Parallel Computing 9 (1988/89) 291–312
North-Holland

A projection method for solving nonsymmetric linear systems on multiprocessors *

Chandrika KAMATH and Ahmed SAMEH

Center for Supercomputing Research and Development, University of Illinois-Urbana, 305 Talbot Laboratory, 104 South Wright Street, Urbana, IL 61801-2932, U.S.A.

Received March 1987
Revised January 1988

Abstract. We consider the iterative solution of large sparse linear systems of equations arising from elliptic and parabolic partial differential equations in two or three space dimensions. Specifically, we focus our attention on nonsymmetric systems of equations whose eigenvalues lie on both sides of the imaginary axis, or whose symmetric part is not positive definite. This system of equations is solved using a block Kaczmarz projection method with conjugate gradient acceleration. The algorithm has been designed with special emphasis on its suitability for multiprocessors. In the first part of the paper, we study the numerical properties of the algorithm and compare its performance with other algorithms such as the conjugate gradient method on the normal equations, and conjugate gradient-like schemes such as ORTHOMIN(k), GCR(k) and GMRES(k). We also study the effect of using various preconditioners with these methods. In the second part of the paper, we describe the implementation of our algorithm on the CRAY X-MP/48 multiprocessor, and study its behavior as the number of processors is increased.

Keywords. Conjugate gradient algorithm, CRAY X-MP/48, multiprocessors, nonsymmetric systems, projection methods, symmetric successive overrelaxation.

1. Introduction

Many problems in engineering and science give rise to one of the most fundamental problems of linear algebra—that of solving linear systems of equations. The use of complex models to describe physical systems often results in handling large sparse linear systems of equations. Since these systems usually occur in the innermost loop of the computational scheme, fast and efficient methods for their solution are very important. One way of speeding up the solution of these systems is by the use of parallel computers. While vector machines such as the CRAY-1 and CYBER 205 have provided speedups over sequential machines, even higher speedups are desired. This increase in speedup can be achieved only through the use of parallelism offered by multiprocessors such as the CRAY X-MP/48 or the ALLIANT FX/80, for example. This, in turn, requires developing algorithms that exploit both concurrency and vectorization.

* This work was supported in part by the National Science Foundation under Grant Nos. US NSF DCR84-10110 and US NSF DCR85-09970, the U.S. Department of Energy under Grant No. US DOE-DE-FG02-85ER25001, the U.S. Air Force Office of Scientific Research under Grant No. AFOSR-85-0211, the IBM Donation, and Digital Equipment Corporation.

In this paper, we present an iterative scheme for the solution of large sparse linear systems of equations

$$Ax = f \tag{1.1}$$

where A is a nonsingular, nonsymmetric $N \times N$ matrix, arising from the numerical handling of (say) elliptic partial differential equations in two or three space dimensions. Effective solution techniques of nonsymmetric systems of equations have thus far concentrated on two cases:

(i) the symmetric part of the matrix is positive definite or,

(ii) the eigenvalues of the matrix are all on one side of the imaginary axis.

These algorithms have been either derived from the conjugate gradient algorithm, or the Chebyshev method. Clearly, any general nonsymmetric system may be solved by handling the corresponding normal equations. This approach, though efficient when the matrix is well-conditioned, can lead to loss of information if the entries of the matrix vary widely. It also results in squaring the condition number of the matrix, which is not desirable. Another method which has recently gained popularity is the Generalized Minimal Residual method, GMRES(k) [33]. This method has been shown to converge for any starting vector, provided the number of direction vectors, k, is sufficiently large. In practice, however, the method may not converge either due to the instability of the preconditioner used or due to the fact that the value of k chosen is not large enough [14]. In addition, GMRES(k), especially when used in conjunction with an incomplete LU-type preconditioner, is not ideally suited for implementation on a multiprocessor.

In the next section, we present a projection method for the solution of (1.1), which is ideally suited for multiprocessors. The algorithm that we consider does not place any restrictions on the eigenvalue distribution of the matrix and hence is suitable for the case when the eigenvalues are on both sides of the imaginary axis. It can also be applied when the nonsymmetric matrix has an indefinite symmetric part. The computationally demanding aspect of our algorithm is the solution, in each iteration, of several independent linear least squares problems of much smaller size than the original linear system. For two-dimensional elliptic problems, for example, such least squares problems are handled via a direct solver. For three-dimensional problems, however, efficient iterative solvers for handling these linear least squares problems become necessary. In this paper, we present results pertaining only to 2-D problems even though the projection scheme is applicable to 3-D problems as well.

In Sections 2–4, we outline the algorithm and study its numerical behavior. We also compare the performance of our algorithm with existing methods such as the conjugate gradient method on normal equations, GCR(k), ORTHOMIN(k) and GMRES(k) and study the effect of preconditioners on these methods. In Section 5, we describe the implementation of our algorithm on the CRAY X-MP/48 multiprocessor and observe how the speedup changes as the number of processors is varied.

Unless otherwise specified, lower case italic letters denote vectors and upper case Greek and italic letters in bold type denote matrices.

2. Projection methods

As the name suggests, a projection method can be considered as a method of solution which involves the projection of a vector onto a subspace. This method, which was first proposed by Kaczmarz [22] in 1937, considers each equation as a hyperplane; thus reducing the problem of finding the solution to a set of equations to the equivalent problem of finding the coordinates of the point of intersection of the hyperplanes. The method of solution is to project an initial iterate onto the first hyperplane, project the resulting point onto the second, and continue the

projections on the hyperplanes in a cyclic order, approaching the solution more and more closely with each projection [4,40,41].

Algorithm 2.1 (Kaczmarz Method)
Step 1: Choose x_0; set $k=0$.
Step 2: Compute

$$x_k \leftarrow x_k + \frac{r_k^i}{\| a_i \|_2^2} a_i, \quad i = 1, \ldots, n.$$

Step 3: If a convergence criterion is satisfied, terminate the iteration; else set $k = k+1$ and go to Step 2.

Here r_k^i is the ith component of the residual $r_k = f - Ax_k$ and a_i^T is the ith row of the matrix A. For each k, Step 2 consists of n steps, one for each row of A.

The method of Kaczmarz has recently been used under the name (unconstrained) ART (Algebraic Reconstruction Technique) in the area of image reconstruction [17,18,42]. An extensive study of the convergence properties of this method has been done by Tanabe [38,39], wherein he proves that the method converges for any system of linear equations with nonzero rows, even when it is singular and inconsistent. A further study of this and other similar methods can be found in [1,7,10].

The method due to Kaczmarz was recognized as a projection method by N. Gastinel [15,16]. This idea of projection methods was further generalized by Householder and Bauer [20,21] to include the method of steepest descent, Gauss–Siedel and other relaxation schemes.

Let the error and the residual at the kth step be defined as

$$\delta x_k = x - x_k \tag{2.1}$$

and

$$r_k = f - Ax_k. \tag{2.2}$$

Then a method of projection is one in which at each step, the error δx_k is resolved into two components, one of which is required to lie in a subspace selected at that step, and the other is δx_{k+1}, which is required to be less than δx_k in some norm. The subspace is selected by choosing a matrix Y_k whose columns are linearly independent. Equivalently,

$$\delta x_{k+1} = \delta x_k - Y_k u_k \tag{2.3}$$

where u_k is a vector (or a scalar if Y_k has dimension 1) to be selected at the kth step so that

$$\| \delta x_{k+1} \| < \| \delta x_k \| \tag{2.4}$$

where $\| \cdot \|$ is some vector norm. The method of projection would depend on the choice of the matrix Y_k in (2.3) and the vector norm in (2.4).

If we consider ellipsoidal norms, we can define

$$\| s \|^2 = s^T G s \tag{2.5}$$

where G is a positive definite matrix. Selecting u_k such that $\| \delta x_{k+1} \|$ is minimized, gives

$$u_k = \left(Y_k^T G Y_k \right)^{-1} Y_k^T G \delta x_k \tag{2.6}$$

and

$$\| \delta x_k \|^2 - \| \delta x_{k+1} \|^2 = \delta x_k^T G Y_k u_k. \tag{2.7}$$

However, we know r_k and not δx_k. Therefore to make the process feasible, the condition

$$Y_k^T G = V_k^T A \tag{2.8}$$

should hold, thus expressing u_k in terms of r_k. The matrix V_k will have to be determined at each iteration.

2.1. Block iterative methods

Various choices of G in (2.5) give rise to various projection methods. In this paper, we will concentrate on the case when

$$G = I. \tag{2.9}$$

This gives, from (2.6) and (2.8),

$$u_k = \left(V_k^{\mathrm{T}} A A^{\mathrm{T}} V_k\right)^{-1} V_k^{\mathrm{T}} A \delta x_k \tag{2.10}$$

and

$$x_{k+1} = x_k + A^{\mathrm{T}} V_k \left(V_k^{\mathrm{T}} A A^{\mathrm{T}} V_k\right)^{-1} V_k^{\mathrm{T}} r_k. \tag{2.11}$$

Now let N be even, and let A be partitioned as

$$A = \begin{bmatrix} B_1^{\mathrm{T}} \\ B_2^{\mathrm{T}} \end{bmatrix} \tag{2.12a}$$

where B_i^{T}, $i = 1, 2$, is an $N/2 \times N$ matrix. Let

$$V_1^{\mathrm{T}} = \left[I_{N/2}, 0\right], \qquad V_2^{\mathrm{T}} = \left[0, I_{N/2}\right] \quad \text{and} \quad f^{\mathrm{T}} = \left[f_1^{\mathrm{T}}, f_2^{\mathrm{T}}\right] \tag{2.12b}$$

where f_i, $i = 1, 2$, is an $N/2$ vector.

Then, one iteration of (2.12) can be written as

$$\begin{aligned} &z_1 = x_k, \qquad z_2 = z_1 + B_1\left(B_1^{\mathrm{T}} B_1\right)^{-1}\left(f_1 - B_1^{\mathrm{T}} z_1\right), \\ &z_3 = z_2 + B_2\left(B_2^{\mathrm{T}} B_2\right)^{-1}\left(f_2 - B_2^{\mathrm{T}} z_2\right), \qquad x_{k+1} = z_3. \end{aligned} \tag{2.13}$$

This is essentially the same as applying the Gauss–Siedel method to

$$A A^{\mathrm{T}} y = f, \qquad A^{\mathrm{T}} y = x. \tag{2.14}$$

Variations of (2.13) include the block-row Jacobi, the block-row JOR and the block-row SOR methods [12,24,31,43]. For each of these methods, we also have the corresponding block-column method, obtained by partitioning the matrix in (2.12a) by columns instead of rows. Note that if A had been partitioned into N parts, each part being a row of A, then (2.13) would be equivalent to the Kaczmarz method.

2.2. A block-SSOR method

If we incorporate a parameter ω in (2.13) and follow a forward sweep through the variables by a backward sweep, we obtain the block-row SSOR method [3].

Algorithm 2.2 (Block-row SSOR method)
Step 1: Choose x_0; set $k = 0$.
Step 2: Compute

$$z_1 = x_k$$
$$z_2 = z_1 + \omega B_1 (B_1^T B_1)^{-1} (f_1 - B_1^T z_1)$$
$$z_3 = z_2 + \omega B_2 (B_2^T B_2)^{-1} (f_2 - B_2^T z_2)$$
$$z_4 = z_3 + \omega B_2 (B_2^T B_2)^{-1} (f_2 - B_2^T z_3)$$
$$z_5 = z_4 + \omega B_1 (B_1^T B_1)^{-1} (f_1 - B_1^T z_4)$$
$$x_{k+1} = z_5$$

Step 3: If a convergence criterion is satisfied, terminate the iteration; else set $k = k + 1$ and go to Step 2.

In the above block-row SSOR method, we have assumed the same value of ω for both the forward and backward sweeps. Note that the term $B_i(B_i^T B_i)^{-1} f_i$, $i = 1, 2$ in Algorithm 2.2, can be evaluated just once.

We can thus write one iteration of the block-row SSOR method in the form

$$x_{k+1} = Q x_k + R f \tag{2.15}$$

where

$$Q = Q_1 Q_2 Q_2 Q_1,$$

in which

$$Q_i = (I - \omega P_i), \quad i = 1, 2,$$

with

$$P_i = B_i B_i^+, \qquad B_i^+ = (B_i^T B_i)^{-1} B_i^T,$$

and

$$R = \omega \left[(I + Q_1 Q_2 Q_2)(B_1^+)^T, \; Q_1 (Q_2 + I)(B_2^+)^T \right].$$

Note that P_i are projectors onto the space spanned by the columns of B_i. We have the following result for convergence of the iteration in (2.15).

Theorem 2.3. *Let* $f \in R(A)$ *and* $x_0 \in R(A^T)$. *Then the iteration* (2.15) *converges towards the solution of* (1.1) *for* $0 < \omega < 2$.

The proof of this theorem is based on the facts that the P_i's are projection operators and that the matrix A is nonsingular [23].

We observe that there are no restrictions on the nonsingular matrix A required for convergence; therefore the method is applicable in the case where the matrix A has a general eigenvalue distribution.

2.3. Determination of optimal omega

In this section we consider the optimal value of ω to be used in Algorithm 2.2. Similar to the block-row JOR and the block-column JOR [12,24], we show that the optimal value of ω is 1.0.

A study of the optimal ω for the block SOR method has been carried out by Elfving [12] using the classical SOR theory [44].

Theorem 2.4. *The optimal value of* ω *is* 1.0.

Proof. We would like to determine that value of ω which minimizes the spectral radius of the iteration matrix in (2.15),

$$Q = (I - \omega P_1)(I - \omega P_2)^2(I - \omega P_1),$$

where P_1 and P_2 are as defined above. Let U and V be two orthogonal matrices such that

$$U^T P_1 U = V^T P_2 V = \operatorname{diag}(I_{N/2}, 0).$$

Thus Q has the same spectral radius as

$$Q(\alpha) = DWD^2W^TD$$

where $D = \operatorname{diag}(\alpha I_{N/2}, I_{N/2})$, with $\alpha = 1 - \omega$, and $W = U^TV$. Throughout, we assume that N is even. Let W be partitioned as

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}$$

where each W_{ij} is of order $N/2$, $i, j = 1, 2$. As a result,

$$Q = Q(\alpha) = \begin{bmatrix} C_{11}(\alpha) & C_{12}(\alpha) \\ C_{12}^T(\alpha) & C_{22}(\alpha) \end{bmatrix}$$

where

$$C_{22}(\alpha) = \alpha^2 I_{N/2} + (1 - \alpha^2) W_{22} W_{22}^T$$

and $C_{11}(\alpha)$, $C_{12}(\alpha)$ vanish for $\alpha = 0$. Now, since

(i) all the eigenvalues of $C_{22}(\alpha)$ lie in the interval containing the eigenvalues of $Q(\alpha)$,
(ii) the spectral radius of $C_{22}(\alpha)$ is minimized when α is zero, and
(iii) the spectral radius of $C_{22}(0)$ coincides with that of $Q(0)$,

the spectral radius of $Q(\alpha)$ is also minimized when α is zero, or ω is 1.0. This completes the proof. □

The observation that the optimal value of ω is 1.0, and the fact that P_i is a projection lead to simplification of the expressions for the matrices Q and R,

$$Q = (I - P_1)(I - P_2)(I - P_1), \tag{2.15a}$$

$$R = \left[(I + Q_1Q_2)(B_1^+)^T, Q_1(B_2^+)^T\right] \tag{2.15b}$$

and Algorithm 2.2 is now independent of any parameters.

2.4. Organization on a multiprocessor

From (2.15), we see that this block-SSOR method requires obtaining vectors v of the form

$$(I - P)u = v \tag{2.16}$$

where

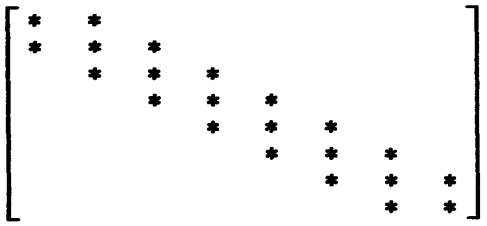
$$P = F(F^TF)^{-1}F^T,$$

F being an $m \times n$ matrix of maximal rank. The vector v may thus be obtained as the residual corresponding to the linear least squares problem

$$\min_{\bar{v}} \| u - F\bar{v} \|_2 . \tag{2.17}$$

If the matrix F consists of smaller independent matrices, then (2.17), would represent several independent linear least squares problems, which would make it well suited for implementation on a multiprocessor.

Consider the case where the matrix A is a block tridiagonal matrix,



(2.18)

with n blocks, each of order n (or n^2), where n is the number of grid points in the x- and y-directions (or x-, y- and z-directions for three-dimensional problems). Thus the total number of unknowns is $N = n^2$ (or n^3). If we apply a suitable permutation $\hat{P}$ to the rows of A, we obtain

$$\hat{A} = \hat{P}A = \left[\begin{array}{cccccccccc} * & * & & & & & & & & \\ * & * & * & & & & & & & \\ & & & * & * & * & & & & \\ & & & & * & * & * & & & \\ \hline & * & * & * & & & & & & \\ & & * & * & * & & & & & \\ & & & & & & * & * & * & \\ & & & & & & & * & * & \end{array}\right] = \left[\begin{array}{c} B_{11}^{\mathrm{T}} \\ B_{12}^{\mathrm{T}} \\ \hline B_{21}^{\mathrm{T}} \\ B_{22}^{\mathrm{T}} \end{array}\right] = \left[\begin{array}{c} B_1^{\mathrm{T}} \\ B_2^{\mathrm{T}} \end{array}\right]. \tag{2.19}$$

From (2.19), we observe that B_i, $i = 1, 2$ has independent blocks or subsystems. This observation has two beneficial side-effects: first the linear least squares solution of each of the blocks can be obtained in parallel, and second, the order of the linear least squares problem being solved is much smaller, leading to a reduction in the memory requirement.

In this paper, we shall assume that the above block-row SSOR method is applied to the matrix $\hat{A}$ given in (2.19).

3. Acceleration of the block-SSOR method

Though the block-SSOR method with the optimal parameter $\omega = 1.0$ converges, the convergence rate can be quite slow. One way of improving the convergence rate of the method is by using an iterative scheme to accelerate the basic scheme (2.15) for solving

$$(I - Q)x = Rf \tag{3.1}$$

where the matrices Q and R are as defined above.

3.1. Conjugate gradient acceleration

Theorem 2.3 implies that, for $\omega = 1$, the matrix $(I - Q)$ is symmetric positive definite. Hence the system in (3.1) is well suited for acceleration using the conjugate gradient method.

Algorithm 3.1 (Conjugate Gradient Method)

Step 1: Choose x_0
Compute $r_0 = Rf - (I - Q)x_0$ and (r_0, r_0)
Set $p_0 = r_0$; Set $i = 0$

Step 2: Compute

$$\alpha_i = (r_i, r_i)/(p_i, (I - Q)p_i)$$
$$x_{i+1} = x_i + \alpha_i p_i$$
$$r_{i+1} = r_i - \alpha_i (I - Q) p_i$$
$$\beta_i = (r_{i+1}, r_{i+1})/(r_i, r_i)$$
$$p_{i+1} = r_{i+1} + \beta_i p_i$$

Step 3: If convergence criterion is satisfied, terminate the iterations; else set $i = i + 1$ and go to Step 2.

Note that the multiplication of the matrix $(I - Q)$ by a vector requires the solution of linear least squares problems. Throughout our experiments, we terminate the iterations whenever

$$\frac{\| f - Ax_i \|}{\| f - Ax_0 \|} \leqslant 10^{-6}. \tag{3.2}$$

Note that we calculate the relative norm for the original system (1.1). However one can also consider a convergence criterion of the form

$$\frac{\| r_i \|}{\| r_0 \|} \leqslant \epsilon \tag{3.3}$$

where r_i and r_0 are obtained from Algorithm 3.1 and ϵ is some small quantity. This criterion would not require the product of the matrix A by a vector, and can be obtained from Algorithm 3.1 at no extra cost. In Step 1, we choose the initial iterate as the zero vector.

Two more observations. First, as an alternative to the CG acceleration one could have used Chebyshev type methods [2,13,26,27,32,35,37]. These are iterative methods based on Chebyshev polynomials and can be used when the system has to be solved many times with different right-hand sides; a case which arises in certain time dependent problems. Second, one could use polynomial preconditioning in Algorithm 3.1 for further acceleration. This was not attempted here, however.

4. Numerical experiments

In this section, we present the numerical results obtained during the study of the behavior of our block-SSOR method. We first describe the partial differential equations that give rise to the model problems which are solved using this method. Unless otherwise stated, all the numerical experiments in this paper are carried out on one CPU of the CRAY X-MP system (machine epsilon ≈ 0.3E–14) at Boeing Computer Services. The compiler used is CFT version 1.11 which performs vectorization for only the most obvious cases. The results of the implementation of the algorithm on more than one CPU of the CRAY X-MP are presented in Section 5.

4.1. Generation of the linear system of equations

We shall restrict the study of the block-SSOR method to the case where the matrix A in (1.1) is obtained by the application of the finite difference method to a two-dimensional elliptic

partial differential equation of the form

$$-bu_{xx} - cu_{yy} + du_x + eu_y + fu = g \tag{4.1}$$

where

$$u = k(x, y), \quad x, y \in \partial R$$

and

$$R: \quad 0 \leqslant x \leqslant 1; \; 0 \leqslant y \leqslant 1.$$

The exact solution (assumed known) is used to calculate the right-hand side from the five-point centered differencing scheme. We assume that there are n interior grid points in both the x- and the y-directions; thus the total number of unknowns is $N = n^2$. The matrix A therefore has the structure given in (2.18), with each diagonal block being a tridiagonal matrix of order n and the off-diagonal blocks being diagonal matrices.

Model Problem 1 ([36])

$$-u_{xx} - \left[(1+xy)u_y\right]_y - \beta\left[\cos(x)u_x + (e^{-x}+x)u_y\right] + 3u = g,$$
$$u = x + y, \qquad \beta = 10000.0.$$

Model Problem 2 ([13])

$$-(e^{-xy}u_x)_x - (e^{xy}u_y)_y + \beta(x+y)u_y + [\beta(x+y)u]_y + \frac{1}{(1+x+y)}u = g,$$
$$u = x\, e^{xy} \sin(\pi) \sin(\pi y), \qquad \beta = 10000.0.$$

Model Problem 3

$$-u_{xx} - u_{yy} - xu_x + 200yu_y - 300u = g, \quad u = x + y.$$

In Figs. 1–9, we present the eigenvalue distributions for the matrix A, the symmetric part of

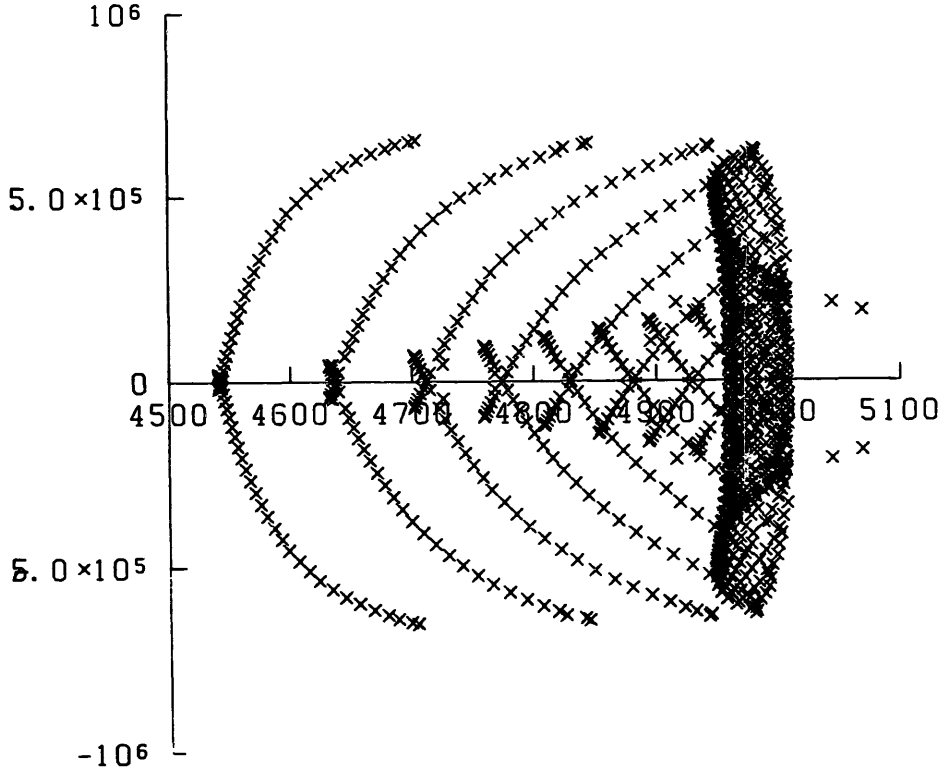


Fig. 1. Eigenvalue distribution of the matrix A for Model Problem 1, $n = 32$.

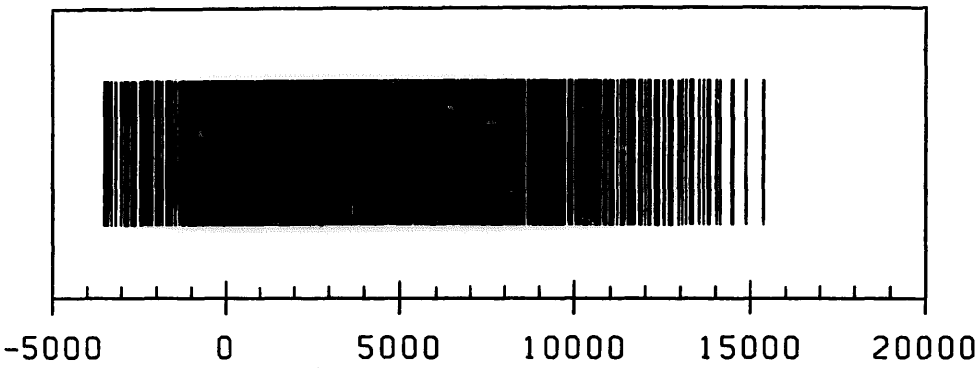


Fig. 2. Spectrum of the symmetric part of A for Model Problem 1, $n = 32$.

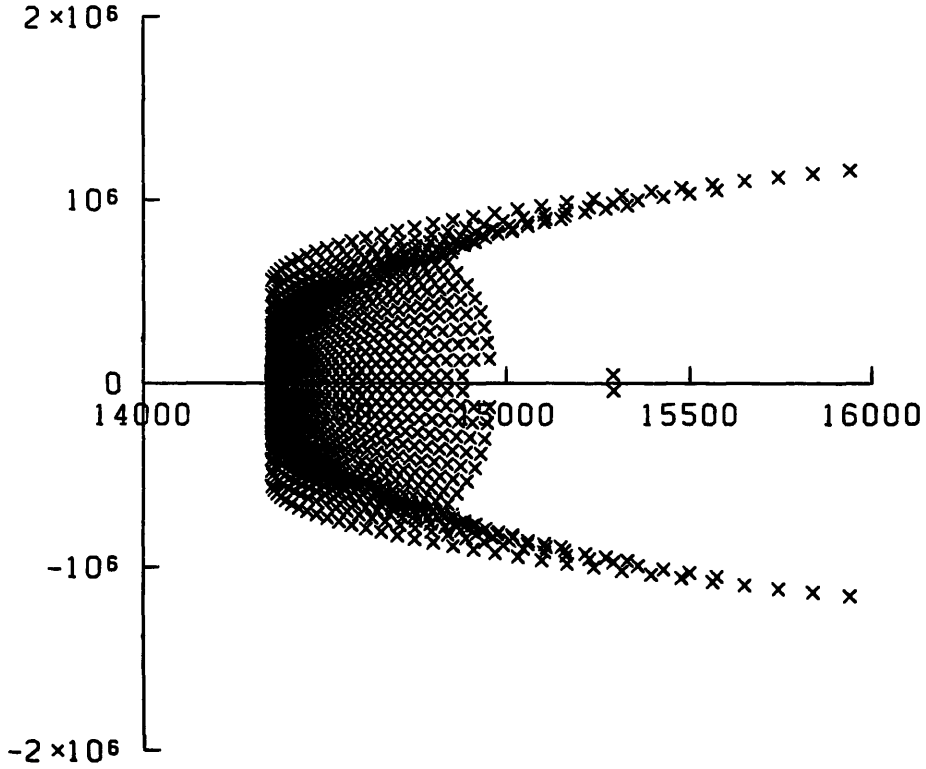


Fig. 3. Eigenvalue distribution of the matrix A for Model Problem 2, $n = 32$.

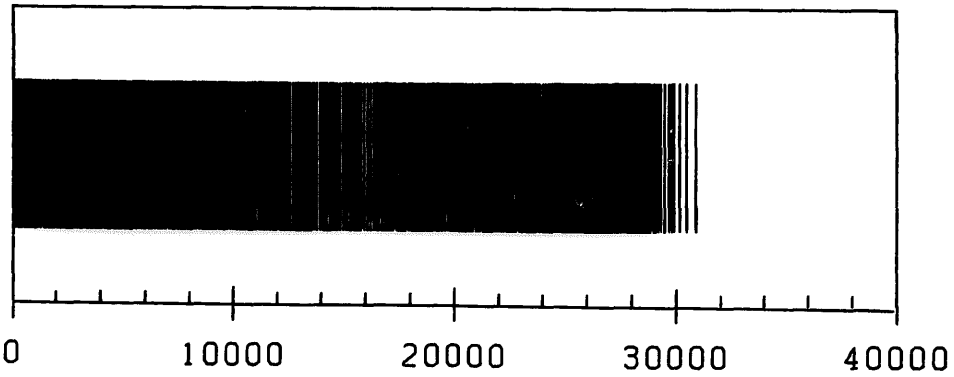


Fig. 4. Spectrum of the symmetric part of A for Model Problem 2, $n = 32$.

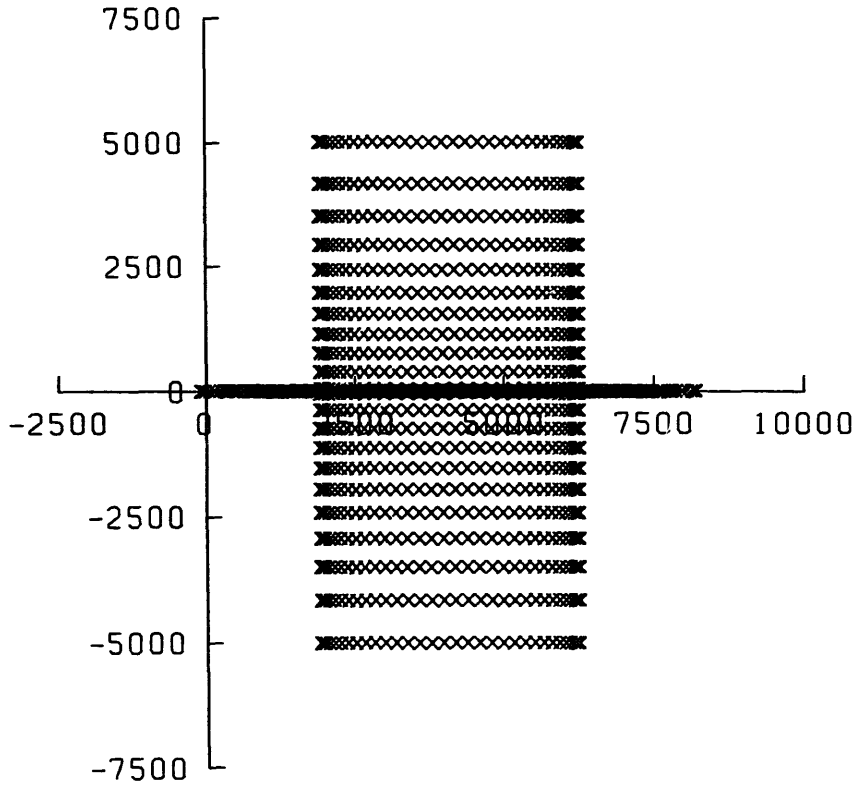


Fig. 5. Eigenvalue distribution of the matrix A for Model Problem 3, $n = 32$.

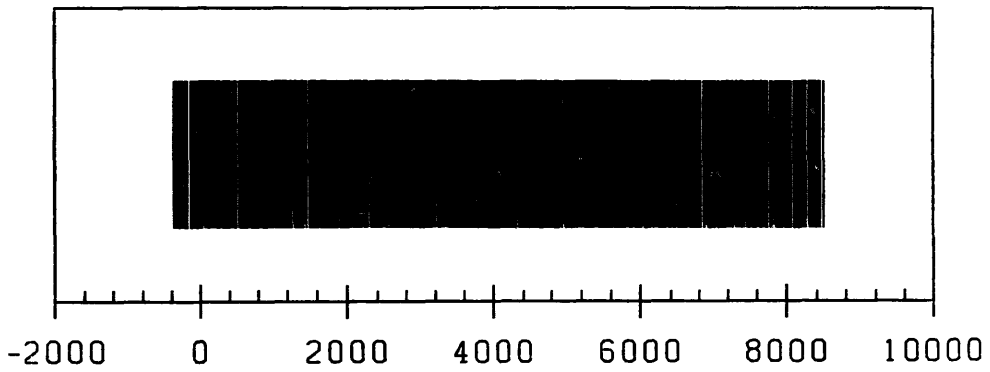


Fig. 6. Spectrum of the symmetric part of A for Model Problem 3, $n = 32$.

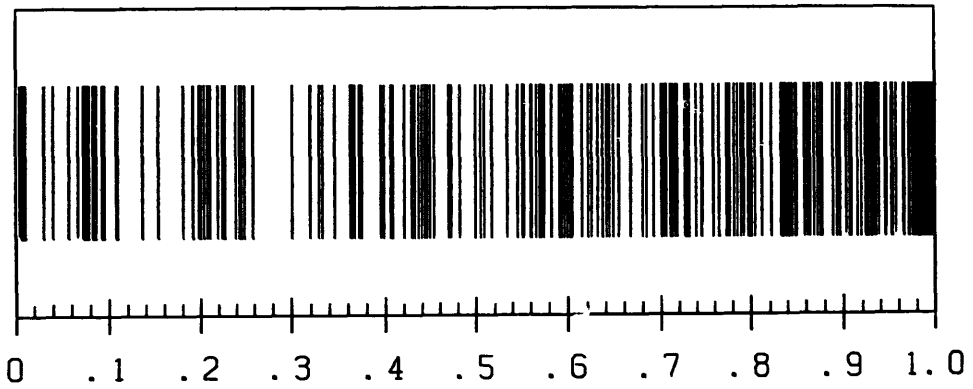


Fig. 7. Spectrum of the iteration matrix Q for Model Problem 1, $n = 32$.

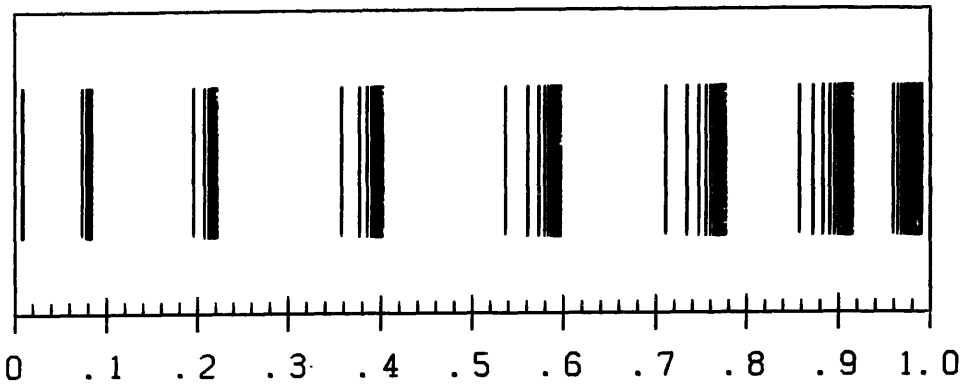


Fig. 8. Spectrum of the iteration matrix Q for Model Problem 2, $n = 32$.

Fig. 9. Spectrum of the iteration matrix Q for Model Problem 3, $n = 32$.

A and the iteration matrix Q for the three model problems with $n = 32$. Model Problems 1 and 2, both have matrices with eigenvalues in the right half plane, while the matrix of Problem 3 has eigenvalues on both sides of the imaginary axis. The symmetric part of the matrix is positive definite for Problem 2 and indefinite for Problems 1 and 3. Table 1 summarizes the estimate of the condition numbers of the model problems.

An important and time-consuming part of the block-SSOR method is the solution of the linear least squares problems. These problems are solved many times, with different right-hand sides; hence, an efficient method of solution would try to utilize the information obtained while solving the least squares problem the first time. This observation points towards the use of a direct method, at least for 2-D problems. The method of choice for solving such problems is via the normal equations. It is this approach that we adopt in handling the linear least squares calculations in Algorithm 2.1. We use routines LLT and LLTS due to Dongarra [9], which are the equivalent of LINPACK routines SPOFA and SPOSL but are more suitable for vector machines. With the partitioning scheme adopted in Section 2.4, the drawbacks usually associ-

Table 1
Estimate of the condition number for the matrices obtained from the model problems

Model problem	$n = 32$
Problem 1	0.9×10^2
Problem 2	0.9×10^2
Problem 3	0.2×10^5

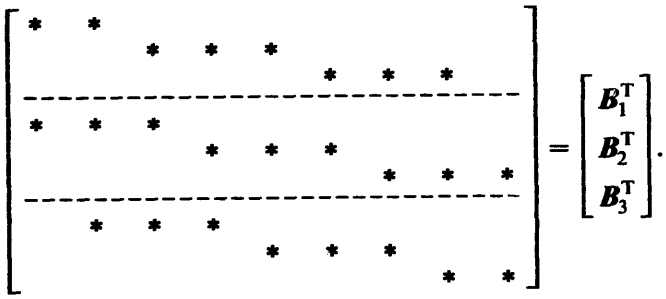
ated with the normal equations do not seem to influence the solution procedure adversely, [23]. The only drawback in this approach is the size of the memory required to store the Cholesky factors of the normal equations. Using the reordered matrix, (2.19), enables us to reduce the size of the linear least squares problems and hence reduce the size of the memory required. If one does not have enough fast memory available for the problem at hand, a different permutation of the rows of the matrix may be used, giving rise to smaller linear least squares problems (see Section 4.3). Other direct methods, such as orthogonal factorization, require more memory than the Cholesky factorization of the normal equations.

Another approach to solving the linear least squares problem is via iterative schemes such as LSQR [30]. Such an approach proved to be more costly than direct methods for the 2-D problems considered here. It becomes only advantageous when one deals with three-dimensional problems.

4.2. A different permutation

The permutation resulting in (2.19) is one of several that the user can utilize to obtain smaller independent subsystems. One could reduce the cost of solving the linear least squares problem by considering permutations that lead to submatrices each with less than $2n$ rows.

In this section, we consider a permutation of the rows of A that yields a matrix consisting of three partitions, B_1^T, B_2^T and B_3^T, each consisting of $n/3$ independent submatrices with n rows each, as shown below,



The block SSOR method for such an ordering is analogous to that given for the partitioning of A into two parts. In addition to the matrix of the normal equations being of order n, we have the added benefit that this matrix is now a pentadiagonal matrix, thus reducing storage considerably. We implemented the above permutation for $n = 36$ and compared it with the results obtained in the case of the permutation leading to two partitions in (2.19). These experiments were performed on the CRAY X-MP at NMFECC, using the CFT 1.14 version of the compiler. The results, both with and without vectorization, are presented in Table 2. For

Table 2
Comparison of results for two and three partitions, $n = 36$

Problem	With vectorization				Without vectorization			
	Two partitions		Three partitions		Two partitions		Three partitions	
	Iterations	Time (s)	Iterations	Time (s)	Iterations	Time (s)	Iterations	Time (s)
1	160	2.1348	221	3.0591	160	8.1180	221	4.9089
2	60	0.9216	53	0.8256	60	3.3235	53	1.2865
3	196	2.5823	260	3.5474	199	9.9960	264	5.8398

the 3 partition case, the linear least squares problems are solved using the LINPACK routines SPBFA and SPBSL.

These results lead to some rather interesting observations. While using two partitions is better with vectorization, lack of vectorization favors the use of three partitions. Also, in all but one of the model problems, the number of iterations required for convergence increases as the number of partitions increases. One might intuitively expect this result, though we do not have an explanation for the contrary result for Problem 2. This numerical experiment shows the versatility of the algorithm. Depending upon the size of the memory available, and the presence or absence of vectorization, the user can choose a permutation that makes the best use of the computational resources.

4.3. Comparison with other methods

In this section, we compare the performance of our algorithm with other methods such as the CG on the normal equations, and CG-like methods such as ORTHOMIN(k), GCR(k) and GMRES(k). We discuss the advantages and disadvantages of each method and the conditions under which it can be applied to solve a particular problem. With the aid of an example, we also show that our method is reliable and is guaranteed to converge even in cases where other competitive methods fail.

4.3.1. Preconditioned CG on the normal equations

One of the methods which can be used for the solution of a general nonsymmetric system of equations

$$Ax = f \tag{4.2}$$

is the CG method applied to the normal equations

$$A^{\mathrm{T}}Ax = A^{\mathrm{T}}f \tag{4.3}$$

directly or by considering an equivalent form of the method with better numerical properties [19]. We consider two preconditioners:

(i) Incomplete Cholesky factorization, IC(0), on the matrix $A^{\mathrm{T}}A$, [28];

(ii) Point SSOR on the matrix $A^{\mathrm{T}}A$, with $\omega = 1$, [3].

4.3.2. Yale-PCGPAK

The Yale package, PCGPAK, consists of the ORTHOMIN(k), GCR(k) and GMRES(k) methods for the iterative solution of sparse nonsymmetric systems of equations [11,13,33,34]. The package provides an option for no preconditioning as well as the ILU(s), MILU(s) and SSOR preconditioners.

ORTHOMIN(k) and GCR(k) are both modifications of the generalized conjugate residual method (GCR), which is modeled after the conjugate residual method. The cost (both storage and computational) of the GCR method may be prohibitively expensive due to the fact that all the direction vectors need to be stored. Modifications to the GCR method to reduce the cost lead to the ORTHOMIN(k) and GCR(k) methods. ORTHOMIN(k) is derived from the GCR method by using only the k previous vectors to evaluate the new direction vector p_{i+1}. GCR(k), on the other hand, saves storage and computation time by restarting the iteration periodically; every $(k+1)$ iterations, the current iterate is considered as the new starting guess.

The GMRES method, which is a generalization of the MINRES algorithm of Paige and Saunders [29], obtains a solution of the form $x_0 + z$ (x_0: initial iterate) which minimizes the residual norm over z in the Krylov space $K_k = \mathrm{span}\{v_1, Av_1, \ldots, A^{k-1}v_1\}$ ($v_1 = r_0/\|r_0\|$; $r_0 = f - Ax_0$). The GMRES(k) method is just the GMRES algorithm restarted every k iterations.

Both ORTHOMIN(k) and GCR(k) are known only to be convergent for problems in which the symmetric part is positive definite. For large enough k, GMRES(k) is convergent for arbitrary nonsingular problems; however, an increase in k also implies an increase in the memory required and the computational work done. Though GMRES(k) does not break down, it may produce residuals which do not converge to zero, even though their 2-norms are nonincreasing. In other words, the restarted GMRES method may become stationary.

Each of the above three methods, i.e. ORTHOMIN(k), GCR(k) and GMRES(k) can also be combined with a preconditioner to improve its convergence. In our experiments with PCGPAK, we consider left oriented preconditioners and stop the iterations whenever

$$\frac{\| r_i \|_2}{\| r_0 \|_2} < 10^{-6}$$

where r_i is the residual of the original system (except for the case of GMRES(k) where the package evaluates the residual of the preconditioned system). We also monitor the residual by requiring that the iterations be stopped if the decrease in the residual size is less than 10^{-5} after 5 iterations. In the case of the ILU and MILU preconditioners, we consider the parameters s and α to be assigned the value 0. The parameter s governs the fill-in for the incomplete factorization and α governs the term defining the diagonal element. The choice of 0 for these parameters has been observed to be a reasonable choice in most cases [5,13].

4.3.3. Results

In this section, we present the numerical results obtained by comparing the CG accelerated block-SSOR method with other competitive methods. Tables 3–5 present the results for our model problems for $n = 32$. The last column in each table comments on the results obtained. "Error 8" indicates that the residual stagnates, and "maxm. iter." denotes that the method fails to converge. It should be mentioned that none of these codes has been well optimized for one CPU of the CRAY X-MP, the machine on which these experiments have been performed.

As seen from the numerical results, the block-SSOR method compared quite favorably with the preconditioned CG scheme on the normal equations but is much slower than the methods given in PCGPAK. Later, however, we present a fourth model problem which causes the routines in PCGPAK to fail for reasonable values of the parameters k and s. A summary of our observations follows:

(i) *Normal equations:* Except for Problem 2, the block SSOR method with CG acceleration requires less time than the normal equations with both the IC(0) and the SSOR precondition-

Table 3
Comparison of different methods, Model Problem 1, $n = 32$

Method	Iterations	Time (s)	Comments
Block SSOR&CG	167	1.3125	–
CGNE with IC(0)	155	5.1915	–
CGNE with SSOR	174	5.7786	–
ORTHOMIN(3)	110	0.4668	diverges
GCR(3)	300	1.1668	maxm. iter.
GMRES(3)	300	1.1708	maxm. iter.
ORTHOMIN(3) & ILU	20	0.3681	diverges
GCR(3) & ILU	300	4.4731	diverges
GMRES(3) & ILU	4	0.0919	–
ORTHOMIN(3) & MILU	1	0.0779	–
GCR(3) & MILU	1	0.0780	–
GMRES(3) & MILU	1	0.0673	–

Table 4
Comparison of different methods, Model Problem 2, $n = 32$

Method	Iterations	Time (s)	Comments
Block SSOR&CG	**51**	**0.5446**	-
CGNE with IC(0)	5	0.3372	-
CGNE with SSOR	74	2.5566	-
ORTHOMIN(3)	95	0.4323	diverges
GCR(3)	300	1.2056	maxm. iter.
GMRES(3)	300	1.2106	maxm. iter.
ORTHOMIN(3) & ILU	5	0.1671	-
GCR(3) & ILU	5	0.1668	-
GMRES(3) & ILU	5	0.1338	-
ORTHOMIN(3) & MILU	4	0.1553	-
GCR(3) & MILU	**4**	**0.1553**	-
GMRES(3) & MILU	**4**	**0.1269**	-

ing. While the use of the normal equations can be both practical and easy to implement, it does suffer from the drawback that the number of iterations required for convergence can be quite large when the matrix is ill-conditioned.

(ii) *PCGPAK with no preconditioning:* The results with the three methods—ORTHOMIN(3), GCR(3) and GMRES(3)—without any preconditioning show that these methods do not converge for any of our model problems. For Model Problems 1 and 3, ORTHOMIN and GCR are not expected to converge since these problems have an indefinite symmetric part. One possible explanation of the results could be that the value of k used was not large enough for these methods to converge.

(iii) *PCGPAK with* ILU(0) *and* MILU(0) *preconditioning:* The use of MILU(0) preconditioning appears to be better than the use of ILU(0) preconditioning for our model problems. In fact, for some of the model problems we find that the MILU(0) preconditioner provides a very good approximation to the matrix under consideration and thus the iterative scheme acts as an iterative refinement of the method, leading to convergence in one iteration. Among the three methods—ORTHOMIN(k), GCR(k) and GMRES(k)—the performance of GMRES(k) seems to be better than the other two. This is presumably due to the fact that GMRES(k) is applicable to all types of problems, whereas ORTHOMIN(k) and GCR(k) are restricted to matrices whose symmetric part is positive definite. GMRES(k) is also cheaper than the other two methods in terms of the work per iteration.

Table 5
Comparison of different methods, Model Problem 3, $n = 32$

Method	Iterations	Time (s)	Comments
Block SSOR&CG	**153**	**1.2105**	-
CGNE with IC(0)	105	3.5534	-
CGNE with SSOR	254	8.3626	-
ORTHOMIN(3)	95	0.3898	diverges
GCR(3)	300	1.1555	maxm. iter.
GMRES(3)	300	1.1599	maxm. iter.
ORTHOMIN(3) & ILU	83	1.2909	diverges
GCR(3) & ILU	300	3.3694	maxm. iter.
GMRES(3) & ILU	29	0.3109	error 8
ORTHOMIN(3) & MILU	1	0.0690	-
GCR(3) & MILU	**1**	**0.0690**	-
GMRES(3) & MILU	**1**	**0.0578**	-

From the remarks above it appears that the methods in PCGPAK, in particular GMRES(k), provide a quick and efficient way of solving a general linear system of equations. The question might then arise as to why one would like to consider other methods for the solution of a nonsymmetric linear system.

From the results presented, we see that although the block-SSOR method with CG acceleration works for all kinds of problems, it can be much slower than PCGPAK, on a vector machine such as the CRAY X-MP, especially in cases where the algorithms in PCGPAK act as iterative refinement schemes. There are certain drawbacks to the methods in PCGPAK, however. These are outlined below:

(i) *User input:* While the block-SSOR method requires no input from the user (as the optimal value of ω is 1.0), PCGPAK requires that the user select not only a method of solution and a preconditioner but also selects a value for k and the parameters for the preconditioner. The user thus has to be aware of the properties of the matrix and therefore cannot use the package as a black-box. As we have observed, the choice of k may not be an easy one and since there is no way to determine the least value of k required for convergence, the user may have to try different values before reaching convergence. In addition, the user may have to try different combinations of methods and preconditioners before finding one that works. One must also keep in kind that as k increases, the memory requirement increases as kN and computation increases as k^2N, where N is the number of equations.

(ii) *Multiprocessor implementation:* As shown in Section 2.3 (and elaborated upon in the next section), the block-SSOR method is very suitable for implementation on a multiprocessor. With the current trend towards the use of parallel computers this would be a desirable property for an algorithm to possess. The routines in the present version of PCGPAK, however, are not very suitable for implementation on a multiprocessor, especially when used with preconditioners of the incomplete LU form.

(iii) *Reliability of the method:* For the above three model problems, we found that both PCGPAK and block-SSOR converge to the correct solution of the problem. As remarked earlier, we also found that the methods in PCGPAK work best when used in conjunction with either the ILU(0) or the MILU(0) preconditioners. However, for certain problems, with certain mesh sizes, these preconditioners can lead to instability causing the residuals to either stagnate or not reduce sufficiently to ensure convergence after a reasonable number of iterations [14]. Elman suggests the use of upwind differencing to get a stable preconditioner; this would however reduce the accuracy of the computed solution. While the model problems we considered above, converged for some combinations in PCGPAK, we present the following model problem as an example to show that PCGPAK may fail for certain problems.

Model Problem 4

$$-u_{xx} - u_{yy} + 1000\, e^{xy} u_x - 1000\, e^{xy} u_y = g, \quad u = x + y.$$

For $n = 32$, the eigenvalues of A lie in the right half plane and the symmetric part of A has a small percentage of negative eigenvalues. Table 6 gives the results obtained using PCGPAK, CG on normal equations with IC(0) and the above CG-accelerated block-SSOR on this problem. The methods which converge are highlighted in bold type. As observed, PCGPAK failed to converge even when the value of k is 20, while the block-SSOR method converged. It must be mentioned that the conditions for convergence for the methods in PCGPAK are governed by the eigenvalue distribution of the preconditioned system and not the original

Table 6
Comparison of methods for Model Problem 4

n	Method	Prec.	Iter.	Time (s)	Comments
32	**Block-SSOR**	**-**	**69**	**0.6280**	**-**
32	**CGNE**	**IC(0)**	**89**	**3.1667**	**-**
32	ORTHOMIN(10)	ILU	17	0.1826	Error 8
32	ORTHOMIN(10)	MILU	17	0.2053	Error 8
32	GCR(10)	MILU	49	0.5033	Error 8
32	GMRES(10)	MILU	400	3.8663	Max. iter.
32	GMRES(10)	ILU	16	0.1986	Error 8
32	GMRES(10)	SSOR	27	0.2726	Error 8
32	GMRES(20)	MILU	400	4.0255	Max. iter.
48	**Block-SSOR**	**-**	**99**	**2.1499**	**-**
48	GMRES(10)	MILU	345	7.4624	Error 8
64	**Block-SSOR**	**-**	**127**	**5.2454**	**-**
64	GMRES(10)	MILU	123	4.8580	Error 8

system. Also, CG on normal equations with IC(0) converges, but is more time-consuming than the block-SSOR method.

5. Multiprocessor implementation

In this section, we present the implementation of our algorithm on the CRAY X-MP/48 at Cray Research, Mendota Heights. This system, [6], has four processors which share an 8 Mwords central bipolar memory organized in 32 interleaved memory banks. All the banks can be accessed independently and in parallel during each machine clock period. All the processors are controlled synchronously by a central clock with a cycle time of 9.5 ns. The processors are identical and symmetric in their programming functions, i.e. there is no permanent master/slave relation among them.

In addition to the vector capability of each processor, it is also possible to run related tasks of a single job on multiple processors (multitasking), with loosely coupled tasks communicating through shared memory and tightly coupled tasks communicating through shared registers. Depending on the granularity of tasks and the software implementation techniques, the overhead of task initiation is small—O(1 μs) to O(1 ms).

Multitasking is the structuring of a program into two or more tasks that can execute concurrently. A task is a unit of computation that can be scheduled [8]. There are basically two approaches to the implementation of an algorithm on a multiprocessor: macrotasking and microtasking. As the name suggests, the task size in macrotasking is very large. In the case of the CRAY X-MP/48, macrotasking has to be done by the user by explicitly splitting the job into tasks, which may be difficult as one has to achieve data independence. The use of macrotasking entails the declaration of shared variables in task common, explicit synchronization of processors and the use of critical regions to ensure that reading/writing of data is taking place as required by the code. Needless to say, this could involve a substantial change in the program and reduce the portability of the code due to the calls to the multitasking routines. Microtasking, on the other hand, does not require the task size to be very large, its overhead is smaller, and implementation on more than one processor is done through compiler directives, which retains the portability of the code. Microtasking, in addition to being easier to implement, also provides the possibility of being automated [25].

In this paper, we adopt the latter approach towards multitasking, namely, microtasking. The CRAY X-MP/48 provides compiler options and compiler directives for vectorizing the code

and for declaring certain parts of the code to be executed concurrently. Variables in a COMMON block, the argument list, SAVE statement and DATA statement are all considered as global variables. All the remaining variables are put in a stack, including the TASK-COMMON. Local variables exist in the stack for each processor which enters a microtasked subroutine.

5.1. Implementation of the algorithm

The main aspect of the implementation of our code on the multiprocessor is the evaluation of the matrix-vector product

$$y = (I - Q)x$$

where the matrix Q is defined by (2.15). This involves the solution of independent linear least squares problems. These may be solved on 4 processors simultaneously, with each processor working on an independent problem. Since this is a major time-consuming part of our algorithm, efficient implementation on the CRAY X-MP is of paramount importance.

Another aspect to the implementation of the CG algorithm on a multiprocessor is the convergence criterion. The convergence criterion used requires the evaluation of the residual of the original system

$$\| f - Ax \|_2$$

which is the same as

$$\| \hat{P}f - \hat{P}Ax \|_2$$

where $\hat{P}$ is a permutation matrix. Since the matrix $\hat{P}A$ has independent blocks, the required matrix-vector multiplication can be done in parallel.

From the implementation of the algorithm described above, we see that a large percentage of the code can be run in parallel. Thus one could expect a reasonably high speedup, when the code is run on more than one processor.

5.2. Experimental results

In this section, we present the results obtained from the implementation of the code on the CRAY X-MP/48. Tables 7–9 present the results for Model Problems 1, 2 and 3, for $n = 96$. These results show how the speedup and computational rate vary as the number of processors are varied. The code is run on 1, 2 and 4 processors in a dedicated mode. The time is measured in seconds using the routine IRTC. The iteration time is the time spent in the CG iterations and excludes any time spent in setting up the problem. The total time is the time for the execution of the whole code.

Table 7
Performance of Model Problem 1 on the CRAY X-MP/48

n	#processors	Iter. time	Total time	Speedup(i)	Speedup(t)	Megaflops
96	1	24.87	27.38	1.0	1.0	62.90
	2	12.69	13.96	1.96	1.96	123.37
	4	6.79	7.44	3.66	3.68	231.48

Table 8
Performance of Model Problem 2 on the CRAY X-MP/48

n	#processors	Iter. time	Total time	Speedup(i)	Speedup(t)	Megaflops
96	1	15.99	19.70	1.0	1.0	61.88
	2	8.68	10.05	1.96	1.96	121.30
	4	4.64	5.35	3.66	3.68	227.85

Table 9
Performance of Model Problem 3 on the CRAY X-MP/48

n	#processors	Iter. time	Total time	Speedup(i)	Speedup(t)	Megaflops
96	1	85.07	87.54	1.0	1.0	63.75
	2	43.56	44.81	1.95	1.95	124.54
	4	23.49	24.13	3.62	3.63	231.28

The speedup is defined as

$$\text{Speedup} = \frac{\text{Execution time on one processor}}{\text{Execution time on } p \text{ processors}}.$$

Speedup(i) refers to the speedup obtained by considering only the time spent in the CG iterations and speedup(t) refers to the speedup obtained by considering the total time. The computational rate (in Megaflops) for the whole code is obtained from the routine FLOPTRACE, which also gives the user megaflop rating of each subroutine.

From the results presented, we see that the CG-accelerated block-SSOR method is well suited for implementation on a multiprocessor. More work need to be done, however, in making better use of the vector capability of each CPU of the CRAY X-MP/48. For reasonable size problems, we obtain approximate speedups of 1.95 for 2 processors and 3.6 for 4 processors. We also obtain a computational rate in the range of 170–240 Megaflops for 4 processors. Note that the size of the problems chosen is such that the number of linear least squares problems being solved is distributed equally among the processors, resulting in no idle processors.

6. Conclusions

In this paper, we presented an iterative scheme for the solution of a large sparse nonsymmetric system of linear equations with a general eigenvalue distribution. We saw that a simple permutation of the rows of the matrix, made the CG-accelerated block-SSOR scheme very suitable for implementation on multiprocessors.

Our numerical experiments showed that the scheme is reliable, requires no user input, and performs relatively well on the 4-processor CRAY X-MP. There is, however, room for improvement. This would be brought about by the use of fast, efficient linear least squares solvers, as well as by efficient implementation of the algorithms through better management of the vector registers in each CPU.

Acknowledgment

We wish to thank Jack Dongarra, John Larson, Youcef Saad, and the referees for their helpful suggestions, and Cray Research Inc. for the use of the CRAY X-MP/48. C. Kamath would also like to acknowledge the support provided by an IBM graduate fellowship.

References

[1] R. Ansorge, Connections between the Cimmino method and the Kaczmarz method for the solution of singular and regular systems of equations, *Computing* **33** (1984) 367–375.
[2] S.F. Ashby, CHEBYCODE: A Fortran implementation of Mantueffel's adaptive Chebyshev algorithm, M.S. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1985.
[3] A. Bjorck and T. Elfving, Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations, *BIT* **19** (1979) 145–163.
[4] E. Bodewig, *Matrix Calculus* (North-Holland, Amsterdam, 1959).
[5] R. Chandra, Conjugate gradient methods for partial differential equations, Yale University Research Report 129, 1978.
[6] S.S. Chen, Large-scale and high-speed multiprocessor system for scientific applications: Cray X-MP series, in: J.S. Kowalik, ed., *High Speed Computation* (Springer, Berlin, 1984) 59–67.
[7] G. Cimmino, Calcolo approssimato per le soluzione di sistemi di equazioni lineari, *Ricerca Sci. II* **9** (I) (1938) 326–333.
[8] Cray Computer Systems Technical Note SN-0222, Multitasking User Guide, 1985.
[9] J.J. Dongarra, Private Communication.
[10] J. Dyer, Acceleration of the convergence of the Kaczmarz method and iterated homogeneous transformations, Ph.D. Thesis, Department of Mathematics, University of California, Los Angeles, 1965.
[11] S.C. Eisenstat, H.C. Elman and M.H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.* **20** (1983) 345–357.
[12] T. Elfving, Block iterative methods for consistent and inconsistent linear equations, *Numer. Math.* **35** (1980) 1–12.
[13] H.C. Elman, Iterative methods for large, sparse, nonsymmetric systems of linear equations, Yale University Research Report 229, 1982.
[14] H.C. Elman, A stability analysis of incomplete LU factorizations, *Math. Comp.* **47** (1986) 191–217.
[15] N. Gastinel, Procédé iteratif pour la résolution numérique d'un systeme d'équations linéares, *Compte Rendue* **246** (1958) 2571–2574.
[16] N. Gastinel, *Linear Numerical Analysis* (Academic Press, New York, 1980); Translated from the original French text *Analyse Numérique Linéaire* (Hermann, Paris, 1966).
[17] R. Gordon, R. Bender and G.T. Herman, Algebraic reconstruction photography, *J. Theor. Biol.* **29** (1970) 471–481.
[18] R. Gordon and G.T. Herman, Three-dimensional reconstruction from projections, a review of algorithms, *Internat. Rev. Cytologi* **38** (1974) 111–115.
[19] M.R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards* **49** (6) (1952) 409–436.
[20] A.S. Householder and F.L. Bauer, On certain iterative methods for solving linear systems, *Numer. Math.* **2** (1960) 55–59.
[21] A.S. Householder, *The Theory of Matrices in Numerical Analysis* (Dover, New York, 1964).
[22] S. Kaczmarz, Angenäherte Auflösung von Systemen linearer Gleichungen, *Bull. Internat. Acad. Polon. Sci. Cl. A* (1937) 355–357.
[23] C. Kamath, Solution of nonsymmetric systems of equations on a multiprocessor, Ph.D. Thesis, University of Illinois, Center for Supercomputing Research and Development, Report No. 591, 1986.
[24] A.S. Kydes and R.P. Tewarson, An iterative method for solving partitioned linear equations, *Computing* **15** (1975) 357–363.
[25] J.L. Larson, Private Communication.
[26] T.A. Mantueffel, The Tchebychev iteration for nonsymmetric linear systems, *Numer. Math.* **28** (1977) 307–327.
[27] T.A. Mantueffel, Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration, *Numer. Math.* **31** (1978) 183–208.
[28] J.A. Meijerink and H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comp.* **31** (137) (1977) 148–162.

[29] C.C. Paige and M.A. Saunders, Solution of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* **12** (4) (1975) 617–629.

[30] C.C. Paige and M.A. Saunders, LSQR: An algorithm for sparse linear equations and sparse least squares, *ACM Trans. Math. Software* **8** (1) (1982) 43–71.

[31] W. Peters, Lösung linearer Gleichungssysteme durch Projektion and Schnittraume von Hyperebenen und Berechnung einer verallgemeinerten Inversen, *Beitrage Numer. Math.* **5** (1976) 129–146.

[32] Y. Saad and A. Sameh, Iterative methods for the solution of elliptic difference equations on multiprocessors, in: *CONPAR 81*, Lecture Notes in Computer Science **111** (Springer, Berlin, 1981) 395–413.

[33] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* **7** (1986) 856–869.

[34] Y. Saad and M.H. Schultz, Conjugate gradient-like algorithms for solving nonsymmetric linear systems, Yale University Research Report 283, 1983.

[35] Y. Saad, A. Sameh and P. Saylor, Solving elliptic difference equations on a linear array of processors, *SIAM J. Sci. Statist. Comput.* **6** (1985) 1049–1063.

[36] A.H. Sherman, An empirical investigation of methods for nonsymmetric linear systems, in: M.H. Schultz, ed., *Elliptic Problem Solvers* (Academic Press, New York, 1981) 429–434.

[37] E. Stiefel, Kernel polynomials in linear algebra and their numerical applications, *Nat. Bur. Standards, Appl. Math. Series* **49** (1958) 1–22.

[38] K. Tanabe, Projection methods for solving a singular system of linear equations and its application, *Numer. Math.* **17** (1971) 203–214.

[39] K. Tanabe, Characterization of linear stationary iterative processes for solving a singular system of linear equations, *Numer. Math.* **22** (1974) 349–359.

[40] R.P. Tewarson, Projection methods for solving sparse linear systems, *Computer J.* **12** (1969) 77–80.

[41] C.B. Tompkins, Methods of steep descent, in: E.F. Beckenbach, ed., *Modern Mathematics for the Engineer* (McGraw-Hill, New York, 1956) Chapter 18.

[42] M.R. Trummer, A note on the ART of relaxation, *Computing* **33** (1984) 349–352.

[43] R.L. Wainwright and R.F. Keller, Algorithms for projection methods for solving linear systems of equations, *Comput. Math. Appl.* **3** (1977) 235–245.

[44] D.M. Young, *Iterative Solution of Large Linear Systems* (Academic Press, New York, 1971).