ELSEVIER

Parallel Computing 22 (1996) 623–641

PARALLEL COMPUTING

Parallel iterative S-step methods for unsymmetric linear systems [1]

A.T. Chronopoulos [a,*], C.D. Swanson [b]

[a] *Computer Science Department, Wayne State University, State Hall 431, 5143 Cass Avenue, Detroit, MI 48202, USA*

[b] *Cray Research, Inc., 655A Lone Oak Drive, Eagan, MN 55121 USA*

Received 2 May 1995; revised 23 August 1995

Abstract

GCR (Generalized Conjugate Residual) and *Omin* (Orthomin) are iterative methods for approximating the solution of unsymmetric linear systems. The S-step generalization of these methods has been derived and studied in past work. The S-step methods exhibit improved convergence properties. Also, their data locality and parallel properties are enhanced by forming blocks of s search direction vectors. However, s is limited (to $s \leq 5$) by numerical stability considerations. The following new contributions are described in this article. The Modified Gram–Schmidt method is used to $A^T A$-orthogonalize the s direction vectors within each S-step block. It is empirically shown that use of values of s, up to $s = 16$, preserves the numerical stability of the new iterative methods. Finally, the new S-step Omin, implemented on the CRAY C90, attained an execution rate greater than 10 *Gflops* (Billion Floating Point Operations per sec).

Keywords: Iterative methods; S-step Orthomin; Modified Gram–Schmidt; Cray C90

1. Introduction

We consider a linear system of equations

$$Ax = f \tag{1}$$

where A is a *real unsymmetric matrix of size n*. In this article we use modified Gram–Schmidt to orthonormalize the direction vector blocks in the S-step Omin (GCR)

* Corresponding author. Email: chronos@cs.wayne.edu.

[1] This work was supported in part by NSF under grant CCR-9496327. Cray Research, Inc. and the Pittsburgh Supercomputing Center provided the CRAY C90 time.

(introduced in [8]). This new approach shows (empirically) that s can be extended up to 16 without loss of numerical stability. The new methods have still attractive parallel properties.

Several algorithms which improve the data locality for dense linear algebra problems exist for shared memory systems [12,14]. These algorithms are coded using the Basic Linear Algebra Subroutines (*BLAS*) [1,12]. Three different classes of BLAS exist: BLAS1 (based on single vector operations), BLAS2 (based on matrix times vector operations), BLAS3 (based on matrix times matrix operations). BLAS can be coded as high performance kernels on shared memory computers either with vector registers (BLAS1 and BLAS2) or local memory (or cache) (BLAS3). One important advantage of these algorithms over the standard ones is their *low ratio of memory references over floating point operations*. This allows efficient use of *vector registers* and *local memories*. It also enhances parallelism by reducing the need for frequent synchronizations of the processors.

In the area of iterative methods for solving linear systems, BLAS1 or BLAS2 module implementations consisting of one or more single vector operations have been studied in [10–12,25,30]. The S-step iterative methods [6–8,16–18] can be expressed in terms of BLAS2 and BLAS3 operations. The S-step methods form (at each iteration) a block of s independent direction vectors using repeated matrix-vector products of the coefficient matrix with a single residual vector. Then the solution is advanced simultaneously using the s direction vectors. Compared to the standard methods the S-step methods have improved data locality properties and parallelism. However, s is limited to $s \leq 5$ due to numerical stability considerations. An alternative approach to the S-step methods is offered by the block methods. The block methods use many independent *initial residual* vectors and they can also be expressed in terms of BLAS2 and BLAS3 operations. However, finding several independent initial residual vectors is a very difficult task. Some representative recent references in this area are [5,22–24].

The main goal of this work was to use the Modified Gram–Schmidt method to orthonormalize the direction vectors within each block of the S-step (GCR) Orthomin. The new methods have slightly worse data locality properties and slightly more operations than the S-step methods. However, the block size s can be increased (up to $s = 16$) without affecting the numerical stability of the method. Thus, the overall parallelism is enhanced. We studied the convergence and robustness of the new methods and implemented them on a 16 processor CRAY C90. We ran tests with a linear system arising in the discretization of a (2-dimensional) partial differential equation at a sustained execution rate greater than 10 GFlops.

The following notation will be used throughout the article. The *transpose* of the matrix A will be denoted as A^T. Lower case *Greek* characters will denote *scalars or real functions* and lower case *English* letters will denote *vectors* except for i,j,k,l,m,n,s which will denote *positive integers*. If the *symmetric part* of A (i.e., $A^T + A/2$) is *positive definite* then the matrix A will be called *definite*; otherwise it will be called *indefinite*. We also call the linear system with an *(in)definite* coefficient matrix an *(in)definite* linear system. We define the *minimal polynomial* of a nonzero vector v with respect to matrix A to be the least degree monic polynomial $q_k(\lambda)$ such that $q_k(A)v = 0$. The lth power of the matrix A is denoted by A^l.

The article follows the following structure. In Section 2, the orthogonal S-step methods are presented. In Section 3, the convergence and robustness of the new methods are discussed. In Section 4, an unsymmetric linear system arising in the discretization of a partial differential equation is presented. In Section 5, a parallel preconditioner (proposed in the past and used here) is outlined. In Section 6, the implementation of the new methods on a shared memory multiprocessor using BLAS is discussed. In Section 7, the results of the parallel implementation are discussed. Section 8 contains a summary and conclusions.

2. Orthogonal S-step GCR and S-step Omin

Omin and GCR [13,27] are iterative methods which apply to unsymmetric definite linear systems. The S-step Orthomin and S-step GCR have been derived and studied in [8]. For $s=1$, these methods are identical to Omin and GCR respectively. However, for $s>1$ they are more powerful in terms of their convergence and parallel properties.

In order to understand these S-step methods, we first outline an s-dimensional steepest descent method called the S-step Minimal Residual (*MR*) method. Let x_1 be an initial guess to the solution of (1) and let $r_1=f-Ax_1$ be the initial residual. For $i=1,\dots,$ S-step MR computes a *block* of direction vectors denoted by the *matrix* $[r_i, Ar_i,\dots,A^{s-1}r_i]$ (of dimension $n\times s$) and uses them to update the solution vector

$$x_{i+1}=x_i+\alpha_i^1 r_i+\dots+\alpha_i^s A^{s-1}r_i,$$

where α_i^j are the steplengths that minimize $\|r_{i+1}\|_2$ over the affine Krylov subspace

$$\left\{x_i+\sum_{j=1}^{s}\alpha_j A^{j-1}r_i : \alpha_j \text{ scalars and } r_i=f-Ax_i\right\}.$$

This method is theoretically equivalent to GMRES(s) [8]. S-step MR is not stable as s increases because of loss of orthogonality of the direction vectors used. However, it is useful in understanding other Krylov subspace iterative methods.

For integers i,k and $1\le i,k$, let $j_i=1$ for S-step GCR and $j_i=\max(1,i-k+1)$ for S-step Omin(k). In S-step GCR and S-step Omin(k), each iteration generates a *block* of s direction vectors, which are denoted by the *matrix* $P_i=[p_i^1,\dots,p_i^s]$. Here, unlike S-step MR, these blocks are created to be $A^{\mathrm{T}}A$-orthogonal. P_i is obtained from the column vectors $[r_i, Ar_i,\dots,A^{s-1}r_i]$, by simultaneously $A^{\mathrm{T}}A$-orthogonalizing them against the preceding blocks of direction vectors $\{[p_j^1,\dots,p_j^s]\}$, for $j_i\le j\le i-1$. However, the direction vectors within each block (P_i) are not $A^{\mathrm{T}}A$-orthogonalized. The norm of the residual $\|r_{i+1}\|_2$ is minimized simultaneously in all s new direction vectors in order to obtain x_{i+1} [8].

Here, we propose the following modification to the S-step GCR and S-step Omin(k). At each iteration i, we apply the Modified Gram–Schmidt method (*MGS*) (see [3]) to orthogonalize the column vectors of the matrix AP_i. This yields the orthogonal S-step methods. The orthogonal S-step GCR (Omin(k)) will be denoted by *OSGCR (OSOmin(s,k))*.

The following notation facilitates the description of the algorithm.

- The vector $\underline{\alpha}_i = [\alpha_i^1, \ldots, \alpha_i^s]^T$ (of dimension s) are the steplengths that minimize $\|r_{i+1}\|_2$ over the affine Krylov subspace
$$\left\{ x_i + \sum_{j=1}^{s} \alpha_j A p_i^j : \alpha_j \text{ scalars} \right\}.$$
- For indices $l = 1, \ldots, s$ and $j_i \le j \le i$, $\underline{\beta}_j^l = [\beta_j^{(l,1)}, \ldots, \beta_j^{(l,s)}]^T$ are vectors (of dimension s) of parameters used in orthogonalizing AP_i against AP_j.

For integers i,k and $1 \le i,k$, let $j_i = 1$ for OSGCR and $j_i = \max(1, i-k+1)$ for OSOmin(s,k). We summarize OSGCR and OSOmin(s,k) in the following algorithm.

Algorithm 1. OSGCR and OSOmin(s,k)

```
Compute $r_1 = f - Ax_1$.
**For** $i = 1, \ldots$ **until** convergence **do**
    1. Compute $AP_i = [Ar_i, A^2 r_i, \ldots, A^s r_i]$ and set $P_i = [r_i, Ar_i, \ldots, A^{s-1} r_i]$ **If** $(1 < i)$ **then**
    2. compute $\underline{\beta}_j^l = [(A^l r_i)^T Ap_j^1, \ldots, (A^l r_i)^T Ap_j^s)]^T$, where $l = 1, \ldots, s$ and $j = j_{(i-1)}, \ldots, i-1$
    3. $AP_i = AP_i - \sum_{j=j_{(i-1)}}^{i-1} AP_j [\underline{\beta}_j^l]_{l=1}^s$
    4. $P_i = P_i - \sum_{j=j_{(i-1)}}^{i-1} P_j [\underline{\beta}_j^l]_{l=1}^s$
    **EndIf**
    5. Apply MGS to the matrix $AP_i$ to obtain final $AP_i$ and $P_i$
    6. Compute $\underline{\alpha}_i = [r_i^T Ap_i^1, \ldots, r_i^T Ap_i^s]^T$
    7. $r_{i+1} = r_i - AP_i \underline{\alpha}_i$
    8. $x_{i+1} = x_i + P_i \underline{\alpha}_i$
**EndFor**
```

We note that the column vectors AP_j, for $j = j_{(i-1)}, \ldots, i-1$, are orthonormal. For $s = 1$ Algorithm 1 is the (GCR) Omin(k) algorithm.

Remark 1. The following alternative computations are possible in Algorithm 1.

(a) Step 7 can be replaced by direct computation of the residual $r_{i+1} = f - Ax_{i+1}$. This may enhance the efficiency and robustness of the method for larger s. In our implementation we used this approach for $s \ge 8$.

(b) The computation of AP_j can be carried out directly if *matrix vector products* are faster than linear combinations in steps 3 and 4. We note that there are s completely parallel matrix-vector products. We did not use this approach in our implementation.

We next display the storage and computational work for *a single iteration* of Algorithm 1 in Table 1. Storage includes the matrix A and the matrices:

$$x_i,\ r_i,\ \{P_j\}_{j=j_{(i-1)}}^{j=i},\ \{AP_j\}_{j=j_{(i-1)}}^{j=i}.$$

Table 1
Number of vector operations for the j-th iteration of OSGCR and OSOmin(s,k) given in terms of n

Vops	OSGCR	OSOmin
Stg	$2(j+1)s$	$2(k+1)s$
Dpr	$(2j+1)s^2+3s$	$\min([(2j+1)s^2+3s], [(2k+1)s^2+3s])$
Mv	s	s
Lc	$[(4j+2)s^2+2s]$	$\min([(4j+2)s^2+2s], [(4k+2)s^2+2s])$

We only count vector operations on vectors of dimension (of the linear system) n. We use the following notation:

- *Vops* (Vector operations);
- *Dpr* (Dot products);
- *Mv* (Matrix times vectors);
- *Lc* (Linear combinations);
- *Stg* (Storage requirements for vectors besides the Matrix A).

The number of dot products required to apply MGS to AP_j (step 5 of Algorithm 1) equals $s(s+1)/2$. Updating AP_j and P_j then requires $s(s-1)$ additional vector operations for linear combinations. Counting the vector operations in the rest of the steps of the Algorithm 1 is an easier task and yields the totals in Table 1.

3. Convergence properties

In this section we discuss the main convergence results and the robustness of (OSGCR) OSOmin(s,k).

It has been proved that S-step (GCR) Omin(k) (with $1 \le s,k$) converges for unsymmetric definite linear systems and for a class of unsymmetric indefinite linear systems [8]. These results also apply to (OSGCR) OSOmin(s,k). Unlike Omin(k), which only can be applied to definite linear systems, S-step Omin(k) and therefore OSOmin(s,k) converges for at least the same class of linear systems for which GMRES(s) converges. For fixed s, the choice of the parameter k affects the *convergence speed* of OSOmin(s,k) (similarly to Omin(k)). However, the choice of the parameter s affects the *convergence* and *parallel properties* of the method.

The main convergence theorem for (OSGCR) OSOmin(s,k) can be stated as follows.

Theorem 1. *Let $1 \le s$ and assume that the degree of the minimal polynomial of r_1 is greater than s. Assume that for each iteration $i = 1, \ldots$ (in Algorithm 1) (a* definiteness *condition) $r_i^T \; A^j r_i \neq 0$ holds for some j $(1 \le j \le s)$. Then (OSGCR) OSOmin(s,k) and GMRES(s) converge to the solution.*

Proof. Given in [8] □

The condition $r_i^T A^j r_i \neq 0$ for some j $(1 \le j \le s)$, provides that one of the steplengths α_i^j is not zero and thus progress towards the solution is made at the ith iteration of

Algorithm 1. Following this theorem we can describe the cases for which Algorithm 1 may break down.

Remark 2. The conditions of Theorem 1 can be violated as follows.

(a) This is a harmless violation. The degree of the minimal polynomial of the residual, $s1$, is less than s. This is detected in step 5, because the vectors AP_i are then linearly dependent. Algorithm 1 converges in one iteration. However, in the implementation s must replaced by $s1$ in steps 6, 7, 8. OSOmin(s1,k) also converges in this case.

(b) The violation of the definiteness condition in Theorem 1 is detected by using a parameter (**Count**(α)) that counts the number of nonzero steplengths (in step 6). If **Count**(α) $= s$, then Algorithm 1 is unable to advance to the next iteration. In this case, the method must be restarted with a different initial solution or a larger s.

We next present examples of contrived linear systems to illustrate the convergence properties of the methods. In all the matrix examples in this section *unlabeled entries* are equal to zero. The following example makes both GMRES(s) and OSOmin(s,k) break down for s less than the dimension (n) of the system [20,4,15].

Problem 1. Let the linear system matrix (of dimension n) be

$$\begin{bmatrix} 0 & & & \cdots & & 1 \\ 1 & 0 & & & & \\ & 1 & 0 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & 0 & \\ & & & & 1 & 0 \end{bmatrix} \tag{2}$$

and let the right hand side be $[1,0,\ldots,0]^{\mathrm{T}}$. Then for $x_0 = 0$, $r_i^{\mathrm{T}} A^j r_i = 0$ for all j, with $1 \leq j < n$. This causes both GMRES(s) and OSOmin(s,k), with $s < n$, to break down. For OSOmin(s,k) this breakdown is detected by checking if $\alpha_i = 0$. A remedy for this is to restart the method (with the current residual r_i) and apply (in a preprocessing stage) a single step of the Conjugate Residual method (e.g., expressed by Algorithm 1 as the OSOmin(1,1)) applied to the normal equations and then continue with OSOmin(s,k). The normal equations formulation of this problem makes the coefficient matrix ($A^{\mathrm{T}}A$) equal to the identity and thus one step of OSOmin(1,1) would find the solution.

The following theorem concerns the convergence of OSOmin(s,1) for symmetric or skew-symmetric linear systems.

Theorem 2. *Let $s \geq 2$ and assume that the degree of the minimal polynomial of r_1 is greater than s. Furthermore, assume that either A is (skew-)symmetric or $A = I - N$, where N is skew-symmetric. Then (OSGCR) OSOmin(s,k) converges to the solution. Also, OSOmin(s,1) is equivalent to OSGCR and thus it converges in at most $\lceil n/s \rceil$ iterations.*

Proof. Given in [8] □

Theorem 2 shows the advantage of OSOmin(s,1) (a truncated method) over GMRES(s) (a restarted method). The second method may break down for skew-symmetric linear systems, as shown in the following example.

Problem 2. Let the linear system (skew-symmetric) matrix of dimension n be

$$\begin{bmatrix} 0 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix} \tag{3}$$

and the right hand side $[1/\sqrt{2},0,\dots,0,1/\sqrt{2}]^T$. This linear system arises naturally in discretizing the boundary value problem[

$$-\rho\psi_{\xi_1\xi_1} + \psi_{\xi_1} = \chi \text{ on } 0 < \xi_1 < 1,$$

with boundary conditions $\psi(0) = \psi(1) = 0$, when $\rho \to 0$. For problem 2 with initial solution equal to zero, GMRES(s) does not converge unless $s = n$ [4]. For this problem OSOmin(2,1) converges in at most $\lceil n/2 \rceil$ iterations.

Remark 3. Although we have not made a theoretical study of the error properties of Algorithm 1 we make two observations:

(a) The modified Gram–Schmidt method error is proportional to the condition number of matrix AP_i (**Cond**(AP_i)) [3]. We have not estimated the **Cond**(AP_i) in our tests. However, it did not affect the accuracy of the solution for an ill-conditioned problem that we tested OSOmin(s,k) (with $s \le 16$).

(b) The linear system can be *scaled* (by the maximum absolute value of the (row) column entries) to achieve *(row) column equilibration* [31]. This scaling modifies the matrix entries so that they do not exceed one, so past errors do not accrue due to matrix-vector products (in step 1). This is not a prohibitive task for sparse matrices. Also, equilibrated diagonal linear systems are solved in one iteration of Algorithm 1.

We next consider an ill-conditioned linear system and test the robustness of OSOmin(s,k) for several values of s ($1 \le s \le 16$). This problem has been used by H.F. Walker to test the robustness of GMRES(32) [28].

Problem 3. Let the linear system matrix of dimension $n = 100$ be

$$\begin{bmatrix} 1 & & & & & \alpha \\ & 2 & & & & \\ & & 3 & & & \\ & & & \ddots & & \vdots \\ & & & & 99 & \\ & & & & 0 & 100 \end{bmatrix}, \tag{4}$$

where $\alpha = 10^3$, the right hand side is $[1,1,\dots,1]^T$ and $x_0 = 0$. We ran OSOmin(s,k) with $(s,k) = (1,8)$, $s = 2, 4, 8, 16$ and $k = 1$. Each iteration of Algorithm 1 requires s

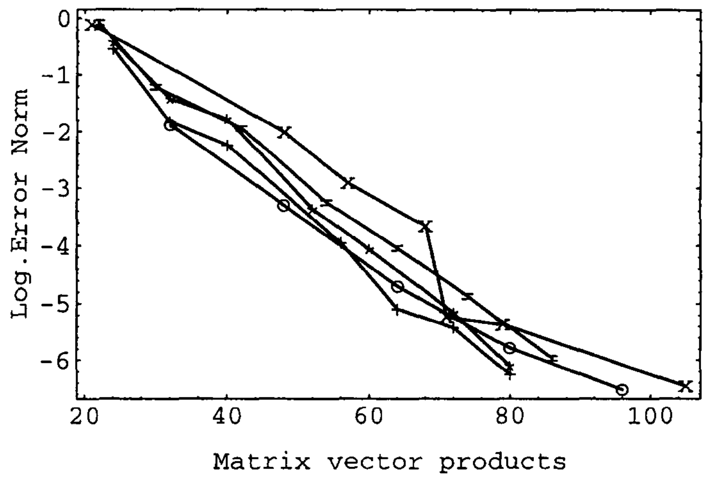

Fig. 1. Convergence of true error for Problem 3.

matrix-vector products. The plotting of the error versus the number of matrix-vector products is a good quick way to compare the various methods for convergence speed. The $\log_{10}$ of the Euclidean norm of the true (residual) errors in terms of the number of matrix-vector products are plotted in Figs. 1 and 2. It can be seen that the methods' convergence is unaffected for up to $s = 16$. For $s = 1$ the method broke down for $k < 26$. The linear system was not equilibrated (as described in Remark 2). Using OSOmin(s,k) without equilibration runs the risk of magnifying errors (in carrying out step 1). Despite this and the large condition number the method remains robust.

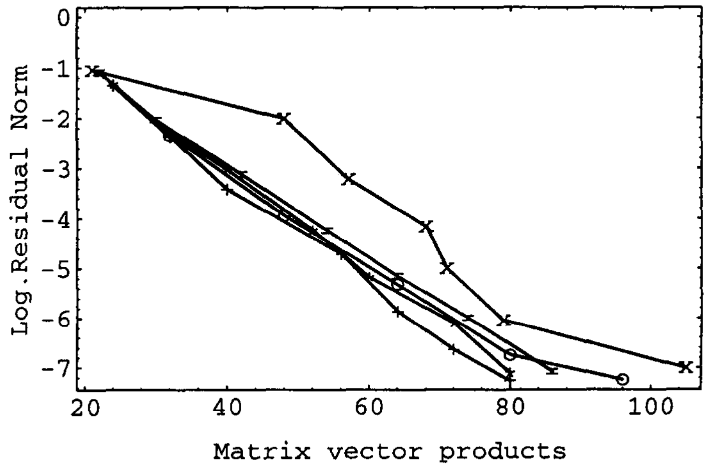

Fig. 2. Convergence of residual error for Problem 3.

4. An unsymmetric large sparse linear system

We have discretized a partial differential equation boundary value problem on a square region by the method of finite differences. This is a standard elliptic problem which can be found in [19] and the right hand side function is constructed so that the analytic solution is known.

Problem 4.

$$-(\rho\psi_{\xi_1})_{\xi_1}-(\sigma\psi_{\xi_2})_{\xi_2}+(\tau\psi)_{\xi_1}+(\zeta\psi)_{\xi_2}+\phi\psi=\chi,$$

where

$$(\xi_1,\xi_2)\in\Omega=(0,1)\times(0,1),$$

$$\rho(\xi_1,\xi_2)=e^{-\xi_1\xi_2},\sigma(\xi_1,\xi_2)=e^{\xi_1\xi_2},\tau(\xi_1,\xi_2)=\tilde{\beta}\cdot(\xi_1+\xi_2),$$

$$\zeta(\xi_1,\xi_2)=\tilde{\gamma}\cdot(\xi_1+\xi_2),\phi(\xi_1,\xi_2)=\frac{1}{(1+\xi_1\xi_2)},$$

$$\psi(\xi_1,\xi_2)=\xi_1e^{\xi_1\xi_2}\sin(\pi\xi_1)\sin(\pi\xi_2),$$

with Dirichlet boundary conditions and $\chi(\xi_1,\xi_2)$ the corresponding right hand side function. By controlling $\tilde{\gamma}$ and $\tilde{\beta}$, we could change the degree of nonsymmetry of the problem. We chose $\tilde{\gamma}=50.0$, $\tilde{\beta}=1.0$.

If this problem is discretized using the centered difference scheme on a uniform $n_x\times n_y$ grid (where $n_x=n_y$) with mesh size $1/(n_x+1)$, we obtain a linear system of equations

$$Ax=f$$

of size $n=n_x^2$. If we use natural ordering of the grid points, then the matrix A is a block tridiagonal matrix of the form

$$A=[C_{k-1},D_k,B_k],\quad 1\le k\le n_x, \tag{5}$$

where C_{k-1}, D_k, B_k are matrices of size n_x; and $C_0=B_{n_x}=0$. The matrices C_{k-1}, B_k are diagonal matrices and D_k are tridiagonal matrices. The matrix is *large*. For example, if $n_x=100$, then the dimension of A is $n=10^4$.

5. Preconditioning

The convergence rate of the Krylov subspace methods (such as OSGCR, OSOmin) is closely related to the condition number of the system matrix. Matrices with high condition numbers may lead to slow convergence rates. A strategy which is often implemented in conjunction with iterative methods is to apply a preconditioner which transforms the original system into one with a matrix of a smaller conditioner number or with clustered eigenvalues. The transformed system is then solved by the iterative solver at a faster convergence rate. Let K be the *right* preconditioning matrix. System (1) is transformed to

$$(AK)K^{-1}x=f, \tag{6}$$

which is then solved by the iterative solver.

Either K is a close approximation to the inverse of A i.e.

$$K \approx A^{-1}$$

or AK has clustered eigenvalues. The preconditioner K must be easily invertible, so that the system $K^{-1}x=b$ is easy to solve. In combining *right* preconditioning with Algorithm 1, we only need to change step 1, as follows:

1. Compute $P_i = [Kr_i, K(AK)r_i, \ldots, K(AK)^{s-1}r_i]$, $AP_i = [AKr_i, (AK)^2 r_i, \ldots, (AK)^s r_i]$.

We choose the ILU factorization preconditioner [26]. The basis for the ILU factorization method is that the matrix A is decomposed into upper and lower triangular matrices U and L such that $A = LU + \Delta$, where $K^{-1} = LU$ and Δ is an error matrix. Also, we assume that if $A_{i_1,i_2} = 0$, then both U_{i_1,i_2} and $L_{i_1,i_2} = 0$. In other words, L and U have the same sparsity patterns as A. This is the ILU(0) method. For more details see [19,26].

Although ILU preconditioning can improve the convergence rate of the iterative solvers considerably, the preconditioner itself may have very slow execution rates if not implemented properly on a vector-parallel computer. This is due to the following fact. Let L and U be the incomplete LU factors of A. Then solution of the two triangular systems $Ly = b$ and $Ux = y$ requires back-solving, which is a serial operation.

Let us now consider ILU(0) for the block tridiagonal A of our model problem. This can be implemented in vector mode by using a Neumann series expansion, proposed by van der Vorst in 1982 [26]. The original problem can be scaled so that the diagonal entries of L and U are 1. Suppose E is the matrix consisting of the subdiagonal of L (L_{i_1,i_1-1}) and F is the matrix consisting of the remaining subdiagonals L_{i_1,i_2}, $i_2 < i_1 - 1$. Then $L = I + E + F$. So the system to be solved is:

$$(I+E+F)y=b \tag{8}$$

or, in block form

$$(I+E_k)y_k = b_k - F_{k-1}y_{k-1}, \tag{9}$$

where E_k, F_k are the diagonal and subdiagonal blocks of L and b_k and y_k are the corresponding block subvectors. Assuming that the norm of E is small relative to the norm of I, this can be expanded via a Neumann Series,

$$y_k = (I+E_k)^{-1}(b_k - F_{k-1}y_{k-1}) = \left(I - E_k + E_k^2 - E_k^3 + \ldots\right)(b_k - F_{k-1}y_{k-1}). \tag{10}$$

The power series is usually truncated after the second term. The backward solution for U is obtained similarly. Note that one of the conditions for use of the Neumann

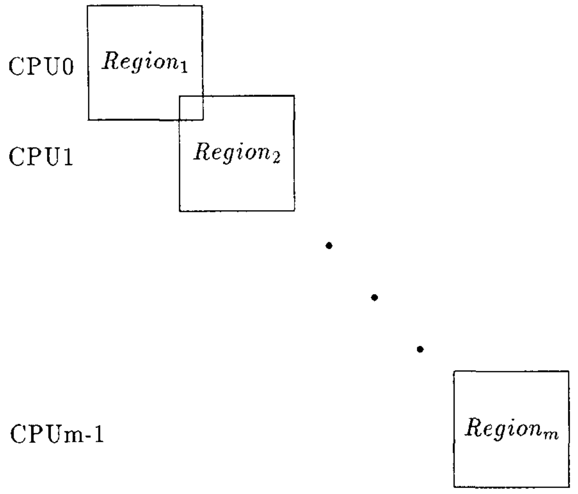

Fig. 3. Parallel execution of ILU for Problem 4.

series is that the norm of E is small relative to I. This will be the case for diagonally dominant systems, but may break down for systems that are not diagonally dominant.

The Neumann expansion leads to vector operations on blocks of length $= n_x$, where n_x is the number of gridpoints in the x direction of the solution mesh. The vectorizable preconditioning shows substantial improvement over serial preconditioning on vector processors. However, no parallelism exists because there is a recurrence (of order 1) in the computation of the different vector blocks. That is, block k uses the result of block $k-1$.

A method for parallelizing the ILU preconditioner was introduced by Radicati di Brozolo and Robert in 1988 [21]. It was proposed to partition the preconditioned matrix into a number (m) of overlapping *submatrix regions*. Each region consists (of a contiguous index sequence) of submatrix blocks of the type $[C_{k-1}, D_k, B_k]$. The loss of connection between the regions is partially compensated for by introducing smaller overlapping *region segments*. Each submatrix region is then executed in parallel on m processors. After the back solution step is carried out (independently in each submatrix region) the overlapped values between the separate regions are set equal to the average of that determined in each region. It is found ([21]) that this overlapping strategy gives better performance than the nonoverlapping one. Here, we use an overlapping of a single submatrix block of the type $[C_{k-1}, D_k, B_k]$, between two successive parallel regions. The parallel implementation of this technique, on m processors (*CPU*s), is illustrated in Fig. 3.

If $m = 1$, this preconditioner is exactly the same as the vectorized ILU preconditioned method. If $m > 1$, this type of preconditioning is slightly less effective at reducing the condition number because of the loss of connection between the submatrix regions. In general, the effectiveness of the preconditioner is reduced as m becomes larger, but the performance on parallel processors improves.

6. The multi-processor implementation

The final numerical experiments (in single user mode) were executed on a *CRAY C90 system at Cray Research, Inc.*. This system has 16 CPUS and 256 million 64-bit words of shared memory. In the CRAY C90, each CPU has two vector pipelines, with each pipeline having one floating-point functional unit for addition and one for multiplication. Thus a CRAY C90 CPU can produce four floating-point results per clock period if addition and multiplication operations can be chained together. The constructs utilized by linear solvers and implemented in BLAS routines take advantage of this feature. The CRAY C90 has a clock period of 4.167 nanoseconds giving a peak performance of 959.9 Mflops per CPU.

Each CPU has eight vector registers (each 128 words long) and four ports to memory. In three-port vector mode, during any given clock period there can be two reads from memory and one write to memory. The fourth port is used for other I/O purposes or for the instruction buffers. To support the dual pipelines in the vector unit, each port is 128 bits (i.e., two words) wide. A fully configured 16 CPU system has a memory bandwidth of 246 Gbytes/sec.

The CRAY C90 operating system is UNICOS, an extended version of UNIX [2] that supports optimizing compilers and the autotasking feature for parallelization. Autotasking automatically detects and exploits parallelism in a program. The user can also insert compiler directives to provide enhanced parallel execution. For example, directives can be used to tell the compiler when potential dependencies inhibiting parallel code generation can safely be ignored.

Finally, the CRAY C90 provides highly optimized scientific mathematics libraries, including efficient implementation of BLAS (1, 2, 3) ([1,12]). Let x and y be vectors and α be a scalar. BLAS1 perform vector-vector operations such as:

$$y \leftarrow \alpha x + y \quad (\mathbf{saxpy})$$

The BLAS1 **sdot, sscal, scopy** were also used in the implementation. The BLAS2 perform matrix-vector operations including the rank-one update (used in the implementation) of the form

$$B = \alpha x y^{\mathrm{T}} + B \quad (\mathbf{sger}),$$

where B is matrix of dimension $n \times s$, α is a scalar, x is a vector of dimension n and y is a vector of dimension s. BLAS3 perform matrix-matrix operations and are designed to take advantage of systems with cache or local memories. The only BLAS3 used in the implementation was **scopy2**, which copies a matrix into another matrix (needed in step 1 of Algorithm 1). Scopy2 enhances the execution rate of the copy operation for problem sizes that exceed the capacity of the CRAY C90 main memory. The performance of BLAS used is discussed below.

[2] UNICOS and CRAY C90 are trademarks of Cray Research, Inc. and UNIX is a trademark of UNIX Systems Laboratories, Inc.

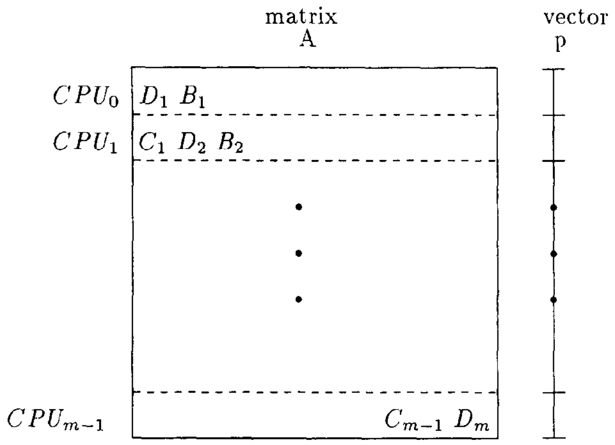

Fig. 4. Parallelization of the matrix-vector product on m CPUs.

Parallelization of the BLAS on a system with m CPUs was implemented by partitioning the vectors of dimension n into m equal subvectors and making m independent BLAS calls. Parallelization of the ILU step was discussed in the previous section and illustrated in Fig. 3. Parallelization of the sparse matrix-vector product (for system size $n = m^2$) was implemented by simply partitioning the matrix and vector along the dimension n as illustrated in Fig. 4.

7. Results

In this section we describe the results of the implementation on the CRAY C90 of OSOmin(s,k) applied to solve *Problem 4*. For the runs described here we used the stopping criterion

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

and the initial solution vector

$$x(i) = 0.05 * \operatorname{mod}(i,50)$$

and dimension $n_x = 512$. The matrix was equilibrated by columns in the unpreconditioned case. We executed OSOmin(s,k) with $(s,k) = (1,4)$ (i.e. the standard Omin(4)) and $(s,k) = (s,1)$ for $s = 2,4,8,16$. Choice of $k = 4$ is (empirically) the best (in terms of number of iterations) for OSOmin(1,k). Similarly, the choice of $s = 2$ is the best for OSOmin(s,1), with $s \geq 2$. Problem 4 is definite so the choice of $s = 1$ can be used for convergence. Here, it must be noted that the choice of s is related both to the convergence and to the parallelism of the methods. The method, applied to Problem 4, converges for any s, $1 \leq s \leq 16$. However, there are more difficult problems [29] which require $s = 16$ for convergence.

Table 2
OSOmin(s,k) single processor performance in Mflops for PDE problem

	$s=16, k=1$	$s=8, k=1$	$s=4, k=1$	$s=2, k=1$	$s=1, k=4$
Mflops/%					
Matvec	691/7	701/13	702/18	703/25	706/17
Saxpy	749/1	759/2	759/4	760/5	781/7
Inner products (sdot)	860/30	869/27	865/24	859/20	861/18
Linear comb. (sger)	816/60	790/55	742/48	705/38	613/51
Scopy	—/1	—/2	—/3	—/5	—/4
ILU preconditioner	470	470	470	470	476
Whole code:					
Unpreconditioned case	817	777	733	688	664
Preconditioned case	768	707	647	597	600

Percentages are for the Unpreconditioned case

Single CPU performance results for the preconditioned and unpreconditioned problem and for the constituent operations are listed in Table 2. Results are in millions of floating point operations per second (Mflops). The percent of time used by each type of operation in the unpreconditioned case is also listed. The algorithm is dominated by linear combinations, implemented using the BLAS2 sger. Performance of sger is seen to degrade as the number of columns (s) decreases to 1. Nevertheless, since the peak single

Table 3
Speed-ups for constituent operations within the OSOmin(s,k) code

	s	k	4 processors	8 processors	16 processors
Inner products	16	1	4.0	7.7	13.4
	8	1	3.9	7.6	12.8
	4	1	3.9	7.2	11.8
	2	1	3.8	7.0	10.1
	1	4	3.8	6.8	10.2
Linear comb.	16	1	4.0	7.9	14.5
	8	1	3.9	7.8	14.3
	4	1	4.0	7.7	14.0
	2	1	4.0	7.6	12.4
	1	4	3.9	7.7	13.1
Saxpy	16	1	4.0	6.9	10.8
	8	1	4.0	6.8	10.6
	4	1	4.0	6.5	10.0
	2	1	4.0	6.3	9.0
	1	4	3.9	6.3	9.2
Scopy			3.9	7.4	12.0
Matvec			3.8	6.9	10.6
ILU preconditioner			3.8	7.3	12.3

Table 4
Wallclock time, iterations and speed-up for the unpreconditioned problem

	s	*k*	1 processor	4 processors	8 processors	16 processors
Wallclock seconds (iterations)						
OSOmin(s,k)	16	1	61.24	14.91	7.85	4.34
			(75)	(73)	(74)	(73)
OSOmin(s,k)	8	1	33.16	8.58	4.38	2.55
			(139)	(139)	(138)	(139)
OSOmin(s,k)	4	1	24.10	6.30	3.43	2.09
			(314)	(314)	(313)	(313)
OSOmin(s,k)	2	1	19.56	5.41	3.11	2.18
			(715)	(720)	(717)	(723)
Omin(k)	1	4	21.60	6.08	3.55	2.45
			(1076)	(1076)	(1076)	(1076)
Speed-up						
OSOmin(s,k)	16	1	1.00	4.11	7.80	14.13
OSOmin(s,k)	8	1	1.00	3.87	7.57	13.00
OSOmin(s,k)	4	1	1.00	3.82	7.03	11.53
OSOmin(s,k)	1	4	1.00	2.99	4.60	6.19
Omin(k)	1	4	1.00	3.55	6.08	8.83

CPU performance for the CRAY C90 is 960 Mflops, the overall results demonstrate efficient vector performance for the OSOmin(s,k) algorithm.

Table 3 shows the parallel performance for the constituent operations, listed as wall-clock time and speed-ups obtained using 4, 8 and 16 CPUs. For scopy, matrix-vector, and the ILU routine, speed-ups were independent of s. For inner products (sdot), linear combinations (sger) and saxpy operations, the speed-ups improved as s increased

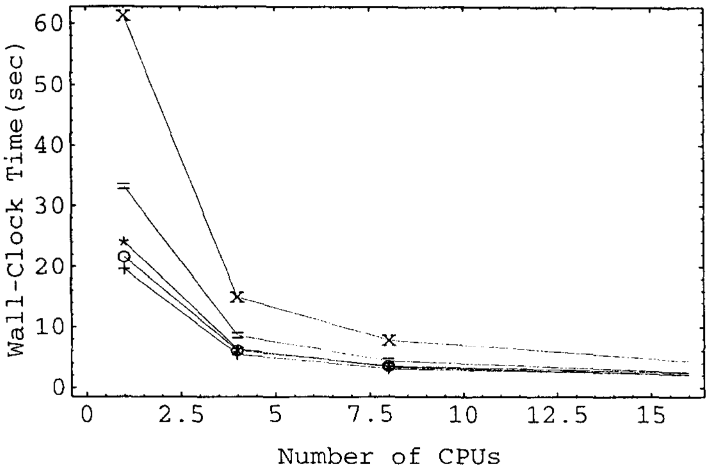

Fig. 5. Run-time for Problem 4 (Unpreconditioned).—x— (s,k) = (1,26), — = — (s,k) = (2,1), — * — (s,k) = (4,1), — + — (s,k) = (8,1), —0— (s,) = (16,1).

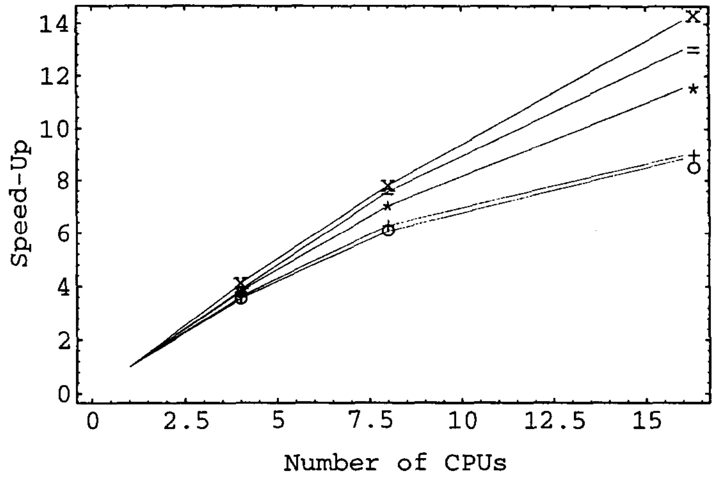

Fig. 6. Speed-up for Problem 4 (Unpreconditioned).

from 2 to 16 with the OSOmin(1,4) results falling between the OSOmin(2,1) and OSOmin(4,1) results.

Wallclock-time and speed-up results for the unpreconditioned problem are given in Table 4 and illustrated in Figs. 5 and 6. The best absolute performance for this problem was achieved with OSOmin(4,1) using 16 CPUs. For the parallel preconditioned implementation, *the number of parallel overlapped blocks m* was chosen to be equal to

Table 5
Wallclock time, iterations and speed-up for the preconditioned problem

	s	*k*	1 processor	4 processors	8 processors	16 processors
Wallclock seconds (iterations)						
SOmin(s,k)	16	1	22.08	6.20	3.44	2.22
			(25)	(26)	(26)	(27)
OSOmin(s,k)	8	1	12.95	3.78	2.17	1.46
			(44)	(46)	(45)	(45)
OSOmin(s,k)	4	1	8.73	2.84	1.86	1.29
			(83)	(91)	(98)	(94)
OSOmin(s,k)	2	1	6.99	2.17	1.43	1.16
			(167)	(161)	(161)	(176)
Omin(k)	1	4	9.25	3.09	2.01	1.49
			(340)	(364)	(368)	(365)
Speed-up						
OSOmin(s,k)	16	1	1.00	3.56	6.43	9.95
OSOmin(s,k)	8	1	1.00	3.43	5.97	8.87
OSOmin(s,k)	4	1	1.00	3.08	4.70	6.79
OSOmin(s,k)	2	1	1.00	3.22	4.88	6.04
Omin(k)	1	4	1.00	2.99	4.60	6.19

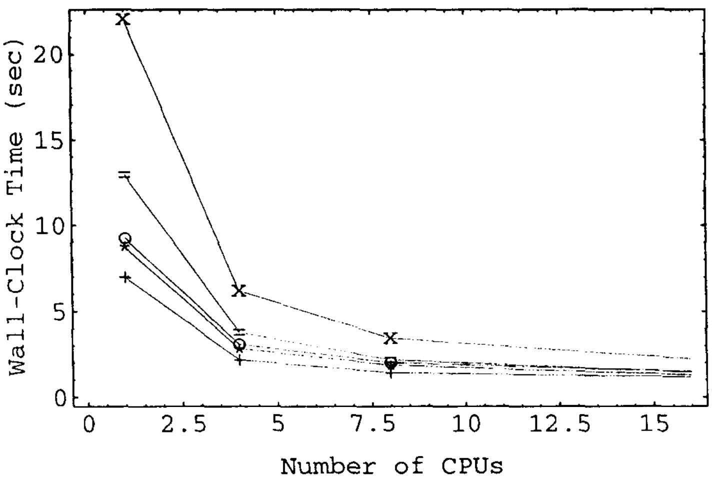

Fig. 7. Run-time for Problem 4 (Preconditoned).

the number of processors of C90. Wallclock-time and speed-up results for the preconditioned problem are given in Table 5 and illustrated in Figs. 7 and 8. The best absolute performance for this problem was achieved with OSOmin(2,1) using 16 CPUs. The fact that we observe speed-ups higher than the number of CPUs is due to the slight variation in the number of iterations for convergence in the parallel execution of the algorithm.

8. Summary and conclusions

We presented an improved version of S-step methods for unsymmetric linear systems in which the s direction vectors within each S-step block are $A^T A$-orthogonalized using

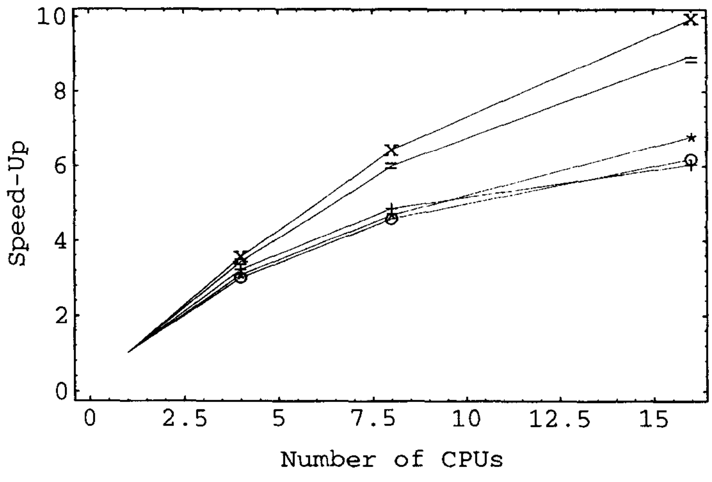

Fig. 8. Speed-up for Problem 4 (Preconditoned).

the Modified Gram–Schmidt technique. This permitted larger values of s (up to $s = 16$) than in the original methods, yielding more robust algorithms. With larger s values, the number of iterations is reduced with more work being done in each iteration, a situation that makes more efficient use of multiple processors. The additional work from the MGS orthogonalization is compensated by enhanced parallel performance (i.e., higher speed-ups) to obtain algorithms with multiple processor performance comparable to Omin. For the large and sparse problem analyzed, the best performance on the 16 CPU CRAY C90 was obtained with $s = 4$ (unpreconditioned) or $s = 2$ (preconditioned). However, there are indefinite systems [29], where $s = 16$ must be chosen to obtain convergence. Here, we demonstrated that the OSOmin(s,k) algorithm scales to 16 processors with good vector and parallel performance.

In general one can determine the best choice of s and k for convergence using a small dimension n, because the convergence properties of the method depend on the spectral properties of the linear system [15,20]. The more difficult the linear system the larger the s and k will be required. Then, one must use s CPUs in executing OSOmin(s,k) to solve the problem (of the large dimension) in order to achieve high efficiency on the parallel system. Finally, as observed in Remark 3, OSOmin(s,k) must always be applied with a good preconditioner or in conjunction with column(row) equilibration of the coefficient matrix.

Acknowledgements

The comments of the anonymous referees, which helped improve the presentation of some parts of the paper are gratefully acknowledged.

References

[1] E. Anderson et al., *LAPACK Users' Guide*, (SIAM, Philadelphia, 1992).
[2] O. Axelsson, A generalized conjugate gradient, Least squares method, *Numer. Math.* 51 (1987) 209–227.
[3] A. Bjork, *Least Squares Methods: Handbook of Numerical Analysis, Vol. 1 Solution of Equations in* R^N (Elsevier, North Holland, Amsterdam, 1988).
[4] P.N. Brown, A theoretical comparison of the Arnoldi and GMRES algorithms, *SIAM J. Sci. Stat. Comput.* 12(1) (1991) 58–78.
[5] C.G. Broyden, Block conjugate gradient methods, *Optimization methods and Software* 2 (1993) 1–17.
[6] A.T. Chronopoulos and C.W. Gear, S-step iterative methods for symmetric linear systems, *Journal of Computational and Applied Mathematics* 25 (1989) 153–168.
[7] A.T. Chronopoulos and C.W. Gear, Implementation of preconditioned S-step conjugate gradient methods on a multiprocessor system with memory hierarchy, *Parallel Computing* 11 (1989) pp. 37–53.
[8] A.T. Chronopoulos, S-step iterative methods for (non)symmetric (in)definite linear systems, *SIAM J. on Num. Analysis* 28(6) (1991) 1776–1789.
[9] J.W. Demmel, M.T. Heath and H. van Der Vorst, *Parallel Numerical Linear Algebra* (Acta Numerica 2, Cambridge University Press, 1993).
[10] E. de Sturler, A parallel variant of GMRES(m), in: J. Miller and R. Vichnevetsky, eds., *Proc. of the 13th IMACS World Congress on Computation and Applied Math.* (Criterion Press, Dublin, 1991) 682–683.
[11] J.J. Dongarra and R.E Hiromoto, A collection of parallel linear equations routines for the Denelcor HEP, *Parallel Computing* 1 (1984) 133–142.

[12] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Parallel Computers* (SIAM publications, 1991).

[13] 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.

[14] K. Gallivan, M. Heath, E. Ng, J. Ortega, B. Peyton, R. Plemmons, C. Romine, A. Sameh and R. Voigt, *Parallel Algorithms for Dense Linear Algebra Computations* (SIAM, Philadelphia, 1990).

[15] A. Greenbaum and Z. Strakos, *Matrices that generate the same Krylov varieties*, Institute of Mathematics and its Applications Volumes in Applied Mathematics, vol. 60, 1994.

[16] S.K. Kim and A.T. Chronopoulos, A class of Lanczos-like algorithms implemented on parallel computers, *Parallel Computing* 17 (1991) 763–778.

[17] S.K. Kim and A.T. Chronopoulos, An efficient nonsymmetric Lanczos on parallel vector computers, *J. of Comput. and Applied Math.* 42 (1992) 357–374.

[18] S.K. Kim and A.T. Chronopoulos, An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices, *The Int. J. on Supercomputing* 6(1) (1992) 98–111.

[19] S. Ma and A.T. Chronopoulos, Implementation of iterative methods for large sparse nonsymmetric linear systems on parallel vector computers, *Int. J. on Supercomputing* 4(4) (1990) 9–24.

[20] N.M. Nachtigal, S.C. Reddy and L.N. Trefethen, How fast are non-symmetric matrix iterations?, *SIAM J. Matrix Anal. Appl.* 13(3) (1992) 778–795.

[21] G. Radicati di Brozolo, Y. Robert, Parallel conjugate gradient-like algorithms for sparse nonsymmetric systems on a vector multiprocessor, *Parallel Computing* 11 (1989) 223–239.

[22] M. Sadkane and B. Vital, Davidson's method for linear systems of equations, Implementation of a block algorithm on a Multi-processor, Technical Report TR/PA/91/60, CERFACS, Toulouse, 1991.

[23] V. Simoncini and E. Gallopoulos, An iterative method for nonsymmetric systems with multiple right hand sides, *SIAM J. Sci. Stat. Comput.* 16 (1995) 917–933.

[24] V. Simoncini and E. Gallopoulos, Convergence properties of block GM-RES and matrix polynomials, to appear in: *Linear Algebra and its Applications*, also: Center for Supercomputing Research and Development, Technical Report No. 1316, April 1994.

[25] D.C. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, *SIAM J. Matrix Anal. Appl.* 13 (1992) 357–385.

[26] H. van der Vorst, A vectorizable variant of some ICGG methods, *SIAM J. Sci. Stat. Comput.* 3 (1982) 350–356.

[27] P. Vinsome, *An iterative method for solving sparse sets of simultaneous equations* (Society of Petroleum Engineers of AIME, SPE 5729, 1976).

[28] H.F. Walker, Implementation of the GMRES method using householder transformations, *SIAM J. Sci. Stat. Comput.* 9 (1988) 152–163.

[29] A.M. Wissink, A.S. Lyrintzis and A.T. Chronopoulos, Efficient iterative methods applied to the solution of transonic flows, *J. of Computational Physics* 123 (1996).

[30] D.M. Young and D.R. Kincaid, The ITPACK software package, in: B. Engquist and T. Smedsaas, eds., *PDE SOFTWARE: MODULES, INTERFACES AND SYSTEMS* (North Holland, Amsterdam, 1984) 193–206.

[31] Z. Zlatev, *Computational Methods for General Sparse Matrices* (Kluwer Academic Publishers, 1991).