·体系结构与软件技术· 文章编号:1000-3428(2016)10-0096-05 文献标志码:A 中图分类号:TP301.6

求解大规模三对角线性方程组的 GaBP 并行算法

陈振武,郑汉垣,兰添才,曾志宏

(龙岩学院 信息工程学院,福建 龙岩 364012)

摘 要: 根据大规模三对角线性方程组求解的特性,结合消息传递接口和开放多处理模型,设计分布式共享内存环境下求解大规模三对角线性方程组的 GaBP 并行算法。在 Intel Xeon E5-2650 并行计算集群环境上进行数值实验,结果表明,与基于消息传递接口的 GaBP 并行算法相比,该算法具有更高的加速比和更好的可扩展性,能充分发挥集群系统的综合计算性能,提高大规模三对角线性方程组的求解速度。

关键词: 三对角线性方程组;GaBP 算法;并行算法;加速比;可扩展性

中文引用格式: 陈振武,郑汉垣,兰添才,等. 求解大规模三对角线性方程组的 GaBP 并行算法[J]. 计算机工程,2016,42(10):96-100.

英文引用格式: Chen Zhenwu, Zheng Hanyuan, Lan Tiancai, et al. Parallel GaBP Algorithm for Solving Large-scale Tridiagonal Linear Equations[J]. Computer Engineering, 2016, 42(10):96-100.

Parallel GaBP Algorithm for Solving Large-scale Tridiagonal Linear Equations

CHEN Zhenwu, ZHENG Hanyuan, LAN Tiancai, ZENG Zhihong

(School of Information Engineering, Longyan University, Longyan, Fujian 364012, China)

【Abstract】 According to the solution characteristics of large-scale tridiagonal linear equations, based on distributed shared memory environment of mixing Message Passing Interface(MPI) with Open Multi-processing(OpenMP), this paper designs a parallel GaBP algorithm to solve large-scale tridiagonal linear equations. Tests are done through parallel computing cluster environment of Intel Xeon E5-2650, and the results show that the proposed algorithm has higher speed up ratio and better scalability, compared with parallel GaBP algorithm based on MPI. It can take fuller advantage of the overall computing performance of cluster system and improve the speed of solving the large-scale tridiagonal linear equations.

【Key words】 tridiagonal linear equations; Gaussian Belief Propagation(GaBP) algorithm; parallel algorithm; speed up ratio; scalability

DOI: 10.3969/j.issn.1000-3428.2016.10.017

1 概述

高性能多处理机的硬件结构有紧耦合和松耦合两种不同的构型,前者通过共享主存实现处理机间通信的;后者则是一个分布式存储系统,任务间的信息流量较小,比较适合于粗粒度的并行计算。分布式共享存储集群系统可以看成是上述两种构型的结合,它具有多个处理器共享内存的节点和连接各个节点的快速网络构成的多级体系结构,在节点内共享存储,在节点间分布式存储[1]。

文献[2]针对分布式共享存储集群系统分析消息传递接口(Message Passing Interface, MPI)和开放多处理(Open Multi-processing, OpenMP)等并行算法的优势与编程实现方法,对稀疏线性方程组、三对角线性方程组的求解也给出了相应的求解方法。文献[3]分析线性方程组的求解在大规模科学工程和数值处理中的普遍应用,综述利用 Krylov 子空间算法进行迭代法求解稀疏线性方程组的过程,并给出在分布式环境下的并行化策略。文献[2-3]指出线性方程组的求解速度与复杂度取决于矩阵系数中非零值的位置。文献[4]通过机群来并行求解虚拟输电分区的电场强度线性方程组,并通过实验表明,减少各处理器间信息交换的通信时间可以提高矩阵矢量积求解的效率,进而加快线性方程组的并行求解速度。

文献[5]针对严格对角占优三对角线性方程组

基金项目: 福建省自然科学基金资助项目(2015J01587);福建省教育厅 A 类基金资助项目(JA09229);福建省科技厅高校基金资助项目(JK2010056);龙岩学院服务海西基金资助项目(JB10160, LYXY2011067)。

作者简介: 陈振武(1963—),男,实验师,主研方向为计算机体系结构;郑汉垣,副教授、博士;兰添才,副教授;曾志宏,讲师、硕士。

收稿日期: 2015-10-12 **修回日期:** 2015-11-17 **E-mail:** lyczw@126.com

提出重叠分割并行近似算法,这种算法不需要大量通信,但只能在机器精度范围内得到与目标要求几乎等价的近似解。文献[6]在文献[5]算法的基础上提出一种可扩展通信的非近似算法用于求解一个对角占优三对角线性方程组的 Toeplitz 系统。

如何充分发挥高性能多处理机系统的性能和对三对角系统进行有效求解,其关键在于设计具有可扩展性好、并行度高、易于实现的并行算法与程序。GaBP(Gaussian Belief Propagation)是在递归更新的概率统计推理的基础上,针对对称对角占优线性代数方程组求解的迭代算法,具有低复杂性和高并行性的特性,并在对称对角占优线性方程组的求解具有良好的收敛性[7-8],因此,GaBP 算法非常适用于处理大规模稀疏线性方程组的求解问题。

MPI 和 OpenMP 是并行计算领域流行的两种编程模型,MPI 适用于分布式存储并采用消息传递机制,硬件结构简单,但细粒度并行环境中常常会引发并行环境大量的数据通信及动态负载平衡困难;OpenMP 充分利用共享存储器消息传递相对简单的特点,兼容了对称式多处理机使用的通信机制,但用软件实现的共享存储器开销很大,严重地限制了共享存储器可支持的应用程序范围[2]。文献[9]在求解线性方程组的 GaBP 算法理论基础上,对比 MPI 和 OpenMP 分别求解线性方程组的性能上的差异,给出利用消息传递技术求解线性方程组的方法与应用验证。文献[10]分析基于多核处理器架构环境中的混合编程模式的现状及 MPI 和 OpenMP 的可扩展性,研究采用 MPI 和 OpenMP 混合的编程模式,发现这种混合编程模式具有更好的性能,并提出在多个线程环境中利用这种混合编程模式求解线性方程组的改进之处。

本文在研究 MPI 和 OpenMP 混合编程的并行算法基础上,对三对角线性方程组求解的 GaBP 算法进行优化,将 MPI 与 OpenMP 混合的 GaBP 并行算法应用于三对角线性方程组的求解,并通过算例进行算法的性能与可行性验证。

2　GaBP 算法与数据结构

2.1　GaBP 算法

GaBP 算法主要有初始化和迭代计算两部分,而迭代计算又可以表述为消息累加、消息传播及更新、求解向量 3 个过程[11-12]。

(1)初始化

$$p_{ij}=0,\mu_{ij}=0(i\neq j) \tag{1}$$

(2)迭代计算

1)消息累加:

$$\tilde{p}_j=A_{jj}+\sum_{k\in N(j)}p_{kj},\tilde{\mu}_j=b_j+\sum_{k\in N(j)}\mu_{kj} \tag{2}$$

2)消息传播及更新:

$$p_{i/j}=\tilde{p}_i-p_{ji},\mu_{i/j}=\tilde{\mu}_i-\mu_{ji} \tag{3}$$

$$p_{ij}=-A_{ij}^2p-1_{i/k},\mu_{ij}=-p-1_{i/j}A_{ij}\mu_{i/j} \tag{4}$$

3)求解向量:

$$\boldsymbol{x}_i=\tilde{\mu}_i/\tilde{p}_i \tag{5}$$

在上述各计算公式中,$N(i)$为第 i 个节点对应邻居节点的集合(不含 i);$N(i)/j$ 表示 $N(i)$中除去节点 j。

由式(2)可知,消息累加和$\tilde{p}_i$ 与$\tilde{\mu}_i$ 是按列进行求和的。因此为了方便求和,采用列块划分方式进行数据划分。

设置数据存储结构有:$\boldsymbol{A},\boldsymbol{b},\boldsymbol{x}$ 分别对应于左端已知矩阵、右端已知向量、左端未知向量;p,u 分别表示对应于求 p_{ij}和μ_{ij}的列排列计算数据;p_r,u_r 表示对应于求 $p_{i/j}$和 $\mu_{i/j}$数据的行排列缓冲区。

2.2　GaBP 编程实现方法

对 GaBP 算法可以有同步算法异步算法两种不同的编程实现方法[8]。

(1)同步 GaBP 算法描述如下:

```
Initialization:
  p_ij = 0; μ_ij = 0;
Iteration:
  repeat
1)消息累加
for all i ∈ 0…n do
     P_i = A_ii; μ_i = b_i;
   for all j ∈ 0…n do
     if(i≠j and A_ij ≠0) then
        P_i = P_i + P_ji
        μ_i = μ_i + μ_ji
     end if
  end for
end for
2)消息传播及更新
for all i ∈ 0…n do
  for all j ∈ 0…n do
    if(i≠j and A_ij ≠0) then
      P_ij = -A_ij^2/(P_i - P_ji)
      μ_ij = -A_ij(μ_i - μ_ji)/(P_i - P_ji)
    end if
  end for
end for
3)求解向量
for all i ∈ 0…n do
  P_i = A_ii + Σ_{k∈N(i)} p_ki
  μ_i = b_i + Σ_{k∈N(i)} μ_ki
  x_i = μ_i/p_i
end for
until convergence: all x_j converged
Output: x* = [x_i]
```

(2)异步 GaBP 算法描述如下:

```
Initialization:
  p_ij = 0; μ_ij = 0;
```

```
    p_i = A_ii ; μ_i = b_i
Iteration:
    repeat
      for all i∈0…n do
    for all j∈0…n do
      if( i≠j and A_ij ≠0) then
        P_ij = -A_ij^2/(P_i - P_ji)
        μ_ij = -A_ij(μ_i - μ_ji)/(P_i - P_ji)
      end if
    end for
      P_i = A_ii ; μ_i = b_i ;
    for all j∈0…n do
      if (i≠j and A_ij ≠0) then
        P_i = P_i+ P_ji
        μ_i = μ_i + μ_ji
      end if
    end for
  end for
  until convergence: all μ_j converged
  for all i∈0…n do
    x_i = μ_i/p_i
  end for
  output: x* = [x_i]
```

2.3 数据划分与数据结构

对应于三对角线性方程组求解中的左端已知矩阵 $\boldsymbol{A}$ 就是一个三对角矩阵,可按如图 1 所示的分裂方式进行数据块划分[14]。

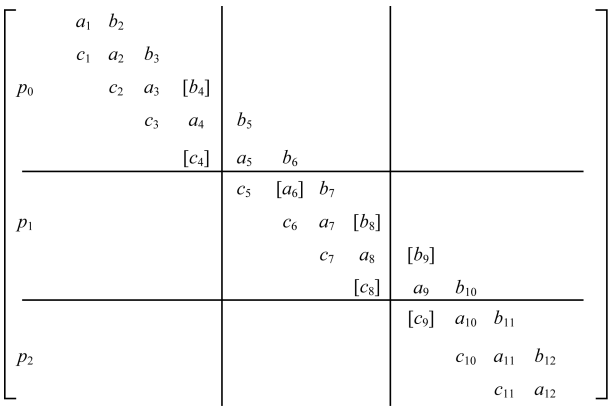

图 1 三对角矩阵的分裂方式

从上述的三对角矩阵来看,每一列只有 3 个元素:$b_j, a_j, c_j (j=1,2,\cdots,n)$。

在用 A_{ij} 格式存储的情况下,当 $i \in N(j)=(j-1),(j+1)$ 时,则第 j 列:$A_{(j-1),j}, A_{jj}, A_{(j+1),j}, (j=1,2,\cdots,n)$。若将它们用 $A[3][n]$ 数组来存储,则有:$b_j=A[1][j], a_j=A[2][j], c_j=A[3][j]$。

对应于 A_{ij} 的 p_{ij}, μ_{ij} 也分别可以用 $P[3][n]$ 和 $U[3][n]$ 存储,而对应于 $\tilde{p}_i$ 与 $\tilde{\mu}_i$,可用 $sp[j]$ 和 $su[j](j=1,2,\cdots,n)$ 来存储。

现令方程组右端已知向量为 $\boldsymbol{d}$,未知向量为 $\boldsymbol{x}$,分别用数组 $d[n], x[n]$ 来存储。

利用上述的数据结构存储数据后,对角线上的元素 A_{jj} 对应的 p_{jj}, μ_{jj} 在计算过程中是一直不变的(初始化时即已确定),因此,对应于第 j 列的 p_{ij}, μ_{ij} 和 $\tilde{p}_i, \tilde{\mu}_i$ 可表示为:$p_{jj}=A_{jj}, \mu_{jj}=b_j$,表示为数组元素的公式则为:$p[2][j]=A[2][j], U[2][j]=d[j]$。而计算 P_{ij}, μ_{ij} 时,对应于第 j 列,只需计算 $P[1][j]$,$U[1][j]$ 和 $P[3][j], U[3][j]$;而计算 $\tilde{p}_i, \tilde{\mu}_i$,则对应于下列公式:

$$sp[j]=p[1][j]+p[2][j]+p[3][j]$$
$$su[j]=U[1][j]+U[2][j]+U[3][j]$$

其中,$P_{ij}(j=1,2,\cdots,n, i=j-1,j+1)$ 则可表示为 $P_{j-1,j}$ 和 $P_{j+1,j}$ 的数组表示 $P[1][j]$ 与 $P[3][j]$。

同理,P_{ji} 则可表示为 $P_{j,j-1}$ 和 $P_{j,j+1}$ 的数组表示 $P[3][j-1]$ 与 $P[1][j+1]$。

计算 $P_{ij}=-A_{ij}^2(\tilde{p}_i-P_{ji})(j=1,2,\cdots,n, i=j-1,j+1)$,则可用数组存储格式表示为:

$$p[1][j]=-A[1][j]\times A[1][j]\times(sp[j-1]-p[3][j-1])$$
$$p[3][j]=-A[3][j]\times A[3][j]\times(sp[j+1]+p[1][j+1])$$

化简后得到公式:

$$p[1][j]=-A[1][j]\times A[1][j]/(p[1][j-1]+p[2][j-1]) \tag{6}$$
$$p[3][j]=-A[3][j]\times A[3][j]/(p[2][j+1]+p[3][j+1]) \tag{7}$$

计算 $\mu_{ij}=A_{ij}(\tilde{p}_i-p_{ji})(\tilde{\mu}_i-\mu_{ji})(j=1,2,\cdots,n, i=j-1,j+1$ 时用数组存储格式表示为:

$$U[1][j]=A[1][j]/(p[1][j-1]+p[2][j-1])\times(U[1][j-1]+U[2][j-1]) \tag{8}$$
$$U[3][j]=A[3][j]/(p[2][j+1]+p[3][j+1])\times(U[2][j+1]+U[3][j+1]) \tag{9}$$

将 A_{jj}, b_j 分别代入式(6)~式(9)可得到:

$$p[1][j]=-A[1][j]\times A[1][j]/(p[1][j-1]+A[2][j-1])$$
$$p[3][j]=-A[3][j]\times A[3][j]/(A[2][j+1]+p[3][j+1])$$
$$U[1][j]=A[1][j]/(p[1][j-1]+A[2][j-1])\times(U[1][j-1]+b[j-1])$$
$$U[3][j]=A[3][j]/(A[2][j+1]+p[3][j+1])\times(b[j+1]+U[3][j+1])$$

用更直观的形式即表示为:

$$p_{1j}=-A_{1j}^2/(p_{1,j-1}+A_{2,j-1})$$
$$p_{3j}=-A_{3j}^2/(p_{3,j+1}+A_{2,j+1})$$
$$U_{1j}=A_{1j}/(p_{1,j-1}+A_{2,j-1})\times(U_{1,j-1}+b_{j-1})$$
$$U_{3j}=A_{3j}/(p_{3,j+1}+A_{2,j+1})\times(U_{3,j+1}+b_{j+1})$$

从以上的算法分析过程可知,计算过程只与矩阵的前一列和后一列有关,而且可以省去对消息累

加$\tilde{p}_i$,$\tilde{\mu}_i$的计算。

对于上面的划分以后,计算节点P_1,只需b_5,c_8两个位置的计算,其他位置的计算都只与本节点内部位置上的数据有关。而计算b_5需前一列的b_4位置的数据,计算c_8需后一列c_9位置的数据。因此,总的来说,在计算过程中只需将计算节点p_1第一列的c_5位置数据往p_0传输,将计算节点p_1的最后一列b_8位置的数据往p_2传输;计算节点p_1要接收p_0的最后一列b_5位置的数据,还要接收p_2的第一列c_9的数据。整个数据通信都是局部的,从并行算法设计的理论上看,随着计算规模的不断扩大,通信时间不变,计算所用的时间必然更小,算法的加速比也必然得到提高,算法是可扩展性并行算法。

3　三对角线性方程的 GaBP 并行算法实现

三对角线性方程 GaBP 的 MPI 并行算法步骤如下:(1)初始化;(2)迭代开始;(3)消息累加;(4)消息更新;(5)求解向量与收敛性判断;(6)数据交换;(7)转到步骤(2)。

给定计算节点迭代步骤的算法主要代码如下:

```
Initialization:
  p[ ][j] = 0;u[ ][j] = 0;x[j] = 0;x0[j] = 0;
Iteration:
  repeat
for j = 0 to m + 1 do
  p[1][j] = p[0][j]/* 接收前一节点的最后一列数据,将第一列的数据发送到前一节点上 */
  u[1][j] = u[0][j]
  p[1][j + 1] = p[0][j]
  u[1][j + 1] = u[0][j]
  p[2][j + 1] = p[1][j + 1]/* 接收后一节点的最前一列数据,将最后一列数据发送到后一节点上 */
  u[2][j + 1] = u[1][j + 1]
  enddo
for j = 1 to m do/* 消息累加 */
  spj = p[1][j] + A[2][j] + p[3][j];
  suj = U[1][j] + b[j] + U[3][j];
  x[j] = suj/suj
 enddo
for j = 1 to m do/* 消息更新从第一列计算到最后一列 */
  p[1][j] = -A[1][j] × A[1][j]/(p[1][j -1] + A[2][j -1];
  p[3][j] = -A[3][j] × A[3][j]/(p[3][j +1] + A[2][j +1];
  U[1][j] = -A[1][j]/(p[1][j - 1] + A[2][j - 1]) × (U[1][j -1] + b[j -1]);
  U[3][j] = -A[3][j]/(p[3][j + 1] + A[2][j + 1]) × (U[3][j +1] + b[j +1]);
 enddo
  until convergence;all μj converged/* 向量求解与收敛性判断 */
  for all j ∈ 0…m do
    xj = μj/pj
  if flag(q) = 1  then/* 容忍度判断 */
    output:x* = [xj]
    else
      x0[j] = x[j]/* 保存上一次求解向量结果 */
    endif
  end for
  for each q ∈ 0…m do/* 节点数据交换与传输 */
    p[1][0] = p[1][mq-1]
    U[1][0] = U[1][mq-1]
    p[3][mq+1] = p[3][1]
    U[3][mq+1] = U[3][1]
  endndo
  for each q ∈ 0…m do/* 节点 q 发送数据的传输 */
    p[3][mq-1 + 1] = p[3][1]
    U[3][mq-1 + 1] = U[3][1]
    p[1][0] = p[1][mq]
    U[1][0] = U[1][mq]
  for j = 1 to m do/* 传输策略,采取异步进行 */
    for each q ∈ 0…m do
      p[1][0] = p[1][mq-1]
      U[1][0] = U[1][mq-1]
      /* 计算第 1 列之前先接收前一节点尾列数据 */
      p[1][j] = -A[1][j] × A[1][j]/(p[1][j - 1] + A[2][j -1];
      U[1][j] = -A[1][j]/(p[1][j - 1] + A[2][j - 1]) × (U[1][j -1] + b[j -1]);
      p[1][0] = p[1][mq-1]
      U[1][0] = U[1][mq-1]
      /* 计算完第 1 列后立即将数据往前一节点传 */
  enddo
    p[3] = [1]p[3][mq-1 + 1]
    U[3][1] = U[3][mq-1 + 1]
    /* 计算最后列前先接收后一节点的首列数据 */
    p[3][j] = -A[3][j] × A[3][j]/(p[3][j + 1] + A[2][j +1];
    U[3][j] = -A[3][j]/(p[3][j +1] + A[2][j +1]) × (U[3][j +1] + b[j +1]);
    p[3][mq-1 + 1] = p[3][1]
    U[3][mq-1 + 1] = U[3][1]
  enddo
```

一个 OpenMP 程序可以抽象成为一个或多个顺序执行或嵌套执行的 for 子程序[14],因此,对于 MPI 与 OpenMP 混合的 GaBP 并行算法实现,需在 GaBP 的 MPI 并行算法的基础上添加 OpenMP 线程级并行的制导控制编译语句,即在上述 GaBP 的 MPI 并行算法的第(4)步中的执行循环体:for each j = jq1 to jqm do 的前面添加#pragma omp parallel for private(…),即:

```
# pragma omp parallel for private(…)
for each j = jq1 to jqm do
…
enddo
```

并将此循环体作为 OpenMp 多线程并行语句块提供调用即可。

4 数值实验

4.1 实验环境

实验环境由 5 个节点、每个节点 2 个 Intel Xeon E5-2650 的 CPU、96 GB 内存、Redhat Linux6.3 X64 操作系统和 Gcc-4.8.1 与 MPICH2 编译与并行计算的集群环境构成。

4.2 实验结果及分析

以美国佛罗里达大学的稀疏矩阵集(UFget)[15]作为实验数据构造 MPI 程序,在每个节点上执行 4 个进程,而对应于 MPI 与 OpenMp 程序则在每个节点上各创建一个进程,每个进程内包含 4 个线程,通过这样的设置,可以对两种算法环境中的并行程序的运行加速比及性能进行比较。

图 2、图 3 给出了该集群系统上 MPI 算法和 MPI 与 OpenMP 多级混合算法在节点个数不同的环境下(总节点数为 13 个)的加速比比较曲线。从图 2 中可看出,当节点数小于 6 时,由于节点个数少通信开销小延迟时间短,MPI 与 OpenMP 的混合并行程序性能低于 MPI 并行程序的性能;从图 3 中可看出,当节点数大于 6 时,MPI 与 OpenMP 的混合并行算法的性能才优于 MPI 并行程序的性能,同时,随着节点数的增加,通信开销不断加大,MPI 与 OpenMP 的混合并行算法的性能明显优于 MPI 并行算法的性能。

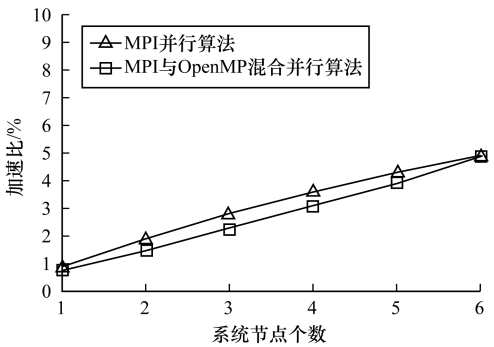

图 2 节点数小于 6 时的加速比曲线比较

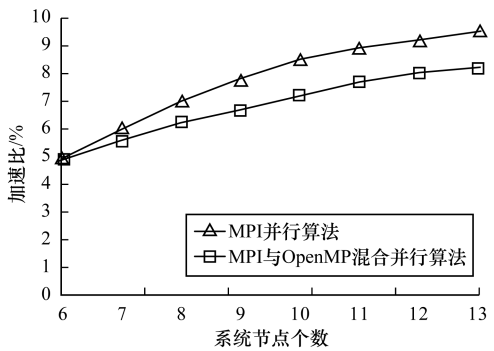

图 3 节点数大于 6 时的加速比曲线比较

由此可见,MPI 与 OpenMP 混合并行算法比单纯的 MPI 算法具有更好的加速比和可扩展性,并且随着系统带宽、线程数的增加、进程间的通信减少,MPI 与 OpenMP 混合并行算法的优势更加明显。

5 结束语

在给出三对角线性方程组求解的 GaBP 优化算法基础上,本文应用多粒度 MPI 与 OpenMP 混合编程模型实现了大规模三对角线性方程组求解的 GaBP 并行算法。实验结果表明,该算法在分布式共享存储系统中比 MPI 编程的 GaBP 并行算法具有更好的性能,能够更充分地发挥集群系统在大规模三对角线性方程组求解应用中的运算能力。

参考文献

[1] 李学干. 计算机系统结构[M]. 西安:西安电子科技大学出版社,2012.

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

[3] 李晓梅,吴建平. Krylov 子空间方法及其并行计算[J]. 计算机科学,2005,32(1):19-20,40.

[4] 厉天威,阮江军,吴 田. 并行计算高压输电线路周围电场[J]. 电工技术学报,2009,24(7):2-15.

[5] Larriba-Pey J L,Jorba A,Navarro J J. Spike Algorithm with Savings for Strictly Diagonal Dominant Tridiagonal Systems[J]. Microprocessing & Microprogramming,1993,39(2-5):125-128.

[6] Mcnally J M,Garey L E,Shaw R E. A Communication-less Parallel Algorithm for Tridiagonal Toeplitz Systems[J]. Journal of Computational & Applied Math-ematics,2008,212(2):260-271.

[7] Ori S,Paul H S,Jack K W,et a1. Gaussian Belief Propagation Solver for Systems of Linear Equations[C]//Proceedings of IEEE International Symposium on Information Theory. Toronto,Canada:ISIT,2008:1863-1867.

[8] El-Kurdi Y,Gross W J,Giannacopoulos D. Efficient Implementation of Gaussian Belief Propagation Solver for Large Sparse Diagonally Dominant Linear Systems[J]. IEEE Transactions on Magnetics,2012,48(2):471-474.

[9] Géraud K,Franck C. Performance Comparison of MPI and OpenMP on Shared Memory Multiprocessors[J]. Concurrency Computation Practice and Experience,2006,18(1):29-61.

[10] Mark B,James E. Performance Evaluation of Mixed-mode OpenMP/MPI Implementations[J]. International Journal of Parallel Programming,2010:38(5/6):396-417.

[11] 李开泰,黄艾香. 有限元方法及其应用[M]. 北京:科学出版社,2006.

[12] Saad Y. 稀疏线性系统的迭代方法(影印版)[M]. 2 版. 北京:科学出版社,2009.

[13] 李太全,肖柏勋. 求解三对角线性方程组的迭代对角占优算法[J]. 计算机应用,2012,32(10):2742-2744.

[14] 李 婷,徐 云,聂鹏宇,等. 一种跨平台的并行编程框架设计与实现[J]. 计算机工程,2014,40(8):43-47.

[15] Davis T A,Hu Yifan. The University of Florida Sparse Matrix Collection[J]. ACM Transactions on Mathematical Software,2011,38(1).

编辑 金胡考