Series Logo
Volume 00, 19xx

A Parallel Preconditioned GMRES Algorithm for Sparse Matrices

J. Erhel

ABSTRACT. This paper describes a parallel preconditioned variant of GMRES. Parallelism is achieved thanks to the so-called Newton-Arnoldi process which first builds a basis of the Krylov subspace then orthogonalises it. Preconditioning is done with an adaptive deflation method which is highly parallel. The target architecture is a distributed memory parallel computer. The basis formation requires sparse matrix-vector products which are parallelized using a communication pattern derived from the matrix structure. The use of local transformations (Givens rotations) allows to distribute efficiently the basis orthogonalisation. Convergence results show the robustness of Newton-Arnoldi and the efficiency of the deflation preconditioner. Performance results on a Paragon prove the scalability of our algorithm.

1. Introduction

The GMRES [**SS86**] algorithm is commonly used to solve large sparse nonsymmetric linear systems. The aim of this paper is to design and implement a parallel version of the preconditioned GMRES algorithm for general sparse matrices, with no particular pattern. The main part of GMRES is the Arnoldi process which generates an orthonormal basis of the Krylov subspace. Because of memory requirements, a restarted version must be used in general. It has been observed that the convergence of the restarted algorithm depends heavily on the dimension of the Krylov subspace and may be slower than in the full case [**HvdV89**]. In most cases, the algorithm must contain a preconditioning step to converge at a satisfactory rate.

The target architecture here is a parallel MIMD machine with a distributed memory. We choose a Single Program Multiple Data 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 parallelisation of GMRES has been studied by several authors. Our parallel version, called Newton-GMRES, is based on the ideas of [**BHR94**]. It first builds a Newton basis which is then orthogonalized. This solution requires the parallelisation of two steps : the basis formation which relies on matrix-vector products including preconditioning, followed by the basis QR factorisation. We

1991 *Mathematics Subject Classification.* 65f10,65f50.

have to distribute the matrix-vector product for a general sparse matrix with an arbitrary pattern of nonzeros. Our present work, described in [**Leg95, Erh96**], uses the library P-SPARSLIB designed in [**SM95**]. In our implementation, the QR factorisation is done either by a Gram-Schmidt process as in [**dSvdV93**] or by a Householder/Givens factorisation as in [**SP94**].

For preconditioning, we adapt to our Newton-GMRES algorithm the deflation method developed in [**EBP96**] for the classical GMRES. The full-GMRES version behaves as if the smallest eigenvalues were removed after some iterations [**vdV93**] but this is no longer true with restarting. After each cycle of GMRES, our deflation method updates a preconditioning matrix by pulling out new eigenvalues. This scheme is different from the flexible GMRES method [**Saa93**] because it executes a true GMRES cycle with a constant preconditioner inside the cycle. The preconditioning step requires merely level 2 dense BLAS operations. It is easy to parallelize using the same distribution as for the Arnoldi process.

The paper is organised as follows. The second part describes our variant of GMRES using a Newton-Arnoldi process and deflation for preconditioning. The third part is devoted to parallelisation. This involves sparse matrix-vector products, preconditioning and a QR factorisation. Numerical experiments in the fourth part first show the impact on convergence of the deflation technique. We then give performance results on a Intel Paragon parallel computer. Concluding remarks follow at the end.

2. DEFL-Newton-GMRES version

In this section, we describe the Newton-GMRES approach combined with a preconditioner based on deflation.

2.1. Sequential version. Consider the linear system $Ax = b$ with $x, b \in \mathbb{R}^n$ and with a nonsingular nonsymmetric matrix $A \in \mathbb{R}^{n\times n}$. The Krylov subspace $K(m; A; r_0)$ is defined by $K(m; A; r_0) = span(r_0, Ar_0, \ldots, A^{m-1}r_0)$. The GMRES algorithm uses the Arnoldi process to construct an orthonormal basis $V_m = [v_1, \ldots, v_m]$ 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 can stall. GMRES(m) leads to the fundamental relation

$$AV_m = V_{m+1}\overline{H}_m,$$

where the matrix $\overline{H}_m = (h_{ij})$ is a upper Hessenberg matrix of order $(m+1) \times m$. The algorithm computes $x = x_0 + V_m y_m$ where y_m is the solution of the least-squares problem

$$\min_{y\in\mathbb{R}^m} \|\beta e_1 - \overline{H}_m y\|,$$

usually solved by a QR factorisation of $\overline{H}_m$ using Givens rotations.

The most time-consuming part in GMRES(m) is the Arnoldi process. Indeed, the resolution of the small least-squares problem of size m represents a neglectible time.

2.2. Newton-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 in the same time. To enhance parallelism, we chose, as advocated for example in [**HW86, CPR93, BHR92, BHR94**], to divide the process into two steps : first build a basis then orthogonalize it. The first step is merely a succession of matrix-vector products and the second step an orthonormalisation process. Following [**BHR94**], we chose in [**Erh96**] to compute a Newton basis $\widetilde{V}_{m+1}$ of the Krylov subspace to ensure numerical stability. We then orthogonalize the basis so that we can write

$$A\widetilde{V}_m = \widetilde{Q}_{m+1}\widetilde{H}_m, \tag{2.1}$$

where $\widetilde{Q}_{m+1}$ is an orthogonal matrix and $\widetilde{H}_m$ is an upper Hessenberg matrix. We end up with a minimization problem similar to GMRES

$$\min_{y \in \mathbb{R}^m} \|\beta e_1 - \widetilde{H}_m y\|. \tag{2.2}$$

2.3. Deflation method. We adapt here the new scheme designed in [**EBP96**] where the preconditioner is allowed to vary from one cycle to another, as opposed to a flexible scheme where the preconditioner changes inside a cycle. Since we still apply GMRES in each cycle, the results for GMRES(m) are still valid. We choose a right preconditioning in order to guarantee a nonincreasing residual.

The objective of the preconditioner is to remove the smallest eigenvalues of A which are known to slow down the convergence of GMRES and to replace them by real positive eigenvalues equal to the largest modulus of the eigenvalues (for scaling reasons). The preconditioner will be based on a deflation technique such that the linear system is solved exactly in an approximate invariant subspace.

We denote by λ_n an approximation of the largest eigenvalue of A and by U an orthonormal basis of an approximate invariant subspace P of dimension r . The preconditioner is given by

$$M^{-1} = I_n + U\ (|\lambda_n|T^{-1} - I_r)\ U^T \tag{2.3}$$

where $T = U^T AU$ and r will increase at each restart. The preconditioner M^{-1} is merely a rank-r update of the identity matrix of the form $I + UXU^T$, where $X = |\lambda_n|T^{-1} - I_r$ is a dense matrix of order r, easy to compute.

After a first cycle of NewtonGMRES(m), we get an orthonormal basis $\widetilde{Q}_m$ of the Krylov subspace and we can easily compute the Hessenberg matrix $C_m = \widetilde{Q}_m^T A\widetilde{Q}_m$. We denote by λ_n the largest eigenvalue of C_m and by S_l the first l Schur vectors of C_m corresponding to its smallest eigenvalues. Then $U = \widetilde{Q}_m S_l$ approximate Schur vectors of A (after the first cycle, U is of size $r(1) = l$).

After each restart we update the matrix M in the following way. We compute in the Krylov subspace new Schur vectors S_l corresponding to approximate eigenvalues of AM^{-1} which in turn approximate remaining eigenvalues of A. They are used to approximate new Schur vectors of A which are added to U to get a more powerful preconditioner. In order to avoid loss of orthogonality, they are orthogonalized against the previous basis U. After each cycle j, U is of size $r(j)$. In the current version a fixed number l of eigenvalues are pulled out after each restart. Below is our algorithm called DEFL-Newton-GMRES(m,l) :

ALGORITHM : DEFL-Newton-GMRES(m,l)

ϵ is the tolerance for the residual norm ;

* Initialisation
Apply GMRES(m) 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 ;
λ_n is the largest eigenvalue ;
convergence = false ;
* Iterative scheme
until convergence **do**
  Compute a Newton basis $\widetilde{V}_{m+1}$;
  Orthogonalize it : $\widetilde{V}_{m+1} = \widetilde{Q}_{m+1}\widetilde{R}_{m+1}$;
  Compute $\widetilde{H}_m$ such that $A\widetilde{V}_m = \widetilde{Q}_{m+1}\widetilde{H}_m$
  Compute C_m such that $\widetilde{Q}_m^T A \widetilde{Q}_m = C_m$;
  find y_m solution of $\min_{y \in \mathbb{R}^m} \|\beta e_1 - \widetilde{H}_m y\|$;
  Compute $x_m = x_0 + \widetilde{V}_m y_m$;
  if $\|b - Ax_m\| < \epsilon$ convergence = true ;
else
  $x_0 = x_m$;
  compute l Schur vectors of C_m noted S_l ;
  orthogonalize $\widetilde{Q}_m C_l$ against U ;
  increase U by $\widetilde{Q}_m S_l$;
  $T = U^T A U$;
  $M^{-1} = I_n + U(|\lambda_n| T^{-1} - I_{r(j)})U^T$;
endif
enddo

where $r(j)$ is the current dimension of the basis U at the j^{th} cycle. Here S_l denotes a set of l Schur vectors in the Hessenberg matrix C_m of order m corresponding to its smallest eigenvalues. The basis U is increased by adding the new vectors $\widetilde{Q}_m S_l$ after they have been orthogonalized against the previous basis U.

DEFL-Newton-GMRES(m) contains two main steps : the computation of the basis $\widetilde{V}_{m+1}$ which involves preconditionings and matrix-vector products and its QR factorisation. We now analyze the parallelisation of these two kernels.

3. Parallelism on distributed memory architectures

The data distribution will be driven by the sparse matrix-vector product which we describe first. Then the distributed preconditioning will be straightforward. Concerning the QR factorisation, we shall describe two variants based on either Gram-Sschmidt or elementary transformations.

3.1. Parallel sparse matrix-vector product. The computation of $\widetilde{V}_{m+1}$ involves operations of the type $y = Ax$. We use the distributed algorithm designed by Saad [**Saa92**], where the matrix is distributed by rows and the vectors accordingly. 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 only local row and column indices :

$$A_{loc} = \{a_{ij}, y_i \text{ is local and } x_j \text{ is local}\}, \tag{3.1}$$

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

$$A_{ext} = \{a_{ij}, y_i \text{ is local and} x_j \text{ is external}\}, \tag{3.2}$$

such that the matrix-vector product is then written

$$y_{loc} = A_{loc} * x_{loc} + A_{ext} * x_{ext}. \tag{3.3}$$

Communications occur to guarantee the correct use of x_{ext}. This algorithm overlaps communication with computations and is scalable. It can be improved by a relevant row distribution based on renumbering techniques.

3.2. Parallel preconditioning. The preconditioning step is of the form

$$y = M^{-1}x = (I + UXU^T)x \tag{3.4}$$

where $U \in \mathbb{R}^{n \times r}$ and $X = |\lambda_n|T^{-1} - I_r \in \mathbb{R}^{r \times r}$.

Here all computations are dense. We choose naturally to distribute the vectors in U and the vectors x, y accordingly to the previous distribution. The product by U^T is simply composed of r independent dot-products. Global communications for the sums can be overlapped with local computations. The product by U is a pure parallel BLAS2 operation. The product by X (in fact triangular solvers after a Gauss factorisation) is done locally by each processor and is of low-cost since r is small.

3.3. Parallel QR factorisation. The other kernel of the Newton-Arnoldi method is the factorisation $\widetilde{V}_{m+1} = \widetilde{Q}_{m+1}\widetilde{R}_{m+1}$. So we have to design a parallel QR factorisation of a rectangular matrix of size $(m+1) * n$ with $m << n$. Here too, we keep the the data distribution chosen for the matrix-vector product. We implemented two parallel variants [**Leg95, Erh96**].

The first solution is to apply a modified gram-schmidt (MGS) method as in [**dSvdV92**]. It naturally relies on the distribution of the vectors inherited from the matrix distribution. It induces distributed dot-products which requires global communication. However, since several dot-products are independent, it is possible to overlap communications with computations. Though this parallel algorithm is not scalable because of global communications, it should be efficient for a reasonably large number of processors.

The second solution, called RODDEC, combines Householder transformations with Givens rotations as explained in [**Sid94, SP94**]. It requires only local communications and keeps all the processors active. In a first step, each processor computes a local QR factorisation of its own block. Then the algorithm annihilates the coefficients in the blocks below the first one by Givens rotations. Parallelism is achieved by mapping a logical ring structure and by sending data on this ring.

4. Numerical experiments

4.1. Convergence analysis of the preconditioner. We implemented in Matlab the deflation preconditioner using the classical Arnoldi process. Results on various matrices are given in [**EBP96**]. Here we give two examples with matrices of the form $A = SDS^{-1}$ with $A, S, D \in \mathbb{R}^{100 \times 100}$ and with $S = (1, \beta)$ a bidiagonal matrix with 1 on the diagonal and β on the upper subdiagonal. The matrices have

method	r_{max}	iterations	flops
FULL	0	54	1.2167
GMRES(10)	0	101	1.5601
GMRES(20)	0	96	1.5494
DEFLGMRES(10,1)	1	97	1.5629
DEFLGMRES(10,1)	2	81	1.4105
DEFLGMRES(10,1)	3	70	1.2820
DEFLGMRES(10,1)	4	64	1.2528
DEFLGMRES(10,1)	5	63	1.2993
DEFLGMRES(10,1)	6	62	1.3395

TABLE 1. Results for Example 1

the following characteristics

No.	β	D
1	0.9	$D_1 = diag(1, 2, \ldots 100)$
2	0.9	D_2

where D_2 is a block diagonal matrix with 2 by 2 blocks, with 12 complex eigenvalues and other eigenvalues evenly distributed : $1 \pm i, 2 \pm 2i, 3 \pm 3i, -3 \pm i, -2 \pm 2i, -1 \pm 3i, [13..100]$.

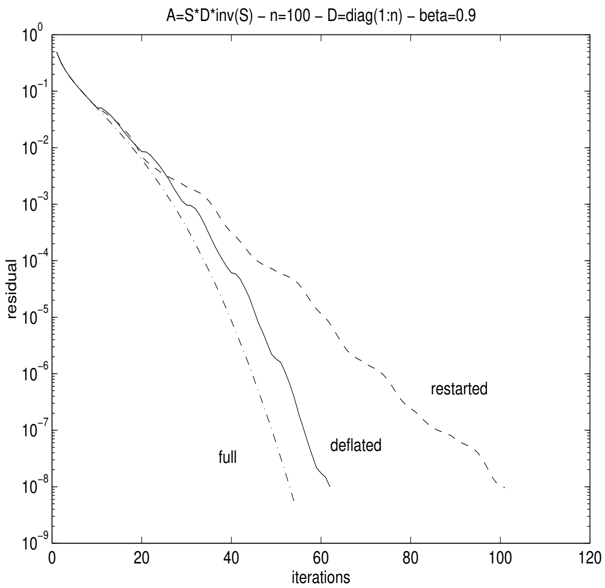

FIGURE 1. Convergence rates for Example 1

The system $Ax = b$ is solved for right-hand sides $b = (1, \ldots, 1)^T$ and GMRES starts with $x_0 = 0$. The tolerance ϵ for convergence is set to 10^{-8} . In the deflated version, noted DEFLGMRES(1), one real or two complex conjugate eigenvalues are pulled out at each restart ($l = 1$ or 2). The total number of extracted eigenvalues is bounded by r_{max}. Once this number is reached, the preconditioning is no more updated. Figures (1) and (2) show the convergence rate for FULL-GMRES

method	r_{max}	iterations	flops
FULL	0	69	1.7620
GMRES(10)	0	∞	*
GMRES(40)	0	∞	*
GMRES(50)	0	447	9.6937
DEFLGMRES(10,1)	8	∞	*
DEFLGMRES(10,1)	12	195	4.5698
DEFLGMRES(10,1)	17	143	3.7658

TABLE 2. Results for Example 2

noted full, GMRES(10) noted restarted, DEFLGMRES(10,1) with $r_{max} = n$ noted deflated. GMRES(10) converges only for example 1. On the other hand, DEFLGMRES(10,1) converges for both examples. GMRES(m) converges slowly for example 2 when increasing sufficiently m , as shown in tables 1 and 2. These tables compare the number of iterations and number of operations, given in the column flops in millions of operations. The memory requirements, measured by the number of vectors of size n , are given in the full case by the number of iterations, for GMRES(m) by m and for DEFLGMRES(m,l) by $m + 2r_{max}$.

These examples show the robustness and efficiency of DEFLGMRES(m) as compared to GMRES(m). It has a lower memory cost than FULL-GMRES(m) and should be comparable in time when increasing n. We did not yet implement DEFL-Newton-GMRES(m) but we can reasonably expect similar results.

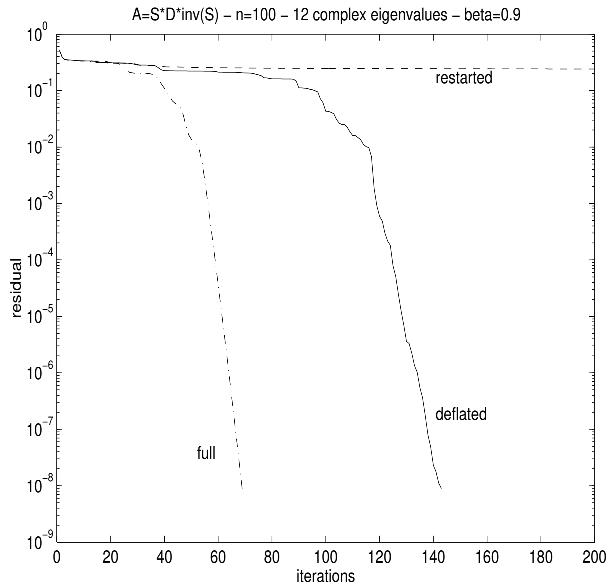

FIGURE 2. Convergence rates for Example 2

4.2. Performance results. We implemented three algorithms on a Paragon computer [**Erh96**]. Our Paragon has 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 with a distributed sparse matrix-vector product using a basic partition by blocks of rows. The two others, called NewtonGMRES(m)-MGS and NewtonGMRES(m)-RODDEC, implement the Newton-Arnoldi process respectively with the distributed MGS version and the RODDEC scheme.

We tested in [**Erh96**] the convergence of Newton-GMRES(m) and observed convergence rates similar to GMRES(m) provided that the Newton basis is well conditioned. For performance measures, we will consider only a non preconditioned version with a number of iterations chosen to 50.

Figure 3 gives the speedup over one processor for a varying number of processors with the matrix BCSSTK24 ($n = 3254, nz = 159910$) taken from the Boeing-Harwell collection [**DGL89**] for the three schemes using $m = 30$. Clearly, NewtonGMRES(m) is more parallel than the basic version. The RODDEC algorithm is more scalable than MGS because it avoids dot-products hence global communications.

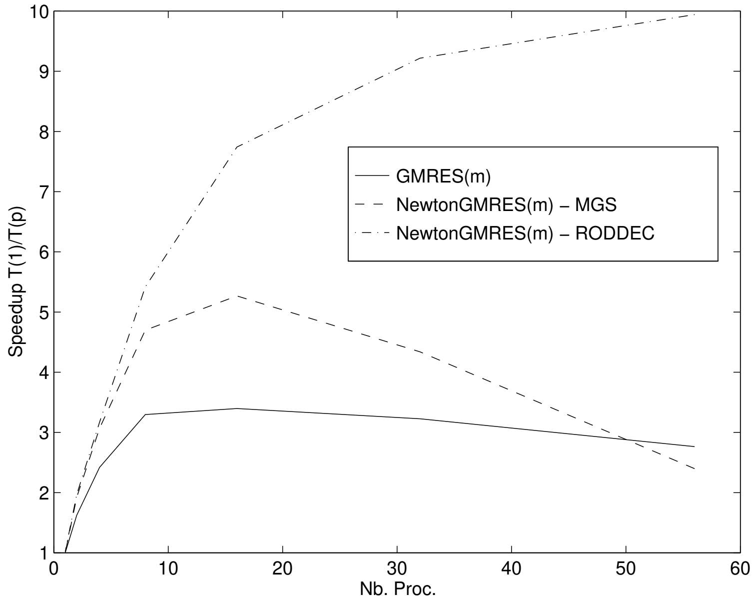

FIGURE 3. Speed-up (matrix BCSSTK24)

Figure 4 gives the speedup over four processors for a varying number of processors using a grid matrix ($n = 90000$) coming from the discretisation of the Laplacian on a unit square with a regular grid and a five-point difference scheme with $m = 10$. Here too, RODDEC is more scalable, though performances are similar up to about 30 processors. Here we exhibit a superlinear speedup due to memory effects. On four processors, the program has to swap quite often because the available memory is too small.

Percentages of each step in Newton-GMRES for a varying problem size, using a grid matrix with 100 rows per processor, are shown in Figures 5 and 6.

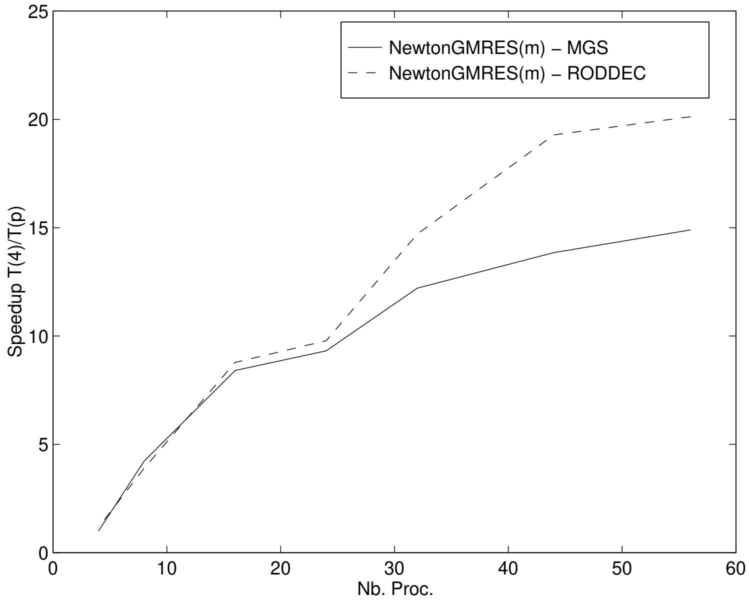

FIGURE 4. Speed-up (grid matrix, n=90000)

Newton-GMRES is decomposed into the basis formation involving sparse matrix-vector products, the basis factorisation done with either MGS or RODDEC, the initialisation including the computation of the Leja points and the least-squares resolution. As expected, the basis factorisation 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.

5. Concluding remarks

This paper presents an efficient parallel preconditioned variant of restarted GMRES for general sparse matrices. This algorithm is implemented on a Paragon which is a distributed memory parallel computer. Convergence results show the efficiency of our preconditioner. We obtain very good performances on a Paragon for the unpreconditioned version. We plan now to improve them by using graph partitioning methods to distribute the sparse matrix. We will also implement the preconditioner on Paragon to prove its parallel efficiency. Further experiments on large unsymmetric matrices will finally give more insight on the convergence behaviour and on the parallel scalability.

References

[BHR92] Z. Bai, D. Hu, and L. Reichel. An implementation of the GMRES method using QR factorisation. In J. Dongarra et al., editor, *Fifth SIAM conference on Parallel Processing for Scientific Computing*, pages 84–91, 1992.

[BHR94] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14:563–581, 1994.

[CPR93] D. Calvetti, J. Petersen, and L. Reichel. A parallel implementation of the GMRES method. In R. S. Varga L. Reichel, A. Ruttan, editor, *Numerical Linear Algebra*, pages 31–46, de Gruyter, Berlin, 1993.

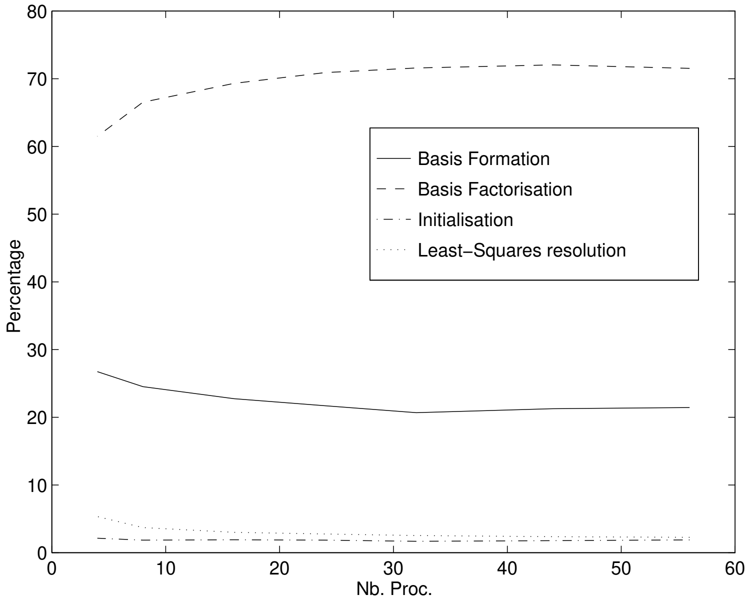

FIGURE 5. Breakdown of CPU time - Newton-MGS (100 rows/processor)

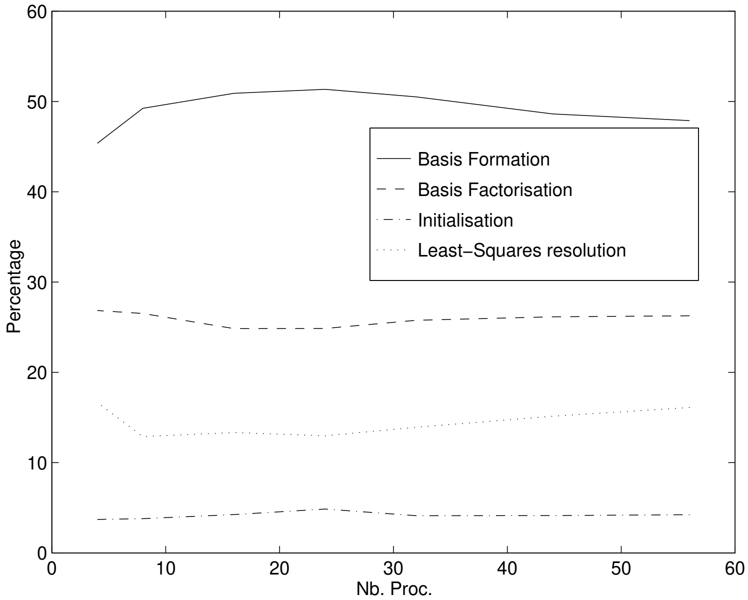

FIGURE 6. Breakdown of CPU time - Newton-RODDEC (100 rows/processor)

[DGL89] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM transactions on mathematica software*, pages 1–14, 1989.

[dSvdV92] E. de Sturler and H. A. van der Vorst. Communication cost reduction for Krylov methods on parallel computers. In Y. Saad, editor, *Numerical methods for large eigenvalue problems*, pages 190–196. Manchester University Press, 1992.

[dSvdV93] E. de Sturler and H.A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. Technical Report 832, Utrecht University, Department of Mathematics, October 1993.

[EBP96] J. Erhel, K. Burrage, and B. Pohl. Restarted gmres preconditioned by deflation. *Journal of Computational and Applied Mathematics*, to appear, 1996.

[Erh96] J. Erhel. A parallel gmres version for general sparse matrices. *Electronic Transactions on Numerical Analysis*, 1996.

[HvdV89] Yunging Huang and Henk van der Vorst. Some observations on the convergence behavior of gmres. Technical Report 89-09, Faculty of technical mathematics and informatics, Delft, 1989.

[HW86] A.C. Hindmarsh and H.F. Walker. Note on a Householder implementation of the GMRES method. Technical Report UCID-20899, Laurence Livermore Laboratory, 1986.

[Leg95] J-M. Legrand. Parallélisation de gmres(m). Dea report (in french), INRIA, 1995.

[Saa92] Y. Saad. Krylov subspace methods in distributed computing environments. Technical Report 92-126, Army high performance computing center, 1992.

[Saa93] Y. Saad. A flexible inner-outer preconditioned gmres algorithm. *SIAM J. Sci. Stat. Comput.*, 14(2):461–469, March 1993.

[Sid94] R.B. 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.

[SM95] Y. Saad and A.V. Malvesky. P-sparslib: a portable library of distributed memory sparse iterative solvers. Technical Report UMSI 95/180, University of Minnesota, September 1995.

[SP94] R. B. Sidje and B. Philippe. Parallel Krylov subspace basis computation. In *CARI'94*, pages 421–440, 1994. 2ème Colloque Africain sur la Recherche en Informatique.

[SS86] Y. Saad and M. H. Shultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM, J. Sci. Statist. Comput.*, 7:856–869, 1986.

[vdV93] H.A. van der Vorst. The superlinear convergence behaviour of gmres. *Journal of Computational and Applied Mathematics*, 48:327–341, 1993.

INRIA/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
E-mail address: erhel@irisa.fr