A Parallel Two-Level Hybrid Method for Tridiagonal Systems and Its Application to Fast Poisson Solvers

Xian-He Sun, *Senior Member*, *IEEE*, and Wu Zhang

Abstract—A new method, namely, the Parallel Two-Level Hybrid (PTH) method, is developed to solve tridiagonal systems on parallel computers. PTH has two levels of parallelism. The first level is based on algorithms developed from the Sherman-Morrison modification formula, and the second level can choose different parallel tridiagonal solvers for different applications. By choosing different outer and inner solvers and by controlling its two-level partition, PTH can deliver better performance for different applications on different machine ensembles and problem sizes. In an extreme case, the two levels of parallelism can be merged into one, and PTH can be the best algorithm otherwise available. Theoretical analyses and numerical experiments indicate that PTH is significantly better than existing methods on massively parallel computers. For instance, using PTH in a fast Poisson solver results in a 2-folds speedup compared to a conventional parallel Poisson solver on a 512 nodes IBM machine. When only the tridiagonal solver is considered, PTH is over 10 times faster than the currently used implementation.

Index Terms—Parallel processing, scalable computing, tridiagonal systems, Poisson solver.

1 Introduction

Solving tridiagonal systems is one of the key issues in numerical simulations of many scientific and engineering problems. However, solving tridiagonal systems efficiently is a difficult task on parallel computers due to their inherent data dependencies and low computation to communication ratio. For this reason, parallel tridiagonal algorithms have been studied extensively and remain an active research area [3], [4], [9], [19]. Notable tridiagonal solvers include the recursive doubling reduction method (RCD) developed by Stone [11], and the cyclic reduction or odd-even reduction method (OED) developed by Hockney [5], which are able to solve an n-dimensional tridiagonal system in $O(\log n)$ time using p processors with $p = n$ and are efficient for the case of fine grain computing. Several algorithms were proposed for medium and coarse grain computing, i.e., for the case of $p < n$, or $p << n$ [2], [8], [18]. The algorithms of Lawrie and Sameh [8] and Wang [18] are called divide-and-conquer methods, which partition the original problem into subproblems. The subproblems are then solved in parallel, and the final solution is obtained by combining the solutions of the subproblems. Later, Sun [12] and Sun et al. [15] proposed three parallel algorithms for solving tridiagonal systems. All three algorithms are divide-and-conquer and are based on the Sherman-Morrison matrix modification formula [1]. Two of them, the Parallel ParTition LU (PPT) algorithm and the Parallel Partition Hybrid (PPH) algorithms, are fast and able to incorporate limited pivoting. The third algorithm, the Parallel Diagonal Dominant (PDD) algorithm, is designed for diagonal dominant systems. The PDD algorithm is the most efficient among these three algorithms. Compared to the usually required $O(\log(p))$ communication cost, the PDD algorithm has only two communications, independent of the number of processors, and has a balanced workload on processors. In reality, the PDD algorithm is perfectly scalable in terms of iso-speed or iso-efficiency scalability [12], [14]. Since its creation, many practitioners have adopted PDD as their choice [7]. PDD, however, has its limitations. It requests diagonal dominance. Studies of the application, accuracy, and performance of PDD algorithm can be found in [12], [13].

Most parallel tridiagonal solvers trade computation for increasing parallelism. For solving multiple tridiagonal systems or systems with multiple right-hand sides, pipelining can be used with the best sequential algorithm. Pipelining works by passing the partial solutions of a subset of systems onto the next processor before solving another subset of the systems. The pipelining approach is computationally efficient, but has a communication cost of $O(p)$. It is a good choice when p, the number of processors, is small. When p is large, its performance drops dramatically due to communication costs and pipeline delays.

A novel Parallel Two-level Hybrid (PTH) method for tridiagonal systems is proposed in this study based on the PDD and PPT algorithms. With a two-level partition, PTH has two levels of parallelism: The first level (outer level) is based on PDD or PPT, while the second level (inner level) can choose different parallel tridiagonal solvers based on the underlying application. For instance, PPT is a good candidate for single systems and pipelining is a natural choice for solving multiple systems. PDD is highly scalable,

- *X.-H. Sun is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616. E-mail: sun@cs.iit.edu.*
- *W. Zhang is with the College of Computer Science and Engineering, Shanghai University, Shanghai, 200072, P.R. China. E-mail: wzhang@mail.shu.edu.cn.*

Manuscript received 19 Aug 2002; revised 17 Jan. 2003; accepted 19 June 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 117147.

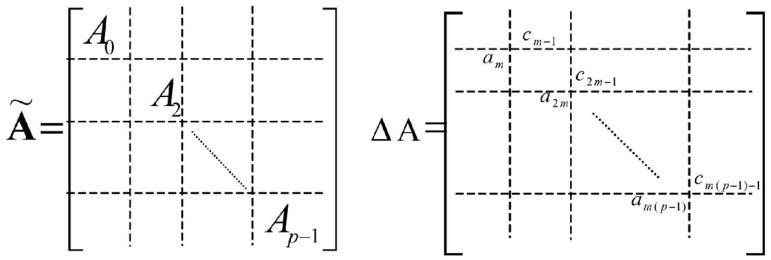

Fig. 1. $m \times m$ tridiagonal matricies.

but becomes inaccurate when the subsystems are small. PPT and pipelining are efficient when p is small but not scalable. PTH algorithm can overcome the shortcomings of the outer and inner solver, and makes use of the merits of both. A representative PTH algorithm is the combination of PDD and the pipelined method that is highly efficient and more applicable than PDD. PTH is thus significantly more appropriate for massively parallel computers than existing tridiagonal solvers.

This paper is organized as follows: Section 2 gives the background and reviews of the PPT, PDD, and the pipelined method. The pros and cons of the algorithms are discussed. The PTH method is introduced in Section 3. Section 4 provides numerical experiments on a teraflop-scale parallel computer for solving Poisson equations. Finally, conclusions are given in Section 5.

2 Existing Parallel Tridiagonal Solvers

A tridiagonal system is a linear system of equations

$$Ax = \begin{pmatrix} b_0 & c_0 & & & \\ a_1 & b & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & a_{n-1} & b_{n-1} \end{pmatrix} x = d, \qquad (1)$$

where $x = (x_0, x_1, \ldots, x_{n-1})^T$ and $d = (d_0, d_1, \ldots, d_{n-1})^T$ are n-dimensional vectors and A is a tridiagonal matrix of dimension n. A is called diagonally dominant if $|b_i| > |a_i| + |c_i|$, for $0 \le i < n$. Assume A, x, and d have real coefficients (extension to the complex case is straightforward). We now introduce PPT, PDD, and the pipelined algorithm for solving system (1).

To solve (1) efficiently on parallel computers, we partition A into two parts, the main part $\widetilde{A}$ and the residue ΔA. For convenience, we assume that $n = p \cdot m$, where m is the subsystem size (in general, if $n - k = p \cdot m$, where $k < p$, then we have k subsystems of order $m+1$ and $p-k$ subsystems of order m). The tridiagonal matrix A can be written as

$$A = \widetilde{A} + \Delta A, \qquad (2)$$

where $\widetilde{A}$ is a block diagonal matrix with diagonal submatrices $A_i (i = 0, 1, \ldots, p-1)$. The submatrices $A_i (i = 0, 1, \ldots, p-1)$ are $m \times m$ tridiagonal matrices shown in Fig. 1.

Let e_i be a column vector with its ith element, $0 \le i < n-1$, being one and all the other entries being zero. We have,

$$\Delta A = [a_m e_m, c_{m-1} e_{m-1}, a_{2m} e_{2m}, c_{2m-1} e_{2m-1}, \ldots, c_{(p-1)m-1} e_{(p-1)m-1}] \cdot \begin{bmatrix} e^T_{m-1} \\ E^T_m \\ \cdot \\ \cdot \\ \cdot \\ e^T_{(p-1)m-1} \\ e^T_{(p-1)m} \end{bmatrix} = VE^T, \qquad (3)$$

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

$$A = \widetilde{A} + VE^T. \qquad (4)$$

Based on the matrix modification formula originally defined by Sherman and Morrison [10] for rank-one changes, (1) can be solved by

$$x = A^{-1}d = (\widetilde{A} + VE^T)^{-1}d, \qquad (5)$$

$$x = \widetilde{A}^{-1}d - \widetilde{A}^{-1}V(I + E^T\widetilde{A}^{-1}V)^{-1}E^T\widetilde{A}^{-1}d. \qquad (6)$$

Note that I is an identity matrix. $Z' = (I + E^T\widetilde{A}^{-1}V)$ is a pentadiagonal matrix of order $2(p-1)$. We introduce a permutation matrix P such that

$$Pz = (z_1, z_0, z_3, z_2, \cdots, z_{2p-3}, z_{2(p-2)})^T, \text{ for all } z \in R^{2(p-1)}. \qquad (7)$$

From the property that $P^{-1} = P$, (6) becomes

$$x = \widetilde{A}^{-1}d - \widetilde{A}^{-1}VP(P + E^T\widetilde{A}^{-1}VP)^{-1}E^T\widetilde{A}^{-1}d. \qquad (8)$$

The intermediate matrix, $Z + (P + E^T\widetilde{A}^{-1}VP)$, is a $2(p-1) \times 2(p-1)$ tridiagonal system, which leads to a reduced computation cost. The modified solving sequence becomes,

$$\widetilde{A}\widetilde{x} = d, \qquad (9)$$

$$\widetilde{A}Y = VP, \qquad (10)$$

$$h = E^T\widetilde{x}, \qquad (11)$$

$$Z = P + E^TY, \qquad (12)$$

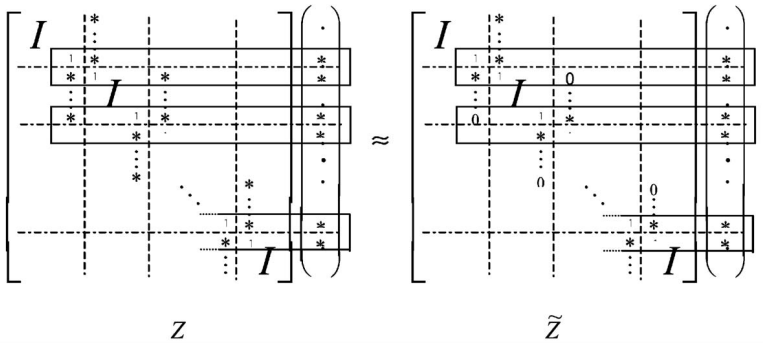

Fig. 2. The matrix form of Z and $\widetilde{Z}$.

$$Zy = h, \tag{13}$$

$$\Delta x = Yy, \tag{14}$$

and (8) becomes

$$x = \widetilde{x} - \Delta x. \tag{15}$$

In (9) and (10), $\widetilde{x}$ and Y are solved by the LU decomposition method. By the structure of $\widetilde{A}$ and V, these are equivalent to solving

$$\begin{aligned} &A_i[\widetilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im}e_0, c_{(i+1)m-1}e_{m-1}], \\ &i = 0, 1, \cdots, p-1. \end{aligned} \tag{16}$$

Here, $x^{(i)}$ and $d^{(i)}$ are the ith block of $\widetilde{x}$ and d, respectively, and $v^{(i)}$ and $w^{(i)}$ are possible no-zero column vectors of the ith row block of Y. Equation (16) implies that we only need to solve three linear systems of order m with the same LU decomposition for each $i, i = 0, 1, \cdots, p-1$.

2.1 PPT: The Parallel Partition LU Algorithm

Based on the matrix partitioning technique above, the PPT algorithm can use p processors to solve (1) concurrently. The steps are as follows:

Step 1. Allocate $A_i, d^{(i)}$ 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 (16). All computations can be executed in parallel and independently on p processors.

Step 3. Send $\widetilde{x}_0^{(i)}, \widetilde{x}_{m-1}^{(i)} v_0^{(i)}, v_{m-1}^{(i)}, w_0^{(i)}, w_{m-1}^{(i)} (0 \le i \le p-1)$ to all other nodes from the ith node to form matrix Z and vector h (11-12) on each node. Here and throughout, the subindex indicates the component of the vector.

Step 4. Use the *LU* decomposition method to solve (13) on all nodes simultaneously. Note that Z is a $2(p-1)$ dimensional tridiagonal matrix.

Step 5. Compute (14) and (15) in parallel on p processors. We have

$$\begin{aligned} \Delta x^{(i)} &= [v^{(i)}, w^{(i)}] \begin{bmatrix} y_{2i-1} \\ y_{2i} \end{bmatrix} \\ x^{(i)} &= \widetilde{x}^{(i)} - \Delta x^{(i)}. \end{aligned}$$

2.2 PDD: Parallel Diagonal Dominant Algorithm

When A is diagonal dominant, an interesting mathematical property is that the off diagonal coefficients of the matrix $\widetilde{A}^{(i)}V$ decay to 0 exponentially with the order of the matrix. Therefore, the coefficients can be dropped within machine accuracy when $p \ll n$, that is,

$$Z \approx \widetilde{Z}, \tag{17}$$

or in matrix form shown in Fig. 2.

As shown in [12], [13], [15], for most diagonal dominant systems, when the subsystem size is greater than 64, the reduced matrix Z is equivalent to $\widetilde{Z}$ within machine accuracy for numerical computing. PDD uses $\widetilde{Z}$ for the solution and needs only two neighboring communications. The computation and communication pattern of PDD is shown in Fig. 3. The optimal and simple communication property makes PDD algorithm an ideal algorithm for massively parallel computing.

The resulting PDD algorithm can be described in the following five steps:

Step 1. Allocate $A_i, d^{(i)}$ 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

$$A_i[\widetilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im}e_0, c_{i+1)m-1}e_{m-1}].$$

All computations can be executed in parallel and independently on p processors.

Step 3. Send $\widetilde{x}_0^{(i)}, v_0^{(i)}$ from the ith node to the $(i-1)$th node for $1 \le i \le p-1$.

Step 4. Solve

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

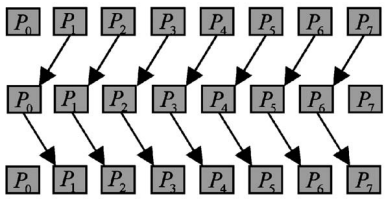

Fig. 3. The computation and communication pattern of PDD.

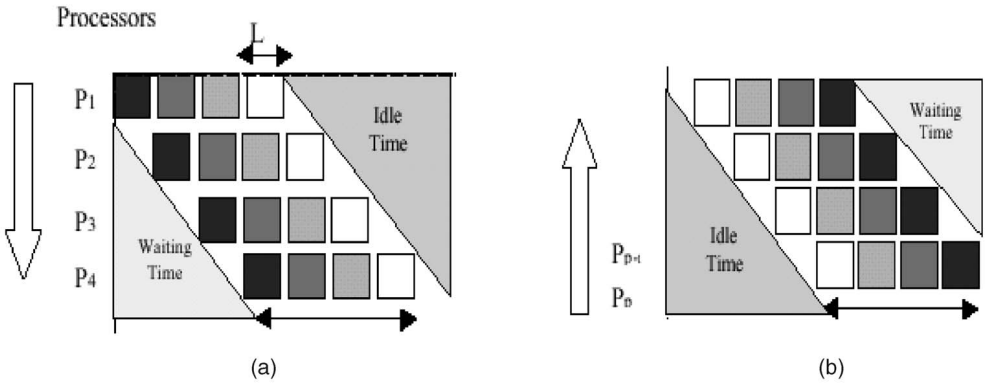

Fig. 4. Pipelined algorithm for the substitutions. (a) Forward. (b) Backward.

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

Step 5. Compute in parallel on p processors

$$\Delta x^{(i)} = [v^{(i)}, w^{(i)}] \begin{bmatrix} y_{2i-1} \\ y_{2i} \end{bmatrix},$$
$$x^{(i)} = \widetilde{x}^{(i)} - \Delta x^{(i)}.$$

2.3 The Pipelined Method for Multiple Systems

PPT and PDD can be applied to either single tridiagonal systems or multiple systems where multiple independent systems or a system with multiple right-hand sides have to be solved.

Like most parallel tridiagonal solvers, PPT and PDD algorithms have a nonoptimal computation count. For solving multiple systems, however, an optimal sequential algorithm can be used to achieve parallel processing via pipelining [3].

Pipelining works by passing the intermediate results from solving a subset of the systems onto the next processor before continuing. Let K be the number of systems to be solved. We partition the K systems into M sets. Each set has L systems: $K + m \cdot L$. The pipelining procedure is given below (see Fig. 4).

- In the first pass, processor 0 solves the first part of the first L systems whereas processors 1 to $p-1$ are idle.
- In the second pass, processor 1 works on the second part of the first L system (using the results of the first pass) while processor 0 works on the first part of the second L systems; processors 2 to $p-1$ are idle.
- In the ith pass, processor $i-1$ solves the ith part of the first L systems, processor $i-2$ solves the $(i-1)$th part of the second L systems, ..., processor 0 solves the first part of the ith L systems. The subsequent passes continue until eventually running out of work, and processors one by one (starting with processor 0) go idle.

There are three rounds of computations for solving a tridiagonal system (or systems) via the conventionally used tridiagonal solver, the Thomas algorithm [12]. One round is for *LU* decomposition, one is for forward substitution, and another one is for backward substitution. Because each round of computation requires $p-1$ communications, the communication cost is high, increasing linearly with the number of processors. Also unfortunate is the $O(p-1)$ pipeline delay, illustrated in Fig. 4. However, the communication costs are balanced by the fact that the amount of computation is optimal, as it uses the best sequential method. When the number of processors is small, and there are multiple systems to be solved, the pipelined method becomes a good candidate.

3 PTH: The Parallel Two-Level Hybrid Method

PDD is highly scalable [12]. However, when the number of processors, p, is close to the order of the tridiagonal matrix, n, the size of the submatrices may become too small to maintain accuracy. PPT does not require diagonal dominance and does not have the accuracy problem. PPT, however, has an $O(\log(p))$ communication complexity and is nonoptimal in computation. The pipelined algorithm achieves the optimal amount of computation, but pays with $O(p)$ communication. An ideal tridiagonal solver would combine the scalability of the PDD algorithm with the accuracy of the PPT algorithm and the pipelined method.

The Parallel Two-Level Hybrid (PTH) method is proposed in this study to combine the merits of PDD, PPT, and the pipelined methods. The basic idea of PTH is to embed an inner tridiagonal solver into PDD or PPT to form a two-level hierarchical parallelism. The base algorithm is PDD or PPT. For diagonal dominant tridiagonal systems, the system is first partitioned based on PDD. However, the subsystems may be too small for the accuracy concerned if we use PDD directly with the one-processor one-subsystem approach. To overcome the limitation of PDD, we group each k processors together to solve a supersubsystem (see Fig. 5). Each supersubsystem is an independent tridiagonal system and can be solved by any direct parallel tridiagonal solver that does not introduce approximation error. For single tridiagonal systems, a good choice for the inner tridiagonal solver would be PPT algorithm [15]. PPT has a good parallelism and a $\log(p)$ communication cost. For multiple tridiagonal systems, the pipelined method would be a natural candidate for the inner solver. Both PPT and the pipelined method require global communication and are otherwise efficient. When they are embedded into PDD, they are used on a small number of processors for solving

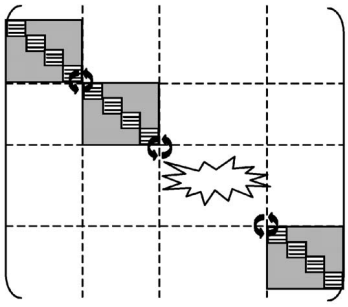

Fig. 5. PTH method: a two-level hierarchical approach.

the supersubsystems, and the communication cost is small. The two-level hybrid method takes the advantage of PDD and inner solvers. PDD takes care of the scalability issue and can be scaled efficiently on massively parallel machines. The inner solvers conduct efficient computation at the local level and provide an adequate solution for accuracy concerned. In addition, by adjusting the size of the supersubsystem, PTH method can scale-up and scale-down on different parallel machines based on the number of processors available. When the number of supersubsystem equals one, PTH is the inner tridiagonal solver. When the number of supersubsystem equals p, the number of processors, PTH becomes the outer algorithm. The optimal number of the supersubsystem is a function of the machine parameters and the desired error bounds. For instance, if communication cost is the concern, then we may use PDD as the outer solver and make the size of the supersubsystems as small as the given error bound that can be met; if computing cost is the concern, we may use the pipelining algorithm as the inner solver and make the size of the supersubsystems as large as possible, until communication becomes a concern or the number of supersubsystem becomes one. When PDD and the pipelined algorithm is chosen as the outer and inner solver, respectively, we call the resulting PTH algorithm *Partition Pipelined Diagonal Dominant (PPD)* algorithm. For general tridiagonal systems, PPT can be used as the outer solver and combined with the pipelined method for multiple systems. The communication complexity of $O(\log(p))$ is not as good as that of PDD but is better than the pipelined method.

The PTH method can be described in the following two steps.

Step 1. Use an accurate parallel tridiagonal solver to solve the m supersubsystems concurrently, each with k processors, where $p = l \cdot k$ and solving three unknowns as given in the *Step 2* of PDD algorithm.

Step 2. Modify the solutions of *Step* 1 with *Steps* 3-5 of the PDD algorithm, or of PPT algorithm if PPT is chosen as the outer solver.

If the accurate parallel tridiagonal solver of *Step* 1 is the pipelined method and the algorithm used in *Step 2* is the PDD algorithm, the resulting PTH algorithm is the PPD algorithm. PPD is practically important for many technical applications and deserves a name on its own. Table 1 summarizes some of the tridiagonal algorithms that we have studied and the PTH method.

Table 2 gives the computation and communication counts of the pipelined method, PPT, PDD, PPD, and other PTH-based algorithms, respectively. α is the communication startup time. β is the data transmission time per byte, normalized to the computing time. The parameters n_1, n, and k in Table 2 stand for the number of tridiagonal systems, the order of the systems and the number of subsystems (or the number of processors) for each supersubsystem, respectively. We assume four bytes per word in our analysis. With n_1 tridiagonal systems, if we pipeline these n_1 systems one-by-one, the computation and communication count of the pipelined method is $(n_1 - 1 + p)\frac{8n-7}{p}$ and $(n_1 - 1 + p)(3\alpha + 12\beta)$, respectively. To reduce the communication cost the n_1 systems are often grouped into m groups, each group with m_1 systems, such as $n_1 = m \cdot m_1$, and the tridiagonal systems is pipelined group by group. The computation and communication count of the general pipelined algorithm is used in Tables 2 and 3.

The computation and communication count of PDD and PPT are copied from [15]. PDD requires only two neighboring communications. Its communication count is machine (connection) independent. PPT requires a data-gathering communication. Its $O(log(p))$ communication cost is based on the assumption that a butterfly communication is supported [15]. We use the same assumption in the communication analysis of this study. In PPD, the pipelined method is used to solve (16). As pointed out in [15], we only

TABLE 1
The PTH Method and Related Tridiagonal Algorithms

Abbreviation	Full Name	Explanation
PPT	Parallel ParTition LU Algorithm	A parallel tridiagonal solver based on rank-one modification
PDD	Parallel Diagonal Dominant Algorithm	A variant of PPT algorithm with dropping, only applicable for diagonal dominant systems
PTH	Parallel Two-level Hybrid Method	A novel two-level approach for solving tridiagonal systems
PPD	Partition Pipelined diagonal Dominant Algorithm	A PTH based alg. where the outer solver is PDD and the inner solver is the pipelined alg.

TABLE 2
Computation and Communication Count (Multiple Independent Systems)

Algorithm	Computation	Communication
The Pipelined Algorithm	$(m-1+p)\frac{8n-7}{p}m_1$	$(m-1+p)(3\alpha+12m_1\beta)$
PPT (nonpivoting)	$\left(17\frac{n}{p}+16p-45\right)n_1$	$(\log(p))\alpha+16(p-1)n_1\beta$
PDD	$\left(17\frac{n}{p}-14\right)n_1$	$2\alpha+12n_1\beta$
PPD (PDD/Pipeline)	$(m-1+k)\frac{13n}{p}m_1+\left(\frac{4n}{p}+7\right)n_1$	$(m-1+k)(3\alpha+24m_1\beta)+$ $(2+\log(k))\alpha+(8\log(k)+12)n_1\beta$
PPT/Pipeline	$(m-1+k)\frac{13n}{p}m_1+$ $\left(\frac{4n}{p}+16\frac{p}{k}-23\right)n_1$	$(m-1+k)(3\alpha+24m_1\beta)+$ $(\log(p))\alpha+$ $\left[16\left(\frac{p}{k}-1\right)+8\log(k)\right]n_1\beta$
PDD/PPT	$\left(30\frac{n}{p}+21k-41\right)n_1$	$(2+2\log(k))\alpha+$ $(16(k-1)+8\log(k)+20)n_1\beta$

TABLE 3
Computation and Communication Count (Multiple Right Sides)

Algorithm	Computation	Communication
The Pipelined Algorithm	$(m-1+p)\frac{5n-3}{p}m_1$	$(m-1+p)(2\alpha+8m_1\beta)$
PPT (nonpivoting)	$\left(9\frac{n}{p}-10p-11\right)n_1$	$\log(p)\alpha+8(p-1)n_1\beta$
PDD	$\left(9\frac{n}{p}+1\right)n_1$	$2\alpha+8n_1\beta$
PPD (PDD/Pipeline)	$(m-1+k)\frac{5n}{p}m_1+\left(\frac{4n}{p}+7\right)n_1$	$(m-1+k)(2\alpha+8m_1\beta)+$ $(2+\log(k))\alpha+$ $8(1+\log(k))n_1\beta$
PPT/Pipeline	$(m-1+k)\frac{5n}{p}m_1+\left(\frac{4n}{p}+\frac{10p}{k}-11\right)n_1$	$(m-1+k)(2\alpha+8m_1\beta)+$ $(\log(p))\alpha+$ $8\left[\frac{p}{k}-1+\log(k)\right]n_1\beta$
PDD/PPT	$\left(13\frac{n}{p}+10k-4\right)n_1$	$(2+2\log(k))\alpha+$ $8(k+\log(k)+1)n_1\beta$

need one LU-decomposition for solving and can skip the forward substitutions for the third set of systems. The communication of the pipelined inner solver is $(m-1+k)(3\alpha+24\cdot m_1\beta)$ for solving (16); the communication cost for solving the reduced system of PDD is $(2\alpha+12n_1\beta)$ (the reduced system is generated by the boundary equations of the l supersubsystems); the communication cost for propagating the solution of the reduced system inside each supersubsystem is $\log(k)(\alpha+8n_1\beta)$. The summation of the three is the communication count

$$(m-1+k)(3\alpha+24\cdot m_1\beta)+(2+\log(k))\alpha+(8\log(k)+12)n_1\beta.$$

We use PPT/Pipeline to denote the PTH algorithm where PPT is used as the outer solver and the pipelined algorithm is used as the inner solver. We can reach the computation and communication count of the PPT/Pipeline algorithm by following a similar deduction of that used on PPD. PPT/Pipeline is a general tridiagonal solver and does not require diagonal dominance. Alternatively, we can use PDD as the outer solver and PPT as the inner solver. The resulting PDD/PPT PTH algorithm may provide better performance than PPD if n_1 is small or k is big. Notice that, for the PDD/PPT approach, both the inner and outer solvers need to solve three equations, though the third

equation can skip the forward substitution step. In addition, the reduced systems of the second and third equations of the PPT inner solver can also skip some forward substitution. The computation and communication count of the three steps of the PDD/PPT approach is:

$$n_1\left(17\frac{n}{p}+16k-47\right)+(\log(k))\alpha+16(k-1)n_1\beta,$$
$$n_1\left(\frac{9n}{p}\right)+5k-3)+8n_1\beta, \text{ and}$$
$$n_1\left(\frac{4n}{p}\right)+7)+(2+\log(k))\alpha+(8\log(k)+12)n_1\beta,$$

respectively. Table 2 lists the summation of the costs.

The computation and communication costs for solving tridiagonal systems increase with the parameter n_1 for all algorithms. Compared with the pipelined algorithm, the PPD algorithm reduces the communication cost significantly when p is big due to the fact that $k << p$. In general, in PPD, we choose the smallest k that maintains the accuracy. From Table 2, we can see that when p is small, the pipelined method is the best among the three. When p is big, PDD provides the best performance. PDD, however, may lose accuracy and, therefore, become inapplicable when p is big. When PDD is inapplicable, PPD is the leading algorithm. The range of p for the performance rank change is machine and application dependent.

Table 2 is for solving n_1 independent tridiagonal systems. Table 3 gives the computation and communication count of solving n_1 multiple right-sides systems, where the LU decomposition cost is not considered. For many applications, the computation costs of PDD and other PDD-based algorithms can be reduced by using the Reduced PDD algorithm. Interested readers can refer [12] for further study.

4 Poisson Solver Application and Numerical Experiments

Tridiagonal solvers can be used to solve systems that arise in a variety of applications. In this section, we present experimental results from using the methods described in this paper to the fast Poisson solvers.

4.1 Application: Fast Poisson Solver

Solving Poisson equations has received continuous attention for several decades. Sequential Poisson solvers are very close to optimal and are unlikely to be further improved. Existing parallel Poisson solvers, however, do not meet the demand of scalable computing. They are often being identified as the bottleneck of parallel simulation packages when the number of processors is large [17]. Among these parallel solvers, Hockney's fast Poisson solver, the Fourier Analysis and Cyclic Reduction (FACR) algorithm [6], is the most accepted direct solver. We use FACR(0) as an application of our tridiagonal solvers.

A 2D Poisson equation written in Cartesian coordinates is

$$\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}=f(x,y)\ in\ \Omega, \tag{18}$$

where Ω is a rectangular region and $f(x,y)$ is a given function. Periodic boundary condition in x direction and Dirichlet boundary condition in y direction are considered, respectively.

In the case that the function $\overline{\varphi}^k(y)$ has prescribed values at the boundaries in y direction, we can solve the problem with the following steps [6], which is called the fast Poisson solver or FACR.

Step 1. Conduct Fourier transform on the given function $f(x,y) \rightarrow \overline{f}^k(y)$.

Step 2. Solve $n_1$independent tridiagonal systems, each with order of n, $\overline{f}^k(y) \rightarrow \overline{\varphi}^k(y)$.

Step 3. Conduct Fourier transform on the function $\overline{\varphi}^k(y) \rightarrow \varphi(x,y)$.

Fast Fourier Transform (FFT) is another fundamental kernel solver with great challenge. FFT has an even worse computing/communication ratio than tridiagonal solvers. The best way to solve FACR on parallel machines is to solve FFT sequentially on each processor and solve the tridiagonal systems in parallel. To solve the FFTs in each machine locally, however, makes the resulting tridiagonal systems have a very specific data distribution: Each processor has a submatrix (on the order of n/p) from each of the N_1 tridiagonal systems, so that each processor solves N_1 subsystems sequentially. Even if P is close to N, each processor may still have a lot of computing work to do. When P is close to N, PDD becomes inapplicable.

4.2 Experimental Testing

Experimental testing was performed on the National Partnership for Advanced Computational Infrastructure (NPACI) IBM Blue-Horizon at the San Diego Supercomputing Center (SDSC). The Blue Horizon is a teraflop-scale Power3-based cluster, which contains 1,152 processors and 576 GBytes of main memory, arranged as 144 Symmetric Multiprocessing (SMP) compute nodes. Each node is equipped with 4 GBytes of memory shared among its 8-375 MHz Power3 processors. Each node also has several GBytes of local disk space.

We use FACR(0) to test PTH. Since the tridiagonal systems in FACR are multiple diagonal dominant systems, PPD is the chosen outer solver. For the experiments, we choose the wave number $N_1 = 512$ in *x*-direction and mesh points $n = 4,608$ in *y*-direction. Therefore, we need to solve 512 tridiagonal systems, where each has an order of 4,608. The size of the subsystems is a function of the number of processors used. The performance of 1, 12, 24, 48, 96, 192, 384, and 512 processor implementations are measured, which has the subsystem sizes of 4,608, 384, 192, 96, 48, 24, 12, and 9, respectively.

The experimental results for solving the FACR tridiagonal systems with PDD and the Pipelined method are given in Fig. 6. As shown in Fig. 6, the pipelined method performs better for small ensemble size. At $p = 96$, PDD starts to outperform the pipelined method. The performance gap between PDD and the pipelined method continues to enlarge, as the number of processors increase. At $p = 512$ PDD is 10 times faster than the pipelined method, which is very impressive. The experimental results confirm PDD is highly scalable. For the given application,

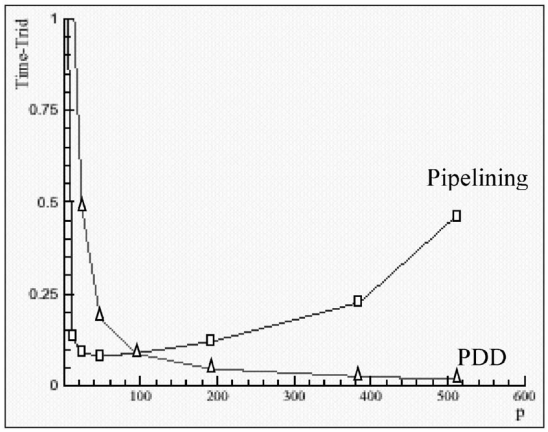

Fig. 6. Tridiagonal solver runtime: Pipelining (square) and PDD (delta).

however, when p is greater than 96, the subsystem size is less than 48 and PDD does not provide accurate results. PDD is not applicable for FACR when p is large.

We now use PPD for FACR and choose k, the number of subsystems in a supersubsystem, to be 16. We take the same set of numbers of processors as above for PPD testing, that is, numbers of processors 1, 12, 24, 48, 96, 192, 384, and 512 are considered, corresponding to the subsystem sizes of 4,608, 384, 192, 96, 48, 24, 12, and 9.

Fig. 7 shows the runtimes of three tridiagonal solvers for the FACR tridiagonal systems. From $p = 1$ to $p = 48$, PPD chooses to take one supersubsystem. That means PPD is the pipelined method for $p < 96$. Starting at $p = 96$, PPD groups each 16 subsystems to form a supersubsystem and use 16 processors to solve each supersubsystem. In this way, PPD reaches the optimal performance. We can see PPD achieves a near PDD performance when p is large and the same performance as the pipelined method when p is small.

Fig. 8 shows the measured accuracy of PDD, PPD, and the pipelined method, compared with the sequential algorithm with the L_∞ norm. The accuracy of PPD coincides with that of the pipelined method in L_∞ norm, where PDD is not. Experimental results confirm that PPD is scalable and applicable. It is a good algorithm for FACR. The measured runtimes of FACR for solving the Poisson equation with

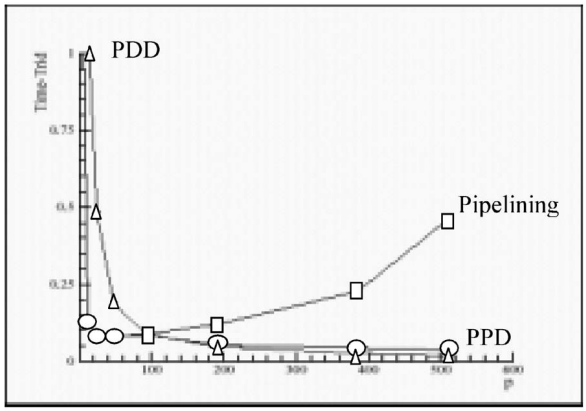

Fig. 7. Tridiagonal Solver Time: Pipelining (square), PDD (delta), and PPD (circle).

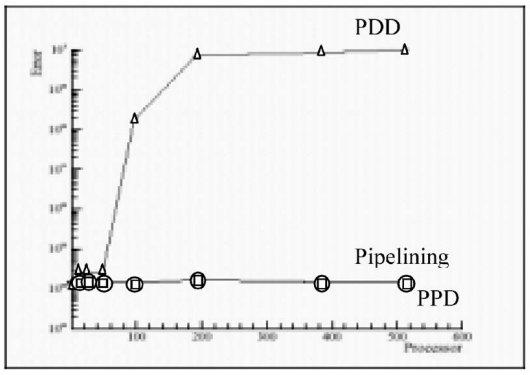

Fig. 8. Accuracy: Pipelining (square), PDD (delta), and PPD (circle).

three different tridiagonal solvers are given in Fig. 9. Please note that the FACR-PDD implementation does not provide an accurate solution. It only can be used as a preconditioner for further computing.

4.3 Performance Prediction

The measured values of the parameters on the IBM Blue Horizon at SDSC are $\alpha = 31.5 \times 10^{-6}$ sec, $\beta = 0.00925 \times 10^{-6}$ sec/*byte*, and $\tau_{comp} = 0.017 \times 10^{-6}$ sec, where α is the communication startup time, β is the data transmission time per byte, and τ_{comp} is the operation time per floating-point operation. We can use the formulas given in Tables 2 and 3 to predict the performance. We first use the measured results to confirm the performance formulas and then use the formulas to predict the performance on even larger computing systems. Since PDD is well-studied in [13], only PPD and the pipelined method are studied here.

Comparisons of the predicted and numerical runtimes for the two algorithms are given in Figs. 10 and 11, respectively. We can see our predicted results match the measured results well. Figs. 12 and 13 give the speedup of the PPD and the pipelined method, respectively. Here, speedup is defined as the single processor execution time of the sequential algorithm over the parallel execution time of the parallel algorithm [16]. From Figs. 12 and 13, we can see that the speedup of the PPD increases linearly with the number of processors, while the linear increasing property does not hold for the pipelined method due to its heavy

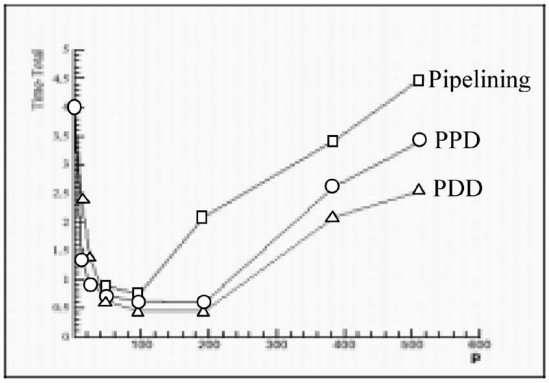

Fig. 9. Total runtime: Pipelining (square), PDD (delta), and PPD (circle).

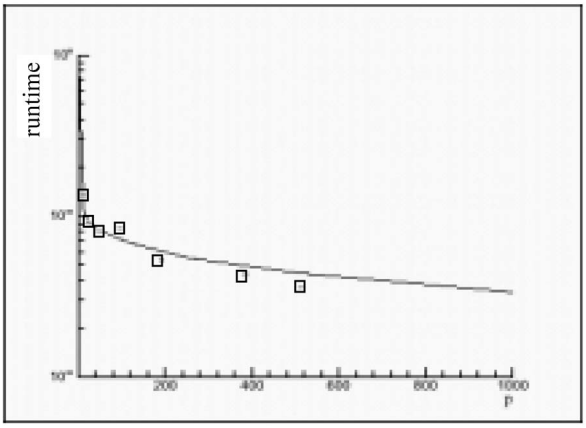

Fig. 10. PPD: The predicted (line) and numerical (square) runtime.

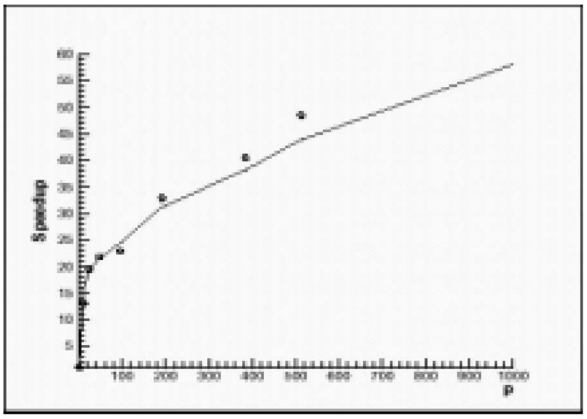

Fig. 12. Speedup of PPD: predicted (line) and numerical (circle).

communication cost. The predicted results show that PPD has a potential for even large computing systems. Since the system arising in the fast Poisson solvers is common in scientific applications, PPD has a real potential for many scientific applications.

5 Conclusions

The PDD algorithm is an efficient algorithm for diagonal dominant tridiagonal systems. Based on the Sherman and Morrison formula [10], PDD can drop elements without losing accuracy for numerical computing. The exponential decay rate of the dropping elements has been mathematically proven. However, it is also known that PDD is inapplicable when the size of the partitioned subsystems is small. In this study, a new method, the Parallel Two-level Hybrid (PTH) method, is proposed to overcome the shortcoming of PDD. PTH consists of two parallel tridiagonal solvers: the outer solver and the inner solver. The outer solver is PDD or PPT algorithm. The inner solver is open and can be application specific. When the outer and inner solver is the PDD and the pipelined method, respectively, the resulting PTH is called the Partition Pipelined diagonal Dominant (PPD) algorithm. PPD maintains PDD's scalability and is feasible while PDD is not. PPD has been examined closely and experimentally tested for the well-known fast Poisson solver originally proposed by Hockney [5]. Experimental analyses show that PPD is fundamentally more appropriate for the fast Poisson solver than existing tridiagonal algorithms. PPD needs to be further studied to improve its performance in solving Poisson equations and other applications. PPD is one of the many possible algorithms that can be generated from the PTH method. For instance, for general tridiagonal systems, PPT can replace PDD and be used as the outer solver. The potential of PTH should also be further investigated.

Acknowledgments

The thought of using PDD in a fast Poisson solver has been with X.-H. Sun for many years. After delivering an invited talk at the Scalability Workshop at the SuperComputing'92 conference, X.-H. Sun had a long and pleasant talk with Professor Roger Hockney on various topics, from performance evaluation to a fast Poisson solver. Both of them realized that PDD had the potential for improving FACR and agreed to incorporate PDD in FACR when time permitted. But alas, they never found the time. Dr. Hockney passed away in 1999 and, since then, improving the performance of FACR has become Dr. Sun's personal commitment. In fiscal year 2000, they received funding

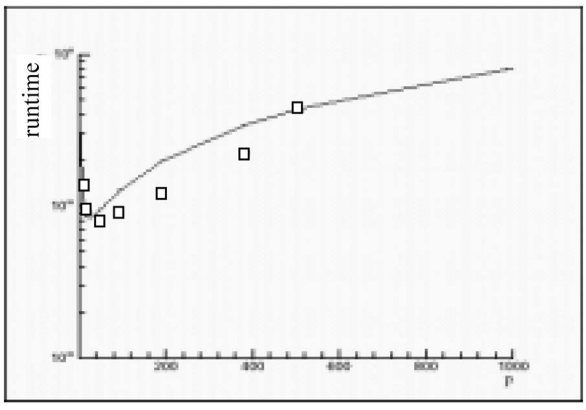

Fig. 11. Pipelining: The predicted (line) and numerical (square) runtime.

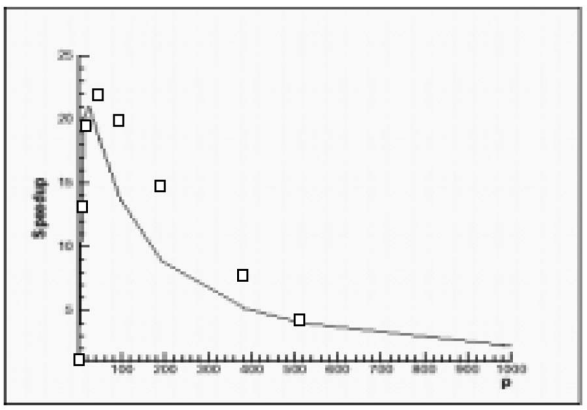

Fig. 13. Speedup of Pipelining: predicted (line) and numerical (square).

from ONR, thanks to the support of Dr. Alan Wallcraft at the Naval Research Laboratory, to improve the performance of NLOM, an ocean simulation code that adopts FACR as its kernel solver. Finally, the PPD algorithm was developed for the fast Poisson solver. This study could be further improved, but they are glad to the current achievement and would like to devote this article to Professor Roger W. Hockney in memory of his outstanding contributions to the high performance computing and parallel processing community.

This research was supported in part by the ONR under PET/Logicon and by the US National Science Foundation under grant CCR-9972251. All numerical experiments were performed on the IBM Blue Horizon operated by the San Diego Supercomputing Center (SDSC). The authors are grateful to NPACI and SDSC for providing the access to this facility, and to the referees for their thoughtful comments and suggestions on the revision of this paper.

REFERENCES

[1] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.
[2] O. Egecioglu, D. Koc, and A. Laub, "A Recursive Doubling Algorithm for Solution of Tridiagonal Systems on Hypercube Multiprocessors," *J. Computational and Applied Math.,* vol. 27, 1989.
[3] T.M. Edison and G. Erlebacher, "Implementation of a Fully-Balanced Periodic Tridiagonal Solver on a Parallel Distributed Memory Architecture," *Concurrency: Practics and Experience,* 1995.
[4] C. Ho and S. Johnsson, "Optimizing Tridiagonal Solvers for Alternating Direction Methods on Boolean Cube Multiprocessors," *SIAM J. Scientific and Statistical Computing,* vol. 11, no. 3, pp. 563-592, 1990.
[5] R.W. Hockney, "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis," *J. ACM,* vol. 12, pp. 95-113, 1965.
[6] R.W. Hockney, *Parallel Computers 2, Architecture, Programming and Algorithms.* Adam Hilger, 1988.
[7] IBM, *Parallel Engineering and Scientific Subroutine Library for AIX,* third ed. 1999, http://www.rs6000.ibm.com.
[8] D. Lawrie and A. Sameh, "The Computation and Communication Complexity of a Parallel Banded System Solver," *ACM Trans. Mathmatic Software,* vol. 10, no. 2, pp. 155-195, June 1984.
[9] J. Ortega and R. Voigt, "Solution of Partial Differential Equations on Vector and Parallel Computers," *SIAM Rev.,* pp. 149-240, 1985.
[10] J. Sherman and W. 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 Statistics,* vol. 20, p. 618, 1949.
[11] H. 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.
[12] X.-H. Sun, "Application and Accuracy of the Parallel Diagonal Dominant Algorithm," *Parallel Computing,* vol. 18, pp. 1241-1267, 1995.
[13] X.-H. Sun and S. Moitra, "Performance Comparison of a Set of Periodic and Nonperiodic Tridiagonal Solvers on SP2 and Paragon Parallel Computers," *Concurrency: Practice and Experience,* vol. 9, no. 8, pp. 781-801, 1997.
[14] X.-H. Sun and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," *IEEE Trans. Parallel Distributed Systems,* pp. 599-613, June 1994.
[15] X.-H. Sun, H. Zhang, and L. Ni, "Efficient Tridiagonal Solvers on Multicomputers," *IEEE Trans. Computers,* vol. 41, no. 3, pp. 286-296, 1992.
[16] X.-H. Sun and J. Zhu, "Performance Considerations of Shared Virtual Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 11, pp. 1185-1194, Nov. 1995.
[17] A.J. Wallcraft and D.R. Moore, "The NRL Layered Ocean Model," *Parallel Computing,* vol. 23, pp. 2227-2242, 1997.
[18] H. Wang, "A Parallel Method for Tridiagonal Equations," *ACM Trans. Math Software,* vol. 7, pp. 170-153, 1981.
[19] P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland, "A Comparison of Parallel Solvers for Diagonally Dominant and General Narrow Banded Linear Systems," *Parallel and Distributed Computing Practices,* vol. 2, pp. 385-400, 1999.

Xian-He Sun received the BS degree in mathematics from Beijing Normal University, Beijing, China, in 1982, the MS degree in mathematics, and the MS and PhD degrees in computer science from Michigan State University, East Lansing, in 1985, 1987, and 1990, respectively. He was a staff scientist at ICASE, NASA Langley Research Center and was an associate professor in the Computer Science Department at Louisiana State University, Baton Rouge. He has been serving as a faculty member of the Computer Science Department at the Illinois Institute of Technology (IIT), Chicago, since 1999. Currently, he is a professor and the director of the Scalable Computing Software Laboratory in the Computer Science Department at IIT, and is a guest faculty member at the Argonne National Laboratory. Dr. Sun's research interests include grid and cluster computing, software system, pervasive computing, performance evaluation, and scientific computing. He has published intensively in the field and his research has been supported by DoD, DoE, NASA, US National Science Foundation, and other government agencies. He is a senior member of the IEEE, a member of the ACM, New York Academy of Science, PHI KAPPA PHI, and has served and is serving as the chairman or on the program committee for a number of international conferences and workshops, including current service as the general co-chair of the Grid and Cooperative Computing (GCC03) workshop, area chair of the technical committee of the IEEE SuperComputing (SC03) conference, vice chair of the programming committee of the International Conference on Parallel Processing (ICPP04), and vice president of the Society of Chinese American Professors and Scientists. He received the ONR and ASEE Certificate of Recognition award in 1999, and received the Best Paper Award from the International Conference on Parallel Processing (ICPP01) in 2001.

Wu Zhang received the BS degree in applied mathematics from Nanjing University of Aeronautics and Astronautics, Nanjing, China, the MS degree in computational mathematics from Xian Jiaotong University, Xian, China, and the PhD degree in engineering mechanics from Northwestern Polytechnic University, Xian, China, in 1988. After graduating, he was a postdoctoral research assistant in Department of Mechanics at Peking University, 1989-1991, and a postdoctoral research assistant in Department of Mathematics at the University of North Carolina, Charlotte, 1996-1998. He has been a faculty member of the Mathematics Department at Xian Jiaotong University since 1993. During the 2000-2001, he was a visiting professor in the Department of Computer Science at the Illinois Institute of Technology, Chicago. He is currently a professor of the School of Computer Science and Engineering at Shanghai University, China. His research interests include parallel computing and grid computing, numerical algorithms and analysis, and platform software of scientific computing. He is a member of the AMS, the Chinese Mathematical Society, and the Computer Society of China, Mathematical Reviewer, and Director of System Architecture Committee, Shanghai Computer Society, China.