_{cu}(10)$及 $t_m(10,11)+t_u(10,11)\leqslant t_{cu}(3)+t_{cu}(4)+t_{cu}(5)+t_{cu}(7)+t_{cu}(9)$时，有 $Tm\leqslant Te$.

可见，此例中边(1,3)的引入，原有只串行的计算(1,2,3,4,5,6,7,8,9,10)，被分成可并发执行的两个子任务(1,2,6,8,10)和(3,4,5,7,9,11). 只要通信的引入，使单个任务计算量的减少，大于通信延迟的开销，就可加快求解速度.

因为利用树的特点，与基于 Elimination－tree 的算法相似，MPFS 算法的并行开发机制也是将 $\sum_{j\in PARTROW(L,i)} l_{ij}*x_j$ 的计算分割开，使之并发进行，加(减)与乘(除)操作均正比于矩阵 L 的非零元素的个数. 由例子分析及 MPFS 算法可见，MPFS 算法仅保证通过增加通信次数使并行性得以开发，而并行性开发的有效性，还取决于其它因素. 基于 Elimination－tree 的算法是将更新向量延着树的边传递，消息个数为 $n-1$. Fan－in 算法中，若 $j\in$PARTROW(L,i)则要将 x_j 造成的更新传至 v_i，消息个数为 $O|L|$，即矩阵 L 非零元个数. MPFS 算法除保持将更新向量按树的边传递外，还要向其它结点发送信息. 但这一部分计算结果不参与更新向量的传递，目的是提高并行性. 通信量一方面取决于矩阵的结构，另一方面取决于任务派生策略，即 map(·)函数. 分析 M－tree 的结构特征，针对计算环境，选择合理的 map(·)函数，将相关性很强的列派生至同一处理器. 该处理器上各列之间的通信操作可转化为内存操作，避免处理机间的通信开销. 同一处理器上计算任务的调度策略，也是 MPFS 算法得以有效实现的关键. 这些将是进一步的研究内容.

4 结 论

本文提出了基于 M－tree 的稀疏三角矩阵 Forward 回代并行算法 MPFS，并且在理论上与原有算法进行了分析和比较. 结果表明，本算法能有效地开发求解过程中的计算并行性，使求解性能得到显著改进. 应说明的是，Elimination－tree 是正定矩阵系统分解算法中引入和使用的. 我们为了与基于 Elimination－tree 的算法作比较，将矩阵 A 限制为正定矩阵. 实际上，由算法及分析可见，基于 M－tree 的算法对 LU 分解后产生的下三角矩阵及其它下三角矩阵的 Forward 求解同样适用，并不仅仅局限于正定矩阵系统. 所以，M－Tree 算法适用范围更广. 研究表明依据稀疏矩阵结构特征，建立结构优化的树结构，确定合理的任务派生与调度策略，对于提高并行求解效率极为有利.

参考文献

1 Ortega J M. Introduction to parallel and vector solution of linear systems. New York: Plenum Press, 1988.

2 George A, Heath M T, Liu J W H *et al*. Solution of sparse positive definite systems on a hypercube. J. Comput. Applied Math., 1989,**27**:129－156.

3 Li G, Coleman T F. A parallel triangular solver for a distributed memory multiprocessor. SIAM J. Sci. Stat. Comput.,1988,**9**:485－502.

4 Duff I S, Erisman A M, Reid J K. Direct methods for sparse matrices. London: Oxford University Press, 1987.

5 Liu J W H. The role of elimination tree in sparse factorization. SIAM J. Matrix Anal. Appl., 1990,**11**:134－172.

6 Schreiber R. A new implementation of sparse gaussian elimination. ACM Trans. Math. Software, 1982,**8**:256－276.

7 Kumar P S, Kumar M K, Basu A. Parallel algorithms for sparse triangular system solution. Parallel Computing, 1993,19:187—196.

8 George J A, Liu J W H. Computer solution of large sparse positive definite systems. Englewood Cliffs, NJ: Prentice Hall, 1981.

M—TREE BASED PARALLEL ALGORITHM FOR SOLVING SPARSE TRIANGULAR SYSTEM ON DISTRIBUTED MEMORY MULTIPROCESSOR SYSTEMS

Li Cheng　Tian Xinmin　Wang Dingxing　Zheng Weimin

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract　In this paper, underlying the architecture of distributed memory multiprocessors, a new forward substitution algorithm for the direct solution of large sparse linear system is presented. In the presented parallel algorithm, the Modified tree (M—tree) is proposed and adopted. Furthermore, a detailed comparison with the Elimination—tree based algorithm is carried out specially. The results show that more effective parallelism is exploited, and significant performance improvement can be obtained.

Key words　Distributed memory multiprocessor system, sparse triangular system, sparse Cholesky factorization, elimination—tree.