文章编号: 1000_0887(2006) 10_1173_09

非对称稀疏线性方程组的快速外存解法及其在无网格法计算中的应用*

苑维然[1], 陈 璞[1], 刘凯欣[1,2]

(1. 北京大学 力学与空间技术系, 湍流与复杂系统国家重点实验室, 北京 100871;
2. 北京大学 工程研究院, 北京 100871)

(叶庆凯推荐)

摘要: 针对局部 Petrov_Galerkin 无网格法(MLPG) 等无网格方法的计算所产生的大型非对称稀疏线性方程组, 介绍了一种新的直接解法• 与一般非对称求解过程不同, 该解法从现有的对称正定解法中演变出来, 其分解过程在矩阵的上、下三角阵中对称进行• 新的矩阵分解算法可以通过修改对称矩阵分解算法的代码来实现, 这提供了从对称解法到非对称解法的快捷转换• 还针对 MLGP 法以及有限元法所产生的方程组开发了多块外存算法(multi_blocked out_of_core strategy) 来扩大求解规模• 测试结果证明该方法大幅度提高了大型非对称稀疏线性方程组的求解速度•

关 键 词: 稀疏矩阵; 线性方程组; 无网格法; 高性能计算
中图分类号: O241.6 文献标识码: A

引 言

面向问题和机器的线性代数方程组的解法是计算力学研究中的一个基本课题• 在有限元发展的 50 年间涌现了大量的解法, 它们适用于在不同计算机环境下的有限元分析的求解[1~4]• 近年来随着无网格法的发展, 迫切需要研究无网格法方程组的一般性解法• 与有限元法的方程组相比, 局部 Petrov_Galerkin 无网格法(MLPG)[5,6] 等无网格法的代数方程组的系数矩阵是非对称的, 但它的非零元分布几乎是对称的• 这启示人们可能借用有限元解法的研究成果来发展无网格法的求解器•

由于无网格方法逐渐走向工程问题的求解, 无网络法的线性方程组的未知量数目也随着研究的深入不断变大• 文献中解小例题的方法, 例如: 满矩阵或带宽矩阵的三角分解已不能适应求解大型应用问题的需求• 这一困难主要反映在求解时间和系数矩阵的大小上•

本文研究并实现了一个针对无网格法的直接非对称矩阵求解器, 它能够快速求解中到大

* 收稿日期: 2005-07-25; 修订日期: 2006-04-07
基金项目: 国家自然科学基金资助项目(10232040; 10572002; 10572003)
作者简介: 苑维然(1980—), 男, 辽宁人, 博士生;
陈璞(1962—), 男, 重庆人, 副教授, 博士(联系人. Tel: + 86_10_62751828; E_mail: chenpu@pku.edu.cn);
刘凯欣, (1956—), 吉林人, 教授, 博士.

型的非对称稀疏线性方程组• 其中最重要的部分是本文将着重讨论的三角分解• 近 10 年来, 有限元法的求解技术取得了巨大的进步, 其中最引人注目的是稀疏矩阵替代带宽矩阵和循环展开[7,8]•

本文分为下面几个部分, 第 1 节回顾了稀疏矩阵的概念和相应的内存、外存储方案• 带有循环展开技术的非对称 $\boldsymbol{LDU}$ 分解将在第 2 节中介绍• 在第 3 节中, 我们提出了外存非对称稀疏解法的实现• 最后给出了几个实际工程应用问题的数值测试的结果和分析•

在内存解法时与著名的稀疏求解器 SuperLU[9,10] 相比较, 本文求解无网格法生成的线性方程组所需要的时间有了大幅度减少• 与有限元方法的求解器类似, 我们引入了分块外存解法, 大幅度地提高了在非并行计算机上问题的求解规模•

考虑非对称的线性方程组

$$\boldsymbol{Ax} = \boldsymbol{b}, \tag{1}$$

其中, neq × neq 的矩阵 $\boldsymbol{A} = (A_{i,j})$, $A_{i,j} \in \mathbf{R}$ 是一个大型稀疏矩阵, 向量 $\boldsymbol{x}$ 和 $\boldsymbol{b}$ 分别是待求的未知向量和已知的输入向量•

在 MLPG 等无网格法中, 系数矩阵 $\boldsymbol{A}$ 在数值上是非对称的, 但是 $\boldsymbol{A}$ 的上三角和下三角部分在非零元的分布上几乎是对称的• 对这样的方程组, 可以在矩阵 $\boldsymbol{A}$ 中少量结构非对称的位置填入零, 使矩阵 $\boldsymbol{A}$ 在结构上对称, 然后改造对称矩阵已有的成熟方法进行计算•

当 MLPG 法生成的系数矩阵 $\boldsymbol{A}$ 的各阶顺序主子式全为正时, 矩阵 $\boldsymbol{A}$ 可唯一地分解为

$$\boldsymbol{A} = \boldsymbol{LDU}, \tag{2}$$

其中, $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 分别是单位下三角和单位上三角矩阵, $\boldsymbol{D}$ 是对角矩阵• 在完成三角分解之后, 式 (1) 的求解可以分为 3 个步骤: $\boldsymbol{Ly} = \boldsymbol{b}$, $\boldsymbol{Dz} = \boldsymbol{y}$, $\boldsymbol{Ux} = \boldsymbol{z}$[11]• 当 $\boldsymbol{A}$ 为对称矩阵 – $\boldsymbol{U} = \boldsymbol{L}^{\mathrm{T}}$•

$\boldsymbol{L}$ 和 $\boldsymbol{U}$ 的计算是求解线性方程组 (1) 中计算量最大的步骤• 如果矩阵 $\boldsymbol{A}$ 是稀疏的, 则一般 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 也是稀疏的, 但是矩阵 $\boldsymbol{A}$ 中的非零元数目小于分解之后上、下三角矩阵非零元数目的和•

1 存 储 方 案

在数值计算中, 稀疏矩阵的存储一般可以选用行主元存储或列主元存储• 在行主元存储中, 矩阵被视作紧凑行向量的顺序组合• 在本文中, 系数矩阵使用一种混合的存储方案• 在式 (3) 中, $a, b, \cdots, i$ 是上三角部分的非零元, 在下三角部分它们的对称位置上也有非零元 $a^*, b^*, \cdots, i^*$• 表 1 中的数组 IA, JCA, PA 和 QA 为矩阵 $\boldsymbol{A}$ 的行/列主元存储方案, 其中上三角部分在行主元形式下需要 3 个数组 IA, JCA 和 PA 来表示, 而它的下三角部分可以在列主元形式下用 IA, JCA 和 QA 表示• 数组 IA 是非零元的列/行指标数组 JCA 以及它的数值数组 PA/QA 的行/列索引• 上三角第 k 行的非零元个数为 IA(k) – IA$(k-1)$ (设 IA$(0) = 0$), 这一数值也是下三角第 k 列的非零元个数• 一般称 (IA, JCA) 为符号矩阵, (PA, QA) 为数值矩阵•

$$\boldsymbol{A} = \begin{pmatrix} 11 & & & & a & b \\ & 22 & c & d & e & \\ & c^* & 33 & f & g & \\ & d^* & f^* & 44 & h & \\ a^* & e^* & g^* & h^* & 55 & i \\ b^* & & & & i^* & 66 \end{pmatrix}. \tag{3}$$

1.1 内存储策略

在无网格法的系数矩阵 $\boldsymbol{A}$ 中, 一个节点相关的方程经常具有连续的编号, 这样在 $\boldsymbol{A}$ 中相应的行和列具有相同的稀疏结构• 这样一组连续的方程定义为 $\boldsymbol{A}$ 的超方程• 超方程的第 1 行称为主方程(master_equation), 其他行称为从方程(slave_equation)• 由于超方程技术在有限元求解中成功的应用[7], 很自然的考虑到应用超方程来提高非对称情形下解法的性能• 一个整型数组 MASTERA(neq) 便可表示超方程• 值为正的 MASTERA(i) 表示从第 i 个方程开始的超方程的大小, 而值为 0 的 MASTERA(i) 表示第 i 个方程是一个从方程• 例如, 根据超方程的定义, 式(3)中矩阵 $\boldsymbol{A}$ 有 4 个超方程, 分别为$\{1\}$、$\{2,3,4\}$、$\{5\}$、$\{6\}$•

这种表示为超方程的稀疏矩阵 $\boldsymbol{A}$ 可以用 Sherman 压缩存储方案[12]来减少存储方案中列/行指标数组 JCA 的存储量• 修正后的列/行指标数组为 JA• 为了使用 JA, 还需要一个辅助数组 KA, 来指出每一列/行的指标修正在 JA 中的开始位置• 稀疏矩阵 $\boldsymbol{A}$ 的存储方案如表 1 所示, 其中 neq 是系数矩阵 $\boldsymbol{A}$ 的阶数, nja 是压缩的 JA 的个数, nzr 是刚度矩阵上/下三角部分非零元个数, 即矩阵 $\boldsymbol{A}$ 中总的非零元个数为 nzr × 2− neq• 通常 nja≪nzr•

表 1 矩阵存储方案

方程	1			2				3			4		5		6
IA(neq)	3			7				10			12		14		15
JA(nzr)	1	5	6	2	3	4	5	3	4	5	4	5	5	6	6
PA(nzr)	11	a	b	22	c	d	e	33	f	g	44	h	55	i	66
QA(nzr)		a^*	b^*		c^*	d^*	e^*		f^*	g^*		h^*		i^*	
KA(neq)	1			4				5			6		8		10
JA(nja)	1	5	6	2	3	4	5						5	6	6
MASTERA(neq)	1			3				0			0		1		1

对于 $\boldsymbol{A}$ 的因子矩阵 $\boldsymbol{L}/\boldsymbol{U}$, 可以采用相同的数据结构 IU、JU、MASTERU、PU 和 QU 来表示, 但要注意它们的非零元个数与 $\boldsymbol{A}$ 的上下三角部分不同, 这是因为在矩阵的三角分解过程中将产生许多新的非零元, 一般称之为填充元(fill_in' s)• 与 $\boldsymbol{A}$ 的超方程定义不同的是, 填充元的产生将使得 $\boldsymbol{U}/\boldsymbol{L}$ 中的不少相邻的行/列的对角块是满上/下三角矩阵, 而其余的部分具有完全相同的列/行指标集, 或者说这些行/列具有相同的稀疏性(sparsity)• 这些相邻的行/列所对应的方程被定义为 $\boldsymbol{U}/\boldsymbol{L}$ 的超方程[13]• 相应的数据结构是 IU, JU, MASTERU, PU 和 QU•

1.2 外存储策略

如果没有外存储策略, 大型问题的一般求解方法是依靠虚拟内存系统, 即允许操作系统在内存和磁盘之间移动数据• 它的优点是不需要修改内存算法的程序, 但经验表明它的速度很慢•

本文针对非对称矩阵的特殊性扩展了文献[13~15]中针对有限元计算的外存算法• 将矩阵 $\boldsymbol{A}$ 和它的分解因子上、下三角阵 $\boldsymbol{L}$、$\boldsymbol{U}$ 分别划分为相似大小的块• 根据消元的要求, 至少有两块可以同时安置于内存• 在本文中, 块的大小要满足 5 块以上可以同时放在内存中• 符号矩阵和数值矩阵应该分别存储在不同的文件中, 因为符号矩阵在数值矩阵分解之前生成•

表 2 列出了针对 Sherman 压缩稀疏存储方法的外存储方案• 矩阵 $\boldsymbol{A}$ 的结构按行/列对称的切分为 lblk 块, 因子 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 分别按行和列切分为 lblu 块• 每一块具有最大 laxt 个记录• 整型数组 BOUND(lblk+ lblu) 表示每个块的主方程• 在 $\boldsymbol{LDU}$ 分解和消元与回代的过程中, 表 2

中控制信息保留在内存中,它决定了内、外存储方案的选择•

表 2 外存数据文件和分块

内容	数组	文件类型	记录长度	记录数目
控制信息	IA(neq), KA(neq), KU(neq), MASTERA(neq), MASTERU(neq), BOUND(lblk+ lblu)	顺序文件		
符号矩阵	JA(laxt, lblk), JU(laxt, lblu)	直接文件	laxt 整型	lbla+ lblu
数值矩阵	PA(laxt, lblk), QA(laxt, lblk), PU(laxt, lblu), QU(laxt, lblu)	直接文件	laxt 浮点型	lblk+ lblu

2 稀疏 *LDU* 分解

当矩阵 ***A*** 是各阶主子式大于零时, 求解器一般可分为几个独立的阶段: 符号刚度矩阵的组装、填充元优化、符号分解、数值刚度矩阵的组装、数值分解以及消元与回代• 对比本文解法在 ***LDU*** 分解过程和有限元线性方程组解法在$(\boldsymbol{LDU})^{\mathrm{T}}$分解过程[13]中的相应阶段: 本文使用混合行_列主元的存储方法, 使符号分解只需要在矩阵上、下三角阵中的一个进行, 对称的非零元位置使我们可以使用为对称方程开发的优化算法, 如 AMD[16]和 METIS[17]• 本文解法在对称情形下做尽量少的更改, 以期提供一个广泛适用的从对称求解器过渡到非对称求解器的解决方案• 其中, 符号组装过程要考虑到无网格算法输出格式与有限元单刚矩阵的不同; 数值组装过程要考虑矩阵上、下两部分的数值数组 PA 和 QA, 所以数值部分所需的内存空间是对称情形的近两倍; 数值分解过程则要考虑上、下三角相互分解; 消元与回代过程则分别使用上、下三角, 而对称情形时则使用两次下三角•

2.1 基本的 *LDU* 分解

由于使用混合行_列主元的存储方法, 本文的 ***LDU*** 分解为 left_looking 的形式(或者称为 backward_reference)[18]• 算法 1 描述了矩阵 ***A*** 的 left_looking 形式的 ***LDU*** 分解算法• 子过程 RowColumn_Task(k, j)对$A_{j,k}$和$A_{k,j}$消元, 即目标第 j 行/列用上/下三角阵 ***U***/***L*** 的第k 行/列的倍数来消元• 子过程RowColumn_Task(j, j)把目标j 行/列非对角线元素除以该行/列的对角线元素•

文献[13]提出了对称情形的直接更新法(direct updating method)来替代传统的分散_收集(scatter_gather)操作• 在算法 2 中, 将这种方法应用到非对称的情形, 它用前面的

算法 1 Left_looking ***LDU*** 分解

```
For row j = 1: neq
  For k = all appropriate rows
    RowColumnTask(k, j)
  End
  RowColumnTask(j, j)
End
```

算法 2 Left_looking ***LDU*** 分解的 FORTRAN 伪码

```
c_loop for all appropriate k
k= CHAIN(j)
do while (k. gt. 0)
  kk= k
  k= CHAIN(k)
c_RowColumnTask(k, j)
  ss= QU(IU(kk- 1)+ 1)* QU(UPD(kk))
  tt= PU(IU(kk- 1)+ 1)* PU(UPD(kk))
  do i= UPD(kk), IU(kk)
    PU(UPD(JU(i)))= PU(UPD(JU(i)))- ss* PU(i)
    QU(UPD(JU(i)))= QU(UPD(JU(i)))- tt* QU(i)
  end do
...
end do
...
c_RowColumnTask(j, j)
ss= 1.0d0/PU(IU(j- 1)+ 1)
do i= IU(j- 1)+ 2, IU(j)! off_diagonal
  PU(i)= PU(i)* ss
  QU(i)= QU(i)* ss
end do
...
```

行/列来更新第 j 行/列的数值$\{PU(l), QU(l) \mid IU(j-1) < l \leqslant IU(j)\}$• 在这种情况下, $UPD(j:neq)$ 作为从第 j 行/列的列号/行号到它们的 PU/QU 位置的映射•

2.2 带有循环展开技术的 *LDU* 分解

循环展开可以高效地结合在非对称稀疏求解器中• 在算法 1 中, RowColumnTask (k,j) 和 RowColumnTask (m,j) 是相互独立的• 这样, k 循环内可不必顺序执行• 一种把 k 循环内组织起来的方法就是将若干连续的 k 循环体(即超方程)放在一起• 算法 3 列出了带有循环展开的 left_looking 实现, 它展开了 k 循环•

算法 3 带有循环展开的 Left_looking ***LDU*** 分解

```
For row j = 1: neq
  For mk = all appropriate master_rows
    SuperRowColumnTask(mk, j)
  End
  RowColumnTask(j, j)
End
```

算法 4 带有循环展开的 Left_looking ***LDU*** 分解稀疏算法的 FORTRAN 伪码

```
c SuperRowColumnTask(mk, j)
if (size(mk).eq.1) then
  ss= QU(IU(kk- 1)+ 1)* QU(UPD(kk))
  tt= PU(IU(kk- 1)+ 1)* PU(UPD(kk))
  do i= UPD(kk), IU(kk)
    PU(UPD(JU(i)))= PU(UPD(JU(i)))- ss* PU(i)
    QU(UPD(JU(i)))= QU(UPD(JU(i)))- tt* QU(i)
  end do
else if (size(mk).eq.2) then
  ss0= QU(IU(kk- 1)+ 1)* QU(UPD(kk))
  tt0= PU(IU(kk- 1)+ 1)* PU(UPD(kk))
  ss1= QU(IU(kk   )+ 1)* QU(UPD(kk)+ ksh1)
  tt1= PU(IU(kk   )+ 1)* PU(UPD(kk)+ ksh1)
  do i= UPD(kk), IU(kk)
    PU(UPD(JU(i)))= PU(UPD(JU(i)))- ss0* PU(i)- ss1* PU(i+ ksh1)
    QU(UPD(JU(i)))= QU(UPD(JU(i)))- tt0* QU(i)- tt1* QU(i+ ksh1)
  end do
else if (size(mk).eq.3) then
  ...
  ...
```

这里, SuperRowColumnTask 可以由一系列 RowColumnTask 组合起来• 然而, 本算法对目标 j 行/列消元的过程中, 是以从第 mk 行/列开始的超方程中的主方程和从方程作为一个 SuperRowColumnTask 的• 超方程的大小 $n+1$ 作为循环展开的深度• 研究表明循环展开给矩阵计算的效率带来显著的提高• 如果目标 j 行/列是从 mk 开始的超方程的一个从方程, 这种情况下从 mk 开始的超方程大小可临时改为 $j-mk$•

超方程中的行/列有相同的稀疏结构, 即除了对角元处的子矩阵外, 这些行/列的元素具有相同的列号/行号索引• 这样, 很容易将算法 3 中的伪代码改为算法 4 中循环展开的形式•

在现代计算机体系结构中, 广泛的应用了高速缓冲存储器、指令级的并行等特性来提高计

算机的性能[19]• 循环展开技术使编译器减少了索引变量的开支, 稀疏 ***LDU*** 分解中的循环展开增强了算法使用指令交错(instruction interleaving) 的能力• 如果两条指令的结果没有相互依存的关系, CPU 可以在前一条指令没有完成之前发出新指令而不会引起硬件冲突• 指令交错就是使用这种指令级的并行性的技术• 在指令交错中, 如果一条指令需要等待前一条指令的处理结果, 其他的指令将插入到这两条指令之间来利用这个等待的时间• 要使用指令交错, 循环体需要足够大, 这样可以有足够多相互之间独立的指令• 循环展开技术就是通过增加循环体的大小来达到这种目的•

3 外存储 *LDU* 分解

可以观察到, 方程_超方程形成了 ***LDU*** 分解过程的层次结构• 用前面提到的分块的概念作为这个层次结构的前一层, 分块分解的算法可如算法 5 描述•

在这里, 根据不同的稀疏算法, 一个 BlockTask 可包括多个 RowColumnTask 或 SuperRow-ColumnTask• 与算法 2 和算法 4 类似, 可以建立一个连接块的链表 BCHAIN(nblu) , 来链接与当前被消元的块相关的所有块• 除此之外, 在 BlockTask(BK, BJ) 中需要一个链表筛选器(chain filter) 来标识当前方程的消去链中所链接的超方程是否在块 BK 中•

算法 5 Left_looking 分块 ***LDU*** 分解

```
For BJ= 1: nblu
  Initialize block BJ
  For BK= all appropriate block
    Load block BK
    BlockTask( BK, BJ)
  End
  BlockTask( BJ, BJ)
  Save block BJ
End
```

算法 6 Left_looking 分块 ***LDU*** 分解算法

```
For BJ= 1: nblu: ntblk
  Initialize blocks BJ, BJ+ 1, …, BJ+ ntblk- 1 to a big block BJJ
  For BK= all appropriate block
    Load block BK
    BlockTask( BK, BJJ)
  End
  BlockTask( BJJ, BJJ)
  Save block BJ, BJ+ 1, …, BJ+ ntblk- 1
End
```

当 ntblk+ 1(ntblk> 1) 个块可以被安置在内存中时, 我们将连续的 ntblk 个目标块放在内存中• 算法 6 描述了我们的多块 ***LDU*** 分解(multi_block ***LDU*** factorization) 的实现• 当然, 如果设块 BJ 为块 BJ, BJ+ 1, …, BJ+ ntblk- 1 的主块, 块链表 BCHAIN(nblu) 现在只需要从每 ntblk 块开始• 在我们的外存储算法的测试中, 算法 6 一般比算法 5 需要更少的 ***LDU*** 分解时间•

多块 ***LDU*** 分解使用目标块循环的循环展开技术来减少外存储算法的磁盘 I/ O• 将内存空间分为两块时, 所有的相关块都需要在每次目标块需要的时候读取一次• 而在多块 ***LDU*** 分解算法中, 内存空间被分为 ntblk+ 1 块• 这样将目标块循环展开后, 所有相应的块在每 ntblk 个目标块需要的时候才读取一次• 数值试验表明, 与两块算法相比, 使用多块 ***LDU*** 分解算法时, 磁盘 I/ O 可以大幅度减少•

4 数 值 算 例

首先用局部 Petrov_Galerkin 无网格法(MLPG) 生成的算例(PKUML*) 来评估本文解法的性能, 进一步测试了选自欧洲 PARASOL 项目收集的对称算例, 用对称矩阵退化到非对称来验证解法的正确性• 这些算例相应的系数矩阵的阶数在 34 191 到 501 501 之间, 都已在实际工程或科研项目中应用• 各算例的简要说明见表 3•

表 3 测试算例说明

题目	描述	结构	neq	nzr
PKUML01	MLPG 方法的方板线性静力分析,131× 131	非对称	34 191	1 968 955
PKUML04	MLPG 方法的 2 维静态裂纹分析,201× 201	非对称	80 601	4 653 525
PKUML05	MLPG 方法的梁线性静力分析, 151× 391	非对称	117 691	8 817 691
PKUML07	MLPG 方法的梁线性静力分析, 161× 701	非对称	225 021	16 900 141
PKUML08	MLPG 方法的方板线性静力分析,501× 501	非对称	501 501	33 554 431
Hood	篷盖	对称	220 542	5 494 489
BMW7st_1	轿车车体的线性静力分析	对称	141 347	3 740 507
BMWCRA_1	轿车曲轴模型	对称	148 770	5 396 386
CRANKSG2	机轴细节的线性静力分析	对称	63 838	7 106 348
PWTK	受压的风洞	对称	217 918	5 926 171

4.1 内存算法数值测试

内存算法数值测试的目的是与其他稀疏直接解法进行对比, 本文的比较对象是广泛应用于科学工程计算的稀疏求解器 SuperLU• 该测试是在基于 Intel Pentium IV 3.0G CPU 的 PC 上完成的, 其中本文解法在 Windows 平台运行, SuperLU 在 Linux 平台运行• 独立的测试表明, Windows 和 Linux 下的数值运算速度没有本质的差别• 表 4 的内存解法项中列出了以上算例在两种解法中测试的表现• 可以看出在两种解法中, 本文的解法总的来说比 SuperLU 需要较少的求解时间•

4.2 外存算法数值测试

当求解超出内存范围的大型算例时, 内存算法无法求解, 而使用操作系统的虚拟内存时效率很低, 所以需要使用本文的分块外存算法• 本测试的目的是, 内存大小分别限制为 128 Mb、256 Mb 和 512 Mb 时, 比较外存算法对求解规模和求解效率的影响• 从表 4 中本文算法与 SuperLU 的对比可以看出, 当在小内存机器上求解大型题目时, 本文的外存算法优势明显• 所以当机器内存较大时, 可求解一般算法在非并行机上不可解的大型题目, 比如具有 501 501 个方程数和 33 554 431 个非零元的算例 PKUML08 在 512 Mb 的内存条件下即可求解• 由此可见, 外存算法可以求解更大型的题目, 并提高求解的效率•

表 4 算例测试

题目	本文分解时间 t/s				SuperLU 分解时间 t/s			
	128 Mb	256 Mb	512 Mb	内存解法	128 Mb	256 Mb	512 Mb	内存解法
PKUML01	80.98	60.24	–	30.62	174.11	47.90	–	37.19
PKUML04	466.20	307.05	210.45	114.14	3 492.00	380.97	244.12	194.91
PKUML05	498.55	473.34	329.81	187.20	N/A	888.53	496.02	196.11
PKUML07	N/A*	1 079.56	825.03	472.33	N/A	N/A	N/A	N/A
PKUML08	N/A	N/A	3 449.41	N/A	N/A	N/A	N/A	N/A
Hood	39.29	37.78	–	8.20	2 172.37	395.57	73.22	28.33
BMW7st_1	37.53	–	–	10.88	803.91	214.60	59.80	40.35
BMWCRA_1	227.25	229.00	224.56	52.61	8 322.17	1 622.85	907.08	336.55
CRANKSG2	161.22	160.70	–	41.72	4 645.28	712.51	401.73	228.29
PWTK	151.91	75.39	–	21.06	1 756.12	753.79	301.34	102.46

* “N/A” 指在该算例在所限制内存条件下无法求解;“–” 指该算例在此内存条件下为内存解法•

5 结 论

本文详细讨论了带有循环展开和外存储策略的非对称稀疏矩阵 ***LDU*** 分解的算法和实现• 介绍了矩阵 ***A*** 和因子 ***L***、***U*** 的结构定义的超方程, 并由此引入了可以大幅度减少符号矩阵大小的混合行 _列主元的存储方案• 并且, 因子 ***L*** 和 ***U*** 可以共同使用相同的超方程• 对应于这种压缩存储方案, 超方程很方便的应用在 ***LDU*** 分解的符号阶段和数值阶段• 讨论了外存储策略而扩展了该算法的求解范围, 使其广泛适用于无网格法• 基于适当的存储方案, 实现了循环展开的稀疏矩阵 ***LDU*** 分解• 分块外存算法在外存储策略的基础上进一步提高了算法的效率• 数值测试表明, 本文的求解过程在时间和存储空间上非常高效•

[参 考 文 献]

[1] CHEN Pu, Runesha H, Nguyen D T, et al. Sparse algorithms for indefinite systems of linear equations [J]. Comput Mech J, 2000, **25**(1): 33—42.

[2] Damhaug A C, Reid J, Bergseth A. The impact of an efficient linear solver on finite element analysis [J]. Comput Struct, 1999, **72**(4/5): 594—604.

[3] Weinberg D J. A performance assessment of NE/Nastran's new sparse direct and iterative solvers [J]. Adv Engng Software, 2000, **31**(8/9): 547—554.

[4] Wilson E L, Bathe K J, Doherty W P. Direct solution of large system of linear equations[J]. Comput Struct, 1974, **4**(2): 363—372.

[5] Atluri S N, Zhu T. A new meshless local Petrov_Galerkin (MLPG) approach in computational mechanics[J]. Comput Mech, 1998, **22**(2): 117—127.

[6] Atluri S N, Zhu T. The meshless local Petrov_Galerkin (MLPG) approach for solving problems in elasto_statics[J]. Comput Mech, 2000, **25**(2/3): 169—179.

[7] Ng E G, Peyton B W. Block sparse Cholesky algorithm on advanced uniprocessor computers[J]. SIAM J Sci Comput, 1993, **14**(5): 1034—1055.

[8] Pissanetzky S. Sparse Matrix Technology [M]. London, Orlando: Academic Press, 1984.

[9] Demmel W J, Eisenstat C S, Gilbert J R, et al. A supernodal approach to sparse partial pivoting[J]. SIAM J Matrix Anal Appl, 1999, **20**(3): 720—755.

[10] Li S X. An overview of superLU: algorithms, implementation, and user interface[J]. ACM Trans Math Software, 2005, **31**(3): 302—325.

[11] Runesha H B, Nguyen D T. Vector_sparse solver for unsymmetrical matrices[J]. Adv Engng Software, 2000, **31**(8/9): 563—569.

[12] Sherman A H. On the efficient solution of sparse systems of linear and nonlinear equations[D]. Rept No. 46. Ph D Dissertation. New York: Dept of Computer Science, Yale University, 1975.

[13] CHEN Pu, ZHENG Dong, SUN Shu_li, et al. High performance sparse static solver in finite element analyses with loop_unrolling[J]. Adv Engng Software, 2003, **34**(4): 203—215.

[14] Fellipa C A. Solution of linear equations with skyline_stored symmetric matrix[J]. Comput Struct, 1975, **5**(1): 13—29.

[15] Wilson E L, Dovey H H. Solution or reduction of equilibrium equations for large complex structural system[J]. Adv Engng Software, 1978, **1**(1): 19—26.

[16] Amestoy R P, Enseeiht_Irit, Davis A T, et al. Algorithm 837: AMD, an approximate minimum degree

ordering algorithm[J]. ACM Trans Math Software, 2004, **30**(3): 381—388.

[17] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs[J]. SIAM J Sci Comput, 1998, **20**(1): 359—392.

[18] Zheng D, Chang T Y P. Parallel Cholesky method on MIMD with shared memory[J]. Comput Struct, 1995, **56**(1): 25—38.

[19] Dowd K, Severance C R. High Performance Computing[M]. 2nd ed. Cambridge: Sebastopol, CA O'Reilly & Associates, 1998.

High Performance Sparse Solver for Unsymmetrical Linear Equations With Out_of_Core Strategies and Its Application on Meshless Methods

YUAN Wei_ran[1], CHEN Pu[1], LIU Kai_xin[1,2]

(1. LTCS & Department of Mechanics and Aerospace Engineering, Peking University, Beijing 100871, P. R. China;
2. Engineering Research Institute, Peking University, Beijing 100871, P. R. China)

Abstract: A new direct method for solving unsymmetrical sparse linear systems(USLS) arising from meshless methods was introduced. Computation of certain meshless methods such as meshless local Petrov_Galerkin (MLPG) method need to solve large USLS. The proposed solution method for unsymmetrical case performs factorization processes symmetrically on the upper and lower portion of matrix, which differs from previous work based on general unsymmetrical process, and attains higher performance. It is shown that the solution algorithm for USLS can be simply derived from the existing approaches for the symmetrical case. The new matrix factorization algorithm in the method can be implemented easily by modifying a standard JKI symmetrical matrix factorization code. Multi_blocked out_of_core strategies were also developed to expand the solution scale. The approach convincingly increases the speed of the solution process, as is demonstrated with the numerical tests.

Key words: sparse matrices; linear equations; meshless methods; high performance computation