A Parallel Two–Level Hybrid Method for Diagonal Dominant Tridiagonal Systems

Xian-He Sun and Wu Zhang
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616
sun@cs.iit.edu

Abstract

A new method, namely the Parallel Two-Level Hybrid (PTH) method, is developed to solve tridiagonal systems on parallel computers. PTH is designed based on Parallel Diagonal Dominant (PDD) algorithm. Like PDD, PTH is highly scalable. It provides accurate solutions when PDD may not be applicable and maintains a near PDD performance when the underlying machine ensemble size is large. By controlling its two-level partition, PTH can deliver optimal performance for different machine ensemble and problem sizes. Theoretical analyses and numerical experiments indicate that PTH is significantly better than existing methods for many scientific and engineering applications.

1. Introduction

Solving tridiagonal systems is one of the key issues of numerical simulations in many scientific and engineering problems. Parallel tridiagonal algorithms have been studied extensively and remain an active research area. The algorithms of Lawrie and Sameh [5] and Wang [8] are called divided-and-conquer methods, which partition the original problem into sub-problems. The sub-problems are then solved in parallel, and the final solution is obtained by combining the solutions of the sub-problems. Later, Sun, Zhang and Ni [6,7] proposed three parallel algorithms for solving tridiagonal systems. All three algorithms are divided-and-conquer methods and are based on Sherman-Morrison matrix modification formula [1]. Two of them, the Parallel ParTition LU (PPT) algorithm and the Parallel Partition Hybrid algorithms, are fast and able to incorporate limited pivoting. The third algorithm, the Parallel Diagonal Dominant (PDD) algorithm, is designed for diagonal dominant systems. PDD algorithm is the most efficient among all three algorithms. Compared with other tridiagonal solvers, which usually require $O(\log p)$ communications, PDD algorithm has only two communications independent of the number of processors, and has a balanced workload on processors.

Most parallel tridiagonal solvers trade computation with parallelism. For solving multiple tridiagonal systems or systems with multiple right-hand-sides, pipelining can be introduced with the best sequential algorithm. 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 pipelining delay.

A novel Parallel Two-level Hybrid (PTH) method for diagonal dominant tridiagonal systems is proposed in this study based on PDD algorithm. With a two-level partition, PTH has two levels of parallelism. The first level (outer level) is based on PDD. 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. PTH algorithm overcomes the shortcoming of PDD and the pipelined method and makes use of the merits of both. It is highly efficient and more applicable than PDD. PTH provides the best performance for many applications.

2. Existing Parallel Tridiagonal Solvers

A tridiagonal system is a linear system of equations

$$Ax = d \qquad (1)$$

Where x and d are n-dimensional vectors, and $A = [a_i, b_i, c_i]$, is a tridiagonal matrix with dimension n. A is called diagonal dominant if $|b_i| > |a_i| + |c_i|$, for $0 \le i < n$. Assume that A, x and d have real coefficients, to solve (1) efficiently on parallel computers, we partition A into two parts, the main part $\tilde{A}$ and the residue ΔA.

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

Where $\tilde{A}$ is a block diagonal matrix with diagonal submatrices $A_i (i = 0,1,\Lambda\ , p-1)$. We assume that

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

$n = p \cdot m$. Thus, $A_i (i = 0,1,\Lambda\ , p-1)$ are $m \times m$ tridiagonal matrices. Let e_i be a column vector with its *i*th, $0 \le i <$ *n*-1, element being one and all the other entries are zero. We have

$$\Delta A = VE^T \quad (3)$$

where

$$V = [a_m e_m, c_{m-1} e_{m-1}, a_{2m} e_{2m}, ..., c_{(p-1)m-1} e_{(p-1)m-1}]$$

and $E = [e^T{}_{m-1}, e^T{}_m, ..., e^T{}_{(p-1)m-1}, e^T{}_{(p-1)m}]$ are $n \times 2(p-1)$ matrices. Thus, we have

$$A = \tilde{A} + VE^T \quad (4)$$

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

$$x = \tilde{A}^{-1} d - \tilde{A}^{-1} V (I + E^T \tilde{A}^{-1} V)^{-1} E^T \tilde{A}^{-1} d \quad (5)$$

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

$$Pz = (z_1, z_0, z_3, z_2, \Lambda\ , z_{2p-3}, z_{2(p-2)})^T, \quad (6)$$

From the property that $P^{-1} = P$, equation (5) becomes

$$x = \tilde{A}^{-1} d - \tilde{A}^{-1} VP (P + E^T \tilde{A}^{-1} VP)^{-1} E^T \tilde{A}^{-1} d \quad (6)$$

The intermediate matrix, $Z = (P + E^T \tilde{A}^{-1} VP)$, is a $2(p-1) \times 2(p-1)$ tridiagonal system, which leads to a reduced computation cost. The solving sequence of (6) is,

$$\tilde{A}\tilde{x} = d \quad (7)$$

$$\tilde{A} Y = VP \quad (8)$$

$$h = E^T \tilde{x} \quad (9)$$

$$Z = P + E^T Y \quad (10)$$

$$Zy = h \quad (11)$$

$$\Delta x = Yy \quad (12)$$

$$x = \tilde{x} - \Delta x \quad (13)$$

In (7) and (8), $\tilde{x}$ and *Y* are solved by the LU decomposition method. By the structure of $\tilde{A}$ and V, these are equivalent to solving

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

$$i = 0,1,\Lambda\ , p-1 \quad (14)$$

Here $x^{(i)}$ and $d^{(i)}$ are the *i*th block of $\tilde{x}$ and d, respectively, and $v^{(i)}$ and $w^{(i)}$ are possible no-zero column vectors of the *i*th row block of *Y*. Equation (14) implies that we only need to solve three linear systems of order *m* with the same LU decomposition for each $i (i = 0,1,\Lambda\ , p-1)$.

2.1. PPT: The Parallel Partition LU Algorithm

Basing on the matrix partitioning technique above, using *p* processors to solver (1), PPT consists of the following steps,

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

Step 2. Use the *LU* decomposition method to solver (14). All computations can be executed in parallel and independently on *p* processors.

Step 3. Send $\tilde{x}_0^{(i)}, \tilde{x}_{m-1}^{(i)}, v_0^{(i)}, v_{m-1}^{(i)}, w_0^{(i)}, w_{m-1}^{(i)}$ $(0 \le i \le p-1)$ to all other nodes from the *i*th node to form matrix *Z* and vector *h* (9-10) on each node. Here are throughout the subindex indicates the component of the vector.

Step 4. Use the *LU* decomposition method to solver (11) on all nodes simultaneously. Note that *Z* is a 2(*p*-1) dimensional tridiagonal matrix.

Step 5. Compute (12) and (13) in parallel on *p* processors. We have

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

$$x^{(i)} = \tilde{x}^{(i)} - \Delta x^{(i)}$$

2.2 PDD: Parallel Diagonal Dominant Algorithm

When *A* is diagonal dominant, the most interesting mathematical properties is that the off diagonal coefficients of the matrix $\tilde{A}^{-1} V$ have an exponentially decay to *0*. Therefore, the coefficients of $v_{m-1}^{(i)}, w_0^{(i)}, (0 < i < p-1)$ can be dropped within machine accuracy when *p* << *n*.

As shown in [6,7], for most diagonal dominant systems, when the subsystem size is greater than 64, the reduced *Z* with the dropping is equivalent to Z within machine accuracy for numerical computing. PDD uses the dropping for the solution and needs only two neighboring communications. The optimal and simple communication

property makes PDD algorithm an ideal algorithm for massively parallel computing.

The resulting PDD algorithm is similar with PPT except that Step 3 and Step 4 are modified as given below.

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

Step 4. Solve

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

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

2.3 The Pipelined Method for Multiple Systems

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, PDD has a non-optimal computation count. For solving multiple systems, however, optimal sequential algorithm can be used to achieve parallel processing via pipelining [2].

Pipelining works by passing the intermediate results from solving a subset of the system 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. The pipelining procedure is given below.

- In the first pass, processor 0 solves the first part of the first *L* systems whereas processors 2 to *p*-1 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 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 [6]. One round is for *LU* decomposition, one is for forward substitution, and another one is for backward substitution. Each round of computation requires a *p*-1 communication. The communication cost is high. It increases linearly with the number of processors. In addition, there is a pipeline delay of (*p*-1). The trade-off is that the best sequential method can be used. The computation is optimal. When the ensemble size is small, the pipelined method is a good candidate for parallel processing.

3. PTH: The Parallel Two-Level Hybrid Method

The Parallel Two-Level Hybrid (PTH) method is proposed in this study to combine the merits of both PDD and Pipelined methods. The basic idea of PTH is to embed an inner tridiagonal solver into PDD to form a two-level hierarchical parallelism. The base algorithm is PDD. The tridiagonal system is first partitioned based on PDD. However, the subsystems may be too small for the accuracy concern 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 super-subsystem (see Figure 1). Each super-subsystem 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 [7]. PPT introduces a good parallelism and has *log*(*p*) communication cost. For multiple tridiagonal systems, the pipelined method would be a natural candidate for the inner solver. Both of PPT and the pipelined method require global communications and otherwise efficient. When they are embedded into PDD, they are used on a small number of processors for solving the super-subsystems, 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 concern. In addition, by adjusting the size of the super-subsystem, PTH method can be scale-up and scale-down on different parallel machines based on the number of processors available. When the number of super-subsystem equals to one, PTH is the inner tridiagonal solver. When the number of super-subsystem equals to *p*, the number of processors, PTH becomes PDD. The optimal number of the super-subsystem is a function of machine parameters and error tolerance. When the pipelined algorithm is chosen as the inner solver, we call the resulting algorithm Partition Pipelined diagonal Dominant (PPD) algorithm.

Algorithm	Computation	Communication
Pipelining	$(n_1 - 1 + p)\frac{8n-7}{p}$	$(n_1 - 1 + p)(3\alpha + 12\beta)$
PDD	$n_1(17\frac{n}{p} - 14)$	$(2\alpha + 12n_1\beta)$
PPD	$[(n_1 - 1 + k)\frac{13n}{p} + n_1(\frac{4n}{p} + 4)]$	$(n_1 - 1 + k)(3\alpha + 12\beta) +$ $(2 + \log(k))(\alpha + 12n_1\beta)$

Table 1. Formulas of computation and communication

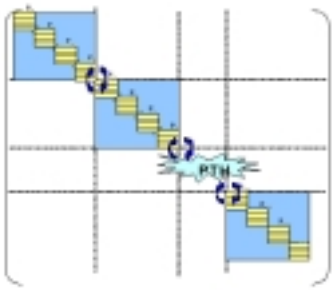

Figure 1. PTH: a Two-level hierarchical algorithm

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

Step 1. Use an accurate parallel tridiagonal solver to solve the *m* super-subsystems concurrently, each with k processors, where $p = m \cdot k$.

Step 2. Modify the solutions of *Step* 1 with *Steps* 3-5 of PDD algorithm and consider communications only between the *m* super-subsystems.

If the accurate parallel tridiagonal solver of *Step* 1 is the pipelined method, the resulting PTH algorithm is the PPD algorithm. We use PPD for our performance analysis.

Table 1 gives the computation and communication counts of the pipelined method, PDD and PPD algorithm, 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 1 stand for the number of tridiagonal systems, the order of the systems and the number of subsystems (or the number of processors) for each super-subsystem, respectively. The computation and communication costs for solving tridiagonal systems increase with the parameter n_1 for all algorithms. Compared with the pipelined algorithm, 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 1, 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. It can be determined when the application and the underlying hardware are given.

4. Numerical Experiments

Tridiagonal solvers have many applications. Here we present the experimental testing of one application, solving Poisson equations. Hockney's fast Poisson solver, the Fourier Analysis and Cyclic Reduction (FACR) algorithm [3,4], is a most accepted direct solver. We use FACR as an application of our tridiagonal solvers.

A 2-D Poisson equation, written in Cartesian coordinates, is

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

where Ω is a rectangle region and $f(x,y)$ is a given function. Let $\overline{\varphi}^k(y)$ and $\overline{f}^k(y)$ be the Fourier coefficients of the k^{th} wave number of $\varphi(x,y)$ and $f(x,y)$ Fourier expansion in x direction, 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 [4], which is called the fast Poisson solver or FACR.

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

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

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

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 having a very distinguished data distribution: each processor has a submatrix (in the order of *n/p*) from each of the n_1 tridiagonal systems, so that each processor solvers n_1 subsystems sequentially. Even *p* is close to *n*, each processor still has enough computing to do. When *p* is close to *n*, however, PDD becomes inapplicable.

4.1 Experimental Testing

Experimental testing performed on the NPACI IBM Blue-Horizon at the San Diego Supercomputing Center (SDSC). The Blue Horizon is a teraflop-scale Power3 based cluster. The machine 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. It is well suited to run straight MPI applications.

We use FACR to test PTH. Since the tridiagonal systems in FACR are multiple systems, PPD is the chosen PTH. For the experiments, we choose the wave number $n_1 = 512$ in *x*-direction and mesh points *n* = 4608 in *y*-direction. Therefore, we need to solve 512 tridiagonal systems, which each has an order of 4608. 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 4608, 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 Figure 2. As shown in Figure 4, the pipelined method performs better for small ensemble size. At *p* = 96, PDD starts to over perform of the pipelined method. The performance gap between PDD and the pipelined method becomes larger and larger, as the number of processors used becomes bigger and bigger. At *p* =512 PDD is more than ten times faster than the pipelined method. It is a very impressive result. The experimental results confirm PDD is highly scalable. For the given application, however, when *p* is greater than 96, the subsystem size is less than 48. PDD does not provide accurate results when *p* reaches 96. It is not applicable for FACR when *p* is large.

We now use PPD for FACR tridiagonal systems and choose *k*, the number of subsystems in a super-subsystem, 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 4608, 384, 192, 96, 48, 24, 12 and 9.

Figure 3 shows the runtimes of three tridiagonal solvers for the FACR tridiagonal systems. From *p* =1 to *p* =48, PPD chooses to take one super-subsystem. That means PPD is the pipelined method for *p* < *96*. Starting at

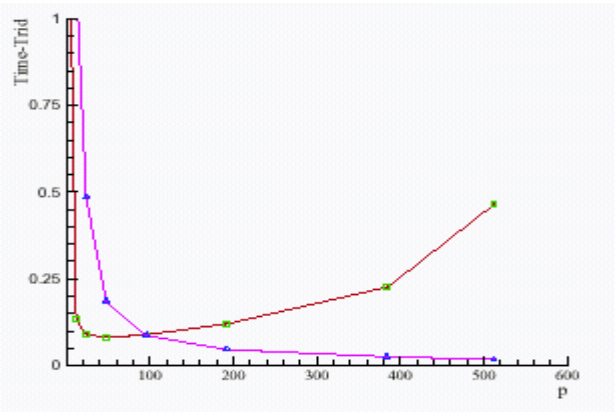

Figure 2. Tridiagonal solver runtime: Pipelining (square) and PDD (delta)

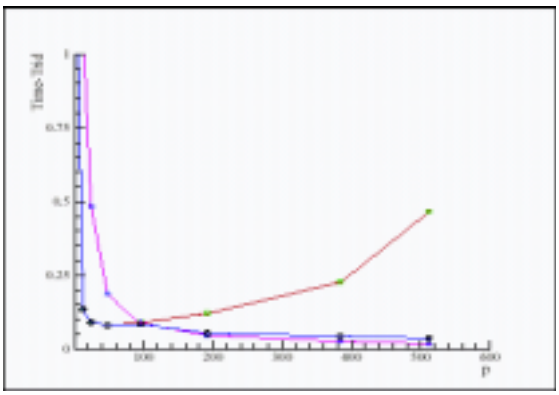

Figure 3. Tridiagonal Solver Time: Pipelining (square), PDD (delta), PPD (circle)

p = 96, PPD groups each 16 subsystems to form a super-subsystem and use 16 processors to solve each super-subsystem. 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.

Figure 4 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. 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 three different tridiagonal solvers are given in Figure 5. Please notice that the FACR-PDD implementation does not provide an accurate solution. It only can be used as a pre-conditioner for further computing.

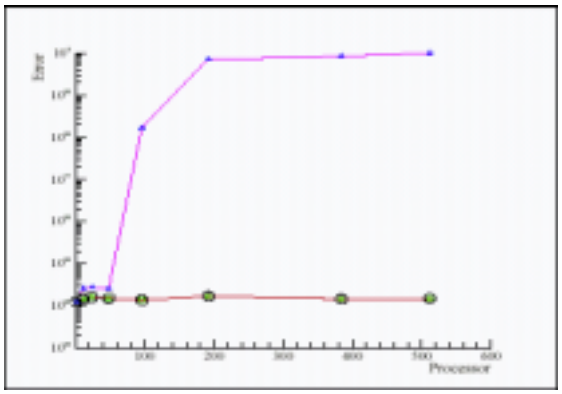

Figure 4. Accuracy: Pipelining (square), PDD (delta), PPD (circle)

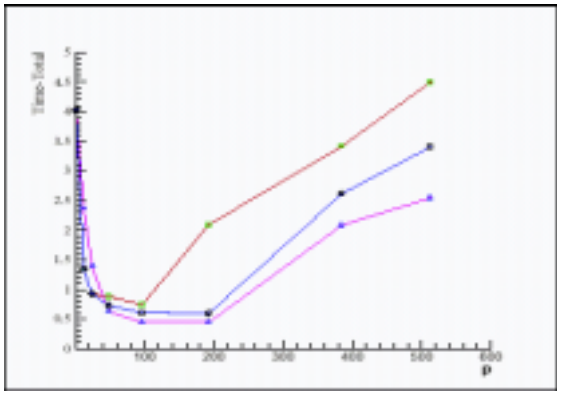

Figure 5. Total runtime: Pipelining (square), PDD (delta), PPD (circle)

5. Conclusions

PDD algorithm is an efficient algorithm for diagonal dominant tridiagonal systems. Partitioned via the Sherman and Morrison formula [1], PDD can drop communications 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. The inner solver is open and can be application specific. When the inner solver is the pipelined method, 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 [3,4]. 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 PTH method. The potential of PTH also should be further investigated.

Acknowledgements

This research was supported in part by ONR under PET/Logicon and by NSF under NSF 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.

References

[1] I. Duff, A. Erisman and J. Reid, *Direct Methods for Sparse Matrices* (Clarendon Press, Oxford, 1986)

[2] 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

[3] R.W. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, *J. Assoc. Comput. Mach*., 12 (1965), 95-113

[4] R.W. Hockney, *Parallel computers 2, Architecture, Programming and algorithms*, Adam Hilger, 1988

[5] D. Lawrie and A. Sameh, The computation and communication complexity of a parallel banded system solver, *ACM Trans. Math. Soft.* 10 (2), June 1984, 155-195

[6] X.-H. Sun, Application and accuracy of the parallel diagonal dominant algorithm, *Parallel Computing*, 18(1995), 1241-1267

[7] X.-H. Sun, H. Zhang and L. Ni, Efficient tridiagonal solvers on multicomputers, *IEEE Trans. Comput*., 41(3) (1992), 286-296

[8] H. Wang, A parallel method for tridiagonal equations, *ACM Trans. Math Software*, 7(1981), 170-153