Efficient Tridiagonal Solvers on Multicomputers

Xian-He Sun, *Member, IEEE*, Hong Zhang, and Lionel M. Ni, *Senior Member, IEEE*

Abstract—**Three parallel algorithms, namely the parallel partition LU (PPT) algorithm, the parallel partition hybrid (PPH) algorithm, and the parallel diagonal dominant (PDD) algorithm are proposed for solving tridiagonal linear systems on multicomputers. These algorithms are based on the divide-and-conquer parallel computation model. The PPT and PPH algorithms support both pivoting and nonpivoting. The PPT algorithm is good when the number of processors is small; otherwise, the PPH algorithm is better. When the system is diagonal dominant, the PDD algorithm is highly parallel and provides an approximate solution which equals to the exact solution within machine accuracy. Both computation and communication complexities of the three algorithms are presented. All three methods proposed in this paper have been implemented on a 64-node nCUBE-1 multicomputer. The analytic results match closely with the results measured from the nCUBE-1 machine.**

Index Terms—**Communication complexity, divide-and-conquer, LU decomposition, matrix partitioning, parallel numerical algorithms, multicomputers, tridiagonal systems.**

I. Introduction

We are interested in solving the following tridiagonal linear system of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{d} \tag{1.1}$$

where $\boldsymbol{A}$ is a tridiagonal matrix of order n

$$\boldsymbol{A} = \begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix} \tag{1.2}$$

$\boldsymbol{x} = (x_0, \cdots, x_{n-1})^T$ and $\boldsymbol{d} = (d_0, \cdots, d_{n-1})^T$. We assume that $\boldsymbol{A}$, $\boldsymbol{x}$, and $\boldsymbol{d}$ have real coefficients. Extension to the complex case is straightforward.

Solving tridiagonal linear systems is one of the most important problems in scientific computing. It is involved in the solution of differential equations and in various other areas of science and engineering. A variety of methods have been developed for solving (1.1) on parallel computers. A good survey of these methods can be found in [9], [13], [14], and [18]. Among them, the *recursive doubling method* (RCD) developed by Stone [22] and the *cyclic reduction* or *odd–even reduction method* (OER) developed by Hockney [10] are able to solve (1.1) in $O(\log n)$ time using n processors. Here and throughout, log denotes $\log_2$. However, in a realistic parallel processing environment, the number of processors, p, is usually less or much less than the dimension of the matrix, n. Lawrie and Sameh [16] proposed an algorithm in 1984 for a practical parallel environment. Recently, a *parallel prefix* (PP) method developed by Egecioglu, Koc, and Laub [7] modifies the RCD method to be applied to a limited number of processors, i.e., $p < n$. *Wang's partition* algorithm [23] is another relatively new method which is designed for a realistic parallel environment, and it has been efficiently implemented in [11]. While the OER algorithm is a popular choice for vector machines, the Lawrie–Sameh's algorithm, RCD algorithm, and Wang's algorithm are good candidates for multiprocessors.

A *multicomputer* is a distributed-memory multiprocessor in which each memory module is physically associated with each processor. A point-to-point interprocessor communication network provides a mechanism for communication between processors [12]. Multicomputers with hundreds or thousands of processors are commercially available. Multicomputers hold the promising potential for providing massive parallelism. It is for this reason the algorithms for multicomputers are studied.

Three parallel algorithms are proposed in this paper. All these algorithms are developed based on Sheman and Morrison matrix modification formula [20]. To start with, a tridiagonal matrix $\boldsymbol{A}$ is partitioned into p submatrices. Solving these p subsystems in parallel, it leads to a reduced tridiagonal system of order $2(p-1)$. If p is small, the *parallel partition LU* (PPT) algorithm solves this small tridiagonal system in a serial mode. If p is large, the reduced system can be solved in parallel by the PP method. This algorithm is called the *parallel partition hybrid* (PPH) algorithm. In many applications, the tridiagonal matrix $\boldsymbol{A}$ is evenly diagonal dominant. In this case half of the off-diagonal elements of the small $2(p-1)$-dimensional matrix converge to zero exponentially. A fast and highly parallel algorithm, namely the *parallel diagonal dominant* (PDD) algorithm, is proposed. All the three algorithms have less communication requirement than Wang's algorithm [17], [23]. The PPT algorithm achieves a time complexity very close to the algorithm proposed by Ho and Johnsson [11]. Based on a matrix modification formula, our matrix partitioning scheme ends up very similar to that of Lawrie and Sameh [16]. However, our algorithms use different strategies in solving the reduced system, which reduce the communication overhead

Manuscript received March 3, 1989; revised October 15, 1990 and March 7, 1991. This work was supported in part by the NSF Grants CCR-87-16833, ECS-88-14027, and by the Applied Mathematical Sciences Program of the Ames Laboratory, U.S. Department of Energy under Contract W-7405-ENG-82.

X.-H. Sun was with the Department of Computer Science, Michigan State University, East Lansing, MI 48824. He is now with the Ames Laboratory, Iowa State University, Ames, IA 50011.

H. Zhang is with the Mathematical Sciences Department, Clemson University, Clemson, SC 29634-1907.

L. M. Ni is with the Department of Computer Science, Michigan State University, East Lansing, MI 48824.

IEEE Log Number 9104167.

significantly. Particularly, the PDD algorithm gives a fast solution that approximates the exact solution to the machine accuracy when $n/p \gg 1$. Although the matrix modification formula being used is not crucial in designing these algorithms, it provides a nice mathematical equation that an error analysis can be conducted formally [25].

Communication mechanism has a great impact on the performance of multicomputers. Thus, the communication pattern of parallel algorithms should be carefully designed to reduce the communication complexity. In this study, the algorithms are evaluated based on both computation and communication complexities.

This paper is organized as follows. Section II briefly reviews the parallel prefix method, which is used in the development of our parallel algorithms. The matrix partitioning method is presented in Section III. Based on the matrix partitioning scheme, three parallel algorithms, PPT, PPH, and PDD are discussed in Section IV, Section V, and Section VI, respectively. These parallel algorithms are implemented on a 64-node nCUBE multicomputer. Section VII presents analytic and experimental results and comparison discussion. Advantages of our proposed algorithms are summarized in Section VIII.

II. Preliminary

Two methods will be used in our algorithms. They are LU decomposition method and parallel prefix method. For completeness, the latter is briefly presented in this section.

The RCD method uses $O(\log n)$ parallel computation time to solve (1.1) on a parallel computer with n processors [10], [22]. The PP method [7] modifies the RCD method for p processors with $p < n$.

Equation (1.1) can be written as

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i. \qquad 0 \le i \le n-1 \tag{2.1}$$

where $x_{-1} = x_n = 0$. Solving for x_{i+1}, we have

$$x_{i+1} = (-\frac{b_i}{c_i})x_i + (-\frac{a_i}{c_i})x_{i-1} + (\frac{d_i}{c_i}) = \alpha_i x_i + \beta_i x_{i-1} + \gamma_i. \tag{2.2}$$

Here $c_i \neq 0$ is assumed. Equation (2.2) then can be written in the matrix form as

$$\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix}.$$

Define

$$\boldsymbol{X}_{i+1} = \begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} \qquad \text{with } x_{-1} = x_n = 0.$$

Equation (2.2) becomes

$$\boldsymbol{X}_{i+1} = \boldsymbol{B}_i \boldsymbol{X}_i. \qquad 0 \le i \le n-1, \tag{2.3}$$

and $\boldsymbol{X}_i$ $(1 \le i \le n)$ can be expressed in terms of $\boldsymbol{X}_0$ as

$$\boldsymbol{X}_i = \boldsymbol{B}_{i-1}\boldsymbol{B}_{i-2}\cdots\boldsymbol{B}_1\boldsymbol{B}_0\boldsymbol{X}_0, \qquad 1 \le i \le n.$$

Now solving (1.1) becomes the finding of all the partial products of matrices $\boldsymbol{B}_i$ for $0 \le i \le n-1$ and x_0. If $p < n$, we first evenly distribute matrices $\boldsymbol{B}_i$'s to p processors, perform sequential matrix multiplication on each processor, then use the prefix method on p processors. There are $(\log p)+1$ parallel communication steps in applying the prefix method with p processors.

Let $\boldsymbol{C}_i^j = \boldsymbol{B}_j\boldsymbol{B}_{j-1}\cdots\boldsymbol{B}_i$. Then $\boldsymbol{C}_i^j$ is a 3×3 matrix. Since the last row of $\boldsymbol{C}_i^j$ always equals [0,0,1], only six entries of $\boldsymbol{C}_i^j$ need to be transferred at each parallel communication. Fig. 1 shows the communication pattern of the parallel prefix method for obtaining all the partial matrix products with $n = 16$ and $p = 8$.

The actual communication complexity of the PP method depends on the mapping of computation units into processors, the interconnection topology of the multicomputer, and the underlying communication mechanism. In the case of hypercube topology, the communication pattern shown in Fig. 1 can be mapped such that the dilation cost is no greater than 2 [13].

One drawback of the PP method is that the method is unstable when $|c_i|$ is small relative to $|a_i| + |b_i|$ in (2.1). It has been shown that some stability problems arise in the use of the algorithm when the size of the system is large [6].

III. A Matrix Partitioning Technique

Our parallel algorithms are based on the divide and conquer model of parallel computation. In the divide part, the matrix $\boldsymbol{A}$ is partitioned into submatrices. For convenience we assume that $n = pm$. The matrix $\boldsymbol{A}$ in (1.2) can be written as

$$\boldsymbol{A} = \tilde{\boldsymbol{A}} + \Delta\boldsymbol{A}, \tag{3.1}$$

where

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_0 & & & \\ & \mathbf{A}_1 & & \\ & & \ddots & \\ & & & \mathbf{A}_{p-1} \end{bmatrix} \qquad \Delta\mathbf{A} = \begin{bmatrix} & c_{m-1} & & \\ a_m & & \ddots & \\ & \ddots & & c_{(p-1)m-1} \\ & & a_{(p-1)m} & \end{bmatrix}$$

The submatrices $\boldsymbol{A}_i$ $(i = 0, \cdots, p-1)$ are $m \times m$ tridiagonal matrices. Let $\boldsymbol{e}_i$ be a column vector with its ith $(0 \le i \le n-1)$ element being 1 and all the other entries being zero. We have

$$\Delta\boldsymbol{A} = [a_m\boldsymbol{e}_m, c_{m-1}\boldsymbol{e}_{m-1}, a_{2m}\boldsymbol{e}_{2m}, c_{2m-1}\boldsymbol{e}_{2m-1}, \cdots, c_{(p-1)m-1}\boldsymbol{e}_{(p-1)m-1}] \begin{bmatrix} \boldsymbol{e}_{m-1}^T \\ \boldsymbol{e}_m^T \\ \cdot \\ \cdot \\ \cdot \\ \boldsymbol{e}_{(p-1)m-1}^T \\ \boldsymbol{e}_{(p-1)m}^T \end{bmatrix} = \boldsymbol{V}\boldsymbol{E}^T$$

where both $\boldsymbol{V}$ and $\boldsymbol{E}$ are $n \times 2(p-1)$ matrices. Thus, we have

$$\boldsymbol{A} = \tilde{\boldsymbol{A}} + \boldsymbol{V}\boldsymbol{E}^T. \tag{3.2}$$

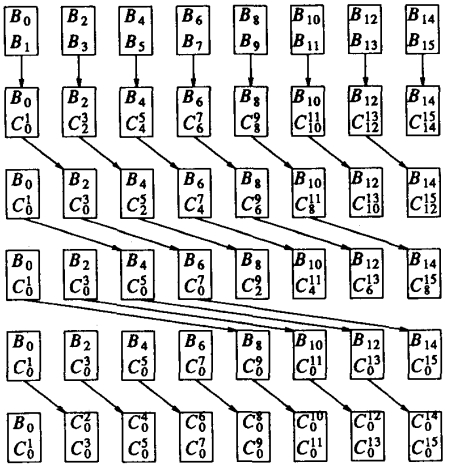


Fig. 1. The communication pattern of the parallel prefix method for obtaining partial products.

Based on the matrix modification formula originally defined by Sherman and Morrison [20] for rank-one changes and generalized by Woodbury [5], [24], and assuming that all A_i's are invertible, (1.1) can be solved by

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{A}^{-1}\boldsymbol{d} = (\tilde{\boldsymbol{A}} + \boldsymbol{V}\boldsymbol{E}^T)^{-1}\boldsymbol{d} \\ &= \tilde{\boldsymbol{A}}^{-1}\boldsymbol{d} - \tilde{\boldsymbol{A}}^{-1}\boldsymbol{V}(\boldsymbol{I} + \boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{V})^{-1}\boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{d}. \quad (3.3) \end{aligned}$$

Note that $\boldsymbol{I}$ is an identity matrix and $\boldsymbol{I} + \boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{V}$ is a pentadiagonal matrix of order $2(p-1)$. We introduce a permutation matrix $\boldsymbol{P}$ such that

$$\boldsymbol{P}\boldsymbol{z} = (z_1, z_0, z_3, z_2, \cdots, z_{2p-3}, z_{2(p-2)})^T \quad \text{for all } \boldsymbol{z} \in \boldsymbol{R}^{2(p-1)}.$$

Equation (3.3) then becomes

$$\boldsymbol{x} = \tilde{\boldsymbol{A}}^{-1}\boldsymbol{d} - \tilde{\boldsymbol{A}}^{-1}\boldsymbol{V}\boldsymbol{P}(\boldsymbol{P} + \boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{V}\boldsymbol{P})^{-1}\boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{d}. \quad (3.4)$$

Note that $\boldsymbol{Z} = (\boldsymbol{P} + \boldsymbol{E}^T\tilde{\boldsymbol{A}}^{-1}\boldsymbol{V}\boldsymbol{P})$ is a tridiagonal matrix of order $2(p-1)$. Let

$$\tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \boldsymbol{d}, \quad (3.5)$$

$$\tilde{\boldsymbol{A}}\boldsymbol{Y} = \boldsymbol{V}\boldsymbol{P}, \quad (3.6)$$

$$\boldsymbol{h} = \boldsymbol{E}^T\tilde{\boldsymbol{x}}, \quad (3.7)$$

$$\boldsymbol{Z} = \boldsymbol{P} + \boldsymbol{E}^T\boldsymbol{Y}, \quad (3.8)$$

$$\boldsymbol{Z}\boldsymbol{y} = \boldsymbol{h}, \quad (3.9)$$

$$\Delta\boldsymbol{x} = \boldsymbol{Y}\boldsymbol{y}. \quad (3.10)$$

From (3.5)–(3.10), (3.4) becomes

$$\boldsymbol{x} = \tilde{\boldsymbol{x}} - \Delta\boldsymbol{x}. \quad (3.11)$$

In (3.5) and (3.6), $\tilde{\boldsymbol{x}}$ and $\boldsymbol{Y}$ are solved by the LU decomposition method. By the structure of $\tilde{\boldsymbol{A}}$ and $\boldsymbol{V}\boldsymbol{P}$, this is equivalent to solve

$$\boldsymbol{A}_i[\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{w}^{(i)}] = [\boldsymbol{d}^{(i)}, a_{im}\boldsymbol{e}_0, c_{(i+1)m-1}\boldsymbol{e}_{m-1}], \quad i = 0, \cdots, p-1. \quad (3.12)$$

Here we have $a_0 = c_{n-1} = 0, \boldsymbol{e}_0, \boldsymbol{e}_{m-1} \in \boldsymbol{R}^m, \tilde{\boldsymbol{x}}^{(i)}$ and $\boldsymbol{d}^{(i)}$ are the ith block of $\tilde{\boldsymbol{x}}$ and $\boldsymbol{d}$, respectively, and $\boldsymbol{v}^{(i)}, \boldsymbol{w}^{(i)}$ are possible nonzero column vectors of the ith row block of $\boldsymbol{Y}$. Equation (3.12) implies that we only need to solve three linear systems of order m with the same LU decomposition for each i $(i = 0, \cdots, p-1)$. In addition, we can skip the first $m-1$ forward substitutions for the third system since the first $m-1$ components of the vector at the right-hand side are all zeros.

Equation (3.7) only picks $2(p-1)$ specified components from the vector $\tilde{\boldsymbol{x}}$. The evaluation of (3.8) uses those possible nonzero entries of specified $2(p-1)$ rows of $\boldsymbol{Y}$ together with $\boldsymbol{P}$ to form matrix $\boldsymbol{Z}$. There is no computation or communication involved in computing $\boldsymbol{h}$ and $\boldsymbol{Z}$.

Solving (3.9) is the major computation involved in the conquer part of our algorithms. Since $\boldsymbol{Y}$ has at most two nonzero entries at each row, the evaluation of (3.10) takes four arithmetic operations per row.

Based on the above observations and together with a careful scaling process, the computational complexity required to solve (1.1) in a sequential processor is stated in the following theorem.

Theorem 3.1: Equation (1.1) can be solved using (3.5)–(3.11) with $17n - 6p - 23$ arithmetic operations without pivoting and $24n - 13p - 34$ arithmetic operations with pivoting.

IV. Parallel Partition LU (PPT) Algorithm

Based on the matrix partitioning technique described previously, in following sections we show how to implement parallel algorithms on multicomputers. The *Parallel Partition LU* (PPT) algorithm is described first in this section.

Using p processors, the PPT algorithm to solve (1.1) consists of the following steps:

Step 1. Allocate $\boldsymbol{A}_i, \boldsymbol{d}^{(i)}$ and elements $a_{im}, c_{(i+1)m-1}$ to the ith node, where $0 \le i \le p-1$.

Step 2. Use the LU decomposition method to solve (3.12). All computations can be executed in parallel and independently on p processors.

Step 3. Send $\tilde{x}_0^{(i)}, \tilde{x}_{m-1}^{(i)}, v_0^{(i)}, v_{m-1}^{(i)}, w_0^{(i)}, w_{m-1}^{(i)}$ $(0 \le i \le n-1)$ to all other nodes from the ith node to form matrix $\boldsymbol{Z}$ and vector $\boldsymbol{h}$ [(3.7) and (3.8)] on each node. Here and throughout the subindex indicates the component of the vector.

Step 4. Use the LU decomposition method to solve $\boldsymbol{Z}\boldsymbol{y} = \boldsymbol{h}$ [(3.9)] on all nodes simultaneously. Note that $\boldsymbol{Z}$ is a $2(p-1)$ dimensional tridiagonal matrix.

Step 5. Compute (3.10) and (3.11). We have

$$\Delta\boldsymbol{x}^{(i)} = \left[\boldsymbol{v}^{(i)}, \boldsymbol{w}^{(i)}\right]\begin{bmatrix} y_{2i-1} \\ y_{2i} \end{bmatrix}, \qquad \boldsymbol{x}^{(i)} = \tilde{\boldsymbol{x}}^{(i)} - \Delta\boldsymbol{x}^{(i)}. \quad (4.1)$$

This step is executed in parallel on p processors.

As mentioned in Section I, the underlying communication mechanism of multicomputers has a great impact on the performance of parallel algorithms. In the *store-and-forward* mechanism, which is used in all first generation hypercube

multicomputers, the message transfer time between two adjacent processors can be expressed as $\alpha + \beta S$, where α is the communication latency, β is the transmission time per byte, and S is the number of bytes in the message [8]. If a message is delivered to h hops away, the message transfer time can be roughly estimated as $h(\alpha + \beta S)$. In second generation multicomputers, advanced communication mechanisms are adopted, such as the *circuit switching* used in iPSC-2 [2] and the *wormhole routing* used in Ametek 2010 [1]. In these new communication paradigms, the message transfer time is almost independent of the distance (number of hops) between processors [2]. In this case, if network contention is not considered, the transfer time of a message with S bytes can always be expressed as $\alpha + \beta S$ regardless of the distance that a message has to traverse.

The PPT algorithm has an all-to-all communication which is also called *total data-exchange* or *all-to-all broadcasting* [15] communication. Fig. 2 shows the communication pattern of the PPT algorithm for the case of $p = 8$.

One of the best ways to handle the all-to-all communication is using the *butterfly* communication fashion as shown in Fig. 3 for the case of $p = 8$.

The butterfly communication pattern can be perfectly embedded in the hypercube topology [19]. Note that in the data transfer, each processor, upon receiving a message, has to add in its own data and forward the enlarged message to the following processors. As a result, the communication time required in total data-exchange can be estimated to be

$$(\log p)\alpha + (p-1)S\beta. \tag{4.2}$$

Equation (4.2) shows that the total data-exchange with butterfly fashion has the same time complexity as using the *spanning binomial tree* technique [15]. By Johnsson and Ho [15], it is optimum within a factor of two. Now we are ready to state the computation and communication complexities of the PPT algorithm.

Theorem 4.1: The PPT algorithm solves (1.1) in $17(n/p) + 16p - 45$ and $24(n/p) + 22p - 69$ parallel arithmetic operations for nonpivoting and pivoting, respectively. With 4-byte per data element, it requires $\alpha \log p + 16(p-1)\beta$ communication times, where p is the number of processors and $n = mp$.

Let T_{LU}, T_{SPT}, and T_{PPT} be the time required to solve (1.1) using the sequential LU decomposition algorithm, the sequential partitioning algorithm [(3.5)–(3.11)], and the PPT algorithm, respectively. Let τ_{comp} represent the unit of a computation operation normalized to the communication time. From those theorems shown previously, we have

$$T_{LU} = (8n-7)\tau_{comp} \qquad \text{without pivoting (4.3)}$$

$$T_{LUP} = (11n-12)\tau_{comp} \qquad \text{with pivoting (4.4)}$$

$$T_{SPT} = (17n-6p-23)\tau_{comp} \qquad \text{without pivoting (4.5)}$$

$$T_{SPT} = (24n-13p-34)\tau_{comp} \qquad \text{with pivoting (4.6)}$$

$$T_{PPT} = (17\frac{n}{p} + 16p - 45)\tau_{comp} + (\log p)\alpha + 16(p-1)\beta \qquad \text{without pivoting (4.7)}$$

$$T_{PPT} = (24\frac{n}{p} + 22p - 69)\tau_{comp} + (\log p)\alpha + 16(p-1)\beta \qquad \text{with pivoting (4.8)}$$

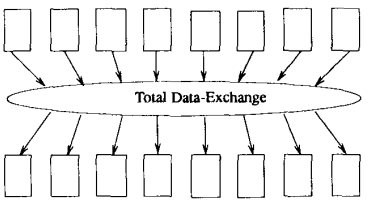


Fig. 2. The communication pattern of the PPT algorithm.

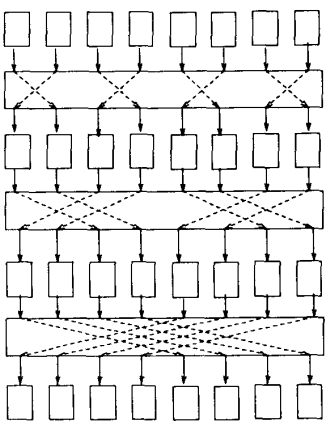


Fig. 3 The butterfly communication pattern.

In Section VII, the theoretical results obtained here will be compared with the measured results obtained from experiments on a 64-node nCUBE-1.

V. The Parallel Partition Hybrid (PPH) Algorithm

The PPT algorithm competes well with any existing parallel algorithms for solving (1.1) when $p \ll n$. However, the efficiency of the PPT algorithm decreases as the number of processors, p, increases. This is due to the fact that the major computation in the conquer part (Step 4 in Section IV) of the PPT algorithm is redundant down on each node. For this reason, we use the PP method (see Section II), the limited processor version of the RCD method, to solve (3.9). In order to apply the PP method, all the superdiagonal elements of the coefficient matrix must be nonzero. The following theorem is needed in applying the PP method to (3.9).

Theorem 5.1: If all the superdiagonal elements of the matrix $\boldsymbol{A}$ are nonzero, the tridiagonal matrix $\boldsymbol{Z} = \boldsymbol{P} + \boldsymbol{E}^T\boldsymbol{A}^{-1}\boldsymbol{V}\boldsymbol{P}$ has nonzero superdiagonal elements.

Proof: The superdiagonal elements of $\boldsymbol{Z}$ are either equal to 1 or the first components of the solutions

$$\boldsymbol{A}_i\boldsymbol{w}^{(i)} = c_{(i+1)m-1}\boldsymbol{e}_{m-1}, \qquad i = 1, \cdots, p-2 \tag{5.1}$$

where $\boldsymbol{w}^{(i)}, \boldsymbol{e}_{m-1} \in \boldsymbol{R}^m$. Suppose $w_0^{(i)} = 0$, then $w_j^{(i)} = 0$ for $j = 1, \cdots, m-1$, since the superdiagonal elements of $\boldsymbol{A}_i$ are nonzero. Therefore, we have $\boldsymbol{A}_i\boldsymbol{w}^{(i)} = 0$, which is a contradiction to (5.1). □

The new algorithm, namely the *parallel partition hybrid* (PPH) algorithm, is similar to the PPT algorithm except the following changes. After Step 2 of the PPT algorithm, the $2i-1$th and $2i$th, for $i=0,\cdots,p-1$, rows of the tridiagonal matrix $\boldsymbol{Z}$ are stored in the ith node (here the -1th and $2(p-1)$th rows of $\boldsymbol{Z}$ are assumed to be zero). Step 3 of the PPT algorithm, which performs total data-exchange, is eliminated. Step 4 of the PPT algorithm is then replaced by the PP method as described in Section II.

The PP method has two communication patterns in Step 4. The first communication pattern shown in Fig. 1 computes all partial matrix products. The second communication pattern is a *broadcast* which broadcasts the computed x_0 to all other nodes. Broadcast can be achieved by using the *spanning binomial tree* technique [15]. Considering second generation multicomputers, based on the communication model discussed in Section IV, the communication time required to obtain all partial products of $\boldsymbol{B}_i$'s (Fig. 1) is $(\log p+1)\alpha+24(\log p+1)\beta$. Here we have $S=24$ because each message transfer has 6 data elements and we assume that each floating point number has 4 bytes. Since the broadcast communication takes $\log p$ steps and the message has one data element, it takes $(\log p)\alpha+(4\log p)\beta$ time. The communication pattern of the PPH algorithm is shown in Fig. 4 for the case of $p=8$. Counting the arithmetic operations and communication time, we have the following theorem.

Theorem 5.2: With $n=pm$, the PPH algorithm solves (1.1) in $17(n/p)+20\log p+17$ and $24(n/p)+20\log p+4$ parallel arithmetic operations for nonpivoting and pivoting, respectively. It requires $(2\log p+1)\alpha+(28\log p+24)\beta$ communication steps.

Let T_{SPH} be the time required to execute the PPH algorithm in a sequential processor and T_{PPH} be the time required to execute the PPH algorithm on p processors, we have

$$T_{SPH}=(17n+8p-41)\tau_{comp} \qquad \text{without pivoting}$$
$$T_{SPH}=(24n-5p-41)\tau_{comp} \qquad \text{with pivoting}$$
$$T_{PPH}=\left(17\frac{n}{p}+20\log p+17\right)\tau_{comp}+(2\log p+1)\alpha+(28\log p+24)\beta \qquad \text{without pivoting}$$
$$T_{PPH}=\left(24\frac{n}{p}+20\log p+4\right)\tau_{comp}+(2\log p+1)\alpha+(28\log p+24)\beta. \qquad \text{with pivoting}$$

It is interesting to notice that the PPH algorithm can reach a speedup of 2 over the PP algorithm for $1<p\ll n$. When $p=n$, no matrix partitioning is needed and the PPH algorithm is virtually the RCD algorithm. When $p=1$, there is no conquer part and as the LU decomposition method is used in the dividing part, the algorithm becomes the LU decomposition algorithm.

As we mentioned earlier, the PP method is unstable if superdiagonal elements are small. This may be true for the PPH method. However, in this method, the PP algorithm is used only for solving a smaller linear system $\boldsymbol{Zy}=\boldsymbol{h}$, and half of superdiagonal elements of the matrix $\boldsymbol{Z}$ equal to 1 as we proved in Theorem 5.1. This indicates that the PPH method might be more stable than the PP method, but still inherits the unstability of the PP method.

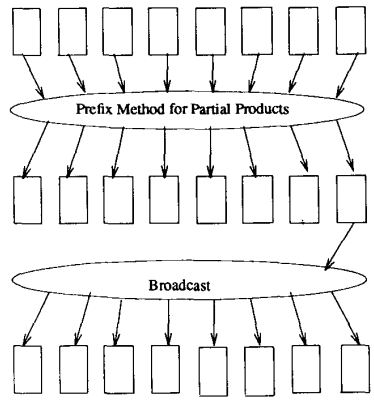


Fig. 4. The communication pattern of the PPH algorithm.

VI. The Parallel Diagonal Dominant (PDD) Algorithm

Theorem 5.1 guarantees that the superdiagonal elements of matrix $\boldsymbol{Z}$ are nonzero if $\boldsymbol{A}$ has the same property. In practice, especially for a diagonal dominant tridiagonal matrix $\boldsymbol{A}$, the magnitude of the last component of $\boldsymbol{v}^{(i)}$ and the first component of $\boldsymbol{w}^{(i)}$, which turn out to be the off-diagonal elements of the matrix $\boldsymbol{Z}$, may be smaller than the machine accuracy when $p\ll n$. In this section we show that the magnitudes of $v_{m-1}^{(i)}$ and $w_0^{(i)}$ converge to zero when $m=n/p\to\infty$. An estimated rate of convergence will also be provided. Thus, when the tridiagonal system satisfies a strong diagonal dominance condition, a fast and highly efficient parallel algorithm gives an approximate solution which equals the exact solution within machine accuracy.

A. Theoretical Results for Diagonal Dominant Systems

For notational convenience, we write a tridiagonal matrix $\boldsymbol{A}$ as a triple of vectors as

$$\boldsymbol{A}=\begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & \cdot & & \\ & \cdot & \cdot & \cdot & \cdot & \\ & & \cdot & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix}=(a_i,b_i,c_i) \quad (i=0,\cdots,n-1)$$

where $a_0=c_{n-1}=0$. The matrix $\boldsymbol{Z}$ and the jth block of the matrix $\boldsymbol{A}$ can be similarly written as

$$\boldsymbol{Z}=(a_i^Z,b_i^Z,c_i^Z) \qquad (i=0,\cdots,2(p-1)-1),$$
$$\boldsymbol{A}_j=(a_i^{(j)},b_i^{(j)},c_i^{(j)}) \qquad (i=0,\cdots,m-1).$$

The super index (j) will be dropped when the context is clear. The matrix $\boldsymbol{Z}$ can be represented as

$$\boldsymbol{Z} = \begin{bmatrix} w_{m-1}^{(0)} & 1 & & & & & & \\ 1 & v_0^{(1)} & w_0^{(1)} & & & & & \\ & v_{m-1}^{(1)} & w_{m-1}^{(1)} & 1 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & & & \\ & & & & \cdot & \cdot & & \\ & & & & & w_{m-1}^{(p-2)} & 1 & \\ & & & & & 1 & v_0^{(p-1)} \end{bmatrix}$$

where $\boldsymbol{v}^{(i)}, \boldsymbol{w}^{(i)}$ for $i = 0, \cdots, p-1$ are the solutions of

$$\boldsymbol{A}_i \boldsymbol{v}^{(i)} = a_0^{(i)} \boldsymbol{e}_0,$$
$$\boldsymbol{A}_i \boldsymbol{w}^{(i)} = c_{m-1}^{(i)} \boldsymbol{e}_{m-1},$$

and 1's come from the permutation matrix $\boldsymbol{P}$.

Definition: A tridiagonal matrix $\boldsymbol{A}$ is *evenly diagonal dominant* if

$$|a_i| \le |b_i/2|, \ |c_i| \le |b_i/2| \text{ and } a_{i+1} \cdot c_i > 0 \qquad i = 0, \cdots, n-2. \quad (6.1)$$

Note that the evenly diagonal dominant is little stronger than diagonal dominant. The submatrix $\boldsymbol{A}_j$ has a LDU factorization

$$\boldsymbol{A}_j = \text{LDU} = (l_i, 1, 0)(0, \delta_i, 0)(0, 1, u_{i+1}) \qquad (i = 0, \cdots, m-1).$$

It can be easily verified that

$$l_i = a_i/\delta_{i-1}, \ u_i = c_{i-1}/\delta_{i-1}, \qquad i = 1, \cdots, m-1. \quad (6.2)$$

Since $\boldsymbol{A}_j^{-1} = \boldsymbol{U}^{-1}\boldsymbol{D}^{-1}\boldsymbol{L}^{-1}$, where

$$\boldsymbol{U}^{-1} = \begin{bmatrix} 1, & -u_1, & u_1u_2, & \cdots, & (-1)^{m-1}u_1 \cdots u_{m-1} \\ & 1, & & & \\ & & 1, & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}.$$

$$\boldsymbol{D}^{-1} = (0, 1/\delta_i, 0)$$

$$\boldsymbol{L}^{-1} = \begin{bmatrix} 1, & & & & & \\ -l_1, & 1, & & & & \\ l_1l_2 & & 1, & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ \cdot & & & & & \cdot \\ (-1)^{m-1}l_1 \cdots l_{m-1}, & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix}.$$

The last component of $\boldsymbol{v}^{(j)} = \boldsymbol{A}_j^{-1}(a_0^{(j)}\boldsymbol{e}_0)$ and the first component of $\boldsymbol{w}^{(j)} = \boldsymbol{A}_j^{-1}(c_{m-1}^{(j)}\boldsymbol{e}_{m-1})$ are

$$v_{m-1}^{(j)} = \frac{(-1)^{m-1}l_1 \cdots l_{m-1}a_0^{(j)}}{\delta_{m-1}} = \frac{(-1)^{m-1}\prod_{i=0}^{m-1} a_i}{\prod_{i=0}^{m-1} \delta_i}.$$

$$w_0^{(j)} = \frac{(-1)^{m-1}u_1 \cdots u_{m-1}c_{m-1}^{(j)}}{\delta_{m-1}} = \frac{(-1)^{m-1}\prod_{i=0}^{m-1} c_i}{\prod_{i=0}^{m-1} \delta_i}.$$

Let $\det(\boldsymbol{A}_j)$ denote the determinant of $\boldsymbol{A}_j$. We have

$$|a_{2j}^Z| = |v_{m-1}^{(j)}| = |\prod_{i=0}^{m-1} a_i^{(j)}|/\det(\boldsymbol{A}_j),$$
$$|c_{2j-1}^Z| = |w_0^{(j)}| = |\prod_{i=0}^{m-1} c_i^{(j)}|/\det(\boldsymbol{A}_j). \quad (6.3)$$

for $j = 1, \cdots, p-2$. Scaling matrix $\boldsymbol{A}$, we have

$$\boldsymbol{A} = \boldsymbol{K}\boldsymbol{A}' = (0, k_i, 0)(a_i', b_i', c_i').$$

where

$$k_i = \text{sign}(b_i) \cdot \text{Max}(|a_i|, |c_i|).$$
$$|a_i'| \le 1, \ |c_i'| \le 1, \ b_i' \ge 2.$$

Equation (6.3) then is written as

$$|a_{2j}^Z| = |\prod_{i=0}^{m-1} a_i'^{(j)}|/\det(\boldsymbol{A}_j').$$
$$|c_{2j-1}^Z| = |\prod_{i=0}^{m-1} c_i'^{(j)}|/\det(\boldsymbol{A}_j').$$

Thus, under the condition (6.1), with no loss of generality, we can assume that $\boldsymbol{A}$ satisfies

$$|a_i'| \le 1, \ |c_i'| \le 1, \ b_i \ge 2, \qquad i = 0, \cdots, n-1,$$
$$a_{i+1} \cdot c_i > 0, \qquad i = 0, \cdots, n-2. \quad (6.4)$$

The next theorem shows that $a_{2j}^Z, c_{2j-1}^Z (j = 1, \cdots, p-2)$ converge to zero as the dimension of submatrices $m = n/p \to \infty$ if $\boldsymbol{A}$ satisfies the condition (6.4), which is equivalent to that matrix $\boldsymbol{A}$ satisfies condition (6.1). The convergent rate is at least exponential if $b_i > 2$ for $i = 0, \cdots, n-1$.

Lemma 6.1: Suppose a real tridiagonal matrix $\boldsymbol{A} = (a_i, b_i, c_i)$ of order n satisfies condition (6.4). Let $\epsilon = \min_{0 \le i \le n-1}(b_i - 2)$. Then we have

$$\det(\boldsymbol{A}) \ge \prod_{s=1}^{n}(\lambda_s + \epsilon) \qquad \text{with } \lambda_s = 2 + 2\cos\frac{s\pi}{n+1}. \quad (6.5)$$

The proof of Lemma 6.1 can be found in the Appendix.

Theorem 6.2: If the matrix $\boldsymbol{A}$ satisfies condition (6.1), then for matrix $\boldsymbol{Z} = (a_i^Z, b_i^Z, c_i^Z)$, we have

$$|a_{2j}^Z| \le 1/\prod_{s=1}^{m}(\lambda_s + \varepsilon).$$
$$|c_{2j-1}^Z| \le 1/\prod_{s=1}^{m}(\lambda_s + \varepsilon) \quad (6.6)$$

where $j = 1, \cdots, p-2, m = n/p$, and $\lambda_s = 2 + 2\cos\frac{\{s\pi\}}{m+1}$.

The proof of Theorem 6.2 is a direct result of (6.3) and Lemma 6.1. It is not difficult to verify that

$$\frac{1}{\prod_{s=1}^{m}(\lambda_s+\epsilon)} \le \frac{1}{(m+1)(1+\epsilon/4)^m}. \tag{6.7}$$

Thus, the convergent rate of those elements is at least exponential if $\epsilon > 0$. Note that the bound in (6.7) is exact for $\epsilon = 0$ and would be more accurate for small ϵ. A sharper bound for larger ϵ is given by [3].

The fast convergence of these elements has a simple intuitive explanation. The solution of a tridiagonal system requires that every equation influences every other equation. For diagonal dominant systems, the influence diminishes with the distance between equations.

As $v_{m-1}^{(i)}, w_0^{(i)} (0 \le i \le 2(p-1))$ become zero within machine accuracy, the matrix $\boldsymbol{Z}$ becomes a diagonal block matrix with each block of size 2×2. In order to carry out the PDD algorithm, we have to guarantee that the 2×2 diagonal blocks of matrix $\boldsymbol{Z}$

$$\begin{bmatrix} w_{m-1}^{(i)} & 1 \\ 1 & v_0^{(i+1)} \end{bmatrix}, \qquad i = 0, \cdots, p-2.$$

are nonsingular. This is proved by the following theorem and its proof is stated in the Appendix.

Theorem 6.3: If a real tridiagonal matrix $\boldsymbol{A}$ of order n satisfies condition (6.1), then

$$|w_{m-1}^{(i)} \cdot v_0^{(i+1)}| < 1, \qquad i = 0, \cdots, p-2. \tag{6.8}$$

B. The Parallel Diagonal Dominant (PDD) Algorithm

Theorem 6.2 implies that the matrix $\boldsymbol{Z}$ is a diagonal block matrix with block size 2×2 within machine accuracy if m is large enough. Then the procedure for solving (3.9), the conquer part of our algorithm, becomes very simple. If (1.1) satisfies condition (6.1), the parallel diagonal dominant (PDD) algorithm has the following steps modified from the PPT algorithm.

3) Send $\tilde{x}_0^{(i)}$, $v_0^{(i)}$ from the ith node to the $(i-1)$th node for $i = 1, \cdots, p-1$.
4) Solve

$$\begin{bmatrix} w_{m-1}^{(i)} & 1 \\ 1 & v_0^{(i+1)} \end{bmatrix} \begin{bmatrix} y_{2i} \\ y_{2i+1} \end{bmatrix} = \begin{bmatrix} \tilde{x}_{m-1}^{(i)} \\ \tilde{x}_0^{(i+1)} \end{bmatrix}$$

in parallel on all ith nodes for $(0 \le i \le p-2)$. Then send y_{2i+1} from the ith node to the $(i+1)$th node for $i = 0, \cdots, p-2$.

Fig. 5 depicts the communication pattern of the PDD algorithm. In Step 3, all nodes transfer a message to their left neighbor. In Step 4, all nodes transfer a message to their right neighbor. Since it is unnecessary for node 0 to communicate with node $p-1$, the algorithm only needs a one dimensional array connection, which can be perfectly embedded in first generation hypercube multicomputers. The optimal and simple communication property makes the PDD algorithm an ideal algorithm for multicomputers. The computation and communication complexities of the PDD algorithm is stated in the following theorem.

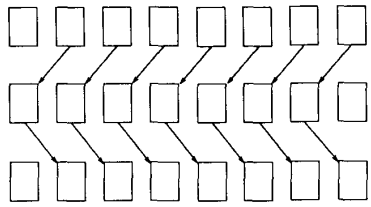


Fig. 5. The communication pattern of the PDD algorithm.

Theorem 6.4: If $n/p \gg 1$ and matrix $\boldsymbol{A}$ satisfies condition (6.1), the PDD algorithm solves (1.1) in $17(n/p) - 14$ parallel arithmetic operations and $2\alpha + 12\beta$ communications on multicomputers with p processors. When the left-hand side of (6.7) is within machine accuracy, the approximate solution equals the exact solution within machine accuracy.

Let T_{SDD} and T_{PDD} be the time required to execute the PDD algorithm in a sequential processor and p parallel processors, respectively. We have

$$T_{SDD} = (17n - 14p - 8)\tau_{comp},$$
$$T_{PDD} = \left(17\frac{n}{p} - 14\right)\tau_{comp} + 2\alpha + 12\beta.$$

Many tridiagonal systems arising in science and engineering problems satisfy condition (6.1). However, condition (6.1) is sufficient but not necessary. The algorithm can be used for any positive definite or diagonal dominant matrix whose resulting $|v_{m-1}^{(j)}|, |w_0^{(j)}|$ [see (6.3)] become underflow as $n \gg p$. One of the sufficient conditions is that the determinants of scaled matrices $\boldsymbol{A}_j'$ $(j = 0, \cdots, p-1)$ become overflow as $n \gg p$. For example, if $\boldsymbol{A} = (1, i, 1)$, $\det(\boldsymbol{A}_j')$ are approximately larger than $m!$ since its ith eigenvalue $\lambda_i \approx i$ except few smallest and largest ones. The magnitude of those entries of $\boldsymbol{Z}$ can be as small as 10^{-64} for $n/p \ge 50$.

VII. Experimental Results

The arithmetic operation counts and communication steps of each individual algorithm are summarized in Table I.

Fig. 6 shows the estimated and measured speedup of the PP, PPT, PPH, and PDD algorithms with respect to the subroutine SGTSL of LINPACK [4]. These algorithms are implemented on a 64-node nCUBE 1 multicomputer. nCUBE 1 is a first generation multicomputer adopting the store-and-forward communication mechanism. In our nCUBE 1 machine, the following system parameters are measured: $\alpha = 5.0$, $\beta = 0.013$, and $\tau_{comp} = 0.08$ (without normalization). The dimension of matrix $\boldsymbol{A}$ is chosen as $n = 6400$. This value is limited by the memory constraint of individual processors. As p increases, the PPH algorithm will outperform the PPT algorithm because the dimension of the matrix $\boldsymbol{Z}$ [(3.9)] increases as p increases, which favors the parallel approach used in the PPH algorithm. The performance of the PPH algorithm seems to be underestimated compared with the

TABLE I
COMPUTATION AND COMMUNICATION COUNTS OF TRIDIAGONAL SOLVERS

Algorithm	Computation	Communication
PPT (nonpivoting)	$17\frac{n}{p} + 16p - 45$	$(\log p)\alpha + 16(p-1)\beta$
PPT (pivoting)	$24\frac{n}{p} + 22p - 69$	$(\log p)\alpha + 16(p-1)\beta$
PPH (nonpivoting)	$17\frac{n}{p} + 20\log p + 17$	$(2\log p + 1)\alpha + (28\log p + 24)\beta$
PPH (pivoting)	$24\frac{n}{p} + 20\log p + 4$	$(2\log p + 1)\alpha + (28\log p + 24)\beta$
PDD	$17\frac{n}{p} - 14$	$2\alpha + 12\beta$

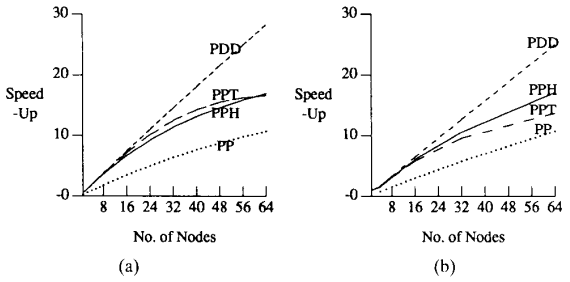


Fig. 6. The speedup over the LU decomposition method, where $n = 6400$. (a) Estimated speedup. (b) Measured speedup.

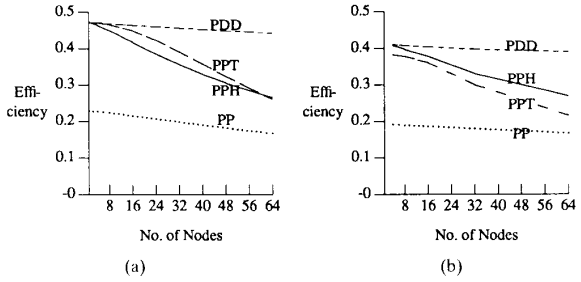


Fig. 7. Efficiency over the LU decomposition method, $n = 6400$. (a) Estimated efficiency. (b) Measured efficiency.

measured results. This is mainly caused by assuming a dilation of 2 for all communications occurred in Fig. 1. However, in actual implementation, some communications have dilation 1.

As n goes to infinity, the asymptotic speedups, compared with the LU decomposition method, of all our methods are $0.471p$. And the asymptotic speedup for the PP method is $0.229p$. Dividing the speedup by the number of processors, p, Fig. 7 shows the *efficiencies* of our methods. For the case of $n = 6400$, only the PDD algorithm has an efficiency closing to the asymptotic efficiency. For all methods, the efficiency decreases as the number of processors p increases. This is mainly caused by the increasing ratio of the communication and the computation complexities.

There are two commonly accepted measures for the speedup and the efficiency of parallel algorithms [18]. One focuses on how much faster a problem can be solved by p processors. Thus, it compares the best serial algorithm with the parallel algorithm under consideration. It is defined as

$$S_p = \frac{\text{execution time using the fastest sequential algorithm on one processor}}{\text{execution time using the parallel algorithm on } p \text{ processors}}.$$

Both Figs. 6 and 7 are based on the above measure, and the best sequential algorithm chosen is LINPACK subroutine, SGTSL. Another measure interests in the parallel inherence of the algorithm and is defined as

$$S'_p = \frac{\text{execution time using one processor}}{\text{execution time using } p \text{ processors}}.$$

We call the latter as the *relative speedup*. The relative speedups

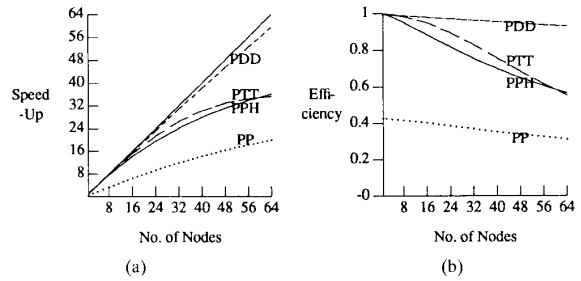


Fig. 8. The performance of relative speedup and relative efficiency, where $n = 6400$. (a) Estimated relative speedup. (b) Measured relative speedup.

and efficiencies of our methods are shown in Fig. 8. As it shows, the relative speedup of the PDD algorithm is very close to the ideal speedup. This is due to the high efficiency of the PDD algorithm as we discussed in Section VI-A.

One of the advantages of our methods is its diversity. In the dividing part, the pivoting may be used when it is necessary. There has been a tacit assumption for most available parallel algorithms that no pivoting is required. Thus, our algorithms may be more stable than others in the cases pivoting is required. In the conquer part, basically the major computation required is to solve (3.9). The methods used in this part are irrelevant to the methods used in the dividing part. Other strategies may also be used to solve (3.9).

The nonsingularity of matrix $\tilde{\boldsymbol{A}}$ is required by the PPT and PPH algorithms. Unless the matrix $\boldsymbol{A}$ is positive definite or diagonal dominant, there is no general rule to guarantee

that $\tilde{A}$ (see Section III) is nonsingular. However, for certain class of matrices A at hand, very often we can find a criterion to avoid the singularity of $\tilde{A}$ and make the algorithms work.

Michielse and Vorst [17] studied the communication pattern of Wang's method [23] for local memory machines. Based on their result, the communication count of Wang's method is $2(p-1)(\alpha+10\beta)+(\alpha+16\beta)$. The recent results given by Ho and Johnsson [11] are more encouraging. The algorithms of Ho and Johnsson are based on substructuring, which is a variant of Wang's method. The computation and communication counts of their algorithms are very close to respective counts of the PPT method. The difference of their algorithms versus ours is that, in the partitioning/substructuring phase, their algorithms require neighboring communications, while ours do not need any communication. The major gain of these extra communications is that their reduced system is of order $p-1$, while ours is of $2(p-1)$. The numbers of communication steps in the conquer phase for all these algorithms are essentially the same.

It is interesting to know that, based on the matrix modification formula (3.3), we are end up a matrix partition which is very similar to that of Lawrie and Sameh [16]. The main difference between our algorithms and theirs is the strategies adopted in solving the reduced system (3.9). Both theoretical and numerical results have shown that strategies of solving the reduced system will affect the overall performance on a multicomputer considerably. It is for this reason, we proposed three algorithms, the PPT, PPH, and PDD, in which, the actual differences are the strategies in solving this reduced system. Table I shows that, the communication complexities of these three algorithms in solving (3.9) are $O(\log p), O(\log p)$, and $O(1)$, respectively, while it is $O(p)$ for the Lawrie–Sameh's algorithm. When p is large (> 16), $O(p)$ communication delay can be significant comparing with $O(\log p)$ and $O(1)$, especially on the nCUBE multicomputer we used, which has a relatively high communication startup latency. This fact is confirmed as shown in Figs. 6–8. Lawrie–Sameh's algorithm was designed for shared-memory multiprocessors, while ours are designed mainly for multiprocessors with distributed memory architecture, although they are well-suited for a variety of architectures.

In solving a multiple tridiagonal system, the procedures (3.5)–(3.11) can be divided into two stages: 1) factoring the coefficient matrices A_i and Z, and formulating $Y = \tilde{A}^{-1}VP$ [see (3.6)]; 2) solving the systems (3.5) and (3.9), then forming the final solution. The number of computation and communication steps in the first stage remains unchanged. In the second stage, both (3.5) and (3.9) become solving a multiple linear system. For a multiple system the vectors d_i of the rightside, $(i = 1, \cdots, k)$, are either available at the beginning of the computation or only available during the solving process. In the formal case, the communication increases slightly, since only the message length is increased due to the multiple rightside vectors. In the latter case, communication has to be repeated for each system, which makes the PPT method more preferable than the PPH method.

VIII. Conclusions

A matrix partitioning method for tridiagonal linear systems is introduced. Based on this partitioning method, three parallel algorithms for solving a tridiagonal system on multicomputers are proposed. All these three algorithms are based on the divide and conquer principle and are designed for the situation where the number of processors p is much smaller than the dimension of the system n. The PPT and PPH algorithms are fast and able to incorporate pivoting. The PPT algorithm is simple and more stable. It is a good choice when the number of processors is small. The PPH algorithm is a hybrid of the LU decomposition method and the recursive doubling method. When $p = n$, the PPH algorithm is virtually the recursive doubling algorithm. When $1 < p \ll n$, the PPH algorithm is faster than the limited processor version of the recursive doubling algorithm and is more efficient than the PPT algorithm. The PDD algorithm is the most efficient algorithm when the system is evenly diagonal dominant. It requires a simple and minimum communication and is an ideal algorithm on multicomputers when the condition is permitted.

The methods introduced in this paper have advantages in the speedup, efficiency, diversity, and the uniqueness of pivoting. All the three algorithms have been implemented on a 64-node nCUBE 1 multicomputer and compared with the sequential LU decomposition algorithm. Our theoretical analysis matches closely with our experimental results. All the three algorithms introduced in this paper can be extended to the banded linear systems and block tridiagonal linear systems. Although the methods proposed here are dedicated to multicomputers, they can be easily modified to run on other multiprocessor architectures.

Appendix

Proof of Lemma 6.1: First let $A^{\epsilon} = (a_i, b_i^{\epsilon}, c_i)$ with $b_i^{\epsilon} = b_i$ for $i \neq j$ and $b_j^{\epsilon} = b_j + \epsilon$, where $\epsilon > 0$. We have

$$\det(A^{\epsilon}) = \det(A) + \det\begin{bmatrix} b_0 & \cdot & & & & & & \\ \cdot & \cdot & \cdot & & & & & \\ & \cdot & \cdot & \cdot & & & & \\ & & b_{j-1} & 0 & & & & \\ & & 0 & \epsilon & 0 & & & \\ & & & 0 & b_{j+1} & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \cdot & \cdot & \cdot \\ & & & & & & \cdot & b_{n-1} \end{bmatrix} > \det(A).$$

Repeating this process we have

$$\det(a_i, b_i + \epsilon_i, c_i) \geq \det(a_i, b_i, c_i) \qquad \text{with } \epsilon_i \geq 0. \quad \text{(A1)}$$

Next let $A^{\epsilon} = (a_i, b_i, c_i^{\epsilon})$ with $c_i^{\epsilon} = c_i$ for $i \neq j$, $c_j^{\epsilon} = c_j + \epsilon^{+}$

such that $|c_j^\epsilon| \le |c_j|$ and $\epsilon^+ \neq 0$. Then we have

$$\begin{aligned}
\det(\boldsymbol{A}^\epsilon) &= \det(\boldsymbol{A}) \\
&\quad + \det\begin{bmatrix} b_0 & \cdot & & & & & \\ \cdot & \cdot & \cdot & & & & \\ & \cdot & \cdot & \cdot & & & \\ & \cdot & b_{j-1} & c_{j-1} & & & \\ & & 0 & 0 & \epsilon^+ & & \\ & & & a_{j+1} & b_{j+1} & \cdot & \\ & & & & a_{j+2} & \cdot & \cdot \\ & & & & & \cdot & \cdot \\ & & & & & & b_{n-1} \end{bmatrix} \\
&= \det(\boldsymbol{A}) - \epsilon^+ a_{j+1} \\
&\quad \cdot \det\begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & & a_{j-2} & b_{j-2} & c_{j-2} \\ & & & & a_{j-1} & b_{j-1} \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} b_{j+2} & c_{j+2} & & & & \\ a_{j+3} & b_{j+3} & c_{j+3} & & & \\ & \cdot & \cdot & \cdot & & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix} \\
&> \det(\boldsymbol{A})
\end{aligned}$$

since $\operatorname{sign}(\epsilon^+ a_{j+1}) = \operatorname{sign}(\epsilon^+ c_j) < 0$. It follows directly that

$$\det(a_i, b_i, c_i + \epsilon_i^+) \ge \det(a_i, b_i, c_i) \qquad \text{with } \epsilon_i \ge 0. \quad \text{(A2)}$$

with

$$|c_i + \epsilon_i^+| \le |c_i|.$$

Similarly we can get

$$\det(a_i + \epsilon_i^-, b_i, c_i) \ge \det(a_i, b_i, c_i) \qquad \text{(A3)}$$

with

$$|a_i + \epsilon_i^-| \le |a_i|.$$

For

$$\epsilon_i = b_i - 2. \qquad \epsilon = \min_{0 \le i \le n-1} \epsilon_i,$$
$$\epsilon_i^- = a_i + 1. \qquad \epsilon_i^+ = c_i + 1$$

using (A1)–(A3) and [21] we get

$$\begin{aligned}
\det(a_i, b_i, c_i) &= \det(-1, \epsilon_i^-, 2 + \epsilon_i, -1 + \epsilon_i^+) \\
&\ge \det(-1, 2 + \epsilon, -1) \\
&= \prod_{s=1}^{n} (\lambda_s + \epsilon).
\end{aligned}$$

□

Proof of Theorem 6.3: It is sufficient to prove (6.8) if matrix $\boldsymbol{A}$ satisfies condition (6.4). In order to prove Theorem 6.3, we need the following lemma:

Lemma: If matrix $\boldsymbol{A}$ satisfies condition (6.4), then the inequality

$$\delta_i \ge \frac{i+2}{i+1} \qquad \text{(A4)}$$

holds. Here $\delta_i (0 \le i \le m-1)$ are the diagonal elements of matrix $\boldsymbol{D}_j$ in the LDU decomposition of $\boldsymbol{A}_j (j = 0, \cdots, p-1)$.

Proof: Since $\delta_i = b_i - a_i c_{i+1}/\delta_{i-1}$ $(i = 1, \cdots, m-1)$ with $\delta_0 = b_0$, then

$$\delta_i \ge 2 + \epsilon_i - 1/\delta_{i-1} \qquad \text{with } \epsilon_i \ge 0.$$

Using mathematical induction, (A4) can be easily verified. □

Using LDU decomposition of $\boldsymbol{A}_j$, it is found that

$$\begin{aligned}
&|w_{m-1}^{(i)} \cdot v_0^{(i+1)}| = \\
&|\frac{c_{m-1}^{(i)}}{\delta_{m-1}^{(i)}} \left(\frac{1}{\delta_0^{(i+1)}} + \frac{u_1^{(i+1)} l_1^{(i+1)}}{\delta_1^{(i+1)}} + \right. \\
&\qquad \left. \cdots + \frac{u_1^{(i+1)} \cdots u_{m-1}^{(i+1)} l_1^{(i+1)} \cdots l_{m-1}^{(i+1)}}{\delta_{m-1}^{(i+1)}} \right) a_0^{(i+1)}| \\
&\le |\frac{1}{\delta_{m-1}^{(i)}} \left(\frac{1}{\delta_0^{(i+1)}} + \frac{c_0^{(i+1)} a^{(i+1)}}{(\delta_0^{(i+1)})^2 \delta_1^{(i+1)}} + \right. \\
&\qquad \left. \cdots + \frac{c_0^{(i+1)} \cdots c_{m-2}^{(i+1)} a_1^{(i+1)} \cdots a_{m-1}^{(i+1)}}{(\delta_0^{(i+1)} \cdots \delta_{m-2}^{(i+1)})^2 \delta_{m-1}^{(i+1)}} \right)| \\
&\le \frac{1}{\delta_{m-1}^{(i)}} \left(\frac{1}{\delta_0^{(i+1)}} + \frac{1}{(\delta_0^{(i+1)})^2 \delta_1^{(i+1)}} + \right. \\
&\qquad \left. \cdots + \frac{1}{(\delta_0^{(i+1)} \cdots \delta_{m-2}^{(i+1)})^2 \delta_{m-1}^{(i+1)}} \right) \\
&\le \frac{m}{m+1} \left(\frac{1}{2} + \frac{1 \cdot 2}{2^2 \cdot 3} + \cdots + \frac{1 \cdot m}{m^2 \cdot (m+1)} \right) \\
&= \frac{m}{m+1} \left(\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \cdots + \frac{1}{m \cdot (m+1)} \right) \\
&= \frac{m}{m+1} \cdot \frac{m}{m+1} < 1 \qquad \text{for any } m > 0.
\end{aligned}$$

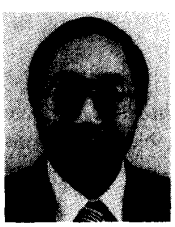
ACKNOWLEDGMENT

The authors are grateful to Prof. S. Eisenstat of Yale University and the referees for their valuable suggestions and comments that helped improve the technical quality and presentation of the paper.

REFERENCES

[1] W. C. Athas, and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Comput. Mag.*, pp. 9–25, Aug. 1988.
[2] L. Bomans and D. Roose, "Benchmarking the iPSC/2 hypercube multiprocessor," *Concurrency: Practice and Experience*, pp. 3–18, Sept. 1989.
[3] S. Demko, W. F. Moss, and P. W. Smith, "Decay rates for inverses of band matrices," *Math. Computat.*, vol. 43, no. 168, pp. 491–499, Oct. 1984.
[4] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack Users' Guide*, SIAM, Philadelphia, PA, 1979.
[5] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. Oxford, England: Clarendon, 1986.
[6] P. Dubois and G. Rodrigue, "An analysis of the recursive doubling algorithm," in *High Speed Computer and Algorithm Organization*, Kuck *et al.*, Eds. New York: Academic, 1977.
[7] O. Egecioglu, C. K. Koc, and Laub, A. J., "A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors," *J. Comput. Appl. Math.*, vol. 27, 1989.
[8] M. R. Grunwald and D. A. Reed, "Benchmarking hypercube hardware and software," Tech. Rep., UIUCDCS-R-86-1303, Dep. Comput. Sci., Univ. of Illinois at Urbana–Champaign, 1986.
[9] D. Heller, "A survey of parallel algorithms in numerical algebra," *SIAM Rev.*, vol. 20, pp. 740–777, Oct. 1978.

[10] R. W. Hockney, "A fast direct solution of Poisson's equation using Fourier analysis," *J. ACM*, vol. 12, pp. 95–113, 1965.
[11] C. T. Ho and S. L. Johnsson, "Optimizing tridiagonal solvers for alternating direction methods on Boolean cube multiprocessors," *SIAM J. Sci. Statist. Comput.*, vol. 11, no. 3, pp. 563–592, May 1990.
[12] K. Hwang, "Advanced parallel processing with supercomputer architectures," *Proc. IEEE*, pp. 33–47, Oct. 1987.
[13] S. L. Johnsson, "Solving tridiagonal systems on ensemble architectures," *SIAM J. Sci. Statist. Comput.*, vol. 8, no. 3, pp. 354–392, May 1987.
[14] ———., "Communication efficient basic linear computations on hypercube multiprocessors," *J. Parallel Distributed Comput.*, no. 4, pp. 133–172, 1987.
[15] S. L. Johnsson and C. T. Ho, "Spanning graphs for optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, vol. 38, Sept. 1989.
[16] D. H. Lawrie and A. H. Sameh, "The computation and communication complexity of a parallel banded system solver," *ACM Trans. Math. Software*, vol. 10, no. 2, pp. 185–195, June 1984.
[17] P. H. Michielse and H. A. Vorst, "Data transport in Wang's partition method," in *Parallel Computing, 7*. New York: North-Holland, 1988, pp. 87–95.
[18] J. M. Ortega and R. G. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Rev.*, pp. 149–240, June 1985.
[19] Y. Saad and M. H. Schultz, "Data communication in hypercube," Res. Rep., YALEU/DCS/RR-428, Oct. 1985.
[20] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix," *Ann. Math. Statist.*, vol. 20, 621, 1949.
[21] G. D. Smith, *Numerical Solution of Partial Differential Equations*. Oxford, England: Oxford University Press, 1985.
[22] H. S. Stone, "An efficient parallel algorithm for the solution of a tridiagonal linear system of equations," *J. ACM*, vol. 20, no. 1, pp. 27–38, Jan. 1973.
[23] H. H. Wang, "A parallel method for tridiagonal equations," *ACM Trans. Math. Software*, vol. 7, pp. 170–183, June 1981.
[24] M. Woodbury, "Inverting modified matrices," Memo. 42, Statistics Research Group, Princeton, NJ, 1950.
[25] H. Zhang, "On the accuracy of the parallel diagonal dominant algorithm," *Parallel Comput.*, pp. 265–272, 1991.



Xian-He Sun (S'88–M'90) received the B.S. degree in mathematics from Beijing Normal University, Beijing, China, in 1982 and the M.S. and Ph.D. degrees in computer science from Michigan State University, East Lansing, MI, in 1987 and 1990, respectively.

He is currently a postdoctoral fellow at Ames Laboratory, Ames, IA. His research interests include parallel processing, performance evaluation, parallel numerical algorithms, and database systems.

Dr. Sun is a member of the IEEE Computer Society and the Association for Computing Machinery.

Hong Zhang received the B.S. degree in mathematics from Beijing Normal University, Beijing, China, in 1982 and the M.S. and Ph.D. degrees in mathematics from Michigan State University in 1985 and 1989, respectively.

She is currently an Assistant Professor at Clemson University, Clemson, SC. Her research interests include numerical and scientific computation. Her current work is on the designing of parallel algorithms for solution of linear algebraic systems and eignevalue problems.

Lionel M. Ni (S'78–M'81–SM'87) received the B.S. degree in electical engineering from National Taiwan University in 1973, the M.S. degree in electrical and computer engineering from Wayne State University, Detroit, MI, in 1977, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1980.

In 1981 he joined the faculty of the Department of Computer Science, Michigan State University, East Lansing, where he is currently a Professor and Director of the Advanced Computer Systems Laboratory. During the summers of 1979 and 1981, he was a Researcher at the IBM San Jose Research Laboratories. During the year 1987–1988, he was a Visiting Scientist in the Division of Mathematics and Computer Science at Argonne National Laboratory. He is an editor of the *Journal of Parallel and Distributed Computing*. His research interests include parallel processing and distributed computing.

Dr. Ni served as a Distinguished Visitor of the IEEE Computer Society from 1985 to 1988. He is a member of the Association for Computing Machinery and SIAM, and served as the program chairman of the Fifteenth Annual International Computer Software and Applications Conference.