广西科学院学报
Journal of Guangxi Academy of Sciences

2008,24(3):248~250
Vol. 24,No. 3 August 2008

基于 OpenMP 的对称矩阵 LDL^T 分解并行算法实现
Parallel Algorithm Based on OpenMP of LDL^T Decomposition for Symmetric Matrix

张锦雄,韦　化
ZHANG Jin-xiong,WEI Hua

(广西大学电气工程学院,广西南宁　530004)
(School of Electrical Engineering,Guangxi University,Nanning,Guangxi,530004,China)

摘要:分析对称矩阵 LDL^T 分解过程的并行结构,给出对称矩阵 LDL^T 分解的并行算法,然后考虑粗粒度组合后的负载平衡,提出基于 OpenMP 的对称矩阵 LDL^T 分解并行算法的实现方案,并对该方案的并行计算性能进行数值实验.实验表明,基于 OpenMP 的对称矩阵 LDL^T 分解并行算法在多核处理器系统中能够显著地提高算法性能.

关键词:并行计算　对称矩阵　LDL^T 分解　OpenMP
中图法分类号:TP312　　**文献标识码**:A　　**文章编号**:1002-7378(2008)03-0248-03

Abstract: By analyzing the parallelism of LDL^T Decomposition for Symmetric Matrix, a parallel algorithm of LDL^T Decomposition for Symmetric Matrix is proposed and an OpenMP-based implementation with load-balancing for rough granularity is obtain. Then the parallel algorithm performance is tested. The numerical experiments show that the performance of algorithm is obviously improved in multi-core processor system.

Key words: parallel computation, symmetric matrix, LDL^T decomposition, OpenMP

人类对计算机性能的无止境要求极大地推动了高速并行计算的发展,而并行计算与并行计算机、并行算法和并行程序设计密切相关.近年来,随着计算机成本的降低,并行计算大众化、平民化已经成为可能.OpenMP 是当前用于并行程序设计的众多编程语言种类中的一种,它是共享存储系统编程的一个工业标准,具有简单、移植性好和可扩展等优点,是对 Fortran、C 等基本语言的扩展[1].OpenMP 中定义的制导指令、运行库和环境变量,能够使用户将已有的串行程序逐步并行化,以实现显式并行程序设计.

一直以来,与矩阵有关的问题的计算是数值计算中的一个基本问题,而系数矩阵为对称矩阵的线性系统在实际工程计算领域中十分常见,一个线性系统的求解归结为其系数矩阵的分解,因此研究对称矩阵分解的并行算法有着重要的意义.本文首先并行化 LDL^T 分解算法,得到对称矩阵 LDL^T 分解的并行算法,然后利用 OpenMP 并行可递增特点充分挖掘 LDL^T 分解过程的并行性,提出考虑负载平衡的基于 OpenMP 的对称矩阵 LDL^T 分解并行算法的实现方案,最后对该方案的并行计算性能进行了数值实验分析.

收稿日期:2008-06-02

作者简介:张锦雄(1969-),男,讲师,主要从事分布并行计算研究。

1　LDL^T 分解并行算法及算法分析

1.1　LDL^T 分解并行算法

1.1.1　LDL^T 分解[2]

设 A 为 n 阶对称阵,且 A 的所有顺序主子式均不为零,则 A 可唯一分解为 $A = LDL^T$,其中 L 为单位下三角阵,D 为对角阵.设 A、L、D 形如:

$$\begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{21} & a_{22} & & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_n \end{bmatrix} \times$$

$$\begin{bmatrix} 1 & l_{21} & l_{31} & \cdots & l_{n1} \\ & 1 & l_{32} & \cdots & l_{n1} \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}.$$

由矩阵乘法,且注意 $l_{jj}=1, l_{jk}=0, (j<k)$,得

$$a_{ij}=\sum_{k=1}^{n}(LD)_{ik}(L^T)_{kj}=\sum_{k=1}^{n}l_{ik}d_kl_{jk}=\sum_{k=1}^{j-1}l_{ik}d_kl_{jk}+l_{ij}d_jl_{jj}. \quad (1)$$

由(1)式得到计算 L 的元素及 D 的对角元素公式为:对于 $i=1,2,\cdots,n$,有

$$(\mathrm{i})l_{ij}=(a_{ij}-\sum_{k=1}^{j-1}l_{ik}d_kl_{jk})/d_j, (j=1,2,\cdots,i-1); \quad (2)$$

$$(\mathrm{ii})d_i=a_{ii}-\sum_{k=1}^{i-1}l_{ik}^2d_k.$$

为了避免重复计算,引入 $t_{ij}=l_{ij}d_j$,由(2)式得到按行计算 L,T 元素的公式为:对于 $i=1,2,\cdots,n$,有

$$(\mathrm{iii})t_{ij}=a_{ij}-\sum_{k=1}^{j-1}t_{ik}l_{jk}, (j=1,2,\cdots,i-1); \quad (3)$$

$$(\mathrm{iv})l_{ij}=t_{ij}/d_j, (j=1,2,\cdots,i-1);$$

$$(\mathrm{v})d_i=a_{ii}-\sum_{k=1}^{i-1}t_{ik}l_{ik}.$$

1.1.2 LDL^T 分解并行算法

由(3)式可以看出,同一行 t_{ij},l_{ij},d_i 的计算存在明显的数据相关,只能依次顺序进行计算,为消除数据相关,便于实现并行计算,将(3)式的计算顺序改为按列进行,并且将累加和改为超前加,从而形成对称矩阵 LDL^T 分解的并行算法如下:

步骤 1:$a_{ij}^1=a_{ij}$, $1\leqslant j\leqslant i\leqslant n$.

步骤 2:对于 $j=1,\cdots,n-1$,

并行步 1:$d_j=a_{jj}^j, t_{ij}=a_{ij}^j, i=j+1,\cdots,n$;

并行步 2:$a_{pq}^{j+1}=a_{pq}^j-t_{pj}*t_{qj}/d_j, j+1\leqslant q\leqslant p\leqslant n$;

并行步 3:$l_{ij}=t_{ij}/d_j, i=j+1,\cdots,n$.

步骤 3:$d_n=a_{nn}^n$.

1.2 LDL^T 分解并行算法的 OpenMP 描述及算法分析

利用 OpenMP 对 LDL^T 并行分解算法进行描述,并对 OpenMP 所描述的算法进行分析,利用 OpenMP 并行可递增特点充分挖掘 LDL^T 分解过程的并行性.

1.2.1 算法的 OpenMP 描述

在 LDL^T 分解的并行算法中,步骤1、步骤3和步骤2中的并行步1无须进行实质操作,实际上只需将步骤 2中的并行步2和并行步3并行化即可,并行步2和并行步3嵌入 C/C++的 OpenMP 描述如下:

```
for(j=1;j<n;j++)
{
    #pragma omp parallel for shared(a,j)
    num _ threads(n-j-1)
      for(int p=j+1;p<=n;p++)
    #pragma omp parallel for shared(a,p
    ,j) num _ threads(p-j)
        for(int q=j+1;q<=p;q++)
            a[p][q]=a[p][q]-
            a[p][j] * a[q][j]/a[j][j];
    #pragma omp parallel for shared(a,j)
    if(n-j>2)
    for(int i=j+1;i<=n;i++)
            a[i][j]=a[i][j]/a[j][j];
}
```

（并行步2、并行步3）

1.2.2 算法过程分析

由于 A 为对称阵,故只需给出下三角阵即可,存储元素共 $n(n+1)/2$ 个,用一维数组 A 存放,顺序为 $\{a_{11},a_{21},a_{22},a_{31},a_{32},a_{33},\cdots,a_{n1},a_{n2},\cdots,a_{nn}\}$,则矩阵元素 a_{ij} 对应一维数组元素 $A(i\cdot(i-1)/2+j)$. 为节省内存空间,用 D 覆盖 A 的对角线相应位置,T、L 则覆盖 A 的对角线以下相应位置,计算过程直接在 A 上进行,最后结果在 A 上型如下式保存 D、L.

$$\begin{bmatrix} d_1 & & & & \\ l_{21} & d_2 & & & \\ l_{31} & l_{32} & d_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & d_n \end{bmatrix}.$$

并行算法随着 j 的变化,矩阵 A 的变化如下:

$$\begin{bmatrix} a_{pq}^1 & 0 \\ 1\leqslant q\leqslant p\leqslant n & \end{bmatrix} \rightarrow \begin{bmatrix} d_1 & \\ l_{21} & 0 \\ l_{31} & \\ \vdots & \\ l_{n1} & a_{pq}^2 \\ & 2\leqslant q\leqslant p\leqslant n \end{bmatrix} \cdots$$

初态　　迭代1

$$\begin{bmatrix} d_1 & & & \\ l_{21} & \ddots & & 0 \\ l_{31} & \vdots & d_j & \\ \vdots & \vdots & \vdots & \\ l_{n1} & \cdots & l_{nj} & a'^{+1}_{pq} \\ & & & j+1\leqslant q\leqslant p\leqslant n \end{bmatrix} \cdots \begin{bmatrix} d_1 & & & & \\ l_{21} & d_2 & & 0 & \\ l_{31} & l_{32} & d_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & d_n \end{bmatrix}.$$

迭代 j 迭代 n

1.2.3 算法复杂性分析

对于 n 阶对称阵，在 p 个处理器 CREW 计算模型的共享存储系统中，并行算法共需 $n-1$ 次迭代. 如果 $p \geqslant \frac{n(n-1)}{2}$ 时，每次迭代每个处理器最多分到乘、除、加运算各一次，并行执行时间为 $T_{pp}=O(n)$. 否则，每次迭代每个处理器分到乘、除、加运算多次，设处理器每次迭代分到乘、除、加运算次数最多为 $m(m \leqslant \frac{n(n-1)}{2p})$ 次，并行执行时间为 $T_{pp}<(n-1)mT_{mad}=O(nm)$，其中，$T_{mad}=T_m+T_a+T_d$，设 T_m 为乘法时间，T_a 为加法时间，T_d 为除法时间.

在单机系统中，串行算法需执行 $\sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$ 次除法、$\sum_{j=1}^{n-1}(n-j)j=\frac{n^3-n}{6}$ 次乘法和加法，那么串行执行时间 $T_s=\frac{n(n-1)}{2}T_d+\frac{n^3-n}{6}(T_m+T_a)$.

由此有 $SP=\frac{T_s}{T_{PP}}>\frac{\frac{n(n-1)}{2}T_d+\frac{n^3-n}{6}(T_m+T_a)}{(n-1)mT_{mad}}$，

经过化简，并行算法的加速比 $SP=\Omega(k_1p+k_2\frac{p}{n-1})$，其中 k_1、k_2 为常数.

易见，当 $n>>p$ 时，并行算法的加速性能随线程数线性提高，但是随着 p 的增加，并行性开销也随着增大，从而使加速达到极限，所以不能盲目地加大 p 值.

2 基于 OpenMP 的对称矩阵 LDL^T 分解并行算法实现

在 LDL^T 分解的并行算法中，并发线程的计算任务仅为乘、除、加运算各一次，颗粒度太细，而并发线程数很多，从而造成并行性开销增大，继而使性能下降. 故应对线程进行适当的粗粒度组合，同时考虑各线程的负载平衡. 为此，进行基于 OpenMP 的对称矩阵 LDL^T 分解并行算法实现时，依照下述方法进行均衡负载.

设有 $p(2p<n)$ 个处理器，对下三角阵 A 从上到下依次按行划分为行块 $A_1, A_2, \cdots, A_{2p}$，每个行块有 $\frac{n}{2p}$ 行，将 A_1、A_{2p} 分配给 P_1，将 A_2、A_{2p-1} 分配给 $P_2, \cdots$，将 A_p、A_{p+1} 分配给 P_p.

3 数值实验

为验证基于 OpenMP 的对称矩阵 LDL^T 分解并行算法的性能，我们在 Intel(R) Pentium(R) D CPU2.80GHz 双核处理器系统上，测试对称阵阶数分别为500，800，1000时串行算法和并行算法的运行时间，结果见表1.

表1 不同对称阵阶数的串行算法和并行算法的运行结果

算法	500阶数		800阶数		1000阶数	
	时间(s)	加速比	时间(s)	加速比	时间(s)	加速比
串行算法	26.57	1.00	109.828	1.00	215.36	1.00
并行算法	15.88	1.67	65.472	1.68	127.982	1.68

表1结果显示，在双核处理器系统上，由于每个线程被分到不同处理器核上进行处理，并行算法比串行算法获得明显的加速。若能在更多核的系统上运行，将能获得更大加速.

4 结束语

本文通过对 LDL^T 分解算法的分析，提出 LDL^T 分解并行算法，并给出基于 OpenMP 的并行算法描述，同时考虑负载平衡改进了算法实现，实验数据表明，基于 OpenMP 的多线程并行计算在日益普及的多核系统中能够显著地提高算法性能. 本文算法的实现为在 CREW 计算模型的共享存储系统上实现 LDL^T 分解并行计算进行了有益的探讨. 对于分布存储系统上的 LDL^T 分解并行计算，则有待于进一步的研究.

参考文献：

[1] 陈国良. 并行计算——结构·算法·编程[M]. 修订版. 北京：高等教育出版社，2003.

[2] 李庆扬，王能超，易大义. 数值分析[M]. 武汉：华中科技大学出版社，2001.

（责任编辑：韦廷宗）