Electronic Transactions on Numerical Analysis.
Volume 3, pp. 160-176, December 1995.

ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A PARALLEL GMRES VERSION FOR GENERAL SPARSE MATRICES *

JOCELYNE ERHEL †

Abstract. This paper describes the implementation of a parallel variant of GMRES on Paragon. This variant builds an orthonormal Krylov basis in two steps: it first computes a Newton basis then orthogonalises it. The first step requires matrix-vector products with a general sparse unsymmetric matrix and the second step is a QR factorisation of a rectangular matrix with few long vectors. The algorithm has been implemented for a distributed memory parallel computer. The distributed sparse matrix-vector product avoids global communications thanks to the initial setup of the communication pattern. The QR factorisation is distributed by using Givens rotations which require only local communications. Results on an Intel Paragon show the efficiency and the scalability of our algorithm.

Key words. GMRES, parallelism, sparse matrix, Newton basis.

AMS subject classifications. 65F10, 65F25, 65F50.

1. Introduction. Many scientific applications make use of sparse linear algebra. Because they are quite time consuming, they require an efficient parallel implementation on powerful supercomputers. An important class of iterative methods are projection methods using a projection onto a Krylov subspace. The aim of this paper is to study the parallelization of such methods for general sparse matrices. We chose to design and implement a parallel version of the GMRES algorithm [21]. The main part of GMRES is the Arnoldi process which generates an orthonormal basis of the Krylov subspace.

The target architecture here is a parallel machine with a distributed memory. We chose a Single Program Multiple Data (SPMD) programming style where each processor executes the same code on different data. Data are distributed according to a static mapping and communications are expressed by means of messages.

The parallelization of GMRES has been studied by several authors. Some of them consider the basic GMRES iteration with the Arnoldi process, for example in [11, 22, 3, 7]. Variants of GMRES such as $\alpha - GMRES$ [26] have also been designed. Hybrid algorithms combining a GMRES basis with a Chebychev basis are also investigated in [15]. Another approach is to define a new basis of the Krylov subspace as, for example, in [16, 8, 9, 10, 5, 2]. Our implementation is based on the ideas develop in [2]. It first builds a Newton basis which is then orthogonalized. This approach requires the parallelization of two steps: the basis formation which relies on matrix-vector products and the basis QR factorization.

As far as the matrix-vector product is concerned, one of two classes of matrices are usually considered: regular sparse matrices arising for example from the discretisation of PDE on a regular grid and general sparse matrices with an arbitrary pattern of nonzeros. This paper deals with the second class. The parallelization of the sparse matrix-vector product in this case has been studied for example in [3, 4, 19]. Our present work uses the library P-SPARSLIB described in [20].

In our implementation, the QR factorization is done either by a Gram-Schmidt process as in [10] or by a Householder/Givens factorization as in [24].

* Received Oct 2, 1995. Accepted for publication November 22, 1995. Communicated by L. Reichel.
† INRIA-IRISA , Campus de Beaulieu, 35042 Rennes, France. erhel@irisa.fr

The paper is organized as follows. The second part describes the basic sequential GMRES version. The third part is devoted to the parallelization of the Arnoldi process with a Newton basis. This involves two main procedures, the sparse matrix-vector product and the QR factorization. Their parallelization is studied in the next two parts. Numerical experiments in the sixth part first show the impact on convergence of the Newton basis. In particular, we exhibit a matrix where the ill-conditioning of the Newton basis degrades drastically the convergence. We then give performance results on a Intel Paragon parallel computer. Concluding remarks follow at the end.

2. Sequential version. We first recall the basic GMRES algorithm [21].

Consider the linear system $Ax = b$ with $x, b \in \mathcal{R}^n$ and with a nonsingular nonsymmetric matrix $A \in \mathcal{R}^{n\times n}$. The Krylov subspace $K(m; A; r_0)$ is defined by $K(m; A; r_0) = span(r_0, Ar_0, \dots, A^{m-1}r_0)$. The GMRES algorithm uses the Arnoldi process to construct an orthonormal basis $V_m = [v_1, \dots, v_m]$ for $K(m; A; r_0)$. The full GMRES method allows the Krylov subspace dimension to increase up to n and always terminates in at most n iterations. The restarted GMRES method restricts the Krylov subspace dimension to a fixed value m and restarts the Arnoldi process using the last iterate x_m as an initial guess. It may stall. Below is the algorithm for the restarted GMRES.

Algorithm 0: GMRES(m)

```
ε is the tolerance for the residual norm ;
convergence = false ;
choose x_0 ;
until convergence do
    r_0 = b − Ax_0 ;
    β = ‖r_0‖ ;
    * Arnoldi process: construct a basis V_m of the Krylov subspace K
    v_1 = r_0/β ;
    for j = 1, ···, m do
    |   p = Av_j ;
    |   for i = 1, ··· j do
    |   |   h_ij = v_i^T p ;
    |   |   p = p − h_ij v_i ;
    |   endfor ;
    |   h_{j+1,j} = ‖p‖_2 ;
    |   v_{j+1} = p/h_{j+1,j} ;
    endfor ;
    * Minimize ‖b − Ax‖ for x ∈ x_0 + K
    * Solve a least-square problem of size m
    compute y_m solution of min_{y∈R^m} ‖βe_1 − \overline{H_m} y‖ ;
    x_m = x_0 + V_m y_m ;
    if ‖b − Ax_m‖ < ε convergence = true ;
    x_0 = x_m ;
enddo
```

The matrix $\overline{H_m} = (h_{ij})$ is a upper Hessenberg matrix of order $(m + 1) \times m$, and we get the fundamental relation

$$AV_m = V_{m+1}\overline{H_m}.$$

The GMRES algorithm computes $x = x_0 + V_m y_m$ where y_m solves the least-squares

ETNA
Kent State University
etna@mcs.kent.edu

problem $\min_{y \in \mathcal{R}^m} \|\beta e_1 - \overline{H_m} y\|$. Usually a QR factorization of $\overline{H_m}$ using Givens rotations is used to solve this least-squares problem.

The linear system can be preconditioned either at left or at right solving respectively $M^{-1}Ax = M^{-1}b$ or $AM^{-1}(Mx) = b$ where M is the preconditioning matrix.

The most time-consuming part in GMRES(m) is the Arnoldi process. Indeed, the solution of the small least-squares problem of size m represents a neglectible time. The Arnoldi process contains three operators: sparse matrix-vector product, orthogonalization and preconditioning. We now study the parallelization of the Arnoldi step.

3. Parallel Arnoldi process. The Arnoldi process builds an orthonormal basis of the Krylov subspace $K(m; A; r_0)$. It constructs the basis and orthogonalizes it at the same time; as soon as a new vector is added, it is orthogonalized relative to the previous vectors and normalized. The modified Gram-Schmidt algorithm is usually preferred for numerical stability. This leads to heavy data dependencies as shown in Figure 3.1.

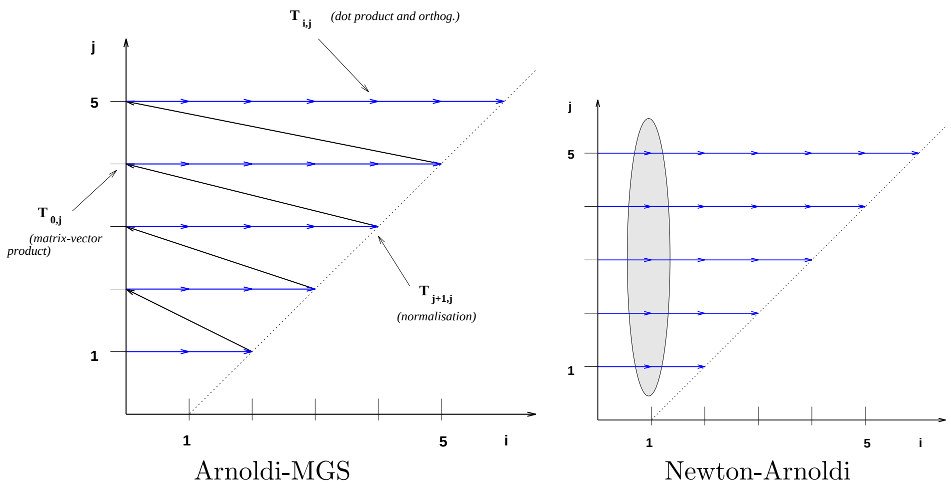

FIG. 3.1. *Dependency graph of Arnoldi method*

Some authors such as [22] have considered a classical Gram-Schmidt version to remove some dependencies, but in that case orthogonalization must then be done twice to ensure numerical stability.

Another solution is to replace the Gram-Schmidt process by Householder transformations as in [25]. This provides a more accurate orthogonal basis at the price of more operations. Parallelism can be achieved using a Householder factorization by blocks [12].

In [16], the basis is built in blocks of size s, each new block being orthogonalized with respect to all previous s-vectors groups. But the convergence becomes slow with this approach.

The solution which we have chosen, advocated for example in [14, 5, 1, 2], is to divide the process into two steps: first build a basis then orthogonalize it. Indeed, it removes the data dependencies on the left so that many tasks are now independent (Figure 3.1). The first step is merely a succession of matrix-vector products and the second step an orthonormalisation process.

ETNA
Kent State University
etna@mcs.kent.edu

Nevertheless, the basis must be well conditioned to ensure numerical stability (more precisely, to avoid loss of orthogonality and to ensure a well conditioned least-squares reduced problem). Following [2], we choose here a Newton basis defined by

$$\widehat{V}_{m+1} = [\sigma_0 v, \sigma_1 (A - \lambda_1 I)v, ..., \sigma_m \prod_{l=1}^{m} (A - \lambda_l I)v]. \tag{3.1}$$

The values λ_l are here estimations of eigenvalues of A ordered with the Leja ordering. We compute m Ritz values in a first cycle of GMRES(m) and order them in a Leja ordering, grouping together two complex conjugate eigenvalues.

We then have a basis with normalized vectors of the Krylov subspace $\widetilde{V}_{m+1}$ such that $\widehat{V}_{m+1} = \widetilde{V}_{m+1}\widetilde{D}_{m+1}$. We compute an orthonormal basis with a QR factorization

$$\widetilde{V}_{m+1} = \tilde{Q}_{m+1}\tilde{R}_{m+1}, \tag{3.2}$$

so that we can write

$$A\widehat{V}_m = \widetilde{Q}_{m+1}\widehat{R}, \tag{3.3}$$

where $\widehat{R} \in \mathcal{R}^{(m+1)\times m}$ is an upper Hessenberg matrix which is easy to compute. We end up with a minimization problem similar to that of GMRES with the function J defined by

$$J(y) = \|\beta e_1 - \widehat{R}y\|. \tag{3.4}$$

To reiterate, we have a new version, called NewtonGMRES(m), which offers more parallelism and which is summarized below:

Algorithm 0: `NewtonGMRES(m)`

ϵ is the tolerance for the residual norm ;
* Initialisation
Apply algorithm 2 to find an approximation x_0
and a Hessenberg matrix $m \times m$, H_m ;
Compute eigenvalues $\lambda(H_m) = \{\lambda_j\}_{j=1}^{m}$ and order λ_j in a Leja ordering ;
convergence = false ;
* Iterative scheme
until convergence **do**
  Compute $\widetilde{V}_{m+1}$;
  Compute the factorization $\widetilde{V}_{m+1} = \widetilde{Q}_{m+1}\widetilde{R}_{m+1}$;
  Compute $\widehat{R}$ such that $A\widehat{V}_m = \widetilde{Q}_{m+1}\widehat{R}$;
  Compute the QR factorization of $\widehat{R}$, and find y_m solution of (3.4) ;
  Compute $x_m = x_0 + \widetilde{V}_m\widetilde{D}_m y_m$;
  if $\|b - Ax_m\| < \epsilon$ convergence = true ;
  $x_0 = x_m$;
enddo

NewtonGMRES(m) contains two main steps: the computation of the basis $\widetilde{V}_{m+1}$ which involves matrix-vector products and its QR factorization. We now analyze the parallelization of these two procedures.

4. Parallel sparse matrix-vector product. The computation of $\widetilde{V}_{m+1}$ involves operations of the type $y = \sigma(A - \lambda I)x$. Without loss of generality, we study here the matrix-vector product $y = Ax$.

For sparse matrices, a natural idea is to distribute the matrix by rows and to distribute the vectors accordingly. Since the vector x is updated in parallel in a previous step, it induces communication to compute y. A straightforward implementation would broadcast the vector x to each processor so that each processor gets a complete copy. However, communication can be drastically reduced thanks to the sparse structure of the matrix A. To compute the component $y(i)$ only a few components $x(j)$ of x are required, namely those corresponding to nonzero entries a_{ij} in row i. Moreover some of these components are local to the processor.

These two remarks are the basis for the distributed algorithm designed by Saad [19]. Let us denote by x_{loc} and y_{loc} the vector blocks distributed to the processor. Local rows are the rows allocated to the processor and local columns refer to local components of vectors. Other columns are called external and we denote by x_{ext} the set of external components required. We also denote by A_{loc} the matrix block with local row and column indices:

$$A_{loc} = \{a_{ij} : y_i \text{is local and } x_j \text{ is local}\}, \tag{4.1}$$

and by A_{ext} the matrix block with local rows but external columns:

$$A_{ext} = \{a_{ij} : y_i \text{ is local and } x_j \text{ is external}\}. \tag{4.2}$$

This block structure is depicted in figure 4.1.

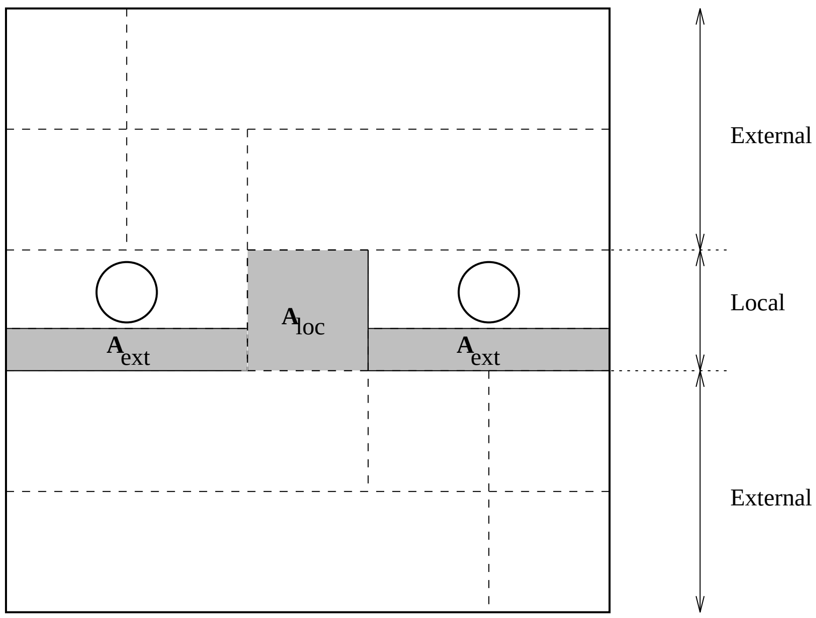

FIG. 4.1. *Distributed matrix*

The matrix-vector product is then written

$$y_{loc} = A_{loc} * x_{loc} + A_{ext} * x_{ext}. \tag{4.3}$$

ETNA
Kent State University
etna@mcs.kent.edu

The processor must then receive x_{ext} from other processors before processing the second term. Converserly, it must send its components to other processors requiring them. The communication pattern is set up in an initialization phase on each processor and stored in data structures. This step requires, on each processor, the knowledge of the distribution of the rows and the local matrix. Each processor has a list $jsend(p)$ of components which will be sent to processor p and a list $jrecv(q)$ of components which will be received from processor q. More details can be found in [20, 17].

Since the matrix is processed by rows, a storage by rows is mandatory. We use a classical compressed sparse row (CSR) format. Each local matrix is stored using a local numbering with a correspondance between local and global numbers.

Finally we get the following distributed algorithm, executed on each processor:
ALGORITHM 0: Distributed sparse matrix-vector product

Send $x_{loc}[jsend(p)]$ to processors p ;
Compute $y_{loc} = A_{loc}x_{loc}$;
Receive $x[jrecv(q)]$ from processors q and build x_{ext} ;
Compute $y_{loc} = y_{loc} + A_{ext}x_{ext}$;

This algorithm allows communication to overlap with computations and is scalable because the volume of communication increases much slower than the volume of computation with the size of the problem. The amount of communication depends on the size of each external block. Renumbering techniques help to reduce this size. They are based on graph partitioning heuristics applied to the adjacency graph of the matrix. The objective is to reduce the size of the interfaces and to balance the size of each subgraph.

5. Parallel QR factorization. The other procedure of the Newton-Arnoldi method is the factorization $\widetilde{V}_{m+1} = \widetilde{Q}_{m+1}\widetilde{R}_{m+1}$.

We have to design a parallel QR factorization of a matrix V which is rectangular of size $(m+1)*n$ with $m << n$. A natural data distribution arises from the data distribution chosen for the matrix-vector product. The column vectors in V are subdivided into blocks defined by the matrix partition, in order to minimize communications.

The first step is to apply a modified Gram-Schmidt (MGS) method as in [9]. Independent operations allow to overlap communications with computations but global communications are still required for the scalar products.

The Householder factorization requires only local communications but more operations. A parallel version using a ring topology [18] and a cyclic partition of the matrix rows to balance the workload has been implemented for GMRES in [1]. A more promising divide-and-conquer approach,following the algorithm designed for shared-memory processors in [6] , is taken in [5]. The idea is to apply first a local QR factorization on the block owned by the processor and then to group the blocks two by two to apply a new QR factorization on the larger blocks. This is done recursively until completion and can be easily modified if the number of processors is not a power of two. The difficulty of this approach is that the number of active processors decrease by half at each stage.However it is efficient for a small number of processors because the block size is large compared to m so that the first step is the most time-consuming one.

A different method which combines Householder transformations with Givens rotations is developed in [23, 24] and is called RODDEC. It still requires only local communications and keeps all the processors active. The first step is similar to the algorithm described previoulsy. It then annihilates all the coefficients in the blocks except the first one by Givens rotations using a ring topology.

We will now describe in more details the two methods we have implemented: MGS and RODDEC. A complete discussion can be found in [17].

5.1. MGS method. There are several ways to parallelize the MGS algorithm. Here we advocate the distribution of the vectors inherited from the matrix distribution. It induces distributed dot-products which require global communication. However, since several dot-products are independent, it is possible to overlap communications with computations [9]. Though this parallel algorithm is not scalable because of global communications, it should be efficient for a reasonably large number of processors. Compared to Arnoldi, we interchange the loops on i and j, so that we orthogonalize remaining vectors as soon as a new vector is computed. Each processor will execute the following algorithm:

ALGORITHM 0: Distributed MGS factorization

for i=1 **to** m **do**
  divide vectors $v_{i+1}, \ldots, v_{m+1}$ into two blocks ;
  Compute local scalar products (LSP) of Block 1 ;
  Send LSP of Block 1 ;
  Compute LSP of Block 2 ;
  Receive and Sum external scalar products of Block 1 ;
  Update v_{i+1}, compute LSP for $\|v_{i+1}\|$, put it in Block 2 ;
  Send LSP of Block 2 ;
  Update vectors in Block 1 ;
  Receive and Sum external scalar products of Block 2 ;
  Update vectors of Block 2 ;
  Normalize v_{i+1} ;
endfor

5.2. RODDEC method. The main advantage of Householder factorization is to avoid dot-products and to require only local communications. Data are still distributed in the same way as for the matrix-vector product. Each processor first executes a QR factorization of its own block with Householder transformations. So, with three processors, we get the matrix structure

$$\left[\begin{array}{cccc} . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \\ 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 \\ \hline . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \\ 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 \\ \hline . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \\ 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 \end{array}\right].$$

Then we implement the algorithm RODDEC (ROw partitioning Diagonal oriented

DECComposition) designed in [24]. The remaining factorization uses Givens rotations to annihilate the entries in the blocks below the first one. Parallelism is achieved by mapping a logical ring structure and by sending data on this ring.

Each processor owns the partition V_{loc} and will execute the following algorithm to compute the factorization $V = QR$. The variables `nbproc`, `myproc`, `myleft` and `myright` hold respectively the number of processors, the current processor number, the processor on the left and on the right. The final value of the matrix R will be hold by the last processor in the ring.

ALGORITHM 0: `RODDEC`

```
* Local QR factorization
[Q_loc, R_loc] = QR(V_loc) ;
* Annihilate entries of R_loc diagonal by diagonal
for d = 1 to m + 1 do
    if myproc = 1 then
        Send row R_loc(d, d : m + 1) to myright
    else
        Recv row (d : m + 1) from myleft
        Compute the rotation to annihilate R_loc(1, d)
        if myproc = nbproc then
            Send updated row (d : m + 1) to myright
        endif
        Compute the rotations to annihilate the d^th diagonal of R_loc
    endif
endfor
```

6. Numerical experiments. We implemented three algorithms on a Paragon computer with 56 nodes plus 8 nodes for I/O organized in a grid topology. Each node has two i860 processors, one for communication and the other for computation, with about 7 MBytes available for the application. For the RODDEC algorithm, we have mapped a logical ring structure onto the physical grid structure.

The first algorithm is the basic GMRES(m) version (algorithm 2) which is distributed using the sparse matrix-vector product data structures (algorithm 4). Here we use a basic partition by blocks of rows. In the future, we plan to apply partitioning techniques to reduce communications. The second algorithm is NewtonGMRES(m)-MGS and uses the Newton basis with a QR factorization (algorithm 3) based on the distributed MGS version (algorithm 5.1). The third one is called NewtonGMRES(m)-RODDEC and uses also the Newton basis but with the QR factorization done with the distributed RODDEC scheme (algorithm 5.2).

We tested these three versions on sparse matrices coming either from the Boeing-Harwell collection [13] or from the discretization of the Laplacian on a unit square with a regular grid and a five-point difference scheme. The matrix WATT is a nonsymmetric matrix of order $n = 1856$ with $nz = 11360$ nonzero entries and is used for convergence analysis. The matrices BCSSTK10 and BCSSTK24 are symmetric but are treated here as nonsymmetric. They are respectively of order $n = 1086$ and $n = 3562$ with $nz = 22070$ and $nz = 159910$ nonzero entries. The grid matrices are of order 10000 and 90000.

6.1. Convergence analysis. We first compare the convergence of the two methods GMRES(m) and NewtonGMRES(m) on the matrices WATT and BCSSTK24.

ETNA
Kent State University
etna@mcs.kent.edu

Results are shown in Figures 6.1 and 6.2. Both methods are similar for the matrix BCSSTK24 and for most matrices.

However, because the Newton basis is ill-conditioned for the matrix WATT, NewtonGMRES(m) does not converge in this case. We therefore implemented a more robust version, called NewtonDbl(m), as advocated in [1]. We execute two cycles of GMRES(m) and sort m eigenvalues from the $2m$ Ritz values computed. The results in Figures 6.1 and 6.2 show a same convergence curve for BCSSTK24 but a considerable improvement for WATT.

Here the Krylov size m is taken to 30 but other values gave similar results.

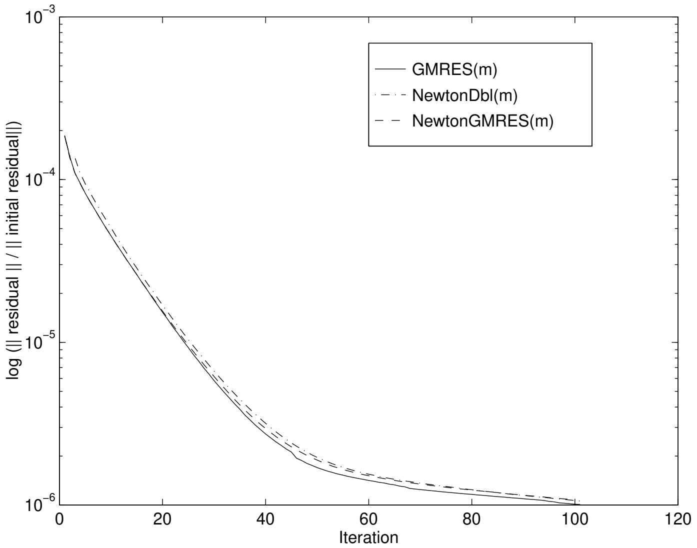

FIG. 6.1. *Convergence history (matrix BCSSTK24)*

6.2. Performance results. For performance measures, we will not consider the NewtonDbl(m) version which adds only a small overhead.

Except otherwise noted, the Krylov size is taken to $m = 30$ for the matrices BCSSTK10 and BCSSTK24 and to $m = 10$ for grid matrices. Since the convergence does not depend too much on the algorithm, we fixed the number of iterations to 50.

Figures 7.1 and 7.2 give the CPU time for a varying number of processors for matrices BCSSTK10 and BCSSTK24 for the three schemes. Clearly, NewtonGMRES(m) is more parallel than the basic version. The RODDEC algorithm is more scalable than MGS because it avoids dot-products and hence global communications. The differences are not so evident for large matrices, as shown in figures 7.3 and 7.4 for grid matrices of size 10000 and 90000. However, the curves would behave similarly if we should increase the number of processors.

We also studied the effect of the Krylov size m on the performances. Results are given in Figures 7.5 and 7.6 for the matrix BCSSTK24 with respectively NewtonGMRES(m) - MGS and NewtonGMRES(m) - RODDEC implementations. It appears that MGS is more sensitive to m than RODDEC which shows almost no variation. The speed-up increases with m for MGS because the blocks become larger and allow more overlap between communication and computation.

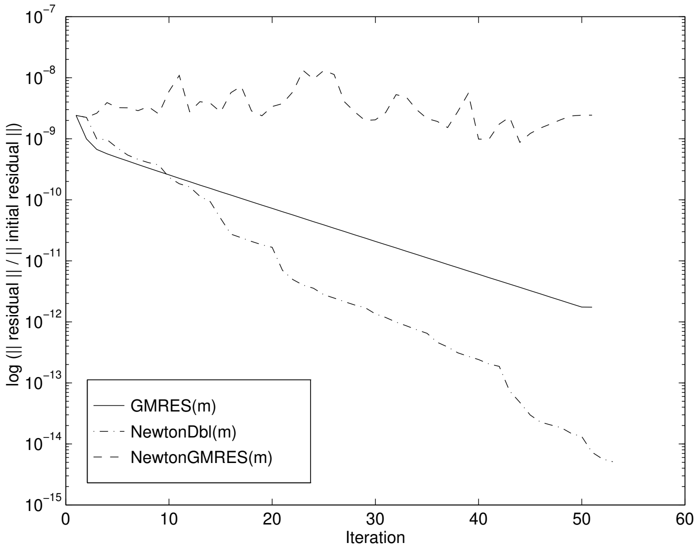

FIG. 6.2. *Convergence history (matrix WATT)*

We analyse the scalability of NewtonGMRES(m) using grid matrices with a varying size. We allocate a fixed number of rows per processor, taken to 100 and 1000, increasing the problem size with the number of processors. Results are given in Figures 7.9 and 7.10 and show the better scalability of RODDEC, though the differences become smaller with larger problems.

Finally, we analyze the various parts of the algorithm. Figures 7.7 and 7.8 show the percentage of each step in Newton-GMRES for a varying problem size, using a grid matrix with 1000 rows per processor. Newton-GMRES is decomposed into the basis formation involving sparse matrix-vector products, the basis factorization done with either MGS or RODDEC, the initialisation including the computation of the Leja points and the least-squares resolution. As expected, the basis factorization takes more time with MGS than with RODDEC. For RODDEC, the least-squares resolution is more expensive, because it is done on only one processor which has to broadcast the results and this global communication slightly slows down the execution time. The performance of the basis formation can still be improved by using a better graph partitioning and by overlapping communications with computations between consecutive matrix-vector products.

7. Concluding remarks. GMRES is based on the projection onto a Krylov subspace. We obtained a very efficient algorithm to compute a Krylov basis. Results on our parallel version of GMRES show good performances, even on small matrices. We plan to tested it with large unsymmetric unstructured matrices to confirm our results. We want also to include a partitioning technique to reduce the communication overhead.

ETNA
Kent State University
etna@mcs.kent.edu

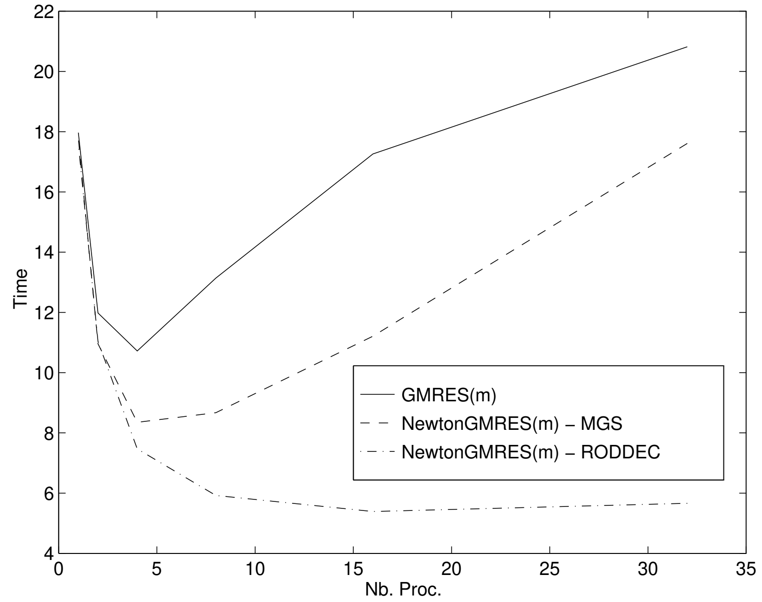

FIG. 7.1. *CPU time (matrix BCSSTK10)*

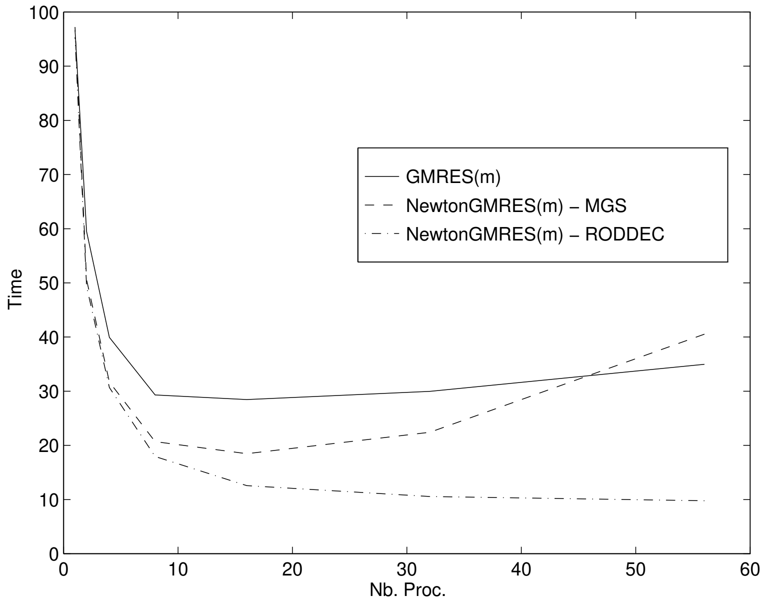

FIG. 7.2. *CPU time (matrix BCSSTK24)*

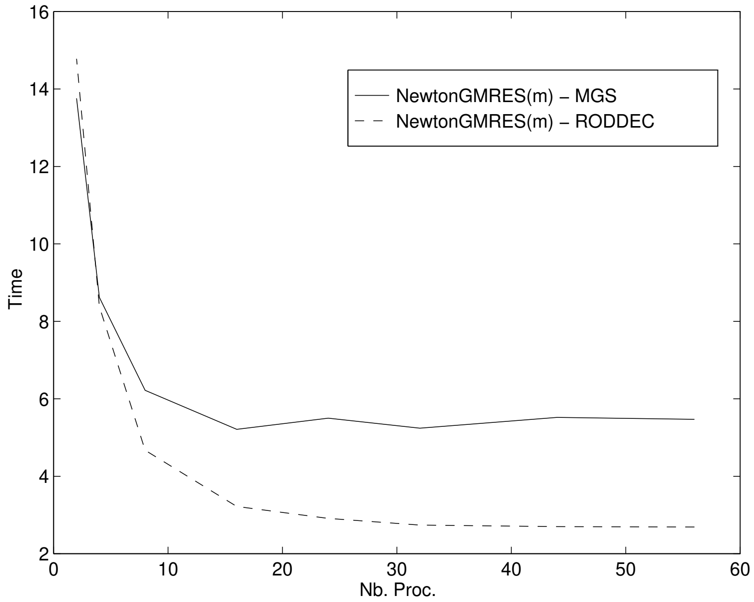

FIG. 7.3. *CPU time (grid matrix,* $n = 10000$*)*

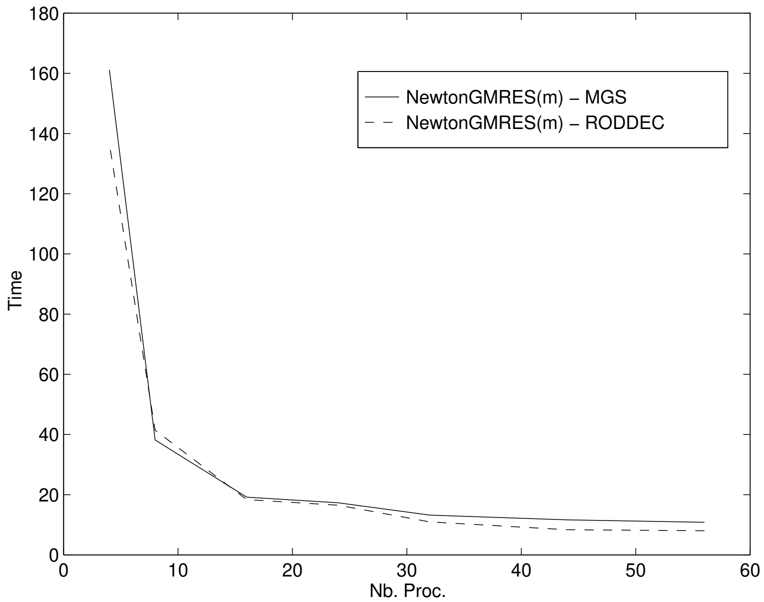

FIG. 7.4. *CPU time (grid matrix,* $n = 90000$*)*

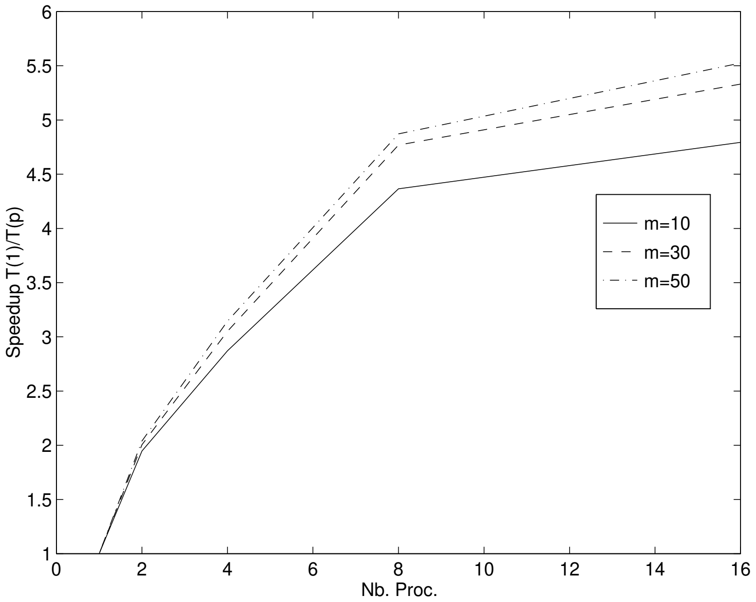

FIG. 7.5. *Speedup NewtonGMRES(m)-MGS (matrix BCSSTK24)*

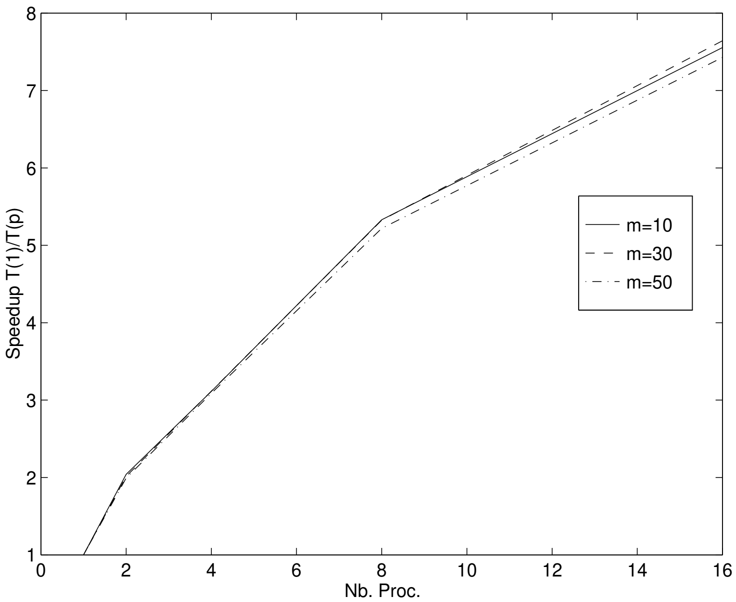

FIG. 7.6. *Speedup NewtonGMRES(m)-RODDEC (matrix BCSSTK24)*

ETNA
Kent State University
etna@mcs.kent.edu

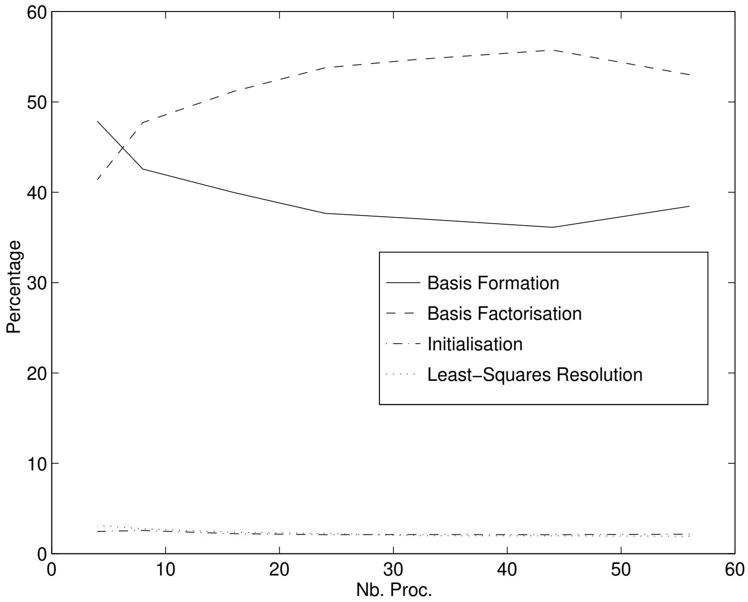

FIG. 7.7. *Breakdown of CPU time - Newton-MGS (1000 rows/processor)*

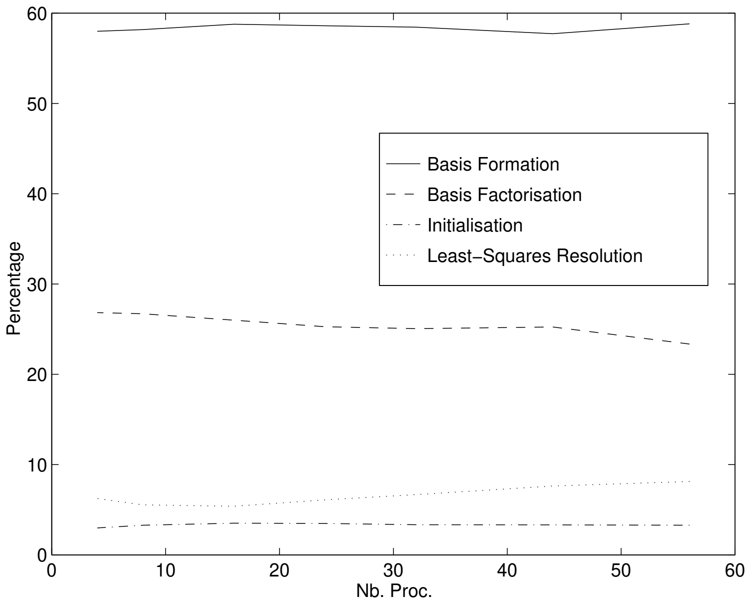

FIG. 7.8. *Breakdown of CPU time - Newton-RODDEC (1000 rows/processor)*

ETNA
Kent State University
etna@mcs.kent.edu

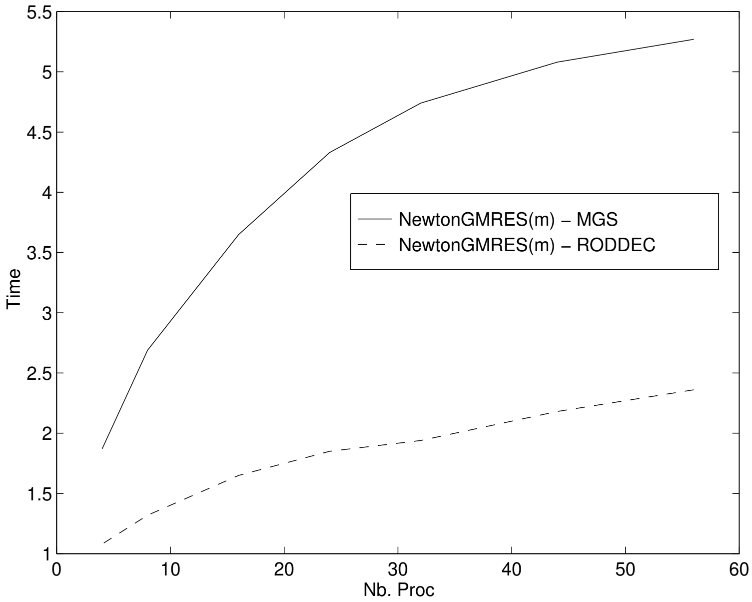

FIG. 7.9. *CPU time (grid matrix, 100 rows/processor)*

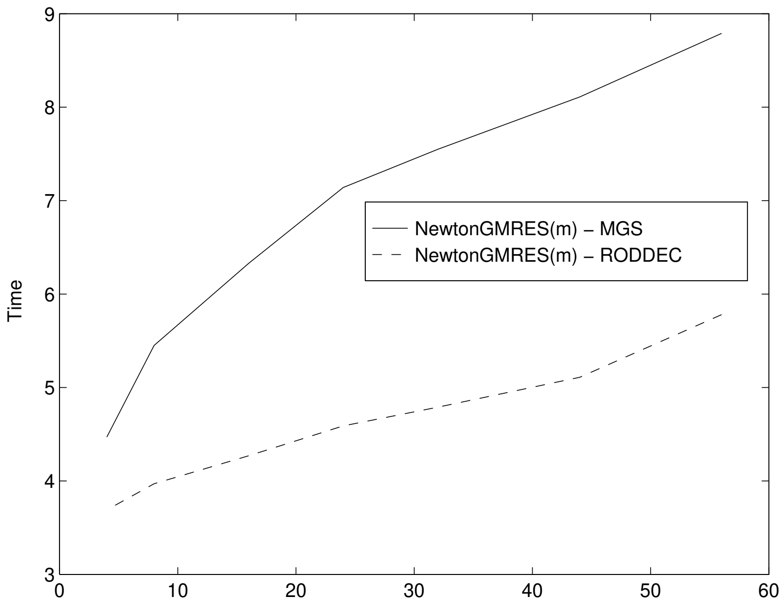

FIG. 7.10. *CPU time (grid matrix, 1000 rows/processor)*

REFERENCES

[1] Z. Bai, D. Hu, and L. Reichel, *Implementation of GMRES method using QR factorisation*, in Proc. Fifth SIAM Conference on Parallel Processing for Scientific Computing, eds. J. Dongarra, K. Kennedy, P. Messina, D. C. Sorensen and R. G. Voigt, SIAM, Philadelphia, 1992, pp. 84–91.

[2] ———, *A Newton basis GMRES implementation*, IMA J. Numer. Anal., 14 (1994), pp. 563–581.

[3] R. H. Bisseling, *Parallel iterative solution of sparse linear systems on a transputer network*, in Parallel Computation, eds. A. E. Fincham and B. Ford, Oxford University Press, Oxford, 1993, pp. 253–271.

[4] F. Bodin, J. Erhel, and T. Priol, *Parallel sparse matrix vector multiplication using a shared virtual memory environment*, in Proc. Sixth SIAM Conference on Parallel Processing for Scientific Computing, eds. R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, D. A. Reed, SIAM, Philadelphia, 1993, pp. 421–428.

[5] D. Calvetti, J. Petersen, and L. Reichel, *A parallel implementation of the GMRES method*, in Numerical Linear Algebra, eds. L. Reichel, A. Ruttan and R. S. Varga, de Gruyter, Berlin, 1993, pp. 31–45.

[6] E. Chu and A. George, *QR factorization of a dense matrix on a shared memory multiprocessor*, Parallel Computing, 11 (1989), pp. 55–71.

[7] R. D. da Cunha and T. Hopkins, *A parallel implementation of the restarted gmres iterative algorithm for nonsymmetric systems of linear equations*, Adv. Comp. Math., 2 (1994), pp. 261–277.

[8] E. de Sturler, *A parallel variant of GMRES(m)*, in Proc. of the 13th IMACS World Congress on Computation and Applied Mathematics, ed. J. J. H. Miller and R. Vichnevetsky, IMACS, Criterion Press Dublin, 1991, pp. 682–683.

[9] E. de Sturler and H. van der Vorst, *Communication cost reduction for Krylov methods on parallel computers*, in Numerical Methods for Large Eigenvalue Problems, ed. Y. Saad, Manchester University Press, 1992, pp. 190–196.

[10] ———, *Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers*, Tech. Rep. 832, Utrecht University, Department of Mathematics, Oct. 1993.

[11] G. R. di Brozolo and Y. Robert, *Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor*, Parallel Comput., 11 (1989), pp. 223–239.

[12] 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, 1991.

[13] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[14] A. C. Hindmarsh and H. F. Walker, *Note on a Householder implementation of the GMRES method*, Tech. Rep. UCID-20899, Laurence Livermore Laboratory, 1986.

[15] W. D. Joubert and G. F. Carey, *Parallelizable restarted iterative methods for nonsymmetric linear systems i- theory, ii- parallel implementation*, Int. J. Comput. Math., 44 (1992), pp. 243–290.

[16] S. K. Kim and A. Chronopoulos, *An efficient Arnoldi method implemented on parallel computers*, in 1991 International Conference on Parallel Processing, vol. III, 1991, pp. 167–196.

[17] J.-M. Legrand, *Parallélisation de GMRES(m)*, DEA Report (in French), INRIA, 1995.

[18] D. P. O'Leary and P. Whitman, *Parallel QR factorization by Householder and modified Gram-Schmidt algorithms*, Parallel Comput., 16 (1990), pp. 99–112.

[19] Y. Saad, *Krylov subspace methods in distributed computing environments*, Tech. Rep. 92-126, Army High Performance Computing Center, Minneapolis, 1992.

[20] Y. Saad and A. Malvesky, *P-Sparslib: a portable library of distributed memory sparse iterative solvers*, Tech. Rep. UMSI 95/180, University of Minnesota, Sept. 1995.

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

[22] J. N. Shadid and R. S. Tuminaro, *Sparse iterative algorithm software for large-scale MIMD machine: an initial discussion and implementation*, Tech. Rep., DOE's Massively Parallel Computing Research Laboratory, Sandia National Laboratories, Albuquerque, 1991.

[23] R. Sidje, *Algorithmes parallèles pour le calcul des exponentielles de matrices de grande taille : Application au calcul du régime transitoire des processus de Markov*, PhD thesis, Université de Rennes I, 1994.

[24] R. B. Sidje and B. Philippe, *Parallel Krylov subspace basis computation*, in CARI'94, 1994, pp. 421–440. 2ème Colloque Africain sur la Recherche en Informatique.

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

[26] X. Xu, N. Qin, and B. Richards, *Alpha-gmres-a new parallelizable iterative solver for large sparse nonsymmetrical linear systems arising from CFD*, Int. J. Numer. Meth. Fluids, 15 (1992), pp. 613–623.