ELSEVIER Parallel Computing 23 (1997) 1005-1019

PARALLEL COMPUTING

A block variant of the GMRES method on massively parallel processors

Guangye Li [*]

Cray Research, A Silicon Graphics Company, 655E Lone Oak Drive, Eagan, MN 55121, USA

Received 25 January 1996; revised 2 November 1996

Abstract

This paper presents a block variant of the GMRES method for solving general unsymmetric linear systems. This algorithm generates a transformed Hessenberg matrix by solely using block matrix operations and block data communications. It is shown that this algorithm with block size s, denoted by BVGMRES(s, m), is theoretically equivalent to the GMRES(s, m) method. The numerical results demonstrate that this algorithm can be more efficient than the standard GMRES method on a cache based single CPU computer with optimized BLAS kernels. Furthermore, the gain in efficiency is more significant on MPPs due to both efficient block operations and efficient block data communications. Preliminary numerical results on some real-world problems also show that this algorithm may be stable up to some reasonable block size.

Keywords: Linear system solver; GMRES; Krylov subspace; Unsymmetric matrices; Block variant; Distributed memory multiprocessors

1. Introduction

This paper is concerned with the iterative solution of the system of linear equations

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

where A is an unsymmetric real matrix of order n. When n is large and A is sparse, iterative methods are usually attractive, especially on parallel computers. The GMRES method and its variant GMRES(m) proposed by Saad and Schultz [19] are robust Krylov subspace methods for general unsymmetric systems. The convergence properties of the

[*] Fax: +1-612-6833699; e-mail: gli@crav.com.

GMRES method have been studied in [19] and further studied by Nachtigal, Reddy and Trefethen [14].

In the standard GMRES method, in addition to (sparse) matrix–vector multiplications, most computations are vector–vector operations. In the language of the basic linear algebra subprograms (BLAS) [13], they are primarily SDOTs (inner-products) and SAXPYs (vector updates) i.e. level 1 BLAS operations. It is well-known that BLAS 2 [9] and BLAS 3 [8] operations based on blocks of submatrices are much more efficient than BLAS 1 operations on parallel computers with optimized BLAS kernels. Thus, recent research has focused on the application of block operations in iterative algorithms.

The block conjugate gradient (BCG) [15] and block GMRES (BGMRES) methods, based on the block Arnoldi method [3,18,20,21], are good candidates for systems with multiple right-hand sides. However, the above block methods have some room for improvement, particularly in the single right-hand side case. In order to use the block algorithms in this case, additional artificial right-hand sides and their corresponding residual vectors must be created to fill up the blocks and this increases the cost of each iteration. Assume the block size of the block algorithm is s. Then, each iteration of the block algorithm takes at least s (sparse) matrix–vector multiplications. It is the hope that the number of iterations needed for convergence by the block algorithm will be reduced by a factor of s over the standard algorithm. If that were the case, the total number of (sparse) matrix–vector multiplications needed in the block method would be the same as the standard method. Unfortunately, the convergence rates of the block algorithms are far more complicated, especially when restarting has to be used [21]. Furthermore, current block algorithms have the additional problem that the block residuals may not have full rank, even when the matrix is nonsingular. Therefore, in order to obtain a stable algorithm, one has to deal with the possible block residual rank deficiency problem.

Another approach using BLAS 3 operations in iterative methods for unsymmetric systems is the s-step technique proposed by Chronopoulos and Kim [4–7]. In s-step methods, at each iteration, a block of the Krylov subspace $V=(r, Ar, A^2r, \dots, A^{s-1}r)$ is generated from a given vector $r \in R^n$. Then, a block Gram–Schmidt procedure is applied to orthogonalize V or V^+ to previous blocks using BLAS 3 operations, where $V^+=(Ar, A^2r, \dots, A^sr)=AV$. One of the advantages of the s-step methods over the standard block methods is that V or V^+ has full rank provided that the degree of the minimal polynomial of r is greater than s. Moreover, the number of iterations needed for convergence is n/s under some reasonable conditions (see Refs. [4,5]).

However, there is an inherent difficulty in the original s-step methods, for in the block orthogonalization procedure, a series of linear systems with V^TV or $V^{+T}V^+$ as coefficient matrix must be solved. In general, these linear systems are ill-conditioned when s is large. Consequently, the s-step methods are usually stable only for small s, say, less than 4 [5,23]. O'Leary [15] laid out the advantages of using a modified Gram–Schmidt algorithm to compute the orthonormal basis of a block of the Krylov subspace. O'Leary and Whitman implemented parallel QR factorization by Householder and modified Gram–Schmidt algorithms [16]. Chronopoulos and Kim [5] proposed using a similar idea to orthogonalize V or V^+ to overcome the ill-conditioning in the s-step methods. Swanson and Chronopoulos [23] showed that this idea works well in the s-step ORTHOMIN method. However, there are some difficulties in applying this idea to the s-step GMRES method because GMRES is far more complex than the ORTHOMIN method. In ORTHOMIN, obtaining an A^TA-orthogonal basis for the Krylov subspace is sufficient to carry out the iterative procedure. In contrast, the GMRES method must generate an upper Hessenberg matrix as well as an orthogonal basis of the Krylov subspace. Some related work on this subject may be found in Refs. [1,11,17,22].

In this paper, we propose an s-step type algorithm which generates orthonormal basis of the Krylov subspace. This algorithm computes a transformed Hessenberg matrix block–column by block–column by using BLAS 3 operations and then solves a transformed linear least squares problem. This technique avoids additional (sparse) matrix–vector multiplications and some complicated computations. Although, formally, this algorithm seems different from the standard GMRES method, we will show that it is equivalent to the standard GMRES in the sense that they generate the same basis of the Krvlov subspace and they solve equivalent least squares problems. Our numerical results show that it is stable up to some reasonable size of s. The numerical examples also show that the new algorithm uses the same number of (sparse) matrix–vector multiplications to converge as the standard GMRES does in many cases, which is the best one can expect from block type algorithms.

While our preliminary numerical results show that the new algorithm can be more efficient than the standard GMRES method on a cache based single CPU computer with optimized BLAS kernels, the speed-up is much more significant on distributed memory parallel computers, especially MPPs. This is due to both efficient block operations and efficient block data communications among PEs.

The rest of this paper is organized in the following way. Section 2 contains a simplified description of the block variant of the GMRES method and some properties of it. Section 3 provides implementation details of our algorithm and discusses implementation issues on distributed memory computers. Section 4 briefly discusses some efficiency and accuracy issues. Given in Section 5 are some numerical results of our algorithm in comparison to the standard GMRES method on a Cray T3D computer with 1 to 128 PEs. We conclude this paper with some comments and possible future work in Section 6.

2. A block variant of the GMRES method

As mentioned in the previous section, the difficulties for the s-step Arnoldi and GMRES algorithms arise in generating the block Hessenberg matrix H. In Chronopouios and Kim's s-step Arnoldi algorithm [5], H is generated in such a way that at step k, the essential equation

$$AV = VH + w_k e_l^T \tag{2.1}$$

is satisfied, where $l = s \cdot k$, V is the Krylov subspace with dimension of l, $w_k \in R^n$ and

e_l is the lth column of the $l \times l$ identity matrix. Thus, H_{ik}, the $(i,\ k)$th block of the block Hessenberg matrix H in Eq. (2.1), can be obtained by solving equations

$$V_i^{\mathrm{T}} V_i H_{ik} = V_i^{\mathrm{T}} A V_k + F_i, \quad i = 1, \ldots, k, \tag{2.2}$$

where

$$V_i = \left(v_i,\ Av_i,\ A^2 v_i,\ \ldots,\ A^{s-1} v_i\right)$$

for a given v_i and F_i stands for some complicated computations by using values computed in previous steps ($i = 1, \ldots, k-1$) [5].

Note that AV_k is actually the last $s-1$ columns of V_k plus one more vector $A^s v_k$. Therefore, at step k, s (sparse) matrix–vector multiplications will be enough to carry out the original s-step Arnoldi procedure. The major problem which causes instability in this procedure with a relatively large s is that the subblock of the Krylov subspace V_k is not orthogonal. Chronopoulos and Kim [5] proposed to solve this problem by using the modified Gram–Schmidt (MGS) method to orthogonalize the columns of V_k to obtain $\overline{U}_k$ and then orthogonalize $\overline{U}_k$ to the previous blocks of the Krylov subspace (U_i, $i = 1, \ldots, k-1$) to get a new block basis U_k. In addition to nonorthonormal U_k because of the block orthogonalization to the previous block basis, a major difficulty arises when solving the equation

$$U_i^{\mathrm{T}} U_i H_{ik} = U_i^{\mathrm{T}} A U_k + F_i \tag{2.3}$$

because, unlike in the case of Eq. (2.2) where AV_k is readily available, one does not have AU_k in hand. If one insists in using the value of AU_k, then some very complicated recursive computations are unavoidable.

We propose to overcome the above difficulties by computing a transformed upper Hessenberg matrix which uses the value of AV_k instead of AU_k. The transformation matrix is an $l \times l$ block upper triangular matrix which is computed block by block in the block orthogonalization and MGS procedures. At each step, a block column of the Hessenberg matrix $((U_1, U_2, \ldots, U_k)^{\mathrm{T}} A V_k)$ is set up by using the values obtained from the block orthogonalization and MGS procedures without additional cost. Furthermore, because U_k is orthonormal, there are no linear systems to be solved in generating the Hessenberg matrix in our algorithm. As a consequence, our algorithm uses about the same number of operations as that used by GMRES in generating the orthonormal basis of the Krylov subspace and the upper Hessenberg matrix. We solve the linear least squares problem by using the transformed Hessenberg matrix and then transform the solution back to the original space. Because the Hessenberg matrix in the original space is never computed, the computation of complicated terms F_i in Eq. (2.3) is avoided.

We will later prove that our algorithm is equivalent to standard GMRES in the sense that they generate the same basis of the Krylov subspace, and they solve equivalent least squares problems. To distinguish it from the block GMRES algorithm (BGMRES), it is called the block variant of the GMRES algorithm (BVGMRES). As in the GMRES(m) method, it is impossible to save the whole basis of the Krylov subspace when the dimension of the problem is large. One variant of the BVGMRES algorithm is to restart the algorithm after every m steps. For the convenience of theoretical discussions, we first give a simplified version of the BVGMRES(s, m) algorithm. For implementation details, please refer to the remarks following this algorithm and Algorithm 3.1.

Algorithm 2.1. Given an $x_0 \in R^n$, and integers $s > 1$ and $m > 0$, for $i = 0$ until convergence do

```
1. Compute $r_i = f - Ax_i$.
2. Compute $\alpha_i = \|r_i\|_2$.
3. Set $v_1 = r_i/\alpha_i$.
4. For $k = 1$ until $m$ do
   (a) Set $v_k^1 = v_k$.
   (b) Compute $V_k = (v_k^1, Av_k^1, A^2 v_k^1, \ldots, A^{s-1} v_k^1)$.
   (c) For $j = 1$ until $k - 1$ do
       i. $R_{jk} = U_j^T V_k$.
       ii. $V_k = V_k - U_j R_{jk}$.
   (d) Apply QR factorization to $V_k$ i.e. compute $V_k = U_k R_{kk}$.
   (e) Compute $w_k = Au_k^s$, where $u_k^s$ is the $s$th column of $U_k$.
   (f) For $j = 1$ until $k$ do
       i. $a_j = U_j^T w_k$.
       ii. $w_k = w_k - U_j a_j$
   (g) Set nonzero part of the $k$th block column of the Hessenberg matrix $H_k' = (U_1, \ldots, U_k)^T AV_k$ by using $R_{jk}$ and $a_j$, $(j = 1, \ldots, k)$ from steps 4(c) and 4(f) (see steps 6(c) and 6(h) of Algorithm 3.1 for implementation details).
   (h) Compute $\beta_k = \|w_k\|_2$.
   (i) Set $v_{k+1} = w_k/\beta_k$.
5. Solve for $x_{i+1}$: Define the block upper triangular matrix
```

$$R = \begin{bmatrix} R_{11} & R_{12} & \cdot & \cdot & R_{1m} \\ 0 & R_{22} & \cdot & \cdot & R_{2m} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 0 & R_{mm} \end{bmatrix} \tag{2.4}$$

and the upper Hessenberg matrix

$$\tilde{H} = \begin{bmatrix} \begin{bmatrix} H_1' \\ h_1^T \\ 0 \end{bmatrix}, & \begin{bmatrix} H_2' \\ h_2^T \\ 0 \end{bmatrix}, \ldots, & H_m' \end{bmatrix}, \tag{2.5}$$

where R_{jk} and H_k' $(k = 1, \ldots, m,\ j \le k)$ are generated at steps 4(c) and 4(g), respectively, and $h_k = (0, 0, \ldots, 0, \beta_k)^T \in R^{s \cdot k}$ $(k = 1, \ldots, m)$. Also define $l = s \cdot m$ and

$$\bar{H} = \begin{bmatrix} \tilde{H} \\ h_m^T \end{bmatrix},$$

where $h_m = (0, 0, \ldots, 0, \beta_m)^T \in R^l$.

(a) Set $b = (\alpha_i, 0, \ldots, 0)^{\mathrm{T}} \in R^{l+1}$.
(b) Apply Givens rotations on $\bar{H}$ and b i.e. compute $Q^{\mathrm{T}}\bar{H} = [{}^{T}_{0}]$ and $\bar{b} = Q^{\mathrm{T}}b$, where Q is an orthonormal matrix and T is an $l \times l$ upper triangular matrix.
(c) Solve the triangular system $T\tilde{y} = \tilde{b}$, where $\tilde{b} \in R^{l}$ contains the first l elements of $\bar{b}$.
(d) Compute $y = R\tilde{y}$.
(e) Compute $x_{i+1} = x_i + Uy$.

Remark 2.1. Note that since v_k^1 is already orthogonal to U_j, $j = 1, \ldots, k-1$, the first column of $R_{.k}$ at step 4(c) should be the identity and the block size in the block modified Gram–Schmidt procedure needs only $s-1$. Also note that w_k at step 4(e) is generated by the last column of U_k instead of V_k. Thus, the last column of $R_{.k}$ should be also the identity when applying R at step 5(d). Therefore, when multiplying $\tilde{y}$ by R_{jk}, $j = 1, \ldots, k-1$ at step 5(d), the matrix size should be $s \times (s-2)$ and the matrix size of R_{kk} should be $(s-1) \times (s-1)$.

Remark 2.2. There is no doubt that preconditioners are very important in practical iterative methods. It should be fairly easy to plug in a preconditioner in the BVGMRES(s, m) algorithm to have a preconditioned BVGMRES(s, m) algorithm.

Next we discuss some properties of Algorithm 2.1. For simplicity, we first introduce some notation omitting subscripts. Let

$$V = (V_1, V_2, \ldots, V_m), \tag{2.6}$$

and

$$U = (U_1, U_2, \ldots, U_m), \tag{2.7}$$

where V_k and U_k ($k = 1, \ldots, m$) are generated at steps 4(b) and 4(d), respectively.

The following results can be easily derived from Algorithm 2.1.

Proposition 2.1. *Let V be defined in Eq. (2.6), U be defined in Eq. (2.7) and $\tilde{H}$ be defined in Eq. (2.5). Then, U is an orthonormal basis of the Krylov subspace $K(v_1, A, l)$ and the upper Hessenberq matrix $\tilde{H}$ satisfies*

$$\tilde{H} = U^{\mathrm{T}}AV.$$

Proposition 2.2. *Let $H = U^{T}AU$ and R be defined in Eq. (2.4). Then,*

$$UR = V, \tag{2.8}$$

and

$$HR = \tilde{H}.$$

The following result is a corollary of the above results. It establishes the theoretical foundation of Algorithm 2.1.

Proposition 2.3. *For $l = s \cdot k < n$, let $U_{l+1} = (U, v_{k+1})$ and*

$$\overline{H} = \begin{bmatrix} \tilde{H} \\ h_k^{\mathrm{T}} \end{bmatrix},$$

where $h_k = (0, 0, \ldots, 0, \beta_k)^{\mathrm{T}} \in R^l$ and β_k is defined in step 4(h) of Algorithm 2.1. Then,

$$AV = U\tilde{H} + w_k e_l^{\mathrm{T}} = U_{l+1}\overline{H}. \tag{2.9}$$

We now apply the idea of solving a least squares problem to get a new approximation to the solution of Eq. (1.1) from the previous step, i.e. given x_0, an initial guess of the solution of Eq. (1.1), let the residual $r_0 = f - Ax_0$ be the starting vector v_1 in Algorithm 2.1, then an approximate solution can be obtained by solving the problem

$$\min_{z \in K} \| f - A(x_0 + z) \|_2, \tag{2.10}$$

where K is the Krylov subspace spanned by the columns of V.

Proposition 2.4. *Let $\bar{y}_\star$ be the solution of the problem*

$$\min_{\bar{y} \in R^l} \| \alpha e_1 - \overline{H}\bar{y} \|_2, \tag{2.11}$$

where $\alpha = \|r_0\|_2$ and e_l is the first column of the identity matrix $I_{l \times l}$. Also let

$$y_\star = R\bar{y}_\star.$$

Then $z_\star = Uy_\star$ is the solution of Eq. (2.10).

Proof. Since U is an orthonormal basis of K, for any $z \in K$, there is an $y \in R^l$ such that $z = Uy$. Define the function

$$J(y) = \| f - A(x_0 + z) \|_2 = \| f - A(x_0 + Uy) \|_2.$$

Let $\bar{y} = R^{-1}y$. Then by Eqs. (2.8) and (2.9),

$$\begin{aligned} J(y) &= \| f - A(x_0 + U_y) \|_2 \\ &= \| r_0 - AURR^{-1}y \|_2 \\ &= \| r_0 - AV\bar{y} \|_2 \\ &= \| r_0 - U_{l+1}\overline{H}\bar{y} \|_2 \\ &= \| U_{l+1}(\alpha e_1 - \overline{H}\bar{y}) \|_2 \\ &= \| \alpha e_1 - \overline{H}\bar{y} \|_2. \end{aligned}$$

which proves the desired result. □

From Proposition 2.4, once the solution of least squares problem (Eq. (2.11)) is obtained, a new approximation to the solution of Eq. (1.1) may be formed by $x_+ = x_0 + Uy_\star$.

The following result is a direct corollary of the Implicit Q Theorem (Theorem 7.4.2 on page 367 of Ref. [10]) and Proposition 2.1.

Lemma 2.1 *The basis of the Krylov subspace U generated by Algorithm 2.1 are 'essentially equal', to that generated by the standard Arnoldi method with the same starting vector v_l. By 'essentially equal', we mean that only signs may be different.*

The following theorem can be easily obtained by using Lemma 2.1 and Proposition 2.4.

Theorem 2.1. *Assume the degree of the minimal polynomial of r_i is greater than $s \cdot m$. Then the BVGMRES(s, m) algorithm produces the same approximate solution x_{i+1} to Eq. (1.1) as that produced by the GMRES($s \cdot m$) method with the same initial guess x_i.*

3. The distributed BVGMRES algorithm

In this section, we provide some implementation details of the BVGMRES(s, m) algorithm as well as discuss how to implement this algorithm on an MPP efficiently. First, since it is not the purpose of this paper to study distributed (sparse) matrix–vector multiplication algorithms, we assume that the user provides an efficient distributed (sparse) matrix–vector multiplication algorithm at steps 4(b) and 4(e) of Algorithm 2.1. Also, we will not discuss the distribution of matrix A among PEs.

Secondly, how to distribute the arrays $U_{n\times l}$, $r_{n\times 1}$, $X_{n\times 1}$, $R_{l\times l}$ and $H_{l\times l}$ is crucial to the efficiency of the BVGMRES algorithm. In our distributed implementation of this algorithm, we distribute large arrays U, r and x in such a way that the first dimension of the arrays is equally divided into p portions, where p is the number of PEs used in this algorithm and each PE holds one portion of these arrays. Let $n_p = n/p$, then each PE holds arrays $U_{n_p\times l}$, $r_{n_p\times 1}$ and $x_{n_p\times 1}$. Fig. 1 shows how the large arrays are distributed among 4 PEs for $m = 4$. Storage of arrays H, R, b and y depends on the implementation of step 5 of Algorithm 2.1. Since the sizes of H, R, b and y are

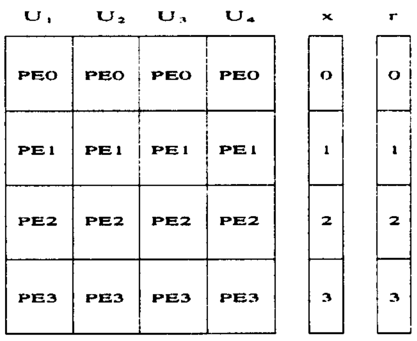

Fig. 1. Distribution of large arrays.

typically small, the time spent on the computations at steps 5(b), 5(c) and 5(d) is usually a very small portion of the whole computation for large problems. We feel that it is not worthwhile to implement these three steps on multi-PEs due to relatively expensive communication cost among PEs. We propose to implement the same computations of steps 5(a) to 5(d) simultaneously on all PEs and then each PE updates its own part of x_i concurrently (step 5(e)). Thus, there is no communication at all at step 5. In doing so, each PE must hold private copies of arrays H, R, b and y. This technique also simplifies the communication at other steps.

Now we are ready to discuss the communication among PEs. Note that due to the nature of the distribution of the arrays discussed above, most of the computationally intensive dense matrix–matrix and dense matrix–vector operations are done in parallel. Steps which involve communication are steps 2, 4(c)-i, 4(d), 4(f)-i and 4(h) without counting the communication in (sparse) matrix–vector operations. If the QR factorization at step 4(d) is implemented by the modified Gram–Schmidt method, then all communication is global summations in inner-products, matrix–transpose–vector and matrix–transpose–matrix operations. For example, at step 4(c)-i, each PE does its own part of the matrix–transpose-matrix operation to produce a private copy of R_{jk} without any communication and then all PEs participate in a global summation procedure GLOBALSUM(R_{jk}, size $= s^2 - s$) (The first column of R_{jk} is zero) to add all private copies of R_{jk} together and then each PE gets a copy of the result. Once the result of the global summation is returned and stored in array R_{jk}, step 4(c)-ii can be carried out entirely in parallel. The distributed algorithm now can be formulated as follows.

Algorithm 3.1. Given $x_0 \in R^n$ and integers $s > 1$ and $m > 0$, for $i = 1$ until convergence do

```
1. Compute $r_i = f - Ax_i$.
2. $\alpha_i = r_i^T r_i$.
3. $\alpha_i =$ GLOBALSUM($\alpha_i$, size = 1).
4. Compute $\alpha_i = \sqrt{\alpha_i}$.
5. Set $v_1 = r_i/\alpha_i$.
6. For $k = 1$ until $m$ do
   (a) Set $v_k^1 = v_k$.
   (b) Compute $V_k^- = (Av_k^1, A^2 v_k^1, \dots, A^{s-1} v_k^1)$.
   (c) For $j = 1$ until $k - 1$ do
       i. Compute the last $s - 1$ columns of $R_{jk}$, i.e. $R_{jk}^- = U_j^T V_k^-$.
       ii. $R_{jk}^- =$ GLOBALSUM($R_{jk}^-$, size $= s^2 - s$).
       iii. $V_k^- = V_k^- - U_j R_{jk}^-$.
       iv. Copy $R_{jk}^-$ to the first $s - 1$ columns of $H'_{jk}$
   (d) Set $V_k = (v_k^1, V_k^-)$
   (e) All PEs participate in the MGS procedure to compute $V_k = U_k R_{kk}$.
   (f) Copy the nonzero part of the last $s - 1$ columns of $R_{kk}$ to the first $s - 1$
       columns of $H_{kk}$.
   (g) Compute $w_k = Au_k^s$, where $u_k^s$ is the $s$th column of $U_k$.
   (h) For $j = 1$ until $k$ do
       i. $a_j = U_j^T w_k$.
```

```
        ii. $a_j = \text{GLOBALSUM}(a_j, \text{size} = s)$.
        iii. $w_k = w_k - U_j a_j$.
        iv. Copy $a_j$ to the last column of $H'_{jk}$.
    (i) Compute $\beta_k = w_k^{\mathrm{T}} w_k$.
    (j) $\beta_k = \text{GLOBALSUM}(\beta_k, \text{size} = 1)$.
    (k) Compute $\beta_k = \sqrt{\beta_k}$.
    (l) Set $v_{k+1} = w_k / \beta_k$.
7. Solve for $x_{i+1}$ (the same as step 5 in Algorithm 2.1. Please note the implementation details mentioned in Remark 2.1. This step is carried out locally on each PE.)
```

It can be seen from Algorithm 3.1 that most computations are done in parallel and most communications are in block global summation operations except for the (sparse) matrix–vector multiplication part. In contrast, the computationally intensive parts of the standard GMRES method are mostly inner-products and the global summations have to be done element by element. This is the reason why Algorithm 3.1 with large s has high efficiency on MPPS.

4. Efficiency and accuracy issues

Because of efficient block operations and block data communications, the BVGMRES algorithm gains efficiency with relatively large block sizes. However, if the block size s is too large, the QR factorization step (step 6(e) in Algorithm 3.1) could become dominant. Unlike in the cases of steps 6(c) and 6(h) of Algorithm 3.1 where a block modified Gram–Schmidt procedure is used, the modified Gram–Schmidt procedure we used at step 6(e) may not take too much advantage of block operations and communications. Therefore the parallel efficiency may be reduced if s is too large.

On the other hand, from the accuracy point of view, the BVGMRES algorithm is, in some sense, inbetween the standard GMRES and the GMRES with classical (unmodified) Gram–Schmidt implementation. When s is small, say $s = 2$, BVGMRES is close to the standard GMRES in the sense of accuracy. With large s, the accuracy of the orthogonalization in the block modified Gram–Schmidt procedures (steps 6(c) and 6(h)) may become worse. We provide some numerical examples in the following section to show the above facts.

5. Numerical results

In this section, we compare the performance of GMRES(m) and BVGMRES(s, m) with different s and m on the Cray T3D. Because the GMRES(m) using classical Gram–Schmidt, abbreviated as GMRESGS, can also take advantage of BLAS 2 operations and block data communications, we also provide some performance data of GMRESGS.

We chose five test problems. The SEISMIC test problem was taken from the finite-difference modeling of 3D migration in seismic data processing ([12], $n = 31752$, nonzero = 190258). The other four problems were from various matrix collections. They are RAEFSKY3 (fluid structure interaction turbulence problem, from Raefsky, in Horst Simon's matrix collection, $n = 21200$, nonzero = 1488768); AF23560 (NACA 0012 AIRFOIL M = 0.8, eigenvalue calculation) from Bai, Day, Demmel and Dongarra, $N = 23560$, nonzeros = 484256); SHERMAN3 (Harwell-Boeing sparse matrix collection, from Duff, Grimes and Lewis, $n = 5005$, nonzero = 20033) and MEMPLUS (memory circuit, from Steve Hamm, Motorola, $n = 17758$, nonzero = 126150). Since there was no right-hand side provided in AF23560, we scaled the matrix by the diagonal and generated a right-hand side by using the solution to be the vector of all ones.

Problem RAEFSKY3 is much denser than other test problems. As expected, the improvement of the parallel efficiency of BVGMRES is less significant than that on very sparse problems because, in this case, sparse matrix–vector multiplications are dominant in the whole computation. Problems AF23560, SHERMAN3 and MEMPLUS were chosen to demonstrate the stability of the algorithms.

For all test problems, the CSR (compressed sparse row) data structure was used to store the matrices. For problems SEISMIC and RAEFSKY3, there were no preconditioners applied except that diagonal scaling was used. For other problems, the incomplete LU (ILU) preconditioner was used in all algorithms. The stopping criterion used in the tests was $\|r\|_2/\|f\|_2 \leq \epsilon$, where r is the true residual and f is the right-hand side. For problems SEISMIC and RAEFSKY3, $\epsilon = 10^{-5}$. For others, $\epsilon = 10^{-7}$. The initial value x_0 was zero.

Due to the fact that the BVGMRES algorithm is theoretically equivalent to the standard GMRES as long as the size of the Krylov subspace for each restart is the same for all algorithms, our numerical experiments were conducted by fixing $s \cdot m$ which is the size of the Krylov subspace for each restart. In GMRES(m) and GMRESGS(m), we chose $m = 48$. In BVGMRES(s, m), we varied s and m, but kept $s \cdot m = 48$ fixed.

Since the standard PVM (parallel virtual machine) [2] is much slower than the Cray T3D specific communication library 'SHMEM', all communications in all algorithms were implemented by calling the 'SHMEM' subroutines. This is also the main reason for not using a publicly available standard GMRES code. Though GMRES(m) is a special case of the BVGMRES(s, m) algorithm when $s = 1$ and $m = s \cdot m$, we coded the standard GMRES separately and made our best efforts to obtain good performance.

We used an optimum communication pattern technique in the parallel sparse matrix vector multiply subroutine, which significantly improved the parallel efficiency in the sparse matrix–vector part.

Table 1 compares the performance of BVGMRES(s, m) with GMRES(m) and GMRESGS(m) on an 128 PE Cray T3D. Since the number of iterations (N_{it}) needed to converge does not have much meaning, we recorded the total number of (sparse) matrix vector multiplications (N_{mv}). Since N_{mv} was the same in all cases for this test problem, the total CPU time spent in (sparse) matrix–vector multiplications (Mv time, in seconds) was about the same in all cases. As a consequence, the total CPU time (total time, in seconds) showed that the performance of the BVGMRES algorithm improved with increasing block size s. Tables 2 and 3 compare GMRES(m) and GMRESGS(m) with BVGMRES(s, m) on a single PE of the Cray T3D. Table 4 shows the number of sparse

Table 1
Performance comparisons of the Seismic problem on an 128 PE Cray T3D

Method	s	m	N_{mv}	Mv time	Total time
BVGMRES	16	3	295	0.242	0.439
BVGMRES	12	4	295	0.242	0.456
BVGMRES	8	6	295	0.242	0.462
BVGMRES	6	8	295	0.242	0.506
BVGMRES	4	12	295	0.242	0.595
BVGMRES	2	24	295	0.242	0.964
GMRES	—	48	295	0.242	1.181
GMRESGS	—	48	295	0.242	0.523

Table 2
Performance comparisons of the Seismic problem on a single PE

Method	s	m	N_{mv}	Mv time	Total time
BVGMRES	16	3	295	20.87	38.82
BVGMRES	12	4	295	20.87	38.58
BVGMRES	8	6	295	20.87	40.15
BVGMRES	6	8	295	20.87	42.99
BVGMRES	4	12	295	20.87	45.28
BVGMRES	2	24	295	20.87	55.87
GMRES	—	48	295	20.87	49.87
GMRESGS	—	48	295	20.87	41.88

Table 3
Performance comparisons of the RAEFSKY3 problem on a single PE

Method	s	m	N_{mv}	Mv time	Total time
BVGMRES	16	3	197	49.02	60.252
BVGMRES	12	4	197	49.02	59.560
BVGMRES	8	6	197	49.02	58.371
BVGMRES	6	8	197	49.02	58.459
BVGMRES	4	12	197	49.02	60.472
BVGMRES	2	24	197	49.02	66.049
GMRES	—	48	197	49.02	63.438
GMRESGS	—	48	197	49.02	60.419

Table 4
Number of sparse matrix–vector's needed to converge on three test problems

Method	s	m	SHERMAN3	MEMPLUS	AF23560
BVGMRES	16	3	196	196	> 500
BVGMRES	12	4	147	147	147
BVGMRES	8	6	147	147	49
BVGMRES	6	8	147	147	49
BVGMRES	4	12	147	147	49
BVGMRES	2	24	147	147	49
GMRES	—	8	147	147	49
GMRESGS	—	48	147	147	> 500

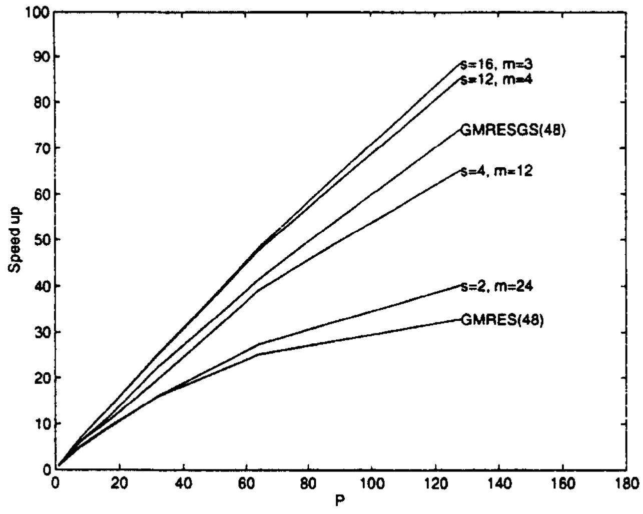

Fig. 2. Speed-up of GMRES, GMRESGS and BVGMRES(s, m) with various s on Seismic problem.

matrix–vector multiplications needed to converge for (ILU) preconditioned GMRES(48), GMRESGS(48) and BVGMRES with various block sizes on three test problems.

Shown in Figs. 2 and 3 are the comparisons of the speed-up of the BVGMRES(s, m) algorithm and that of GMRES(m) and GMRESGS(m) on 1 to 128 PEs of the Cray T3D. In the figures, p stands for the number of PEs used in solving the problem. The speed-up is the total speed-up which equals the total CPU time on 1 PE divided by the total CPU time on p PEs by using the same algorithm. Since the time spent on the

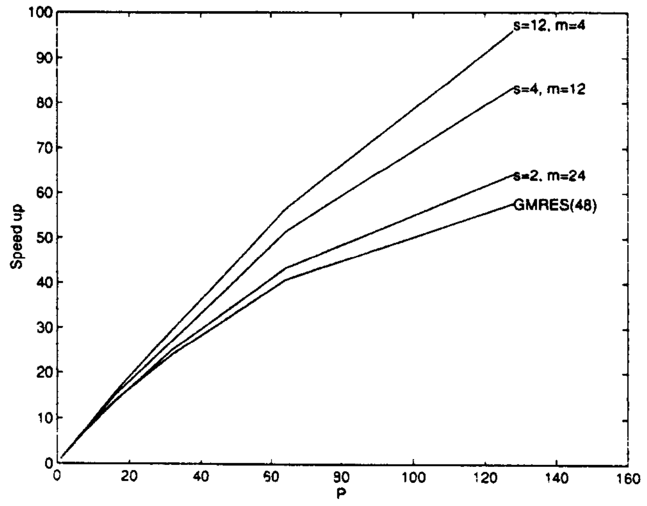

Fig. 3. Speed-up of GMRES and BVGMRES(s, m) with various s on RAEFSKY3 problem.

(sparse) matrix-vector part was about the same for all cases, the difference was made solely by BLAS operations and block data communications.

6. Concluding remarks

A block variant of the GMRES method, named as BVGMRES(s, m), has been implemented on a distributed memory computer. This algorithm generates a transformed Hessenberg matrix by using block matrix operations and block data communications. It has been proved that the BVGMRES($s \cdot m$) is theoretically equivalent to the GMRES($s \cdot m$) method. Our preliminary numerical results show that this algorithm can be more efficient than the standard GMRES method on a cache based single CPU computer with optimized BLAS kernels. Furthermore, the gain in efficiency is more significant on MPPs due to both efficient block operations and efficient block data communications. Our numerical results show that this algorithm may be stable up to some reasonable block size.

However, more numerical experiments and comparisons are needed to demonstrate the efficiency of the BVGMRES(s, m) algorithm on real world problems. Moreover, more study needs to be done on error analysis of the BVGMRES algorithm and on how to choose the parameters s and m. It is planned to extend this work to solving multiple right-hand side systems, i.e. to have a stable s-step block GMRES algorithm.

Acknowledgements

The author would like to thank Professor A.T. Chronopoulos for the discussion with him about their stable s-step ORTHOMIN method and his interest in implementing the block s-step ORTHOMIN algorithm. The author is grateful to Dr. P. Vu, Dr. K. Williamson, Dr. L. Wu, Mr. E. Chow and the referees for their helpful comments and suggestions. Finally, the author would like to thank his employer, Cray Research, for its support and providing its advanced computing facilities.

References

[1] Z. Bai, D. Hu, L. Reichtel, A Newton basis GMRES implementation, Dept. of Mathematics, University of Kentucky, Research Report 91-03, April 1991.

[2] Beguelin, Dongarra, Geist, Manchek, Sunderam, A User's Guide to PVM, ORNL/TM-11826, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, July 1991.

[3] D. Boley, G. Golub, The Lanczos-Arnoldi algorithm and controllability, Syst. Control Lett. 4 (1984) 317–324.

[4] A. Chronopoulos, s-Step iterative methods for (non)symmetric (in)definite linear systems, SIAM J. Num. Anal. 28 (6) (1991) 1776–1789.

[5] A. Chronopoulos, S. Kim, s-Step ORTHOMIN and GMRES implemented on parallel computers, TR 90-15, Computer Science Department, University of Minnesota, Feb. 1990.

[6] A. Chronopoulos, S. Kim, Towards efficient parallel implementation of s-step iterative methods, Supercomputer 47 IX (1) (1992) 4–17.

[7] A. Chronopoulos, M. Pernice, Vector preconditioned s-step methods on the IBM 3090/600S/6VF, Proceedings of the Fifth SIAM Conf. on Parallel Proc., Houston, March 1991, pp. 130–137.
[8] J. Dongarra, J. Ducroz, I. Duff, S. Hammarling, A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Soft. (1989).
[9] J. Dongarra, J. Ducroz, S. Hammarling, R. Hanson, An extended set of Fortran basic linear algebra subprograms, ACM Trans. Math. Soft. 14 (1) (1988) 1–32.
[10] G. Golub, C. Van Loan, Matrix Computation, 2nd ed., The Johns Hopkins University Press, Baltimore and London, 1990.
[11] W. Joubert, G. Carey, Parallelizable restarted iterative methods for nonsymmetric linear systems. Part I: Theory. Part II: Parallel implementation, Int. J. Comput. Math. 44 (1992) 243–290.
[12] J. Kao, G. Li, C. Yang, Preconditioned iterative 3D finite-difference migration or modeling on MPP systems, Proceedings of SHPCC'94, Scalable High-Performance Computing Conference, Knoxville, Tennessee, May 1994, pp. 601–606.
[13] C. Lawson, R. Hanson, D. Kincaid, F. Krogh, Basic linear algebra subprograms for Fortran usage, ACM Trans. Math. Soft. 5 (1979) 308–325.
[14] N. Nachtigal, S. Reddy, L. Trefethen, How fast are matrix iterations?, SIAM Matrix Anal. Appl. 13 (1992) 778–795.
[15] D. O'Leary, The block conjugate gradient algorithm and related methods, Lin. Alg. Appl. 29 (1980) 293–322.
[16] D. O'Leary, P. Whitman, Parallel QR factorization by Householder and modified Gram–Schmidt algorithm, Parallel Comput. 16 (1990) 99–112.
[17] M. Pernice, Implementations of GMRES and their performance on the IBM 30901600S, University of Utah, Utah Supercomputing Institute, USI Report No. 13, May 1991.
[18] Y. Saad, Numerical Methods for Large Eigenvalue Problems, Halstead Press, New York, 1992.
[19] Y. Saad, M. Schultz, A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statis. Comput. 7 (1986) 856–869.
[20] M. Sadkane, Block Arnoldi and Davidson methods for unsymmetric large eigenvalue problems, Numer. Math. 64 (1993) 195–211.
[21] V. Simoncini, E. Gallopoulos, Convergence properties of block GMRES for solving systems with multiple right-hand sides, CSRD Report No. 1316 (1993), Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign.
[22] C. Sturler, A parallel restructured version of GMRES(m)/GMRES II, Copper Mountain Conference on Iterative Methods, April 1992.
[23] C. Swanson, A. Chronopoulos, Parallel iterative s-step methods for unsymmetric linear systems, UMSI 94/105, University of Minnesota Supercomputer Institute Research Report, June 1994.