Comm. Korean Math. Soc. 9 (1994), No. 2, pp. 449–465

EFFICIENT PARALLEL ITERATIVE METHOD FOR SOLVING LARGE NONSYMMETRIC LINEAR SYSTEMS

JAE HEON YUN

1. Introduction

The two common numerical methods to approximate the solution of partial differential equations are the finite element method and the finite difference method. They both lead to solving large sparse linear systems. For many applications, it is not unusal that the order of matrix is greater than 10,000. For this kind of problem, a direct method such as Gaussian elimination can not be used because of the prohibitive cost. To this end, many iterative methods with modest cost have been studied and proposed by numerical analysts.

The classical Conjugate Gradient (CG) method of Hestenes and Stiefel [9] with some preconditioning technique is one of the most effective iterative methods for solving large sparse symmetric positive definite linear systems. However, this algorithm fails in general for nonsymmetric linear systems. In the last 15 years, a large number of generalizations of the CG method have been proposed for solving nonsymmetric linear systems [7, 8, 11, 12, 13]. All of the iterative methods developed to date for nonsymmetric indefinite linear systems are either slow converging or converge fast only for very special cases. The Generalized Conjugate Residual (GCR) method [7] and the Generalized Minimum Residual (GMRES) method [12] are typical examples of those for solving nonsymmetric linear systems. Throughout this paper, consider the linear system $Ax = b$, where A is a large sparse nonsymmetric matrix of order n with real coefficients. Let M denote the symmetric part of the matrix A, that is, $M = (A+A^T)/2$. If the matrix M is (positive or negative) *definite*, then the matrix A is called (positive or negative) *definite*. Given a set of vectors $\{p_0, p_1, \ldots, p_k\}$, let $\langle p_0, p_1, \ldots, p_k\rangle$ denote the subspace spanned by $\{p_0, p_1, \ldots, p_k\}$. For a given vector r_0, let the m-th *Krylov subspace* $K_m(A, r_0)$ denote $\langle r_0, Ar_0, \ldots, A^{m-1}r_0\rangle$. Both

Received August 21, 1993.

This paper was supported (in part) by NON DIRECTED RESEARCH FUND, Korea Research Foundation, 1992

the GCR and GMRES methods are based on minimizing the norm of the residual vector over a Krylov subspace. The disadvantage of these methods is that the computational costs and storage requirements per iteration are prohibitively high when the number of iteration steps is large. To avoid this problem, the restarted versions GCR(k) and GMRES(k) of the above algorithms which involve at most k previous direction vectors are generally used.

More and more vector and parallel computers with a hierarchy of storage, such as the CRAY-2, Alliant FX/8, and IBM 3090, have been developed and made available to the engineers and numerical analysts in recent years. To take advantage of full capabilities of current advanced multiprocessors, a great deal of efforts need to be made in the search for efficient and parallel implementations. For example, to make efficint use of the memory hierarchy we must keep the data either in cache or local memory or in vector registers as long as possible. For multiprocessing, we must reduce the number of synchronization points in numerical computations since processor synchronization is a severe bottleneck in achieving peak performance, see [5] for more details.

The goal of this paper is to develop an efficient parallel iterative algorithm, specially well-suited for computers with a shared memory, which does not break down even for *nonsymmetric indefinite* linear systems. This paper is organized as follows. In Section 2, we establish a necessary and sufficient condition which causes a breakdown of the GCR algorithm when it is applied to indefinite linear systems. It is also shown that the well-preconditioned GCR method does not break down until convergence even for indefinite linear systems. In Section 3, a vector and parallel implementation of the restarted algorithm GMRES(k) is presented, and then another algorithm called GMORTH(k) which performs better on the CRAY-2 than the GMRES(k) for small values of k is proposed and parallelized. In Section 4, we describe the test problem considered and we present performance results of these two algorithms on the CRAY-2 vector and parallel computer. Lastly, performance comparisons for both algorithms are made and some conclusions are drawn.

2. GCR algorithm applied to indefinite linear systems

We first consider the generalized conjugate residual method(GCR) to solve $Ax = b$, where A is a nonsingular and nonsymmetric matrix of order

n. The GCR algorithm that works for the matrix A being (positive or negative) definite is described.

ALGORITHM 1: GCR

1. Choose x_0 and $r_0 = b - Ax_0$
2. Set $p_0 = r_0$
3. Compute Ap_0

   For $i = 0, 1, \ldots, k, \ldots,$ **until satisfied, do:**
4. $\alpha_i = \frac{(r_i, Ap_i)}{(Ap_i, Ap_i)}$
5. $x_{i+1} = x_i + \alpha_i p_i$
6. $r_{i+1} = r_i - \alpha_i Ap_i$

   If $\| r_{i+1} \|_2$ satisfies a certain criterion, then stop
7. Compute Ar_{i+1}
8. $h_{ji} = -\frac{(Ar_{i+1}, Ap_j)}{(Ap_j, Ap_j)}$, $0 \leq j \leq i$
9. $p_{i+1} = r_{i+1} + \sum_{j=0}^{i} h_{ji} p_j$
10. $Ap_{i+1} = Ar_{i+1} + \sum_{j=0}^{i} h_{ji} Ap_j$

Before we go further, the basic relations among the vectors generated by this algorithm which are proved in [7] are given below:

(1.a) $(Ap_i, Ap_j) = 0$ if $i \neq j$, (1.b) $(r_i, Ap_j) = 0$ if $i > j$
(1.c) $(Ap_i, Ap_i) \leq (Ar_i, Ar_i)$, (1.d) $(r_i, Ap_i) = (r_i, Ar_i)$
(1.e) $\langle p_0, \ldots, p_i \rangle = \langle r_0, \ldots, r_i \rangle = K_{i+1}(A, r_0)$
(1.f) x_{i+1} minimizes the residual norm $\| b - Ax \|_2$ over the affine space $x_0 + \langle p_0, \ldots, p_i \rangle$.

It is shown in [7] that if A is definite, then the GCR method produces a sequence of approximations x_k which converges to the exact solution. Now we will show that GCR may break down when A is indefinite. Note that for indefinite A, it may happen that $(r_i, Ap_i) = (r_i, Ar_i) = (r_i, Mr_i) = 0$ and hence α_i in step 4 is equal to 0. It is clear that $p_i \neq 0$ implies $r_i \neq 0$. From now on, it is assumed unless otherwise stated that the GCR method is applied to linear systems with indefinite matrices.

THEOREM 2.1. *Suppose that $p_j \neq 0$ for all $j = 0, 1, \ldots, i$. If $\alpha_i = 0$, then $p_{i+1} = 0$ and hence the GCR breaks down at the $(i+1)$-th iteration.*

Proof. Since $\alpha_i = 0$ and $p_i \neq 0$, $r_{i+1} = r_i \neq 0$. So step 9 becomes $p_{i+1} = r_i + \sum_{j=0}^{i} h_{ji} p_j$. By relation (1.e), $p_{i+1} \in \langle p_0, p_1, \ldots, p_i \rangle$. Thus,

$$Ap_{i+1} \in \langle Ap_0, Ap_1, \ldots, Ap_i \rangle.$$

From relation (1.a), it is clear that Ap_{i+1} belongs to the orthogonal complement of $\langle Ap_0, Ap_1, \ldots, Ap_i \rangle$. Hence $Ap_{i+1} = 0$, so that $p_{i+1} = 0$ since A is nonsingular. At the $(i+1)$-th iteration, a division by zero takes place when computing α_{i+1}, which causes a breakdown of the GCR algorithm.

EXAMPLE 2.2. Let A be a skew-symmetric matrix. Since $A^T = -A$, for any nonzero residual vector r_0

$$(r_0, Ap_0) = (r_0, Ar_0) = -(r_0, Ar_0),$$

which implies $(r_0, Ap_0) = 0$. Thus, $\alpha_0 = 0$, $r_1 = r_0 \neq 0$, and $p_1 = 0$, so that GCR breaks down.

If A is definite, it can be easily shown that $p_i = 0$ implies $r_i = 0$. However, the above example shows that this does not hold for an indefinte matrix A. Therefore, $p_i = 0$ is a happy breakdown for a definite matrix A. In the following theorem, the converse of Theorem 2.1 is established.

THEOREM 2.3. *Suppose that $p_j \neq 0$ for all $j = 0, 1, \ldots, i$ and $r_{i+1} \neq 0$. If $\alpha_i \neq 0$, then $p_{i+1} \neq 0$, i.e., the GCR does not break down at the $(i+1)$-th iteration.*

Proof. First consider the case for $i = 0$. Assume that $p_1 = 0$. Then, from step 9 $r_1 = -h_{00}p_0 = -h_{00}r_0$. Since $r_1 \neq 0$, $h_{00} \neq 0$. From $(r_1, Ap_0) = 0$, we have $(r_0, Ap_0) = 0$ which implies $\alpha_0 = 0$. Hence, the theorem holds for $i = 0$. Next, consider the case for $i \geq 1$. Since $p_j \neq 0$ for all $j = 0, 1, \ldots, i$, by Theorem 2.1 $\alpha_j \neq 0$ for all $j = 0, 1, \ldots, (i-1)$. Assume that $p_{i+1} = 0$. Then, from step 9 and relation (1.e), it can be seen that $r_{i+1} \in \langle r_0, r_1, \ldots, r_i \rangle$. So, there exist constants $\beta_0, \beta_1, \ldots, \beta_i$, not all zero, for which

$$\sum_{j=0}^{i} \beta_j r_j = r_{i+1}. \tag{2.1}$$

Taking the inner product with Ap_0 and using relation (1.b),

$$\beta_0(r_0, Ap_0) = \sum_{j=0}^{i} \beta_j (r_j, Ap_0) = (r_{i+1}, Ap_0) = 0.$$

Since $\alpha_0 \neq 0$, $(r_0, Ap_0) \neq 0$ and hence $\beta_0 = 0$. Thus, equation (2.1) becomes

$$\sum_{j=1}^{i} \beta_j r_j = r_{i+1}.$$

Taking the inner products with $Ap_1, Ap_2, \ldots, Ap_{i-1}$ consecutively, similarly one obtains $\beta_1 = \beta_2 = \cdots = \beta_{i-1} = 0$. Thus $r_{i+1} = \beta_i r_i$ is obtained. Since $(r_{i+1}, Ap_i) = \beta_i(r_i, Ap_i) = 0$, $\beta_i = 0$ or $(r_i, Ap_i) = 0$. On the other hand, $\beta_i \neq 0$ since $r_{i+1} \neq 0$. Hence $(r_i, Ap_i) = 0$, so that $\alpha_i = 0$ which completes the proof.

If $r_{i+1} = 0$ in the above theorem, then GCR algorithm is terminated and gives the exact solution to $Ax = b$. Thus, the hypothesis $r_{i+1} \neq 0$ in Theorem 2.3 just means that the GCR does not produce the exact solution at the i-th iteration. From Theorems 2.1 and 2.3, we established a necessary and sufficient condition which causes a breakdown of the GCR when applied to indefinite linear systems. Now we modify the GCR method to obtain an algorithm which does not break down for indifinite problems. To do this, we need to show a preliminary result.

Suppose that $p_i \neq 0$ and r_i are generated from GCR algorithm. Since $r_{i+1} = r_i - \alpha_i Ap_i$ and $\alpha_i = \frac{(r_i, Ap_i)}{(Ap_i, Ap_i)} = \frac{(r_i, Ar_i)}{(Ap_i, Ap_i)}$, one obtains by (1.c)

$$\begin{aligned} \| r_{i+1} \|_2^2 &= \| r_i \|_2^2 - \alpha_i (r_i, Ar_i) \\ &= \| r_i \|_2^2 \left[1 - \left(\frac{r_i}{\| r_i \|_2}, \frac{Ar_i}{\| Ap_i \|_2} \right)^2 \right] . \\ &\leq \| r_i \|_2^2 \left[1 - \left(\frac{r_i}{\| r_i \|_2}, \frac{Ar_i}{\| Ar_i \|_2} \right)^2 \right] . \end{aligned}$$

Put $\widehat{r_i} = \frac{r_i}{\|r_i\|_2}$, $\widehat{Ar_i} = \frac{Ar_i}{\|Ar_i\|_2}$, and $\| \widehat{r_i} - \widehat{Ar_i} \|_2 = \epsilon$. Then, by a simple calculation, $(\widehat{r_i}, \widehat{Ar_i}) = 1 - \frac{\epsilon^2}{2}$. Hence,

$$\| r_{i+1} \|_2 \leq \frac{\epsilon\sqrt{4-\epsilon^2}}{2} \| r_i \|_2 . \tag{2.2}$$

Let $f(\epsilon) = \frac{\epsilon\sqrt{4-\epsilon^2}}{2}$, where $0 \leq \epsilon \leq 2$. Since $f(\epsilon)$ has the minimum 0 at $\epsilon = 0$ or 2 and the maximum 1 at $\epsilon = \sqrt{2}$, $0 \leq f(\epsilon) \leq 1$ on $[0, 2]$ and the following theorem is obtained.

THEOREM 2.4. *Suppose that $p_i \neq 0$ and r_i are generated from the GCR algorithm, and let ϵ be defined as above. If $\epsilon = 0$ or 2, that is, there exists a nonzero constant δ such that $Ar_i = \delta r_i$, then $\alpha_i \neq 0$ and $r_{i+1} = 0$. If $\epsilon = \sqrt{2}$, then $\alpha_i = 0$ and hence the GCR breaks down.*

The above theorem showed that if r_i is an eigenvector of the matrix A,

then $r_{i+1} = 0$. In practical, it is very unlikely for r_i to become an eigenvector of A. Thus, to make use of the idea in Theorem 2.4 the GCR algorithm is modified as follows. Replace step 2 in the GCR with p_0 equal to a vector w_0 and replace r_{i+1} in steps 7 through 10 with a vector w_{i+1}. For example, step 9 in the GCR is changed to $p_{i+1} = w_{i+1} + \sum_{j=0}^{i} h_{ji}p_j$. Note that $(r_i, Ap_i) = (r_i, Aw_i)$ since (1.b) still holds for this modified GCR. From the similar argument to the proof of Theorem 2.4, if w_i is chosen so that $Aw_i = \delta r_i$ for an arbitrary nonzero constant δ, then $\alpha_i \neq 0$ and $r_{i+1} = 0$. For simplicity, let $\delta = 1$ or -1. The ideal choice for w_i is $\delta A^{-1}r_i$, but solving this linear system has exactly the same kind of difficulties we have at the original linear system. To circumvent these difficulties, we choose a w_i so that Aw_i is a good approximation to r_i. Notice that $\delta = 1$ is chosen since $f(\epsilon)$ converges to 0 more rapidly as ϵ approaches 0 than as ϵ approaches 2, where ϵ denotes $\| \hat{r}_i - \widehat{Aw_i} \|_2$. Since r_i is given, finding a w_i such that $Aw_i \approx r_i$ leads to finding an easily invertible matrix, say M, such that M is a good approximation of A. Here M is called a *preconditioner* in the literature. Some techniques for finding such a M are described in [10]. In general, the choice of an easily invertible matrix M is an art in itself and depends on the application problem, the architecture of high-performance computer, and so on. Once such a M is chosen, we can easily find a w_i such that $Mw_i = r_i$. For this w_i, ϵ may be close to 0, so that from (2.2) the convergence of $\| r_{i+1} \|_2$ to 0 may be accelerated. We call this modified GCR with a preconditioner M the *preconditioned GCR*. It can be easily shown that the preconditioned GCR satisfies (1.a), (1.b), (1.f), $(Ap_i, Ap_i) \leq (Aw_i, Aw_i)$, $(r_i, Ap_i) = (r_i, Aw_i)$, and

$$\langle p_0, p_1, \ldots, p_i \rangle = langle\epsilon w_0, w_1, \ldots, w_i \rangle = K_{i+1}(M^{-1}A, w_0).$$

Using the above relations and $Mw_i = r_i$, Theorems 2.5 and 2.6 whose proofs are analogous to Theorems 2.1 and 2.3 respectively can be obtained.

THEOREM 2.5. *Suppose that $p_j \neq 0$ for all $j = 0, 1, \ldots, i$. If $\alpha_i = 0$, then $p_{i+1} = 0$ and hence the preconditioned GCR breaks down at the $(i+1)$-th iteration.*

THEOREM 2.6. *Suppose that $p_j \neq 0$ for all $j = 0, 1, \ldots, i$ and $r_{i+1} \neq 0$. If $\alpha_i \neq 0$, then $p_{i+1} \neq 0$, i.e., the preconditioned GCR does not break down at the $(i+1)$-th iteration.*

Finally, it will be shown that the preconditioned GCR does not break down until convergence even for indefinite linear systems if the preconditioner M is well chosen.

LEMMA 2.7. *Let $M = A - E$ and let $r_i \neq 0$ be a given vector. If $\| EA^{-1} \|_2 = \| I - MA^{-1} \|_2 < 1$, then M is invertible,*

$$\frac{\| Aw_i - r_i \|_2}{\| r_i \|_2} \leq \frac{\| EA^{-1} \|_2}{1 - \| EA^{-1} \|_2},$$

and

$$\frac{1}{\| A^{-1} \|_2 \| M \|_2} \leq \frac{\| Aw_i \|_2}{\| r_i \|_2},$$

where w_i is a unique vector such that $Mw_i = r_i$.

Proof. Since $M = (I - EA^{-1})A$ and $\| EA^{-1} \|_2 < 1$, it follows that $I - EA^{-1}$, and hence M, is invertible. Moreover, one has $Aw_i = AM^{-1}r_i = (I - EA^{-1})^{-1}r_i$. Since $\| EA^{-1} \|_2 < 1$,

$$(I - EA^{-1})^{-1} = I + EA^{-1} + (EA^{-1})^2 + \cdots .$$

It follows that $Aw_i - r_i = (EA^{-1} + (EA^{-1})^2 + \cdots) r_i$. Hence

$$\| Aw_i - r_i \|_2 \leq \frac{\| EA^{-1} \|_2}{1 - \| EA^{-1} \|_2} \| r_i \|_2,$$

which proves the first inequality. Since $\| r_i \|_2 \leq \| M \|_2 \| w_i \|_2$,

$$\frac{\| Aw_i \|_2}{\| r_i \|_2} \geq \frac{\| Aw_i \|_2}{\| M \|_2 \| w_i \|_2} \geq \frac{1}{\| M \|_2} \inf_{x \neq 0} \frac{\| Ax \|_2}{\| x \|_2}.$$

Since $\| A^{-1} \|_2^{-1} = \inf_{x \neq 0} \frac{\| Ax \|_2}{\| x \|_2}$ [14, p.182], the second inequality is proved.

THEOREM 2.8. *Let $M = A - E$ be a nonzero matrix and let $\alpha = \left(\frac{1}{\|A^{-1}\|_2 \|M\|_2}\right)^2$. Suppose that $r_i \neq 0$ is a given vector. If $\| EA^{-1} \|_2 < \frac{(\alpha+1) - \sqrt{\alpha+1}}{\alpha}$, then M is invertible and $(r_i, Aw_i) > 0$, where w_i is a unique vector such that $Mw_i = r_i$.*

Proof. Since $\frac{(\alpha+1)-\sqrt{\alpha+1}}{\alpha} < 1$ for $\alpha > 0$, by Lemma 2.7 M is invertible. From $\| Aw_i - r_i \|_2^2 = (Aw_i, Aw_i) + (r_i, r_i) - 2(r_i, Aw_i)$, one has

$$\frac{2(r_i, Aw_i)}{(r_i, r_i)} = 1 + \left(\frac{\| Aw_i \|_2}{\| r_i \|_2}\right)^2 - \left(\frac{\| Aw_i - r_i \|_2}{\| r_i \|_2}\right)^2.$$

Using Lemma 2.7, one obtains

$$\frac{2(r_i, Aw_i)}{(r_i, r_i)} \geq 1 - \left(\frac{\| EA^{-1} \|_2}{1 - \| EA^{-1} \|_2}\right)^2 + \alpha.$$

Letting $\beta = \| EA^{-1} \|_2$, this inequality becomes

$$\frac{2(r_i, Aw_i)}{(r_i, r_i)} \geq \frac{\alpha\beta^2 - 2(\alpha+1)\beta + (\alpha+1)}{(1-\beta)^2}.$$

It is easy to show that $\frac{\alpha\beta^2 - 2(\alpha+1)\beta + (\alpha+1)}{(1-\beta)^2} > 0$ for $\beta < \frac{(\alpha+1)-\sqrt{\alpha+1}}{\alpha}$. Therefore, $(r_i, Aw_i) > 0$.

Notice that $\frac{(\alpha+1)-\sqrt{\alpha+1}}{\alpha}$ in Theorem 2.8 is a real number between $\frac{1}{2}$ and 1 for any $\alpha > 0$. From this fact, the following theorem can be obtained.

THEOREM 2.9. *Let $M = A - E$ and $r_0 \neq 0$. If $\| EA^{-1} \|_2 \leq \frac{1}{2}$, then the preconditioned GCR with preconditioner M does not break down until a certain convergence criterion is satisfied.*

Proof. Since $r_0 \neq 0$, $p_0 \neq 0$ and by Theorem 2.8 $(r_0, Aw_0) > 0$. Note that $\alpha_i = \frac{(r_i, Aw_i)}{(Ap_i, Ap_i)}$ and $\| r_{i+1} \|_2^2 = \| r_i \|_2^2 - \alpha_i (r_i, Aw_i)$. Hence $\alpha_0 > 0$ and $\| r_1 \|_2 < \| r_0 \|_2$. If $r_1 = 0$, then the preconditioned GCR produces the exact solution. If $r_1 \neq 0$, then by Theorem 2.6 $p_1 \neq 0$ and by Theorem 2.8 $(r_1, Aw_1) > 0$. Hence $\alpha_1 > 0$ and $\| r_2 \|_2 < \| r_1 \|_2$. Continuing the same procedures as above, we can see that the preconditioned GCR does not break down until a certain convergence criterion is satisfied.

EXAMPLE 2.10. Consider $Ax = b$, where

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Clearly, A is indefinite. Choose $M = \begin{pmatrix} 0 & 0.9 \\ -0.9 & 0 \end{pmatrix}$. Then

$$E = \begin{pmatrix} 0 & 0.1 \\ -0.1 & 0 \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad \text{and} \quad EA^{-1} = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}.$$

Since $\| EA^{-1} \|_2 = 0.1$, by Theorem 2.9 the preconditioned GCR does not break down until convergence. Let's try the preconditioned GCR for this example. For given $x_0 = (0,0)^T$, we have $r_0 = (1,1)^T$, $p_0 = w_0 = M^{-1}r_0 = \frac{10}{9}(-1,1)^T$, and $\alpha_0 = \frac{9}{10}$. Hence $x_1 = (-1,1)^T$ which is the exact solution of $Ax = b$. However, the GCR breaks down for this example.

3. Parallel algorithms

In Section 2, we studied a necessary and sufficient condition which causes a breakdown of the GCR, and we showed that the preconditioned GCR does not break down if a preconditioner matrix M is well chosen. However, choosing such a M is not an easy problem. So we consider the GMRES method [12] which has better parallel structure than the GCR and does not break down for indefinite linear systems. Notice that the GMRES has some practical difficulties. That is, when the number of iteration steps, say l, increases, the number of vectors requiring storage increases like l and the number of arithmetic operations increases like l^2n, where n refers to the order of matrix A. For this reason, only the restarted version of the GMRES denoted by GMRES(k) is described and parallelized below, where k is a fixed integer parameter.

ALGORITHM 2 : GMRES(k)

1. Choose x_0
2. $r_0 = b - Ax_0$ and $p_1 = \frac{r_0}{\|r_0\|_2}$
3. **For** $i = 1, 2, \ldots, k$ **do:**
4. Compute Ap_i
5. $h_{ji} = (Ap_i, p_j)$, $j = 1, 2, \ldots, i$
6. $\hat{p}_{i+1} = Ap_i - \sum_{j=1}^{i} h_{ji}p_j$
   If $\hat{p}_{i+1} = 0$, set $k := i$ and then go to step 9
7. $h_{i+1,i} = \| \hat{p}_{i+1} \|_2$
   if $i = k$, then go to step 9
8. $p_{i+1} = \hat{p}_{i+1}/h_{i+1,i}$
9. **Form the approximate solution :**

10. Define $\overline{H}_k$ to be the $(k+1) \times k$ (Hessenberg) matrix whose nonzero entries are the coefficients h_{lm}, $1 \leq l \leq (m+1)$, $1 \leq m \leq k$ and define $P_k \equiv [p_1, p_2, \ldots, p_k]$. Find the vector y_k which minimizes $\| \| r_0 \|_2 e_1 - \overline{H}_k y \|_2$ over all vectors $y \in R^k$, where $e_1 = [1, 0, \ldots, 0]^T \in R^{k+1}$.
11. $x_k = x_0 + P_k y_k$
12. **Check the residual norm** $\| r_k \|_2$:
    If $\| r_k \|_2$ satisfies a certain criterion, then stop.
    Else, set $x_0 := x_k$ and then go to step 2

The set $\{p_1, p_2, \ldots, p_k\}$ generated from the GMRES(k) algorithm forms an l_2-orthonormal basis of the k-th Krylov subspace $K_k(A, r_0)$ and x_k minimizes the residual $\| b - Ax \|_2$ for all x in the affine subspace $x_0 + K_k(A, r_0)$. It was shown in [12] that if $\hat{p}_{i+1} = 0$ in step 6, then $x_i = x_0 + P_i y_i$ provides the exact solution of $Ax = b$ (i.e., $r_i = 0$). Thus GMRES(k) does not break down even for indefinite linear systems. But, there are instances where the residual norms produced by GMRES(k), although nonincreasing, do not converge to 0. The least squares problem $\min_{y \in R^k} \| \| r_0 \|_2 e_1 - \overline{H}_k y \|_2$ in step 10 is solved using a QR factorization of the upper Hessenberg matrix $\overline{H}_k$. The QR factorization of $\overline{H}_k$ is carried out using Givens rotations, and it is done progressively as each column of it appears, i.e., at every iteration in step 3. This allows the residual norm $\| b - Ax_i \|_2$ to be calculated without computing x_i at each iteration in step 3. See [12] for more details on GMRES(k).

Now we consider the vector and parallel implementation of GMRES(k) on the CRAY-2 computer with vector registers and a shared memory. The basic vector computations required by GMRES(k) are vector inner products (SDOT), vector updates, and sparse matrix times vector operations. Each of steps 2 and 4 involves a sparse matrix times vector operation, so we will first consider parallelization of this operation. The storage format for a sparse matrix A is as follows: Only the non-zero elements of the matrix A are stored in an array, say PA, counting across rows (except for the diagonal entry, which must appear first in each row). In other words, for each row in the matrix A, put the diagonal entry in the array PA and then put in the other non-zero elements going across the row (except the diagonal) in order. Let n_p denote the number of processors of the computer to be used. For simplicity of exposition, assume that n is divisible by n_p. The matrix A is divided into n_p row blocks such that each row block

is an $m \times n$ submatrix of A, where $m = n/n_p$. If we denote n_p row blocks by $A_1, A_2, \ldots, A_{n_p}$, then the computation of Ax can be done in parallel by assigning the computation of $A_j x$ to the Processor j, where $x \in R^n$ and $j = 1, 2, \ldots, n_p$. This provides a good load balancing if the non-zero elements of A are well distributed. Since only the non-zero elements of A are stored, special care is needed for parallel implementation of this operation.

Step 5 has i SDOTs which are parallelized by assigning them to n_p processors. This is done using Autotasking directives on the CRAY-2, see [1] for Autotasking directives. If i is not divisible by n_p, a poor load balancing can be expected. Since the number of floating point operations in SDOT is $O(n)$, this problem can be ignored. Numerical experiments in Section 4 also prove this.

Steps 6 and 11 have i and k vector updates, respectively. Each of multiple vector updates in steps 6 and 11 can be grouped together in a single general matrix times vector operation (SGEMV). The functionalities of SGEMV are described in [6]. The functionality of SGEMV used in these steps is $y := \beta y + \alpha Bx$, where x and y are vectors, α and β are constants, and B is a general matrix. Since SGEMV was already vectorized and parallelized by the vendor on the CRAY-2 [2], no further efforts are necessary for these steps. Step 11 is the typical one that the GMRES(k) has a better parallelism than the GCR. This can be explained as follows. The GCR does only one update of the approximate solution x_0 per iteration, while the GMRES(k) does k updates of x_0 at a time, so that the number of memory references is reduced by reusing vector registers and more coarsely grained parallelism is achieved. The number of synchronization points is also reduced.

Steps 7 and 8 are vectorized, but they are conditionally parallelized. This means that they are parallelized if n is greater than a certain number which is internally set at run-time on the CRAY-2. These steps are parallelized using FPP directives, see [1] for FPP directives. Since k is usually much smaller than n, the number of arithmetic operations involved in step 10 is much smaller than those in other steps. Thus, step 10 is not parallelized and most of its arithmetic operations are executed in scalar mode. In most of application problems, the value of k commonly used by many researchers ranges from 5 to 20.

As was seen in the above, step 10 in the GMRES(k) is not paral-

lelized and is mostly executed in scalar mode. This is a main motivation for proposing another algorithm, called GMORTH(k) from now on, which has better vector and parallel structures than GMRES(k). The algorithm GMORTH(k) first forms an *l_2-orthogonal basis* $\{Ap_1, Ap_2, \ldots, Ap_k\}$ of the subspace $AK_k(A, r_0) = \langle Ar_0, A^2r_0, \ldots, A^kr_0\rangle$ as well as a *basis* $\{p_1, p_2, \ldots, p_k\}$ of the Krylov subspace $K_k(A, r_0)$, and then it requires the residual minimization property over an affine subspace $x_0 + K_k(A, r_0)$. Let $x_k \in x_0 + K_k(A, r_0)$. Since $K_k(A, r_0) = \langle p_1, p_2, \ldots, p_k\rangle$, we can put $x_k = x_0 + \sum_{j=1}^{k} \alpha_j p_j$, where α_j's are some constants. The α_j's are chosen such that the residual norm $\| b - Ax_k \|_2$ is minimized. Using the property $(Ap_i, Ap_j) = 0$ for $i \neq j$, we can easily obtain for $i = 1, 2, \ldots, k$

$$\alpha_i = \frac{(r_0, Ap_i)}{(Ap_i, Ap_i)} \text{ and } \| r_i \|_2^2 = \| r_0 \|_2^2 - \sum_{j=1}^{i} \alpha_j (r_0, Ap_j).$$

Thus, the GMORTH(k) algorithm based on the Arnoldi process for the subspace $AK_k(A, r_0)$ is obtained.

ALGORITHM 3 : GMORTH(k)

1. Choose x_0
2. $r_0 = b - Ax_0$ and compute $\| r_0 \|_2$
3. Set $p_1 := r_0$ and then compute Ap_1
4. **For** $i = 1, 2, \ldots, k$ **do:**
5. $\quad \alpha_i = (r_0, Ap_i)/(Ap_i, Ap_i)$
6. $\quad \| r_i \|_2^2 = \| r_{i-1} \|_2^2 - \alpha_i (r_0, Ap_i)$
   $\quad$ If $\| r_i \|_2$ satisfies a certain criterion, then go to step 15
   $\quad$ If $i = k$, then go to step 11
7. $\quad$ Compute $A^2 p_i = A(Ap_i)$
8. $\quad h_{ji} = (A^2 p_i, Ap_j)/(Ap_j, Ap_j)$, $j = 1, 2, \ldots, i$
9. $\quad p_{i+1} = Ap_i - \sum_{j=1}^{i} h_{ji} p_j$
10. $\quad Ap_{i+1} = A^2 p_i - \sum_{j=1}^{i} h_{ji} Ap_j$
11. **Form the approximate solution and residual:**
12. $\quad x_k = x_0 + \sum_{j=1}^{k} \alpha_j p_j$
13. $\quad r_k = r_0 - \sum_{j=1}^{k} \alpha_j Ap_j$
14. $\quad$ Set $x_0 := x_k$ and $r_0 := r_k$, and then go to step 3
15. $x_i = x_0 + \sum_{j=1}^{i} \alpha_j p_j$ **and then stop**

Next, one important theorem which ensures that the GMORTH(k) does not break down even for indefinite linear systems is provided.

THEOREM 3.1. *Suppose that* $p_j \neq 0$ *for all* $j = 1, 2, \ldots, i$. *If* $r_i \neq 0$, *then* $p_{i+1} \neq 0$, *i.e., the GMORTH(k) does not break down at the* $(i+1)$*-th iteration.*

Proof. Assume that $p_{i+1} = 0$. From step 9, $Ap_i = \sum_{j=1}^{i} h_{ji} p_j$. Since $K_i(A, r_0) = \langle p_1, p_2, \ldots, p_i \rangle$, there exist constants $\beta_1, \beta_2, \ldots, \beta_i$, not all zero, for which $Ap_i = \sum_{j=1}^{i} \beta_j A^{j-1} r_0$. Thus we have

$$\beta_1 r_0 = Ap_i - \sum_{j=2}^{i} \beta_j A^{j-1} r_0. \tag{3.1}$$

Now we want to show that $\beta_1 \neq 0$ in (3.1). Suppose that $\beta_1 = 0$. Then from (3.1), $Ap_i = \sum_{j=2}^{i} \beta_j A^{j-1} r_0$. Since $\langle Ap_1, \ldots, Ap_{i-1} \rangle = \langle Ar_0, \ldots, A^{i-1} r_0 >$ and $p_i \neq 0$, there exist some constants γ_j's, not all of which are 0, such that $Ap_i = \sum_{j=1}^{i-1} \gamma_j Ap_j$. Hence $p_i = \sum_{j=1}^{i-1} \gamma_j p_j$, where constants γ_j's are not all 0. This is a contradiction to the linear independence of $\langle p_1, \ldots, p_i \rangle$. Since it was shown that $\beta_1 \neq 0$, by (3.1) $r_0 \in \langle Ap_1, \ldots, Ap_i \rangle$. Notice that $r_i = r_0 - \sum_{j=1}^{i} \alpha_j Ap_j$ and α_j's are chosen so that $\| r_0 - \sum_{j=1}^{i} \xi_j Ap_j \|_2$ is minimized over all real numbers ξ_j's. Hence, $r_0 \in \langle Ap_1, \ldots, Ap_i \rangle$ implies $r_i = 0$. This completes the proof.

The vector and parallel implementation of the GMORTH(k) on the CRAY-2 is described in the following. Each of steps 2, 3, and 7 involves a sparse matrix times vector operation whose parallelization was already discussed in the GMRES(k). Each of steps 9, 10, 12, and 13 can be replaced with a general matrix times vector operation (SGEMV) which was already optimized for the CRAY-2 with vector and parallel processing. Most of arithmetic operations required in other steps are vector inner products that can be also parallelized using FPP directives on the CRAY-2 when n is large. Notice that almost all of arithmetic operations involved in GMORTH(k) are executed in vector and/or parallel mode. Arithmetic operations executed in scalar mode are 1 division in step 5, 1 multiplication in step 6, and i divisions in step 8. The richness of vector and parallel codes is a big advantage of this algorithm as compared with the GMRES(k).

As can be seen in the GMORTH(k) algorithm, it requires storage for both $\{p_j\}_{j=1}^{k}$ and $\{Ap_j\}_{j=1}^{k}$, whereas the GMRES(k) requires storage for only $\{p_j\}_{j=1}^{k}$. Since k is usually small, this is not a big problem for high performance supercomputers with large memory. The number of vector and scalar operations for both the GMRES(k) and GMORTH(k) is sum-

marized in Table 1 (only multiplications and divisions are counted). In this paper, vector operations refer to arithmetic operations which involve vectors of size n, and scalar operations refer to arithmetic operations which involve vectors of size $\leq k$. The vector operations used in these two algorithms are SDOTs and vector updates, and multiple vector updates are grouped together in a single SGEMV wherever possible. SPMV in Table 1 denote a sparse matrix times vector operation. If we denote the number of nonzero elements in A by NZ, then the number of multiplications required for executing one SPMV operation is NZ.

Table 1. Arithmetic operation counts for completing one period of the GMRES(k) and GMORTH(k)

Algorithm	Scalar Operations	Vector Operations	SPMV
GMRES(k)	$\frac{5k^2+13k}{2}$	$(k^2+4k+1)n$	$k+1$
GMORTH(k)	$\frac{k^2+3k}{2}$	$\frac{(3k^2+5k)n}{2}$	k

Table 1 shows that the GMRES(k) has less vector operations, but more scalar operations, than the GMORTH(k). Since k is usually very small compared with n, the number of total arithmetic operations in GMRES(k) is less than those in GMORTH(k). So, there is no doubt that GMRES(k) will perform better than GMORTH(k) on scalar computers. However, for vector or parallel computers the performance of an algorithm depends upon both the number of total arithmetic operations and how much portion of a code is vectorizable or parallelizable. This is one reason why GMORTH(k) is considered even if it has more arithmetic operations than GMRES(k). Next section will show that for small values of k the GMORTH(k) performs better than the GMRES(k) on the CRAY-2 vector and parallel computer.

4. Numerical results

In this section, we present numerical results for both the parallel GMRES(k) and the parallel GMORTH(k) on the CRAY-2 computer at the National Center for Supercomputing Applications (NCSA) in Illinois. The CRAY-2 has 4 processors and a central memory of 4 gigabytes, and each processor has 8 64-bit vector registers of length 64. Wall-clock time (elapsed time) was measured to evaluate the efficiency of parallel algorithms. All test runs were performed using 64-bit arithmetics (i.e., single

precision on the CRAY).

The test problem considered is to solve a block tridiagonal linear system $Ax = b$ with

$$A = \begin{pmatrix} D & B & & & \\ C & D & B & & \\ & \ddots & \ddots & \ddots & \\ & & C & D & B \\ & & & C & D \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} 4 & \alpha & & & \\ \beta & 4 & \alpha & & \\ & \ddots & \ddots & \ddots & \\ & & \beta & 4 & \alpha \\ & & & \beta & 4 \end{pmatrix},$$

where $\alpha = -1 + \delta$, $\beta = -1 - \delta$, $C = (-1 - \gamma)I$, and $B = (-1 + \gamma)I$. This kind of matrix A comes from the five point discretization of a linear elliptic partial differential equation involving a nonselfadjoint operator. The right-hand side vector b is chosen so that $b = A[1, 1, \ldots, 1]^T$. The initial vector x_0 is set to 0 for both algorithms to make a fair comparison. The termination criterion (i.e., convergence criterion) used for both algorithms is $\frac{\|r_i\|_2}{\|b\|_2} < 10^{-6}$.

Test runs were made for 4 different pairs of δ and γ; $\delta = \gamma = 0.2$, $\delta = 0.2$ and $\gamma = 0$, $\delta = \gamma = 0.01$, and $\delta = 0.01$ and $\gamma = 0$. Since the same conclusions were obtained from these 4 pairs of δ and γ, we only report the performance results for $\delta = \gamma = 0.2$ in Tables 2 to 4. Wall-clock time is reported in seconds. ITER refers to the number of iterations required for getting an approximate solution which satisfies the convergence criterion mentioned above, and n_p denotes the number of processors used for execution. The numbers in the parentheses represent parallel speedups which are calculated using the wall-clock time obtained from serial execution.

From Tables 2 to 4, performance results for both the GMRES(k) and GMORTH(k) can be summarized as follows. The number of iterations to get an approximate solution with a specified accuracy is almost the same for both algorithms, but the GMORTH(k) performs *better* than the GMRES(k) for both $k = 10$ and 20. For example, the execution time for GMRES(10) is 5 to 8% longer than that for GMORTH(10). For both algorithms, $k = 10$ produces better performance than $k = 20$. Unfortunately, the optimal number of k depends upon application problems considered. Notice that the value of k commonly used ranges from 5 to 20. For small values of $k \leq 20$, we recommend the use of GMORTH(k) in solving large sparse nonsymmetric linear systems on vector computers even if it has more arithmetic operations than the GMRES(k). If k is large,

then the use of GMRES(k) is recommended. Both the parallel GMRES(k) and parallel GMORTH(k) perform very efficiently on the CRAY-2 with 4 processors, see parallel speedups in Tables 2 and 3.

Table 2. Performance results for GMRES(10)

$\sqrt{n}$	ITER	$n_p = 1$	$n_p = 2$	$n_p = 4$
48	158	1.2705	0.6611(1.92)	0.3529(3.60)
64	207	2.9980	1.5514(1.93)	0.8057(3.72)
100	261	9.1983	4.6828(1.96)	2.3972(3.84)

Table 3. Performance results for GMORTH(10)

$\sqrt{n}$	ITER	$n_p = 1$	$n_p = 2$	$n_p = 4$
48	160	1.2109	0.6310(1.92)	0.3345(3.62)
64	206	2.7560	1.4302(1.93)	0.7389(3.73)
100	261	8.4880	4.3335(1.96)	2.1974(3.86)

Table 4. Performance results for GMRES(20) and GMORTH(20)

	GMRES(20)		GMORTH(20)	
$\sqrt{n}$	ITER	$n_p = 1$	ITER	$n_p = 1$
48	194	1.5407	196	1.5467
64	258	3.6782	256	3.5932
100	359	12.3478	358	12.0915

References

1. Cray Research, Inc., *Autotasking User's Guide(SN-2088)*, Eagan, Minnesota, 1988.
2. Cray Research, Inc., *Volume 3 : UNICOS Math and Scientific Library Reference Manual(SR-2081)*, Eagan, Minnesota, 1991.
3. A. T. Chronopoulos, *A class of parallel iterative methods implemented on multiprocessors*, Technical Report UIUCDCS-R-86-1267, University of Illinois at Urbana, 1986.
4. A. T. Chronopoulos and C. W. Gear, *On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy*, Parallel Comput., **11**(1989), 37-53.
5. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Solving linear systems on vector and shared memory computers*, SIAM, Philadelphia, Pennsylvania, 1991.
6. J. J. Dongarra, J. J. Du Croz, S. J. Hammarling, and R. Hanson, *An extended set of Fortarn basic linear algebra subprograms*, ACM Trans. Math. Software, **14**(1988), 1-17.
7. 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.

8. H. C. Elman, *Iterative methods for large sparse nonsymmetric systems of linear equations,* Research Report 229, Yale University, 1982.
9. M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems,* J. Res. Nat. Bur. Standards **49**(1952), 409-435.
10. T. Meijerink and H. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix,* Math. Comp. **31**(1977), 148-162.
11. Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems,* Math. Comp. **37**(1981), 105-126.
12. Y. Saad and M. H. Schultz, *GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput. **7**(1986), 856-869.
13. P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput. **10**(1989), 36-52.
14. G. W. Stewart, *Introduction to matrix computations,* Academic Press, Inc., Orlando, Florida, 1973.

Department of Mathematics
College of Natural Sciences
Chungbuk National University
Cheongju, 360-763, Korea
E-mail address: gmjae@cbucc.chungbuk.ac.kr