Journal of Computational and Applied Mathematics 36 (1991) 209–226
North-Holland

A parallel alternating direction implicit preconditioning method *

Hong Jiang and Yau Shu Wong
Department of Mathematics, University of Alberta, Edmonton, Alberta, Canada T6G 2G1

Received 25 May 1990
Revised 3 December 1990

Abstract

Jiang, H. and Y.S. Wong, A parallel alternating direction implicit preconditioning method, Journal of Computational and Applied Mathematics 36 (1991) 209–226.

The alternating direction implicit (ADI) iterative method is an efficient iterative method to solve systems of linear equations due to its extremely fast convergence. The ADI method has also been used successfully as a preconditioner in some other iterative methods, such as the preconditioned conjugate gradient. In this paper a parallel algorithm for the ADI preconditioning is proposed. In this algorithm, several steps of the ADI iteration are computed simultaneously. This means that several tridiagonal systems that are traditionally solved sequentially are now solved concurrently. The high performance of this algorithm is achieved by increasing the degree of parallelism and reducing memory contention. The algorithm can easily be implemented in a multiprocessor architecture. Experiments have been conducted on the Myrias SPS-2 computer with 64 processors and good performance of this algorithm is observed.

Keywords: ADI methods, preconditioners, parallel iterative methods.

1. Introduction

The alternating direction implicit (ADI) iterative method, due to Peaceman and Rachford [27] and Douglas and Rachford [9], is an efficient iterative technique to solve systems of linear equations resulting from discretizations of partial differential equations. For some model problems, an extremely rapid convergence rate can be realized by the proper choice of iteration parameters. Over the past thirty-five years, theoretical and experimental results have demonstrated that ADI is also an effective method for solving a large class of elliptic problems. Dyksen [11,12] applied Tensor Product Generalized ADI (TPGADI) methods to an entire class of separable elliptic problems. The ADI method, when used as a preconditioner in conjunction with another outer iterative scheme, has also been proved to be effective for solving more general problems. It has been shown [7,25] that an arbitrary second-order self-adjoint elliptic partial

* The research was supported by the Natural Sciences and Engineering Research Council of Canada.

differential equation can be preconditioned by the Laplacian on the same grid with appropriate boundary conditions. Thus the solution of the preconditioned system involves solutions of equations with the Laplace operator. The latter can be solved rapidly by the ADI method. The first published discussion of combining several iterations of ADI with an outer iterative method was [10]. Wachspress [33] used the ADI applied to a model problem as a preconditioner for conjugate gradients (CG) applied to diffusion equations with variable diffusion coefficients. Subsequently, the ADI preconditioned CG has been applied to a variety of problems, cf. [2,14,16–18]. Adams [1] analyzed convergence rates of the preconditioned CG method where the preconditioners are based on several iterations of some iterative methods.

One step of the ADI iteration involves two sweeps of mesh in the coordinate directions. In the case of using finite differences, each sweep corresponds to solving a tridiagonal system. A difficulty in parallelizing the ADI method is that classical algorithms for solving tridiagonal systems are sequential in nature and are unsatisfactory for parallel computations. There has been a considerable amount of work to solve tridiagonal systems on parallel computers, see, for example, [23,26,28,29,31]. Another potential problem in parallel computers is to arrange the storage so that transfers between sweeps are minimized. Since each sweep of ADI constitutes a set of independent tridiagonal systems, the solution is parallelizable. However, the transfers between sweeps involve expensive data communications, which causes storage contention and access conflicts on many parallel architectures [22,26]. Johnsson et al. [22] proposed a few implementations of the ADI method on multiprocessors. They described some data structures and algorithms that efficiently use some multiprocessor configurations. Their complexity analysis shows that the ADI method can be made highly efficient on parallel architectures by using some parallel algorithms for solving tridiagonal systems.

Although parallelism is introduced in each step of the ADI method, the successive steps are computed sequentially. The fact that the next step can only be computed when the previous step is completed constitutes a severe bottleneck in achieving high performance on parallel architectures. This is reflected in the large overhead cost for initializing parallel process and excessive memory references. Some degree of parallelism can be achieved by pipelining techniques where the next iteration can begin as soon as some subsets of the unknowns from the previous iteration are computed. To achieve maximum performance on parallel architectures, however, it is desirable to perform several iterative steps of the ADI iteration simultaneously so as to increase the degree of parallelism and reduce the memory contention. Chronopoulos and Gear [4,5] are the first to introduce the concept of s-step iterative methods in which s consecutive steps of a one-step method are performed simultaneously. In [4,5] they derived s-step conjugate gradient methods for symmetric and positive definite linear systems.

In this paper, we propose an algorithm for the ADI method where k step consecutive iterations are computed simultaneously. For example, the k tridiagonal systems that are traditionally solved consecutively are now solved concurrently. We will show by complexity analysis and experiments that the new algorithm can achieve high performance on parallel computers. We note that the present algorithm is different from the m-step preconditioners of Adams [1]. In the latter, m steps of some stationary iterative methods are concerned.

We consider the solution of the partial differential equation

$$-\frac{\partial}{\partial x}\left(a(x, y)\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial y}\left(b(x, y)\frac{\partial u}{\partial y}\right)=f(x, y), \quad (x, y)\in\Omega,$$

$$u=0, \quad (x, y)\in\partial\Omega,$$

where $a > 0$, $b > 0$ and $\Omega = [0, 1] \times [0, 1]$. The numerical solution of the problem may be obtained by the usual procedure of placing a set of mesh points on Ω. This results in a system of linear equations

$$Au = (H + V)u = f. \tag{1.1}$$

For separable problems, it is possible to write H and V in the tensor product form

$$H = A_1 \otimes B_2, \qquad V = B_1 \otimes A_2, \tag{1.2}$$

where A_1, A_2, B_1, B_2 are matrices of order $n \times n$, and u, f are column vectors of size n^2. We assume that A is positive definite, and, furthermore, A_1^{-1}, A_2^{-1} exist and $A_1^{-1}B_1$ and $A_2^{-1}B_2$ are also positive definite. For example, when $a = b =$ constant, the 5-point scheme results in

$$A_1 = A_2 = I, \qquad B_1 = B_2 = \text{trid}(-\tfrac{1}{2}, 1, -\tfrac{1}{2}), \tag{1.3}$$

and the 9-point scheme yields

$$A_1 = A_2 = \text{trid}(\tfrac{1}{10}, 1, \tfrac{1}{10}), \qquad B_1 = B_2 = \text{trid}(-\tfrac{1}{2}, 1, -\tfrac{1}{2}). \tag{1.4}$$

In this paper, we will consider (1.1) in the tensor product form. The tensor product form, which was first studied for the ADI method in [24] and recently in [11], has advantages in both analysis and applications. A detailed discussion of tensor products of linear spaces and operators can be found in [19] and a summary of some properties of tensor products of matrices which are useful in the analysis of the ADI method is given in [11].

An efficient way to solve (1.1) is to apply an iterative method to the system

$$M^{-1}Au = M^{-1}f, \tag{1.5}$$

in which M^{-1} is a preconditioner. A k-term ADI preconditioner is obtained if $M^{-1} = M_k^{-1}$ is defined as the k-iteration ADI operator applied to (1.1), i.e., for any f, $u^k = M_k^{-1}f$ is the kth approximation of the solution of (1.1) by the ADI method.

We describe an algorithm to compute the preconditioning operator M_k^{-1} so that the maximum degree of parallelism is achieved. We will not be concerned with outer iterations, i.e., the iterative methods for solution of (1.2), although in our experiments the conjugate gradient method is used. Discussions of implementations of the CG method on parallel computers can be found in [5,6].

2. The ADI iterative methods

The Peaceman–Rachford alternating direction implicit iterative method applied to (1.1) is the implicit process defined by

$$(H + \rho_{j+1}I)u^{j+1/2} = f - (V - \rho_{j+1}I)u^j, \tag{2.1}$$

$$(V + \rho_{j+1}I)u^{j+1} = f - (H - \rho_{j+1}I)u^{j+1/2}, \tag{2.2}$$

where u^0 is an arbitrary initial vector, and $\{\rho_j\}$, $j = 1, 2, \ldots$, is a set of positive acceleration parameters. If H, V commute, it is possible to express the error of the kth iteration of the ADI

process in terms of eigenvalues and eigenvectors of H and V. Let $e^k = u^k - u^*$ be the error of the kth iteration; then the norm of e^k satisfies

$$\| e^k \| \leqslant \max_{\substack{\lambda_i \in \sigma(H) \\ \mu_i \in \sigma(V)}} \left| \prod_{j=1}^{k} \frac{(\lambda_i - \rho_j)(\mu_i - \rho_j)}{(\lambda_i + \rho_j)(\mu_i + \rho_j)} \right| \| e^0 \|, \tag{2.3}$$

where $\sigma(H)$, $\sigma(V)$ are spectra of H and V, respectively.

The symmetric formulas (2.1) and (2.2) are suitable for analysis, but they are not computationally efficient. Formulas (2.1) and (2.2) can be rewritten so that the multiplications by V and H on the right-hand side are not necessary. Solving for u^{j+1} from (2.1) and (2.2), the result is

$$u^{j+1} = (V + \rho_{j+1} I)^{-1} \Big\{ f - (H - \rho_{j+1} I) \Big[(H + \rho_{j+1} I)^{-1} \big[f - (V - \rho_{j+1} I) u^j \big] \Big] \Big\}.$$

From this, it follows that

$$\begin{aligned}
u^{j+1} &= (V + \rho_{j+1} I)^{-1} \Big\{ (V - \rho_{j+1} I) u^j + \big[f - (V - \rho_{j+1} I) u^j \big] \\
&\qquad - (H - \rho_{j+1} I) \Big[(H + \rho_{j+1} I)^{-1} \big[f - (V - \rho_{j+1} I) u^j \big] \Big] \Big\} \\
&= (V + \rho_{j+1} I)^{-1} \Big\{ (V - \rho_{j+1} I) u^j + \big[I - (H - \rho_{j+1} I)(H + \rho_{j+1} I)^{-1} \big] \\
&\qquad \times \big[f - (V - \rho_{j+1} I) u^j \big] \Big\} \\
&= (V + \rho_{j+1} I)^{-1} \Big\{ (V - \rho_{j+1} I) u^j \\
&\qquad + \big[(H + \rho_{j+1} I)(H + \rho_{j+1} I)^{-1} - (H - \rho_{j+1} I)(H + \rho_{j+1} I)^{-1} \big] \\
&\qquad \times \big[f - (V - \rho_{j+1} I) u^j \big] \Big\} \\
&= (V + \rho_{j+1} I)^{-1} \Big\{ (V - \rho_{j+1} I) u^j + \\
&\qquad 2\rho_{j+1} (H + \rho_{j+1} I)^{-1} \big[f - (V - \rho_{j+1} I) u^j \big] \Big\},
\end{aligned}$$

which we can write as

$$u^{j+1} = (V + \rho_{j+1} I)^{-1} \Big\{ \tilde{u}^j + 2\rho_{j+1} (H + \rho_{j+1} I)^{-1} (f - \tilde{u}^j) \Big\},$$

where

$$\tilde{u}^j = (V - \rho_{j+1} I) u^j.$$

Moreover, note that

$$\begin{aligned}
\tilde{u}^j &= (V - \rho_{j+1} I) u^j \\
&= (V - \rho_{j+1} I)(V + \rho_j I)^{-1} \\
&\quad \times \Big\{ (V - \rho_j I) u^{j-1} + 2\rho_j (H + \rho_j I)^{-1} \big[f - (V - \rho_j I) u^{j-1} \big] \Big\}
\end{aligned}$$

$$= (V-\rho_{j+1}I)(V+\rho_j I)^{-1}\{\tilde{u}^{j-1}+2\rho_j(H+\rho_j I)^{-1}(f-\tilde{u}^{j-1})\}$$
$$= [(V+\rho_j I)-(\rho_{j+1}+\rho_j)I](V+\rho_j I)^{-1}\{\tilde{u}^{j-1}+2\rho_j(H+\rho_j I)^{-1}(f-\tilde{u}^{j-1})\}$$
$$= [I-(\rho_{j+1}+\rho_j)(V+\rho_j I)^{-1}]\{\tilde{u}^{j-1}+2\rho_j(H+\rho_j I)^{-1}(f-\tilde{u}^{j-1})\}.$$

Thus the kth iteration u^k can be computed by the following algorithm.

Algorithm 2.1. Set $\tilde{u}^0=(V-\rho_1 I)u^0$. For $j=1,\ldots,k-1$, compute

$$\tilde{u}^j=[I-(\rho_{j+1}+\rho_j)(V+\rho_j I)^{-1}][\tilde{u}^{j-1}+2\rho_j(H+\rho_j I)^{-1}(f-\tilde{u}^{j-1})].$$

Set $u^k=(V+\rho_k I)^{-1}[\tilde{u}^{k-1}+2\rho_k(H+\rho_k I)^{-1}(f-\tilde{u}^{k-1})]$.

In general, H, V as defined in (1.2) may not commute, but (1.1) can be recast into a system where the commutativity holds. We assume that A_1 and A_2 are nonsingular. When (1.1) is multiplied by $A_1^{-1}\otimes A_2^{-1}$ from the left, the result is

$$(\tilde{H}+\tilde{V})u=\tilde{f}, \tag{2.4}$$

where

$$\tilde{H}=I\otimes A_2^{-1}B_2, \qquad \tilde{V}=A_1^{-1}B_1\otimes I, \tag{2.5}$$

and $\tilde{f}=(A_1^{-1}\otimes A_2^{-1})f$. It is clear that $\tilde{H}$, $\tilde{V}$ defined in (2.5) commute. The result of applying Algorithm 2.1 to (2.4) is the next algorithm.

Algorithm 2.2. Set $\tilde{u}^0=(A_1^{-1}B_1\otimes I-\rho_1 I)u^0$. For $j=1,\ldots,k-1$, compute

$$\tilde{u}^j=[I-(\rho_{j+1}+\rho_j)(A_1^{-1}B_1\otimes I+\rho_j I)^{-1}]$$
$$\times[\tilde{u}^{j-1}+2\rho_j(I\otimes A_2^{-1}B_2+\rho_j I)^{-1}(\tilde{f}-\tilde{u}^{j-1})]. \tag{2.6}$$

Set $u^k=(A_1^{-1}B_1\otimes I+\rho_k I)^{-1}[\tilde{u}^{k-1}+2\rho_k(I\otimes A_2^{-1}B_2+\rho_k I)^{-1}(\tilde{f}-\tilde{u}^{k-1})]$.

In the above formulas $\tilde{f}=(A_1\otimes A_2)^{-1}f$.

Each iteration in (2.6) can be computed in the following steps:

(1) $$r=\tilde{f}-\tilde{u}^{j-1}, \tag{2.7a}$$

(2) $$d=(I\otimes A_2^{-1}B_2+\rho_j I)^{-1}r, \tag{2.7b}$$

(3) $$r=\tilde{u}^{j-1}+2\rho_j d, \tag{2.7c}$$

(4) $$d=(A_1^{-1}B_1\otimes I+\rho_j I)^{-1}r, \tag{2.7d}$$

(5) $$\tilde{u}^j=r-(\rho_{j+1}+\rho_j)d. \tag{2.7e}$$

Mathematically, Algorithm 2.2 is equivalent to the tensor product generalized ADI (TPGADI) of [11]:

$$[A_1\otimes(B_2+\rho_{j+1}A_2)]u^{j+1/2}=f-[(B_1-\rho_{j+1}A_1)\otimes A_2]u^j, \tag{2.8}$$
$$[(B_1+\rho_{j+1}A_1)\otimes A_2]u^{j+1}=f-[A_1\otimes(B_2-\rho_{j+1}A_2)]u^{j+1/2}. \tag{2.9}$$

If the ADI method is used as a preconditioner, then only the last iteration u^k is needed. In this case, Algorithm 2.2 is computationally more efficient than (2.8) and (2.9) for the following reasons. First, each half step in (2.8) or (2.9) requires matrix by vector multiplications on the right-hand side, while only vector updates are involved in (2.7). Second, though both methods require the solution of tridiagonal systems at each half step, (2.7b) or (2.7d) require less computation work. To see this, (2.7b) is written as

$$\left(I \otimes A_2^{-1}B_2 + \rho_j I\right)d = r$$

or

$$\left[I \otimes \left(B_2 + \rho_{j+1}A_2\right)\right]d = (I \otimes A_2)r. \tag{2.10}$$

This requires computing a matrix by vector multiplication and solving one tridiagonal system. On the other hand, (2.8) involves solving the system

$$\left[A_1 \otimes \left(B_2 + \rho_{j+1}A_2\right)\right]u = b \quad \text{or} \quad \left[A_1 \otimes I\right]\left[I \otimes \left(B_2 + \rho_{j+1}A_2\right)\right]u = b,$$

which requires solving two tridiagonal systems. This clearly shows that Algorithm 2.2 not only has fewer computational operations but also is more suitable for parallel computers than (2.8) and (2.9) because the solution of tridiagonal systems is more difficult for parallel computations than the matrix by vector multiplications.

The steps in (2.7) can be efficiently implemented on parallel computers; for details we refer to [20,22]. It was shown in those papers that the computations of (2.7) can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems.

3. A parallel ADI preconditioner

When the k-term ADI iterative method is used as a preconditioner, the preconditioning operator $M = M_k$ is defined in the following way.

For any given vector f, the action of M_k^{-1} on f is defined as

$$M_k^{-1}f = u^k,$$

where u^k is the kth iteration of the ADI method given in Algorithm 2.2 applied to the system

$$(A_1 \otimes B_2 + B_1 \otimes A_2)u = f.$$

The operator M_k thus defined is clearly dependent upon the parameters $\{\rho_j\}$ and the initial u^0, as well as k. At first glance, in order to compute $u^k = M_k^{-1}f$, the vectors

$$\tilde{u}^0, \tilde{u}^1, \ldots, \tilde{u}^{k-1}, u^k$$

must be computed sequentially, which is done in the traditional ADI method.

These sequential computations can be avoided if we assume $\rho_i \neq \rho_j$ for all $i \neq j$ and $u^0 = 0$ as we will see in the following theorem.

Theorem 3.1. *If $\rho_i \neq \rho_j$ for all $i \neq j$ and $u^0 = 0$, then*

$$M_k^{-1} = \sum_{j=1}^{k}\left[I \otimes \left(B_2 + \rho_j A_2\right)\right]^{-1}\sum_{i=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\left[\left(B_1 + \rho_i A_1\right) \otimes I\right]^{-1},$$

where $\gamma_{ij}^{\mathrm{ADI}} = \gamma_{ij}^{\mathrm{ADI}}(\rho)$ *are some parameters defined in the following. Let* $\gamma_{11}^{1} = 2\rho_1$. *For* $p = 2, \ldots, k$, *compute*

$$\gamma_{ij}^{p} = \frac{\rho_p + \rho_i}{\rho_p - \rho_i}\frac{\rho_p + \rho_j}{\rho_p - \rho_j}\gamma_{ij}^{p-1}, \qquad 1 \leqslant i,\ j < p,$$

$$\gamma_{ip}^{p} = -\frac{2\rho_p(\rho_p + \rho_i)}{\rho_p - \rho_i}\sum_{j=1}^{p-1}\frac{\gamma_{ij}^{p-1}}{\rho_p - \rho_j}, \qquad 1 \leqslant i < p,$$

$$\gamma_{pj}^{p} = \gamma_{jp}^{p}, \qquad 1 \leqslant j < p, \tag{3.1}$$

$$\gamma_{pp}^{p} = 2\rho_p\left[1 + 2\rho_p\sum_{i=1}^{p-1}\sum_{j=1}^{p-1}\frac{\gamma_{ij}^{p-1}}{(\rho_p - \rho_i)(\rho_p - \rho_j)}\right].$$

Let $\gamma_{ij}^{\mathrm{ADI}} = \gamma_{ij}^{k}$, $1 \leqslant i,\ j \leqslant k$.

Proof. From Algorithm 2.2, it can be shown that

$$u^k = 2\rho_k\left(\tilde{V} + \rho_k I\right)^{-1}\left(\tilde{H} + \rho_k I\right)^{-1}\tilde{f} + \left(\tilde{V} + \rho_k I\right)^{-1}\left(\tilde{H} + \rho_k I\right)^{-1}\left(\tilde{V} - \rho_k I\right)\left(\tilde{H} - \rho_k I\right)u^{k-1},$$

where $\tilde{H} = I \otimes A_2^{-1}B_2$, $\tilde{V} = A_1^{-1}B_1^{-1} \otimes I$ and $\tilde{f} = (A_1^{-1} \otimes A_2^{-1})f$. By using the commutativity of $\tilde{H}$ and $\tilde{V}$, we obtain

$$u^k = 2\rho_k\left(\tilde{V} + \rho_k I\right)^{-1}\left(\tilde{H} + \rho_k I\right)^{-1}\tilde{f} + \left[I - 2\rho_k\left(\tilde{V} + \rho_k I\right)^{-1}\right]\left[I - 2\rho_k\left(\tilde{H} + \rho_k I\right)^{-1}\right]u^{k-1}.$$

By the assumption that $u^0 = 0$, it is easy to see that u^k is a linear combination of vectors of the form

$$\prod_j\left(\tilde{V} + \rho_j I\right)^{-1}\prod_i\left(\tilde{H} + \rho_i I\right)^{-1}\tilde{f}.$$

Furthermore, since $\rho_i \neq \rho_j$ if $i \neq j$, we have

$$\left(\tilde{V} + \rho_j I\right)^{-1}\left(\tilde{V} + \rho_i I\right)^{-1} = c_1\left(\tilde{V} + \rho_j I\right)^{-1} + c_2\left(\tilde{V} + \rho_i I\right)^{-1}$$

for some constants c_1 and c_2. A similar result holds for $\tilde{H}$. This shows that

$$u^k = \sum_{j=1}^{k}\left(I \otimes A_2^{-1}B_2 + \rho_j I\right)^{-1}\sum_{i=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\left(A_1^{-1}B_1 \otimes I + \rho_i I\right)^{-1}\tilde{f}. \tag{3.2}$$

The formula for $\gamma_{ij}^{\mathrm{ADI}}$ can be obtained by induction. It is easy to verify that (3.2) is equivalent to

$$u^k = \sum_{j=1}^{k}\left[I \otimes \left(B_2 + \rho_j A_2\right)\right]^{-1}\sum_{i=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\left[\left(B_1 + \rho_i A_1\right) \otimes I\right]^{-1}f. \tag{3.3}$$

This completes the proof. □

In general, for any given sets of $\{\gamma_{ij}\}$ and $\{\rho_j\}$, $\rho_j > 0$, we can define a k-term ADI preconditioning operator as

$$M_k^{-1} = \sum_{j=1}^{k} \left[I \otimes (B_2 + \rho_j A_2) \right]^{-1} \sum_{i=1}^{k} \gamma_{ij} \left[(B_1 + \rho_i A_1) \otimes I \right]^{-1}. \tag{3.4}$$

With this definition, the classical ADI method of Peaceman and Rachford is a special case of (3.4) with $A_1 = A_2 = I$ and some particular choice of $\{\gamma_{ij}\}$.

When (3.4) is used in computing $u^k = M_k^{-1} f$, the maximum degree of parallelism can be achieved. If we assume memory space for k vectors $d_1, d_2, \ldots, d_k$ is available, then $M_k^{-1} f$ can be computed in the following algorithm.

Algorithm 3.2.

(1) Compute in parallel for $i = 1, \ldots, k$,

$$d_i = \left[(B_1 + \rho_i A_1) \otimes I \right]^{-1} f. \tag{3.5a}$$

(2) Compute in parallel for $j = 1, \ldots, k$,

$$d_j \leftarrow \sum_{i=1}^{k} \gamma_{ij} d_i. \tag{3.5b}$$

(3) Compute in parallel for $j = 1, \ldots, k$,

$$d_j \leftarrow \left[I \otimes (B_2 + \rho_j A_2) \right]^{-1} d_j. \tag{3.5c}$$

(4) Compute

$$u^k = \sum_{j=1}^{k} d_j. \tag{3.5d}$$

Note that the work in each of the formulas (3.5a)–(3.5d) can also be carried out in parallel.

We compare Algorithm 3.2 with Algorithm 2.2. To compute $M_k^{-1} f$ in Algorithm 2.2, roughly k repeated executions of (2.7a)–(2.7e) are needed. The work in computing $M_k^{-1} f$ is therefore $2k$ tridiagonal systems and $3k$ vector additions. The work in Algorithm 3.2 is dominated by $2k$ tridiagonal systems and $k(k+1)$ vector additions. Thus the extra work for Algorithm 3.2, which is represented in (3.5b), is $2(k(k-2)N)$ floating-point operations, where N is the size of the vectors. The detailed operation counts are given in Table 1. This extra work is a small price paid to improve the parallel properties of the ADI method. The advantages of (3.5) over (2.7) in parallel computations are obvious. Not only the k executions of solving tridiagonal systems can

Table 1
Operation counts for computing $M_k^{-1} f$

Operations	$r \leftarrow r + \rho d$ *	$r \leftarrow (I \otimes A) r$	$r \leftarrow (I \otimes B)^{-1} r$
Algorithm 2.2	$5k-1$	$2k$	$2k$
Algorithm 3.2	k^2+3k	0	$2k$

* Including $B + \rho A$, where A, B are $\sqrt{N} \times \sqrt{N}$ matrices.

be done simultaneously, but also the vector updates in (2.7) are replaced by linear combinations. Linear combinations of the form (3.5b) or (3.5d) have lower cache-miss ratio than the vector updates of the form (2.7a), (2.7c) or (2.7e) [4,5,15]. Operations with lower cache-miss ratio require fewer memory references, reducing memory contentions. The extra work in (3.5b) is not significant if a large number of processors are available, because the linear combination (3.5b) has an arithmetic complexity of $O(\log_2 k)$.

In the case where $A_1 \neq I$, or $A_2 \neq I$, as in the 9-point scheme for the Laplace equation, (2.7) requires even more computational work. In (2.7b), the solution is obtained by solving the system (2.10). Solving (2.10) is more costly than solving (3.5c) because of an additional matrix by vector multiplication on the right-hand side of (2.10). In fact, when (3.5) is used, the work to compute $M_k^{-1} f$ for the 5-point scheme and the 9-point scheme are about the same, because the operations for computing $B_1 + \rho_i A_1$ may be regarded as the same whether A_1 is the identity or a tridiagonal matrix.

Instead of a preconditioner, Algorithm 3.2 can also be regarded as a k-term ADI iterative method to apply directly to (1.1). In other words, M_k^{-1} can be applied to (1.1) repetitively to yield successive approximations $u^m = (M_k^{-1})^m f$. This amounts to restarting the ADI algorithm after every k steps.

4. Acceleration parameters

The set of optimum parameters $\{\rho_j\}$, $j = 1, \ldots, k$, for the ADI method (2.1) and (2.2), or equivalently Algorithm 2.2, can be obtained by minimizing the bound of error e^k of (2.3). Let a, b be the lower and upper bounds of $\sigma(A_1^{-1}B_1)$ and $\sigma(A_2^{-1}B_2)$, i.e., assume

$$\sigma\left(A_1^{-1}B_1\right) \subset [a, b], \qquad \sigma\left(A_2^{-1}B_2\right) \subset [a, b].$$

Then the optimum parameters $\{\rho_j\}$ are determined by minimizing the expression

$$\max_{\substack{\lambda \in [a,b] \\ \mu \in [a,b]}} \left| \prod_{j=1}^{k} \frac{(\lambda - \rho_j)(\mu - \rho_j)}{(\lambda + \rho_j)(\mu + \rho_j)} \right|. \tag{4.1}$$

Wachspress [32] described a procedure to solve this minimax problem in which the unique optimum parameters $\{\rho_j^{\mathrm{op}}\}$ are found in terms of elliptic functions.

Now we consider the acceleration parameters $\{\gamma_{ij}\}$ and $\{\rho_j\}$ in (3.4). One criterion of determining these parameters is that the preconditioned system (1.5) is better conditioned than (1.1), which in general means $\| M_k^{-1}A - I \|_2$ is small. If M_k^{-1} is given by (3.4), we have

$$\begin{aligned} M_k^{-1}A &= \sum_{j=1}^{k} \sum_{i=1}^{k} \gamma_{ij} \left[I \otimes \left(B_2 + \rho_j A_2 \right) \right]^{-1} \left[\left(B_1 + \rho_i A_1 \right) \otimes I \right]^{-1} \left[A_1 \otimes B_2 + B_1 \otimes A_2 \right] \\ &= \sum_{j=1}^{k} \sum_{i=1}^{k} \gamma_{ij} \left[I \otimes \left(A_2^{-1}B_2 + \rho_j I \right) \right]^{-1} \left[\left(A_1^{-1}B_1 + \rho_i I \right) \otimes I \right]^{-1} \\ &\quad \times \left[I \otimes A_2^{-1}B_2 + A_1^{-1}B_1 \otimes I \right]. \end{aligned} \tag{4.2}$$

If $\lambda \in \sigma(A_1^{-1}B_1)$ and $\mu \in \sigma(A_2^{-1}B_2)$, then an eigenvalue of $M_k^{-1}A$ is given by

$$\sum_{j=1}^{k}\sum_{i=1}^{k}\gamma_{ij}\frac{\lambda+\mu}{(\lambda+\rho_i)(\mu+\rho_j)}. \tag{4.3}$$

It follows that $\| M_k^{-1}A - I \|_2$ is bounded by

$$\omega_k(\gamma, \rho) \overset{\text{def}}{=} \max_{\substack{\lambda\in[a,b]\\ \mu\in[a,b]}} \left| \sum_{i=1}^{k}\sum_{j=1}^{k}\gamma_{ij}\frac{\lambda+\mu}{(\lambda+\rho_i)(\mu+\rho_j)} - 1 \right|, \tag{4.4}$$

and the condition number of $M_k^{-1}A$ is bounded by

$$\frac{1+\omega_k}{1-\omega_k},$$

if $\omega_k < 1$. Thus the optimum parameters $\{\gamma_{ij}\}, \{\rho_j\}$ should be chosen to minimize the expression (4.4). We assume these optimum parameters exist, and they are denoted by $\{\gamma_{ij}^*\}, \{\rho_j^*\}$, i.e.,

$$\omega_k(\gamma^*, \rho^*) \leqslant \omega_k(\gamma, \rho), \quad \text{for all } \{\gamma_{ij}\}, \{\rho_j\}. \tag{4.5}$$

Finding these optimum parameters amounts to solving a nonlinear minimax problem, and efficient numerical methods such as the Remez algorithms [3,34] are available. However, the solution of nonlinear best approximations are more difficult than linear approximations from both theoretic and practical viewpoint. A linear problem results if $\{\rho_j\}$ are held fixed and the optimum parameters $\{\gamma_{ij}^{\text{op}}\} = \{\gamma_{ij}^{\text{op}}(\rho)\}$ are sought such that

$$\omega_k(\gamma^{\text{op}}(\rho), \rho) \leqslant \omega_k(\gamma, \rho), \quad \text{for all } \{\gamma_{ij}\}.$$

Clearly, for any $\{\rho_j\}$, with $\rho_j > 0$, $\rho_i \neq \rho_j$ if $i \neq j$, the following inequalities hold:

$$\omega_k(\gamma^*, \rho^*) \leqslant \omega_k(\gamma^{\text{op}}(\rho), \rho) \leqslant \omega_k(\gamma^{\text{ADI}}(\rho), \rho), \tag{4.6}$$

where $\{\gamma_{ij}^{\text{ADI}}\}$ are given by (3.1). In general, for a given set $\{\rho_j\}$, $\{\gamma_{ij}^{\text{op}}(\rho)\}$ need to be computed by the Remez algorithm. However, the following result shows that if $\{\rho_j\} = \{\rho_j^{\text{op}}\}$, the $\{\gamma_{ij}^{\text{op}}(\rho^{\text{op}})\}$ can easily be computed from (3.1).

Theorem 4.1. *If* $\{\rho_j^{\text{op}}\}$ *are the optimum parameters which minimize the expression* (4.1), *then*

$$\omega_k(\gamma^{\text{op}}(\rho^{\text{op}}), \rho^{\text{op}}) = \omega_k(\gamma^{\text{ADI}}(\rho^{\text{op}}), \rho^{\text{op}}).$$

Proof. In view of (4.6), we need to show

$$\omega_k(\gamma^{\text{ADI}}(\rho^{\text{op}}), \rho^{\text{op}}) \leqslant \omega_k(\gamma, \rho^{\text{op}}), \quad \text{for all } \{\gamma_{ij}\}. \tag{4.7}$$

To prove this, we note that

$$\sum_{i=1}^{k}\sum_{j=1}^{k}\gamma_{ij}^{\text{ADI}}(\rho^{\text{op}})\frac{\lambda+\mu}{(\lambda+\rho_i^{\text{op}})(\mu+\rho_j^{\text{op}})} - 1 = \prod_{j=1}^{k}\frac{(\lambda-\rho_j^{\text{op}})(\mu-\rho_j^{\text{op}})}{(\lambda+\rho_j^{\text{op}})(\mu+\rho_j^{\text{op}})} = P(\lambda)P(\mu), \tag{4.8}$$

where

$$P(\lambda)=\prod_{j=1}^{k}\frac{\lambda-\rho_j^{\mathrm{op}}}{\lambda+\rho_j^{\mathrm{op}}}.$$

It is shown in [32] that $P(\lambda)$ has alternance properties, i.e., $P(\lambda)$ assumes its maximum magnitude with alternating sign $k+1$ times in $[a, b]$.

For a given $\mu_0 \in [a, b]$, the function

$$g(\lambda)=\sum_{i=1}^{k}\sum_{j=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\frac{\lambda+\mu_0}{(\lambda+\rho_i^{\mathrm{op}})(\mu_0+\rho_j^{\mathrm{op}})}-1=\sum_{i=1}^{k}\tilde{\gamma}_i^{\mathrm{op}}\frac{\lambda+\mu_0}{\lambda+\rho_i^{\mathrm{op}}}-1$$
$$=P(\lambda)P(\mu_0) \tag{4.9}$$

also has the alternance properties, where

$$\tilde{\gamma}_i^{\mathrm{op}}=\sum_{j=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\frac{1}{\mu_0+\rho_j^{\mathrm{op}}}.$$

Let us consider the system

$$\frac{\lambda+\mu_0}{\lambda+\rho_1^{\mathrm{op}}},\ \frac{\lambda+\mu_0}{\lambda+\rho_2^{\mathrm{op}}},\ \ldots,\ \frac{\lambda+\mu_0}{\lambda+\rho_k^{\mathrm{op}}}. \tag{4.10}$$

Since $\rho_j^{\mathrm{op}}>0$ and $\rho_j^{\mathrm{op}}\neq\rho_i^{\mathrm{op}}$ if $i\neq j$, (4.10) is a Chebyshev system on $[a, b]$ (see [3, p.72]). Thus the minimum of the expression

$$\max_{\lambda\in[a,b]}\left|\sum_{i=1}^{k}\tilde{\gamma}_i\frac{\lambda+\mu_0}{\lambda+\rho_i^{\mathrm{op}}}-1\right|$$

is achieved by the function which has the alternance properties. This function must be g of (4.9). Therefore

$$\max_{\lambda\in[a,b]}\left|\sum_{i=1}^{k}\tilde{\gamma}_i^{\mathrm{op}}\frac{\lambda+\mu_0}{\lambda+\rho_i^{\mathrm{op}}}-1\right|=\max_{\lambda\in[a,b]}|g(\lambda)|$$
$$\leqslant\max_{\lambda\in[a,b]}\left|\sum_{i=1}^{k}\tilde{\gamma}_i\frac{\lambda+\mu_0}{\lambda+\rho_i^{\mathrm{op}}}-1\right|,\quad\text{for all }\{\tilde{\gamma}_i\}. \tag{4.11}$$

Since (4.11) holds for all $\mu_0\in[a, b]$, we have

$$\max_{\substack{\lambda\in[a,b]\\ \mu\in[a,b]}}\left|\sum_{i=1}^{k}\sum_{j=1}^{k}\gamma_{ij}^{\mathrm{ADI}}\frac{\lambda+\mu}{(\lambda+\rho_i^{\mathrm{op}})(\mu+\rho_j^{\mathrm{op}})}-1\right|$$
$$\leqslant\max_{\substack{\lambda\in[a,b]\\ \mu\in[a,b]}}\left|\sum_{i=1}^{k}\sum_{j=1}^{k}\gamma_{ij}\frac{\lambda+\mu}{(\lambda+\rho_i^{\mathrm{op}})(\mu+\rho_j^{\mathrm{op}})}-1\right|,\quad\text{for all }\{\gamma_{ij}\}, \tag{4.12}$$

which is exactly the same as (4.7). □

In some preconditioned iterative methods, such as the preconditioned conjugate gradient method, it is required that the preconditioner M_k be positive definite. The positive definiteness of M_k is guaranteed by the following theorem.

Theorem 4.2. *M_k defined in* (3.4) *is positive definite when the parameters $\{\gamma_{ij}\}$, $\{\rho_j\}$ satisfy one of the following conditions*:

(1) $\rho_j > 0, \quad \rho_i \neq \rho_j \text{ if } i \neq j, \qquad \gamma_{ij} = \gamma_{ij}^{\mathrm{ADI}}(\rho),$

(2) $\rho_j > 0, \quad \rho_i \neq \rho_j \text{ if } i \neq j, \qquad \gamma_{ii} = \gamma_{ij}^{\mathrm{op}}(\rho),$

(3) $\rho_j = \rho_j^*, \qquad \gamma_{ij} = \gamma_{ij}^*.$

Proof. It follows from (4.3) and (4.4) that the eigenvalues of $M_k^{-1}A$ are bounded below by

$$1 - \omega_k.$$

Therefore, if $\omega_k < 1$, then the eigenvalues of $M_k^{-1}A$ are all positive. Since A is positive definite, we conclude that M_k^{-1}, hence M_k, is positive definite if $\omega_k < 1$.

Let $\{\rho_j\}$ be such that $\rho_j > 0$ and $\rho_i \neq \rho_j$. For this set of $\{\rho_j\}$, the ADI method of Algorithm 2.2 is equivalent to (3.4) if $\gamma_{ij} = \gamma_{ij}^{\mathrm{ADI}}(\rho)$. It is easy to verify that

$$\left| \sum_{i=1}^{k} \sum_{j=1}^{k} \gamma_{ij}^{\mathrm{ADI}}(\rho) \frac{\lambda + \mu}{(\lambda + \rho_i)(\mu + \rho_j)} - 1 \right| = \left| \prod_{j=1}^{k} \frac{(\lambda - \rho_j)(\mu - \rho_j)}{(\lambda + \rho_j)(\mu + \rho_j)} \right|. \tag{4.13}$$

The right-hand side of (4.13) is clearly less than 1. Thus we have

$$\omega_k\left(\gamma^{\mathrm{ADI}}(\rho), \rho\right) < 1,$$

and the rest of the proof follows from the inequalities (4.6). □

Finally, if $\{\rho_j\}$ is given such that $\rho_j > 0$ and $\rho_i \neq \rho_j$ if $i \neq j$, the parameters $\{\gamma_{ij}^{\mathrm{ADI}}(\rho)\}$ can be computed from Theorem 3.1. However, we note that (3.1) is not numerically stable. The severity of the instability is dependent upon the choices of parameters $\{\rho_j\}$. Our numerical results show that for practical problems where k is not very large and $\rho_j \approx \rho_j^{\mathrm{op}}$, the instability is negligible.

5. Implementations on a parallel computer

It is often believed that any discussion of parallel numerical algorithms is considered incomplete if issues of architecture/algorithm mapping are not addressed. In this section we discuss some implementations of Algorithm 3.2 on the Myrias multiprocessor architecture. Algorithm 3.2 can also be implemented on hypercube architectures. With data structures similar to those described in [13,20,22] this algorithm is expected to perform well on hypercube architecture.

The Myrias parallel computer is a newly developed massively-parallel computer system. Myrias Parallel Fortran provides access to parallelism and dynamic array allocation. When a "parent" task starts a parallel process, a collection of "child" tasks is created, and the parent task is suspended until all of the child tasks have finished executing. Child tasks inherit identical copies of the parent task's memory state. Each task then executes independently. When all of the child tasks have completed, their memory images are merged together to form a new memory

image for the parent, and the parent task resumes execution [30]. Under this programming model, the way the memory is arranged physically in the machine is different from the way the memory is viewed by a user. The user sees a flat address space and does not have control over where data are actually stored in physical memory. The assignment of child tasks to processors, and the management of the memory spaces of tasks are transparent to the user.

To implement Algorithm 3.2 on the Myrias parallel computer, we first note that step (1) of this algorithm constitutes a set of k independent tridiagonal systems

$$\left[\left(B_1 + \rho_j A\right) \otimes I\right] d_j = f, \quad j = 1, \ldots, k. \tag{5.1}$$

Using the matrix representation for vectors [11], (5.1) can be written as

$$\left(B_1 + \rho_j A_1\right) d_j^{\mathrm{T}} = f^{\mathrm{T}}, \quad j = 1, \ldots, k, \tag{5.2}$$

where d_j and f are represented by $n \times n$ matrices. Let

$$d_j^{\mathrm{T}} = \left[\tilde{d}_j^1, \tilde{d}_j^2, \ldots, \tilde{d}_j^n\right], \qquad f^{\mathrm{T}} = \left[\tilde{f}^1, \ldots, \tilde{f}^n\right], \tag{5.3}$$

where $\tilde{d}_j^i$, $\tilde{f}^i$, $i = 1, 2, \ldots, n$, are column vectors of size n. Then (5.2) consists of n independent tridiagonal systems of size n:

$$\left(B_1 + \rho_j A_1\right) \tilde{d}_j^i = \tilde{f}^i, \quad i = 1, \ldots, n, \tag{5.4}$$

for each $j = 1, 2, \ldots, k$.

Step (1) of Algorithm 3.2, consisting of kn independent systems, can be performed by p tasks, each solving kn/p systems of the form (5.4) by the Gaussian elimination algorithm. The memory storage required for the solution of (5.4) by the Gaussian elimination algorithm, other than f, d and ρ, is created dynamically within the fast local memory, and destroyed after the task is completed. A potential problem in solving (5.4) is that the vector $\tilde{f}^i$ is not stored in consecutive memory locations because of the transpose in (5.3). However, in the Gaussian elimination process only one read of data from the task's own copy of f is required and f needs not be updated when merging takes place at the end of the parallel process. In the merging procedure, only the locations of d_j are updated. The updating is performed with good locality of reference because each task addresses the consecutive memory locations of d_j. Note that for each fixed j, d_j stores d_j^{T}, not d_j.

After step (1) is completed, d_j^{T} can be partitioned as

$$\tilde{d}_j^{\nu}, \quad \nu = 1, \ldots, \kappa, \text{ where } \kappa = \frac{p}{k}.$$

This partition is to balance the work load of the processors, and may not necessarily be the same form as (5.3). Then in step (2) each of the p tasks proceeds to compute

$$\tilde{d}_j^{\nu} \leftarrow \sum_{i=1}^{k} \gamma_{ij} \tilde{d}_i^{\nu}, \tag{5.5}$$

for a fixed $j = 1, \ldots, k$, and $\nu = 1, \ldots, \kappa$. The size of the vectors in (5.5) is $l = kn^2/p$.

Steps (3) and (4) are performed similarly as steps (1) and (2) with appropriate partitions of the vectors.

6. Experimental results

Experiments were conducted on a Myrias SPS-2 with 64 processors. The Laplace equation with Dirichlet boundary conditions over a square domain is discretized using a uniform mesh

$$h = \frac{1}{n+1}.$$

The resulting system is

$$Au = (A_1 \otimes B_2 + B_1 \otimes A_2)u = f, \tag{6.1}$$

where A_j, B_j, $j = 1, 2$, are given by (1.3) for the 5-point approximation and by (1.4) for the 9-point approximation. The number of equations in (6.1) is $N = n^2$.

Equation (6.1) is solved by the preconditioned conjugate gradient method. The following algorithm, which is a modification of the original PCG algorithm [8], was proposed in [4] for parallel computations.

A parallel PCG algorithm.

choose u_0
$r_0 = f - Au_0$, $p_0 = d_0 = M^{-1}r_0$
compute and store Ad_0
$a_0 = (d_0, r_0)/(Ad_0, d_0)$, $b_{-1} = 0$
For $i = 0, 1, \ldots$ until convergence do
(1) $p_i = d_i + b_{i-1}p_{i-1}$
(2) $Ap_i = Ad_i + b_{i-1}Ap_{i-1}$
(3) $u_{i+1} = u_i + a_i p_i$
(4) $r_{i+1} = r_i - a_i Ap_i$
(5) $d_{i+1} = M^{-1}r_{i+1}$
(6) compute and store Ad_{i+1}
(7) compute (d_{i+1}, r_{i+1}), (Ad_{i+1}, d_{i+1})
(8) $b_i = (d_{i+1}, r_{i+1})/(d_i, r_i)$
(9) $a_{i+1} = (d_{i+1}, r_{i+1})/[(Ad_{i+1}, d_{i+1}) - (b_i/a_i)(d_{i+1}, r_{i+1})]$

In the above, M is a preconditioner. We will consider the preconditioners of the ADI methods given by Algorithms 2.2 and 3.2. For simplicity, we denote by TADI the PCG algorithm when the preconditioner is given by Algorithm 2.2, and by PADI the method when the preconditioner is given by Algorithm 3.2.

The eigenvalues of $A_1^{-1}B_1 = A_2^{-1}B_2$ are given by

$$\lambda_j = \begin{cases} \sin^2\left(\dfrac{\pi j}{2(n+1)}\right), & \text{for 5-point,} \\ \dfrac{\sin^2\left(\dfrac{\pi j}{2(n+1)}\right)}{1 + \frac{1}{5}\cos\left(\dfrac{\pi j}{n+1}\right)}, & \text{for 9-point,} \end{cases} \qquad j = 1, 2, \ldots, n.$$

Thus the bounds for $\sigma(A_1^{-1}B_1) = \sigma(A_1^{-1}B_1)$ can be found to be

$$a = \begin{cases} \sin^2\left(\dfrac{\pi}{2(n+1)}\right), & \text{for 5-point,} \\ \dfrac{\sin^2\left(\dfrac{\pi}{2(n+1)}\right)}{1+\frac{1}{5}\cos\left(\dfrac{\pi}{n+1}\right)}, & \text{for 9-point,} \end{cases}$$

$$b = \begin{cases} \sin^2\left(\dfrac{n\pi}{2(n+1)}\right), & \text{for 5-point,} \\ \dfrac{\sin^2\left(\dfrac{n\pi}{2(n+1)}\right)}{1-\frac{1}{5}\cos\left(\dfrac{\pi}{n+1}\right)}, & \text{for 9-point.} \end{cases}$$

The parameters $\{\rho_j\}$ are chosen to be the Peaceman–Rachford parameters according to [35, p.525] for both TADI and PADI:

$$\rho_j = b\left(\frac{a}{b}\right)^{(2j-1)/2k}, \quad j = 1, 2, \ldots, k.$$

This set of parameters approximates the optimum parameters $\{\rho_j^{\text{op}}\}$ which minimize the expression (4.1). The parameters $\{\gamma_{ij}\}$ are chosen as $\{\gamma_{ij}^{\text{ADI}}(\rho)\}$ given by (3.1). With these choices, TADI and PADI are mathematically equivalent.

Implementation of PADI was discussed in Section 5, and each iteration of TADI, given by (2.7), is implemented similarly as for the PADI with $k = 1$. In other words, the TADI is a parallel implementation of Algorithm 2.2. Thus all 64 processors are used in both the TADI and PADI methods.

All the computations are performed in double precision. The iteration process is terminated if the l_∞-norm of the residual $\| r \|_\infty \leqslant 10^{-8}$. As an initial guess for the PCG outer iteration, we take a vector whose elements are random numbers uniformly distributed in [0, 1].

Table 2 compares the CPU-times required for convergence of the two methods with different numbers of terms in the preconditioners. The speedup in the CPU-time of PADI over TADI increases with k. This is due to the fact that TADI requires more overhead cost and more memory references as k is increased.

Table 2
CPU-time in seconds for convergence with $n = 100$

k	2	4	8	10	16	20	21
TADI	112.8	85.4	74.8	68.5	69.3	84.9	88.9
PADI	82.1	42.4	23.4	18.3	14.9	16.4	16.9
Speedup	1.37	2.01	3.02	3.74	4.65	5.18	5.26

Table 3
CPU-time in seconds for computing $M_k^{-1}r$ with $n=100$

k	2	4	8	10	16	20	21
TADI	4.4	8.4	16.5	20.5	32.4	40.2	42.4
PADI	2.7	3.1	3.5	3.8	4.8	5.7	6.0
Speedup	1.63	2.71	4.71	5.39	6.75	7.05	7.07

Table 4
CPU-time in seconds for convergence with different n, $k=16$

n	60	100	120	150	200
TADI	47.5	69.3	94.0	162.0	234.2
PADI	11.0	14.9	17.0	33.2	49.6
Speedup	4.3	4.7	5.5	4.9	4.7

Table 5
CPU-time in seconds for convergence with different n, $k=20$

n	60	100	120	150	200
TADI	57.9	84.9	115.3	133.5	193.5
PADI	11.7	16.4	19.9	26.3	40.3
Speedup	4.9	5.2	5.8	5.1	4.8

In Table 3 we report the CPU-time required to compute $M_k^{-1}r$ in step (5) of the PCG algorithm. As expected, the CPU-time is approximately a linear function of k in the TADI method because the k-term TADI preconditioner roughly repeats k times of the 1-term TADI preconditioner. On the other hand, the CPU-time increases slowly with k in the PADI method.

The results for different numbers of equations are reported in Tables 4 and 5. From these results, we observe that the best speedup in the CPU-time of PADI over TADI appears to occur at $n=120$, but we do not fully understand this behavior.

Finally, Table 6 compares the results of the 5-point scheme and the 9-point scheme by using the PADI method. As we pointed out earlier, the extra work in solving a 9-point system is negligible.

Table 6
Comparison of CPU-time in seconds for the 5- and 9-point schemes

	$k=16$		$k=20$	
	$n=100$	$n=200$	$n=100$	$n=200$
5-point	14.9	49.6	16.4	40.1
9-point	15.0	49.9	16.4	40.3

In the computations for the PADI method, the parameters $\{\gamma_{ij}^{ADI}\}$ were computed from the formulas given in Theorem 3.1. Instability was not observed for k up to 21 and for n ranging from 60 to 300. The computed solutions from both the TADI and the PADI methods agree within the accepted accuracy.

7. Conclusions

We have proposed a parallel algorithm for the ADI preconditioning in tensor product formulations. In this algorithm, several steps of the ADI iteration are computed simultaneously. This results in improvements of both the degree of parallelism and the data locality over classical ADI algorithms. However the new algorithm also introduces some additional computational work. The preliminary experimental results on a parallel computer demonstrate that this algorithm can be implemented efficiently in parallel and good performance is observed. We have also discussed the selection of acceleration parameters and the conditions for the preconditioner to be positive definite. The practical value of this algorithm for implementing the ADI scheme directly, not as a preconditioner, has yet to be investigated.

Acknowledgements

We thank the Myrias Research Corporation for providing computing environment on the Myrias SPS-2 computer. We are also indebted to Dr. M. Walker, Dr. B. Joe and Mr. F. Carlacci for their helps in the course of preparing this paper. We thank the referees for their comments leading to the improvement of this paper.

References

[1] L. Adams, m-step preconditioned conjugate gradient methods, *SIAM J. Sci. Statist. Comput.* **6** (2) (1985) 452–463.

[2] O. Axelsson, A class of iterative methods for finite element equations, *Comput. Methods Appl. Mech. Engrg.* **9** (2) (1976) 123–137.

[3] E.W. Cheney, *Introduction to Approximation Theory* (McGraw-Hill, New York, 1966).

[4] A.T. Chronopoulos and C.W. Gear, s-step iterative methods for symmetric linear systems, *J. Comput. Appl. Math.* **25** (2) (1989) 153–168.

[5] A.T. Chronopoulos and C.W. Gear, On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy, *Parallel Comput.* **11** (1) (1989) 37–53.

[6] B. Codenotti and M. Leoncini, Parallelism and fast solution of linear systems, *Comput. Math. Appl.* **19** (1990) 1–18.

[7] P. Concus and G.H. Golub, Use of fast direct methods for the efficient numerical solution of nonseparable elliptic equations, *SIAM J. Numer. Anal.* **10** (1973) 1103–1120.

[8] P. Concus, G.H. Golub and D.P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in: J.R. Bunch and D.J. Rose, Eds., *Sparse Matrix Computations* (Academic Press, New York, 1976) 309–332.

[9] J. Douglas Jr and H.H. Rachford Jr, On the numerical solution of heat conduction problems in two or three space variables, *Trans. Amer. Math. Soc.* **82** (1956) 421–439.

[10] E.G. D'Yakonov, An iteration method for solving systems of finite difference equations, *Soviet Math. Dokl.* **2** (1961) 647–650.
[11] W.R. Dyksen, Tensor product generalized ADI methods for separable elliptic problems, *SIAM J. Numer. Anal.* **24** (1987) 59–76.
[12] W.R. Dyksen, A tensor product generalized ADI method for elliptic problems on cylindrical domains with holes, *J. Comput. Appl. Math.* **16** (1) (1986) 43–58.
[13] Ö. Eğecioğlu, C.K. Koc and A.J. Laub, A recursive doubling algorithm for solution of tridiagonal systems on hypercube multiprocessors, *J. Comput. Appl. Math.* **27** (1&2) (1989) 95–108.
[14] D.J. Evans and C.R. Gane, Alternating direction preconditioning methods for partial differential equations, in: D.J. Evans, Ed., *Precondition Methods, Theory and Applications* (Gordon and Breach, New York, 1983) 81–114.
[15] K.A. Gallivan, R.J. Plemnons and A.H. Sameh, Parallel algorithms for dense linear algebra computations, *SIAM Rev.* **32** (1990) 54–135.
[16] G.H. Golub and D.P. O'Leary, Some history of the conjugate gradient and Lanczos algorithms: 1948–1976, *SIAM Rev.* **31** (1989) 50–102.
[17] J.E. Gunn, The numerical solution of $\nabla \cdot a \nabla u = f$ by a semi-explicit alternating-direction iterative technique, *Numer. Math.* **6** (1964) 181–184.
[18] J.E. Gunn, The solution of elliptic difference equations by semi-explicit iterative techniques, *SIAM J. Numer. Anal.* **2** (B) (1965) 24–45.
[19] P.R. Halmos, *Finite-dimensional Vector Spaces* (Van Nostrand, Princeton, NJ, 2nd ed., 1959).
[20] C.T. Ho and S.L. Johnsson, Optimizing tridiagonal solvers for alternating direction methods on Boolean cube multiprocessors, *SIAM J. Sci. Statist. Comput.* **11** (1990) 563–592.
[21] B.P. Il'in, Some estimates for conjugate gradient methods, *U.S.S.R. Comput. Math. and Math. Phys.* **16** (1976) 22–30.
[22] S.L. Johnsson, Y. Saad and M.H. Schultz, Alternating direction methods on multiprocessors, *SIAM J. Sci. Statist. Comput.* **8** (1987) 686–700.
[23] J.J. Lambiotte and R.G. Voigt, The solution of tridiagonal linear systems on the CDC STAR-100 computer, *ACM Trans. Math. Software* **1** (1975) 308–329.
[24] R.E. Lynch, J.R. Rice and D.H. Thomas, Tensor product analysis of alternating direction implicit methods, *J. Soc. Indust. Appl. Math.* **13** (1965) 995–1006.
[25] T. Manteuffel and S. Parter, Preconditioning and boundary conditions, *SIAM J. Numer. Anal.* **27** (1990) 656–694.
[26] J.M. Ortega and R.G. Voigt, Solution of partial differential equations on vector and parallel computers, *SIAM Rev.* **27** (1985) 149–240.
[27] D.W. Peaceman and H.H. Rachford Jr, The numerical solution of parabolic and elliptic differential equations, *J. Soc. Indust. Appl. Math.* **3** (1955) 28–41.
[28] A.H. Sameh, S.C. Chen and D.J. Kuck, Parallel Poisson and biharmonic solvers, *Computing* **17** (1976) 219–230.
[29] H.S. Stone, An efficient parallel algorithm for the solution of a tridagonal linear system of equations, *J. Assoc. Comput. Mach.* **20** (1) (1973) 27–38.
[30] T. Stone, B. Joe, A. Behie and M. London, Solution of linear systems on a MIMD parallel computer, presented at Supercomputing Symposium '90, Montreal, 1990.
[31] H.A. van der Vorst, Analysis of a parallel solution method for tridiagonal linear systems, *Parallel Comput.* **5** (3) (1987) 303–311.
[32] E.L. Wachspress, Optimum alternating-direction-implicit iteration parameters for a model problem, *J. Soc. Indust. Appl. Math.* **10** (1962) 339–350.
[33] E.L. Wachspress, Extended application of alternating direction implicit iteration model problem theory, *J. Soc. Indust. Appl. Math.* **11** (1963) 994–1016.
[34] G.A. Watson, *Approximation Theory and Numerical Methods* (Wiley, New York, 1980).
[35] D.M. Young, *Iterative Solution of Large Linear Systems* (Academic Press, New York, 1971).