MULTIPLE EXPLICITLY RESTARTED ARNOLDI METHOD FOR SOLVING LARGE EIGENPROBLEMS

NAHID EMAD[†], SERGE PETITON[‡§], AND GUY EDJLALI[¶]

Abstract.
In this paper we propose a new approach for calculating some eigenpairs of large sparse non-Hermitian matrices. This method, called Multiple Explicitly Restarted Arnoldi (MERAM), is particularly well suited for environments that combine different parallel programming paradigms. This technique is based on a multiple use of Explicitly Restarted Arnoldi method and improves its convergence.

This technique is implemented and tested on a distributed enviromnment consisting of two interconnected parallel machines. MERAM technique is compared to Explicitly Restarted Arnoldi (ERAM) method, and one can notice that the accelaration of convergence is improved effectively. In some cases, more than one two-fold improvement can be seen in MERAM results.

We also implemented MERAM on a cluster of workstations. According to our experiments, MERAM converges better than Explicitly Restarted Block Arnoldi method and, for some matrices, more quickly than PARPACK package which implements Implicitly Restarted Arnoldi Method.

Key words. Large eigenproblem, Arnoldi method, explicit restarting, parallel programming, asynchronous communication, heterogeneous environment.

AMS subject classification. 65F15, 65F50 and 65Y05

Abbreviated title : Multiple Explicitly Restarted Arnoldi Method

1. Introduction.

Hybrid methods were proposed during the last decade to accelerate the convergence and/or to improve the accuracy of the solution of some linear algebra problems. These methods combine several different numerical methods to solve these problems efficiently. For example, both convergence acceleration techniques and preconditioning methods could be used to develop a hybrid method. Hybrid methods were used successfully to solve linear systems, such as the problem introduced by Brézinski and Redivo-Zagila [4], some eigenproblems with Arnoldi-Chebyshev method proposed by Saad [20, 19] and Jacobi-Davidson algorithms from Sleijpen, Van der Vorst and Bai [28].

In this paper, we propose a new approach for calculating some of the eigenpairs in the context of large sparse non-Hermitian matrices. This new technique, called Multiple Explicitly Restarted Arnoldi method (MERAM), is a hybrid method and is based on the Explicitly Restarted Arnoldi method (ERAM). It involves multiple invocations of ERAM and is closely linked to the Explicitly Restarted Block Arnoldi method. Each ERAM invocation is done with different set of parameter values. This approach may also be applied to some other restarted projection methods.

We propose an asynchronous parallel algorithm for Multiple Explicitly Restarted Arnoldi method and show that some of its properties such as asynchronous communications between its coarse grain subtasks, fault tolerance and dynamic load balancing

[†]Laboratoire PR*i*SM, Universit Versailles St-Quentin, 45, av. des tats-Unis, 78035 Versailles, France (emad@prism.uvsq.fr)

[‡]Laboratoire d'Informatique Fondamentale de Lille, Universit des Sciences et Technologies de Lille, Cit Scientifique, 59655 Villeneuve d'Ascq, France (petiton@lifl.fr)

[§]Laboratoire d'Applications Scientifiques du Calcul Intensif, Universit Paris-Sud, 91403 Orsay, France (petiton@asci.fr)

[¶]Dept of Electrical and Computer Engineering, Wayne State University, 5050 Anthony Wayne Drive, Detroit MI, 48202, USA (edjlali@ece.eng.wayne.edu)

make this method well-adapted to the GRID computational environments. Our experiments ran on two different platforms - a network of parallel machines and on a cluster of workstations - showed that MERAM converges quicker than ERAM and the Explicitly Restarted Block Arnoldi method. Moreover, according to our experiments, for some matrices, one have a difficult convergence (slow or impossible) with Implicitly Restarted Arnoldi Method and a fast convergence with MERAM.

Our asynchronous MERAM algorithm can be easily implemented on a cluster of heterogeneous machines as it exhibits a coarse grain parallelism. These machines can be sequential, vector, or parallel. The number of iterations to convergence of the main process is reduced by combining results from other processes at runtime. For example, a dedicated parallel machine can take advantage of the availability of a cluster of non-dedicated processors to speed-up its convergence. We implemented our approach on a CM5 and a small cluster of workstations [7], and were able to show improvement on the number of iterations before convergence by just combining two Sun Sparc machines to our 32 nodes CM5 parallel machine. In this paper, we will propose results when multiple parallel machines are available for computation. In particular, we will show that in some cases we can achieve more than one two-fold improvement in the number of iterations before convergence by combining the results from a faster and a slower parallel machine. We will propose also certain results of comparison of MERAM with Explicitly Restarted Block Arnoldi and Implicitly Restarted Arnoldi Methods on a cluster of workstations.

Before defining the Multiple Explicitly Restarted Arnoldi algorithm in the context of the projection methods, a brief presentation of these methods is given in the next section. An overview of the Arnoldi method and some of its variants is provided in section 3. The MERAM and an aynchronous algorithm of this method are presented in the section 4. It also includes the restarting strategy selected in MERAM, the stopping criterion of the restarts in ERAM/MERAM, and the relationship between MERAM and the Explicitly Restarted Block Arnoldi method. Parallel versions of these methods are discussed in section 5. Section 6 is devoted to numerical experiments. Finally, possible applications of the concept are discussed along with the conclusion in section 7.

2. General Purpose and Notations. Let A be a complex non-Hermitian matrix of dimension $n \times n$ and $\mathbb{K}$ be a subspace of $\mathbb{C}^n$. A Krylov subspace method allows an approximation of an eigenpair (λ, u) of A by a Ritz-elements pair ($\lambda^{(m)} \in \mathbb{C}$, $u^{(m)} \in \mathbb{K}$) where the subspace $\mathbb{K}$ is defined by

$$\mathbb{K}_{m,X} = Span(X, AX, \cdots, A^{m-1}X) \tag{2.1}$$

and X is spanned by a set of μ vectors. The point (resp. block) Krylov subspace methods are characterised by choosing $\mu = 1$ (resp. $\mu > 1$).

A Krylov subspace method approximates s eigenpairs of A by those of a matrix of order m, where $s \leq m \ll n$. This matrix is obtained by orthogonal projection of A onto an m-dimensional subspace $\mathbb{K}_{m,v}$ with $X = Span(v)$. Let W_m be the matrix whose columns $w_1, \cdots, w_m$ constitute an orthogonal basis of $\mathbb{K}_{m,v}$. The problem is to find $\lambda^{(m)} \in \mathbb{C}$ and $y^{(m)} \in \mathbb{C}^m$ such that

$$(H_m - \lambda^{(m)} I) y^{(m)} = 0 \tag{2.2}$$

where the matrix H_m of dimension $m \times m$, is defined by $H_m = W_m^H A W_m$. Note that W_m^H is the transpose conjugate of W_m and $u^{(m)} = W_m y^{(m)}$. Therefore, some

eigenvalues of A can be approximated by the eigenvalues of the matrix H_m. These eigenvalues can be found by building an orthogonal basis of $\mathbb{K}_{m,v}$ and by solving the problem (2.2). There are different ways of building such basis and the most used process is the Arnoldi's orthogonalization.

3. Arnoldi Method and some of its Variants.

3.1. Arnoldi Method. Let the initial guess w_1 be equal to $v/\|v\|_2$. The well-known Arnoldi process generates an orthogonal basis $w_1, \cdots, w_m$ of Krylov subspace $\mathbb{K}_{m,v}$ by using the Gram-Schmidt orthogonalization process:

`Arnoldi Reduction :` $\texttt{AR}(input : A, m, v; output : H_m, W_m)$.

For $j = 1, 2, \cdots, m$ do:

- $h_{i,j} = (Aw_j, w_i)$, for $i = 1, 2, \cdots, j$
- $\bar{w}_j = Aw_j - \sum_{i=1}^{j} h_{i,j} w_i$
- $h_{j+1,j} = \|\bar{w}_j\|_2$
- $w_{j+1} = \bar{w}_j / h_{j+1,j}$

The above algorithm may break down if $w_j = 0$ for some j. This may happen if the minimal polynomial of v is of degree j. In this case, the subspace $\mathbb{K}_{m,v}$ is invariant and the Ritz elements are exact [19].

This method was introduced by W. E. Arnoldi [1] in 1951 to reduce a matrix to an Hessenberg form. The reduced matrix $H_m = (h_{i,j})$ is an Hessenberg representation of A in the orthonormal basis W_m of $\mathbb{K}_{m,v}$ when $m = n$. Arnoldi hinted that the process could give good approximations to some eigenvalues if stopped before completion [1], i.e., when $m < n$. Today, it is the most common used method.

The matrices H_m and W_m issued from the `AR` algorithm and the matrix A satisfy the equation:

$$AW_m = W_m H_m + f_m e_m^H \tag{3.1}$$

where $f_m = h_{m+1,m} w_{m+1}$ and e_m is the m^{th} vector of the canonical basis of $\mathbb{C}^m$. Note that the `AR` algorithm is the standard Gram-Schmidt orthogonalization process, and assumes exact arithmetics. Therefore, the modified Gram-Schmidt or the Householder variants of `AR` are more interesting as they both deal with inaccuracies introduced by the hardware. The cost analysis and comparison of each version are given by Y. Saad in [19].

Once the choice of the orthogonalization process is fixed, the s desired Ritz values (with largest/smallest real part or largest/smallest magnitude) $\Lambda_m = (\lambda_1^{(m)}, \cdots, \lambda_s^{(m)})$ and the corresponding Ritz vectors $U_m = (u_1^{(m)}, \cdots, u_s^{(m)})$ can be calculated as follows[1]:

`Basic Arnoldi Algorithm :` $\texttt{BAA}(input : A, s, m, v; output : r_s, \Lambda_m, U_m)$.

1. Compute an $\texttt{AR}(input : A, m, v; output : H_m, W_m)$ step.
2. Compute the eigenpairs of H_m and select the s desired ones.
3. Compute the s associate Ritz vectors $u_i^{(m)} = W_m y_i^{(m)}$.
4. Compute $r_s = (\rho_1, \cdots, \rho_s)$ with $\rho_i = \|(A - \lambda_i^{(m)} I) u_i^{(m)}\|_2$.

[1] We suppose that the eigenvalues and corresponding eigenvectors of H_m are re-indexed so that the first s Ritz pairs are the desired ones.

If the accuracy of the computed Ritz elements is not good enough, the projection can be restarted again to generate a new $\mathbb{K}_{m,v}$.

The new $\mathbb{K}_{m,v}$ can be defined with *the same* initial vector v and *a larger* m value. It is clear that, according to the hypothesis that v does not belong to any desired invariant subspace, m has to be as large as possible. An important, well-known, shortcoming of this version of the Arnoldi method is its alarmingly large storage space requirement and computational cost for large values of m.

3.2. Explicitly Restarted Arnoldi Method. In this version the projection subspace size is *fixed*. Starting with an initial vector v, it computes `BAA` (Basis Arnoldi Method). The starting vector is updated and a `BAA` process is restarted again until the accuracy of the approximated solution is satisfactory (using appropriate methods on the computed Ritz vectors). This update is designed to force the vector to be in the desired invariant subspace. This goal can be reached by some polynomial restarting strategies proposed in [19] and discussed in section 4.2. This method is called the Explicitly Restarted Arnoldi (ERAM) and its algorithm is given below:

`ERAM`$(input : A, s, m, v, tol; output : r_s, \Lambda_m, U_m)$.

1. `Start.` Choose a parameter m and an initial vector v.
2. `Iterate.` Compute a `BAA`$(input : A, s, m, v; output : r_s, \Lambda_m, U_m)$ step.
3. `Restart.` If $g(r_s) > tol$ then use Λ_m and U_m to update the starting vector v and go to 2.

where tol is a threshold value and the function g defines the stopping criterion of iterations. We will see some possible definition of this function in the next section.

3.3. Implicitly Restarted Arnoldi Method. Another variant of the Arnoldi Method based on restarting technique is proposed by Sorensen [22]. This refined version, called Implicitly Restarted Arnoldi Method (IRAM), is a technique that combines the implicitly shifted QR with an Arnoldi factorization and can be viewed as a truncated form of the implicitly shifted QR technique. This technique involves an *implicit* application of a polynomial in A to the starting vector. It has been shown that IRAM leads to good approximations of Ritz elements of large sparse matrices [22, 11].

Several other versions of Arnoldi method based on the augmented Krylov methods exist including variants proposed by Wu and Simon [29], Stewart [24], Morgan [12] and Morgan and Zeng [13].

4. Multiple Explicitly Restarted Arnoldi Method. The method we propose is charactrised by a new restarting technique to the Arnoldi method. Our restarting technique combines the two algorithms described above (i.e., in last paragraph of 3.1 and in 3.2). In this version, we neither fix the parameter m nor the initial vector v. To overcome the storage dependent shortcoming of the first variant of the Arnoldi method, we introduce a constraint on the subspace size m. More precisely, we suppose that m belongs to a discrete interval $I_m = [m_{inf}, m_{sup}]$. The lower bound m_{inf} and upper bound m_{sup} may be chosen as a function of the computation and storage resources. In addition, the constraint $m_{inf} \leq m_{sup} \ll n$ has to be fulfilled. Let $m_1 \leq \cdots \leq m_\ell$ with $m_i \in I_m$ $(1 \leq i \leq \ell)$, $M = (m_1, \cdots, m_\ell)$ be a set of subspace sizes and $V^\ell = [v^1, \cdots, v^\ell]$ be the matrix of ℓ starting vectors. An algorithm of this

method for calculating s ($s \leq m_1$) desired Ritz elements of A is as follows:

MERAM($input : A, s, M, V^\ell, tol; output : r_s, \Lambda_M, U_M$)

1. `Start.` Choose a starting matrix V^ℓ and a set of subspace sizes $M = (m_1, \cdots, m_\ell)$.
2. `Iterate.` For $i = 1, \cdots, \ell$ do:
   (a) Compute a BAA($input : A, s, m_i, v^i; output : r_s^i, \Lambda_{m_i}, U_{m_i}$) step.
   (b) If $g(r_s^i) \leq tol$ then stop.
3. `Restart.` Update the vectors $v^1, \cdots, v^\ell$ and go to 2.

Note that r_s^i is the vector of the residual norms at the i^{th} iteration.

Suppose that $u_j^{(m_p)}$ is "better" than $u_j^{(m_q)}$ if $\rho_j^p \leq \rho_j^q$. Then an interesting updating strategy would be the choice of v^i as a function f^i of "the best" Ritz vectors:

$$v^i = f^i(U^{best}), \tag{4.1}$$

where $U^{best} = (u_1^{best}, \cdots, u_s^{best})$ and u_j^{best} is "the best" j^{th} Ritz vector. The function f^i has to be chosen in order to force the vector v^i to be in the desired invariant subspace. A simple choice of f^i would be a linear combination of $u_1^{best}, \cdots, u_s^{best}$ vectors. The definition of f^i can be also based on the techniques proposed by Saad in [19]. The latter will be discussed in more detail in section 4.2.

In order to ensure that the BAA processes, used in MERAM algorithm, produce ℓ different Krylov subspaces, $(f^i)_{i=1,\cdots,\ell}$ have to be chosen such that $f^i \neq f^j$ for $i \neq j$. Nevertheless, the vectors v^i and v^j, (for $i \neq j$), of step 3 can be defined using the same restarting strategy. In other words, the equation (4.1) may become:

$$v^i = f(U^{best}) \tag{4.2}$$

where $f = f^i$ (for $i = 1, \cdots, \ell$). The vectors $v^1, \cdots, v^\ell$ will become identical and the Krylov subspaces of different processes would look extremely close to one another after few iterations.

Clearly, the MERAM algorithm is equivalent to a particular use of several ERAMs. It allows the restarting vector v^i of an ERAM to be updated by taking interesting eigen-information obtained by the other ERAMs into account. Another advantage of this algorithm is that it can be parallelised easily, as it contains coarse grain parallelism. In fact, each iteration of step 2 can be executed as an independent process. Suppose that we have ℓ processes and a process (i) is assigned to the iteration (i) of the inner loop of the algorithm. Thereafter, the processes have to synchronise at the step 3. The only drawback of this parallelisation is that the execution time of the algorithm is dominated by the execution time of the slowest process. The algorithm defined in the following section is a remedy for this problem.

4.1. Asynchronous MERAM. We suppose that after each `BAA` step, each process broadcasts its output of interest to all other processes. The computation of a restarting vector v^i on process (i) can than be done by combining "the best" computed Ritz elements *available* within that process. As a consequence, this parallel algorithm *is not* precisely a parallel version of the above MERAM algorithm where `BAA` processes are *all* synchronized in step 3. In fact, the asynchronism introduced at the end of each `BAA` step produces a "parallel" version of MERAM algorithm without

a corresponding "serial" MERAM version[2].

We call a dynamic algorithm, an algorithm whose one or more components are dynamic and a dynamic process a process whose run time behavior can depend on outside events. The above parallel asynchronous MERAM can be considered as a dynamic algorithm. To each run of this version corresponds one parallel static MERAM version in which the matrices U^{best} in the updating strategy (4.1) are different for every ERAM process. That means, the initial vector v^i can be thus defined by $v^i = f^i(U^{best(i)})$ where $U^{best(i)}$ denotes the matrix U^{best} of the ERAM process (i). Consequently, so that MERAM does not produce the same Krylov subspaces, one must have $f^i \neq f^j$ and/or $U^{best(i)} \neq U^{best(j)}$ (for $i \neq j$). In other words, this way of asynchronising communications between the `BAA` processes allows MERAM to produce ℓ *different* Krylov subspaces even with the restarting strategy (4.2)[3].

The asynchronous MERAM consists of ℓ "independent" ERAM processes which cooperate to update their restarting vectors. Let `Send_Eigen_Info` represents the task of sending eigen-information of interest from an ERAM process to all other ERAM processes. Let `Receiv_Eigen_Info` be the task of receiving eigen-information of interest from one or more ERAM processes by the current ERAM process. Finaly, let `Rcv_Eigen_Info` be a boolean variable that is true if the current ERAM process has received eigen-information from the other ERAM processes. An asynchronous parallel version of MERAM is explained in the following:

`An asynchronous MERAM Algorithm.`

1. `Start.` Choose a starting matrix V^ℓ and a set of subspace sizes $M = (m_1, \cdots, m_\ell)$.
2. `Iterate.` For $i = 1, \cdots, \ell$ do in parallel:
   - Computation process (`ERAM` process)
     (a) Compute a `BAA`$(input : A, s, m_i, v^i; output : r_s, \Lambda_{m_i}, U_{m_i})$ step.
     (b) if $g(r_s^i) \leq tol$ stop all processes.
     (c) If (`Rcv_Eigen_Info`) then `hybrid restart`
     else `simple restart`
   - Communication process
     (a) `Send_Eigen_Info`
     (b) `Receiv_Eigen_Info`

In this algorithm, each ERAM process has either a simple or a hybrid restarting strategy. A `simple restart` is a classical restarting strategy taking into account the eigen-information computed by the current ERAM process. A `hybrid restart` is a restarting strategy taking into account the eigen-information received by the other ERAM processes also. A hybrid restart includes data from other processes when eigen-information has been received, while the simple restart uses just the eigen-information computed by the current process. Let $u_j^{(m_k)}$ be the j^{th} Ritz vector computed by the k^{th} process (corresponding to the one with the subspace size m_k) and received by the current process. The matrix U^{best} of (4.1) is then defined by $(u_1^{best}, \cdots, u_s^{best})$ where u_j^{best} is "the best" of $u_j^{(m_k)}$ and the j^{th} Ritz vector computed by the current

[2]The terms parallel and serial are used here to mention the execution in parallel or in serial of just the BAA processes of the MERAM.

[3]with $m_i \neq m_j$ (for $i \neq j$) on homogeneous resources.

process, where $k \in F \subset [1, \ell]$, $cardinal(F) = p$, and p is the number of the processes that sent their eigen-information to the current process. When a new and "better" Ritz pair is received from a process, its older value is overwritten by the new value. In addition to the coarse grain parallelism between ℓ ERAM processes, one can also overlap communication step by computations. More details about the parallelisation of this algorithm is given in section 5.

A consequence of the dynamicity of this algorithm is its large sensitivity to the variations of the system on which it runs. Indeed, the results of this algorithm are a function of architectural parameters such as the load of the interconnection network or that of the processors which constitute the system. One important property of the asynchronous MERAM algorithm is its fault tolerance. That means, the loss of an ERAM process at run time doesn't impede the other ERAM processes to continue to run and finish the algorithm. Another important property of this algorithm is its capacity of dynamic load balancing. Indeed, the complexities in time and in space of an iteration of an ERAM process of MERAM is a function of the subspace size of this process. Consequently, by using these complexities, the system can attribute dynamically the ERAM processes to the processors of the system according to their load. All these properties make the asynchronous MERAM a well suited algorithm to the GRID computational environments.

4.2. Restarting Strategies - Convergence. The restarting strategy is a critical part of both ERAM and MERAM. Saad [20] proposed to restart the iteration of ERAM with a preconditioning vector as to force it to be in the desired invariant subspace. It concerns a polynomial preconditioning applied to the starting vector of ERAM. The aim of the preconditioning is to make sure that the components of the restarting vector are nonzero in the desired invariant subspace and zero in the unwanted invariant subspace:

$$v(k) = p(A)v \tag{4.3}$$

where $v(k)$ is the k^{th} update of the starting vector v of ERAM, $v = v(0)$ is the starting vector and p is a member of the space of polynomials of degree $< m$. One can define p as a Chebyshev polynomial determined from some knowledge about the distribution of A eigenvalues. This restarting strategy is very efficient in speeding the ERAM convergence (For more details see [20, 19]). Another way of defining the polynomial p is to compute the restarting vector with a linear combination of s desired Ritz vectors:

$$v(k) = \sum_{i=1}^{s} \alpha_i u_i^{(m)}(k) \tag{4.4}$$

where $u_i^{(m)}(k)$ denotes the i^{th} Ritz vector computed at the iteration k. This restarting strategy is polynomial because $u_i^{(m)}(k) \in \mathbb{K}_{m,v(k-1)}$ implies $u_i^{(m)}(k) = \phi_i(A)v$ for some polynomial ϕ_i and thus $v(k) = \sum_{i=1}^{s} \alpha_i \phi_i(A)v$. There are several ways to choose the scalar values α_i in (4.4). One choice can be α_i equal to the ith residual norm. Other choices can be $\alpha_i = 1$, $\alpha_i = i$ or $\alpha_i = s - i + 1$ for $1 \leq i \leq s$ (see [21] for more details).

Note that if $v = \sum_{j=1}^{n} \gamma_j u_j$, the IRAM with exact shifts [22, 11] provides a specific selection of expansion coefficients γ_j for a new starting vector as a linear combination of the current Ritz vectors for desired eigenvectors. Implicit restarting provides a means to extract eigen-information of interest from large Krylov subspaces while avoiding the storage and numerical difficulties. This is done by continually

compressing eigen-information of interest into an s-dimensional subspace of fixed size. This means that IRAM continues a `BAA` step, having kept all Ritz vectors of interest.

To define a restarting strategy for MERAM we have to define the functions f^i introduced in (4.1). The latter are the functions with several variables and have to be used as input of MERAM. Their variables are calculated dynamically in the case of the asynchronous MERAM. These functions constitute a very part of this algorithm. Here again we have several possibilities of choosing f^i. The first possibility is to choose them as simple as possible by using the same restarting techniques proposed for ERAM. The second possibility can be defined by using sophisticated restarting techniques such as augmented Krylov or implicit restarting. In the rest of this paper, we use a simple restarting technique defined in the (4.4) and we call it the *hybrid restarting strategy*. According to the hypothesis: $(\lambda_j^{(m_p)}, u_j^{(m_p)})$ is better than $(\lambda_j^{(m_q)}, u_j^{(m_q)})$ if $\rho_j^p \leq \rho_j^q$, the hybrid restarting strategy is defined by:

$$v^i = f^i(U^{best}) = \sum_{j=1}^{s} \alpha_j^i u_j^{best} \tag{4.5}$$

where α_j^i (for $j = 1, s$ and $i = 1, \ell$) are some scalar values. The hybrid restarting strategy corresponding to equation (4.2) is then:

$$v^i = f(U^{best}) = \sum_{j=1}^{s} \alpha_j u_j^{best} \tag{4.6}$$

where $\alpha_j = \alpha_j^1 = \cdots = \alpha_j^\ell$.

We can view $(\ell - 1)$ ERAM processes of MERAM as the convergence accelerators of the last process. We implemented the parallel MERAM with the restarting strategy (4.6). Our experiments in section 6 illustrate that this ERAM process converges quicker than an ERAM with the same subspace size and starting vector. In other words, our experiments show that the restarting vector computed with the hybrid restarting technique is more in the desired invariant subspace than the one computed with (4.4).

Clustered Eigenvalues. If the desired eigenvalues are not well-separated, the restarting strategy (4.4) applied to ERAM will cause many iterations back and forth toward the invariant subspace of interest. This means that the sum of the residual norms (SRN[4]) corresponding to the desired eigenvalues is a *non-monotonic* function of the number of restarts. There are many examples in [8] confirming this convergence behavior for clustered eigenvalues even if the matrix A is symmetric. This convergence history has been observed by other authors including Sleijpen, Van der Vorst and Bai [28] for symmetric and non-symmetric cases.

MERAM is based on ERAM and the restarting strategy (4.6) of MERAM is the corresponding restarting strategy (4.4) of ERAM. One can conclude that as long as the restarting strategy of MERAM is a *linear combination* of "the best" desired Ritz vectors (i.e., strategy (4.6)), the above convergence history should also be valid for MERAM.

[4]SRN is a stronger convergence measure than the residual of each Ritz vector alone. This because SRN bounds the subspace residual : $\|AU_m - U_m\Lambda_m\|_2 \leq SRN = \sum_{i=1}^{s} \|Au_i^{(m)} - \lambda_i^{(m)} u_i^{(m)}\|_2$.

4.3. Stopping Criterion. The residual norms ρ_i defined in the previous section verify the well known equation [19]:

$$\rho_i = \|(A - \lambda_i^{(m)} I)u_i^{(m)}\|_2 = h_{m+1,m}|e_m^H y_i^{(m)}| \tag{4.7}$$

where e_m is the m^{th} vector of the canonical basis of $\mathbb{C}^m$. This enables the computation of the residual norms at low cost because the i^{th} residual norm is equal to the product of the last component of the eigenvectors $y_i^{(m)}$ by $h_{m+1,m}$. However, in practice the residual norms ρ_i can be more indicative of current errors.

Let r_s be a vector of the residual norms $(\rho_1, \cdots, \rho_s)$. To stop the restarts in ERAM one has to define a function g given in step 3. Some typical examples are:

$$g(r_s) = \|r_s\|_\infty \tag{4.8}$$

or by a linear combination of the residual norms

$$g(r_s) = \sum_{j=1}^{s} \alpha_j \rho_j \tag{4.9}$$

where α_j are scalar values[5]. The same technique applied to $r_s^i = (\rho_1^i, \cdots, \rho_s^i)$ can be used to define the stopping criterion of restarts - the function g in step 2.(b) - of MERAM.

4.4. Relationship with Explicitly Restarted Block Arnoldi Method. As shown in section 2, the *block Krylov subspace methods* allow the approximation of $(\lambda^{(m)}, u^{(m)})$ solution of (2.2) with $\mathcal{K}_{m,X}$ an $m \times \mu$-dimensional subspace defined by (2.1), where $X = span(x_1, \cdots, x_\mu)$ and $\mu > 1$. We assume that the vectors $x_1, \cdots, x_\mu$ are linearly independent.

The Explicitly Restarted Block Arnoldi Method (ERBAM) is used to extract the eigenvalues whose algebraic multiplicity is less than or equal to the block size. Let X_1 be the orthonormal matrix represented by the columns $x_1, \cdots, x_\mu$. The block Arnoldi process generates a set of $X_1, \cdots, X_m$ matrices. Let $\boldsymbol{W}_m$ be the $(n, m \times \mu)$-size matrix $[X_1, \cdots, X_m]$. The $m \times \mu$ columns of this matrix constitute an orthogonal basis of Krylov subspace $\mathbb{K}_{m,X_1}$:

`Block Arnoldi Reduction :` `BAR`$(input : A, m, X_1; output : \boldsymbol{H}_m, \boldsymbol{W}_m)$.

For $j = 1, 2, \cdots, m$ do:

- $H_{i,j} = X_i^H A X_j$, for $i = 1, 2, \cdots, j$
- $\bar{X}_j = AX_j - \sum_{i=1}^{j} X_i H_{i,j}$.
- Compute QR decomposition of $\bar{X}_j$:
  $\bar{X}_j = Q_j R_j = X_{j+1} H_{j+1,j}$,

The $H_{i,j}$ matrices generated by the `BAR` process constitute the blocks of an upper block Hessenberg matrix $\boldsymbol{H}_m$. The eigenvalues of this matrix approach the corresponding eigenvalues of A. Then, an equivalent equation to (3.1) will be

$$A\boldsymbol{W}_m = \boldsymbol{W}_m \boldsymbol{H}_m + F_m \boldsymbol{E}_m^H \tag{4.10}$$

[5] if $\alpha_j = 1$ (for $j = 1, \cdots s$) then (4.9) defines the SRN associated with s desired eigenelements.

where $F_m = X_{m+1}H_{m+1,m}$ and $\boldsymbol{E}_m$ is the matrix of the last μ columns of $I_{n\mu}$. The block version of `BAA` is unchanged, except for the first step where we calculate `BAR`$(input : A, m, X_1; output : \boldsymbol{H}_m, \boldsymbol{W}_m)$ instead of `AR`$(input : A, m, v; output : H_m, W_m)$. We call this algorithm `BBAA`$(input : A, s, m, X_1; output : r_s, \Lambda_{m\times\mu}, U_{m\times\mu})$.

Even though both ERBAM and MERAM begin with a set of starting vectors, they are not equivalent. Suppose that $\ell = \mu$ is the number of starting vectors for both methods. $m \times \ell$ is the projection subspace size in ERBAM and m is the size of each projection subspace in MERAM, which means $m_1 = \cdots = m_\ell = m$. The difference between these two methods is that the projection subspace in ERBAM is $\mathbb{K}_{m,X=span(x_1,\cdots,x_\ell)}$ while in MERAM the problem is projected onto ℓ subspaces $\mathbb{K}_{m,x_1}$, $\cdots$, $\mathbb{K}_{m,x_\ell}$. Now, assume that the degree of the minimal polynomial of X_1 is greater than m, these Krylov subspaces satisfy the following relation:

$$\mathbb{K}_{m,X} = \mathbb{K}_{m,x_1} \oplus \cdots \oplus \mathbb{K}_{m,x_\ell}. \tag{4.11}$$

Therefore, if $\dim(\mathbb{K}_{m,X}) = m \times \ell$ we have $\dim(\mathbb{K}_{m,x_i}) = m$ for all $i \in [1, \ell]$ and $\dim(X) = \ell$.

Suppose that $\dim(\mathbb{K}_{m,X}) = m \times \ell$. The construction of an orthogonal basis $((x_1^1, \cdots, x_\ell^1), \cdots, (x_1^m, \cdots, x_\ell^m))$ of $\mathbb{K}_{m,X_1}$ is equivalent to the construction of an orthogonal basis $\tilde{X}_i = (x_i^1, \cdots, x_i^m)$ for each subspace $\mathbb{K}_{m,x_i}$, $1 \le i \le \ell$. The inverse is true only if $\tilde{X}_i$ is orthogonal to $\tilde{X}_j$ for all $i, j \in [1, \ell]$ and $i \neq j$. This means that the block Arnoldi reduction which is also a Gram-Schmidt orthogonalization of Krylov vectors of $\mathbb{K}_{m,X_1}$ defines ℓ Arnoldi reductions which again are ℓ Gram-Schmidt orthogonalizations of Krylov vectors of subspaces $\mathbb{K}_{m,x_i}$ $(1 \le i \le \ell)$ but the inverse is not always true.

In the first method, s desired eigenvalues are approximated by the eigenvalues of an $(\ell \times m)$-size block Hessenberg matrix, while in the second, they are approximated by s "best" eigenvalues of ℓ Hessenberg matrices of order m. We notice also that in ERBAM the relation $m \ge s/\ell$ has to be true while in MERAM the relation $m \ge s$ must be true.

Some parallel and numerical properties of MERAM and ERBAM are discussed in the next two sections.

5. Parallelism Analysis.

5.1. Parallel ERAM. Let us consider the Basic Arnoldi algorithm - `BAA`$(input : A, s,\ m, v; output : r_s, \Lambda_m, U_m)$ - which is the core of ERAM. This algorithm can be considered as four main tasks corresponding to its four steps. The projection phase of the algorithm manipulates the n-sized data sets. The second phase acts on m-sized data sets and the third and fourth phase deal with the n and m sized data sets[6]. Steps 1, 3 and 4 constitute the expensive parts of this algorithm. They are composed mainly of sparse matrix-vector multiplications and triadic and reduction operations.

The global programming model used is message passing, which is crucial for distributed architectures. This model is assuming that the system has a number of processors that have local memories and communicate with each other by means of memory transfer from one processor to another [5]. A classical way to parallelize `BAA` using message passing model is to distribute the large arrays (vectors and/or matrices) among the processors while the small arrays are kept on just one processor or

[6]This is not true for the fourth step if the right hand side of equation (4.7) is used to compute the vector r_s.

redundantly on all processors. This is equivalent to distributing each of the tasks 1, 3 and 4 between the processors and to run step 2 on just one processor or simultaneously on all processors.

In the context of the message passing programming model, the projection phase of `BAA` is highly parallel. The second task can represent a good vector structure[7] and the last two tasks are also parallel. These different types of parallelism can be exploited efficiently by a distributed memory platform formed, for example, by a distributed memory parallel machine and a vector computer or by a distributed memory parallel machine and a sequential machine. We must mention that we do not have to use message passing model for the step 2, as this task would become a bottleneck in the algorithm [17]. This means, in terms of implementation, the architecture used for step 2 ought to be different from the one used for steps 1, 3 and 4 (see figure (1)).

In ERAM algorithm the size of the subspace determines the amount of time necessary to execute each restart (mainly a `BAA` step). The cost of each restart increases with the subspace size, while in general the number of restarts decreases according to the subspace size. We need to find a balance between the execution time of each restart and the number of restarts to minimize the total execution time [17, 6]. Generally, the larger the subspace size, the better the convergence properties, so the subspace size m ought to be chosen as large as possible given the limitations of the target system resources.

5.2. Parallel MERAM. MERAM consists of several ERAMs interacting with some protocol. As we mentioned in section 4, a great advantage of MERAM is its coarse grain parallelism. We consider here the asynchronous parallel MERAM algorithm described in section 4.1. In this algorithm, each ERAM core - `BAA`($input$: $A, s, m, v; output$: r_s, Λ_m, U_m) - can be executed *independently* and *asynchronously* from the others. Consequently, we have ℓ independent coarse grain tasks (`ERAM` processes). For each of them, one can make use of above described parallelism.

In addition to the coarse grain parallelism between ℓ ERAM processes, one can also overlap communication step by computations. The processors would always be active and execute a non-redundant task. Therefore, a significant reduction of the response time of this algorithm is expected. In the next section, we will confirm this with our experiments.

5.3. Parallel ERBAM. Clearly, the paralellisation method described in section 5.1 is also valid for ERBAM (i.e., `BBAA` in section 4.4). However, if the subspace sizes of ERAM and ERBAM are equal to $\mathcal{M} = m \times \mu$ with $\mu > 1$, the coarse grain parallelism in each `BBAA` step is larger than in the corresponding `BAA` step. This is due to the `BAA` core (i.e., `AR`) computes $\mathcal{M} = m \times \mu$ matrix-vector products with A while in `BBAA` core (i.e., `BAR`) the computations of A on μ vectors can be done at once; in roughly the same cost as computing a single matrix-vector product. The `AR` has to compute $\mathcal{M}$ second level BLAS operations while `BAR` has to compute m third level BLAS operations. Consequently, the computation cost of ERAM should be less effective than ERBAM computation cost.

The global programming model used to implement ERBAM and MERAM is message passing. Bearing in mind that large arrays are distributed among the processors and small arrays are kept on one processor or replicated on all processors, this is equivalent to distributing each of the tasks 1, 3 and 4 of `BBAA` among the processors and running step 2 on just one processor or redundantly on all processors.

[7] If we use a classical method like QR and/or inverse iteration to solve this step.

Consider a MERAM with (μ) ERAM processes having the subspace sizes m_i with $\sum_{i=1}^{\mu} m_i = \mathcal{M}$ and $m_1 \leq \cdots \leq m_\mu$. The goal is to determine if this MERAM can be competitive with ERBAM with subspace size $\mathcal{M} = m \times \mu$. In other words, we would like to know if ERAM(m_μ) helped by the other ERAM processes of MERAM (i.e., ERAM(m_i) for $i = 1, \cdots, \mu - 1$) can be competitive with ERBAM.

6. Numerical experiments. Data-parallel and Vector programming models are very well suited for ERAM [17]. To efficiently exploit the natural parallelism of MERAM, one has to use the coarse grain parallel model, where a number of ERAM processes are running concurrently. In addition, one can exploit the fine grain parallelism within each task (ERAM process), in order to maximize the resource usage of a Meta-Computing platform. A meta-Computer consists of heterogeneous and autonomous computers linked by an interconnection network. This architecture has a tremendous amount of memory and computational power [10] to cope with large industrial applications. However, in practice, the number of parallel machines involved in a meta-computer is limited. We present, in this section, some results of our experiments on homogenous and heterogenous programming environments.

We denote by ERAM(m, v) (resp. IRAM(m, v)) an ERAM (resp. IRAM) process with the subspace size m and the initial vector v, by ERBAM(m, V^ℓ)an ERBAM process with the subspace size m and the initial matrix V^ℓ and by MERAM(M, V^ℓ) a MERAM with ℓ ERAM$(m_i, v^i)_{i=1,\ell}$ processes where $M = (m_1, \cdots, m_\ell)$ and $V^\ell = [v^1, \cdots, v^\ell]$. The matrix I_k denotes that one whose columns are $e_1, \cdots, e_k$.

6.1. MERAM and ERAM.

Harware Platform. We implemented ERAM and MERAM according to the above programming models on different heterogeneous environments [8]. These environments were a 32-node Connection Machine (CM5), CM5 and a Sparc 20, a CM5 and 2 Sparc 20, and a CM5 and a 4K Connection Machine 200 (CM200). The CM5 is a distributed memory MIMD[8] architecture [26]. Each node of CM5 consists of a Sparc processor, a 32 mega byte memory and 4 floating point computation unites operating at 32 mega flops each. The CM5 supports message passing model, data parallel model or a combination of the two. The CM5 nodes are connected by three networks. *Data network* is a *fat-tree* point to point communications and has 0.5 Gb per second global bandwidth. *Control network* is a binary tree for global operations (as diffusion, reduction, etc.) by pipelining the messages and *diagnostic network* is dedicated to error messages. The CM200 is a massively parallel SIMD[9] machine consisting of thousands of bit serial data processors and a front-end for management and control [25]. The processors and memory are packaged as 16 in a chip. Each chip also contains the routing circuitry which allows any processor to send and receive messages from any other processor in the system. The system version used has 0.5 giga byte memory and operates at 10 MHz. The CM200 supports data parallel programming model. Both CM5 and CM200 have workstation as front-end. The programming language used on CM5 and CM200 is CM Fortran [27]. This is an extension of Fortran 77 allowing to handle parallel variables distributed on the processors.

In [7], we showed that a significant acceleration can be achieved when the heterogeneous architecture consists of a CM5 and 2 Sparc workstations. In this paper, we present the results of our experiments on a CM5 for ERAM and on a heterogeneous environment consisting of a CM5 and a CM200 interconnected by a 10Mbit/s

[8]Multiple Instruction Multiple Data
[9]Single Instruction Multiple Data

Ethernet link for MERAM.

Implementation. As we have seen in the previous section, the Basic Arnoldi algorithm can be divided in four main tasks: the projection, the computation in the subspace, the Ritz vectors computation and the computation of the vector of the residual norms. These tasks can run on different machines. The projection task, the Ritz vectors computation and the computation of the residual norms are highly data-parallel. They will be mapped on a parallel machine. The computation on the subspace on the other hand is suited for vector programming. It will be mapped onto the front-end (i.e., control processor) of the parallel machine. Figure 1 presents this implementation of ERAM on a distributed architecture consisting of a parallel architecture and a vector/sequential machine. In our experiments the parallel machine is CM5 or CM200 while the other machine is the corresponding front-end.

Figure 2 describes the parallel MERAM algorithm for $\ell = 2$ on a heterogeneous architecture. TCMa and TCMb (resp. Tcomma and Tcommb)) are computation (resp. communication) tasks of the algorithm. TCMa (resp. TCMb) communicates with Tcomma (resp. Tcommb) using shared memory model. Tcomma and Tcommb are connected by a network. These communication tasks are designed to overlap communication with computation phases between parallel machines. TCMa (resp. TCMb) implements an ERAM process. After each iteration, if the convergence is not reached, the eigen-information is stored in the shared memory (SM) and sent by one communication task to the other process. TCMa and TCMb are implemented on a CM5 and a CM200 interconnected by a 10Mbit/s Ethernet link. The subspace computation of TCMa and TCMb are mapped onto the CM5 front-end and the CM200 front-end respectively. PVM library is used to implement the communications between Tcomma and Tcommb.

Test Matrices. The sparse matrices for our experiments have a C-diagonal pattern. The only non-zero elements of a C-diagonal matrix are on the main diagonal and the diagonals immediately around it. C is the maximum number of non-zero elements per row or per column. These sparse matrices have some interesting properties. They are easy to build and well suited for SGP (Sparse General Pattern) data parallel storage format [18]. It has also been shown in [8] that matrices with C-diagonal pattern have good performance in sparse matrix-vector multiplication algorithms executed on data-parallel machines. The non-zero upper-diagonal and lower-diagonal elements of our non-symmetric matrices are chosen in $[-1, +1]$ range while the non-zero diagonal elements are in $[0, C]$. To study the convergence properties of MERAM and to compare it with ERAM, we intentionally limited the sizes of our test matrices. The size of the test matrices is fixed to 1024.

Performances. The execution of ERAM on CM5 is done with starting vector equal to v, the subspace size m and the restarting strategy (4.6) with $\alpha_i = 1$ (for $i = 1, s$). While, two starting vectors (v^1, v^2), two subspace sizes (m_1, m_2), and two restarting strategies are needed to execute MERAM on both CM5 and CM200. We suppose that we have the same parameters on the CM5 for ERAM and for MERAM. This means that $v^1 = v$, $m_1 = m$ and the restarting strategies for ERAM and MERAM will be the same if the restart of MERAM is simple on the CM5. In the case of hybrid restart, we have two sets of eigen-information $E_{m_1} = (\Lambda_{m_1}, U_{m_1}, r_s^1)$ and $E_{m_2} = (\Lambda_{m_2}, U_{m_2}, r_s^2)$. We choose "the best" to form a new set of eigen-information: $E_{best} = (\Lambda^{best}, U^{best}, r_s^{best})$. Then, with these s Ritz vectors, we compute the restarting vector by (4.6) with $\alpha_i = 1$ (for $i = 1, s$). The speedup, S, is calculated as the ratio of the time needed to solve the problem with MERAM (on CM5 and CM200)

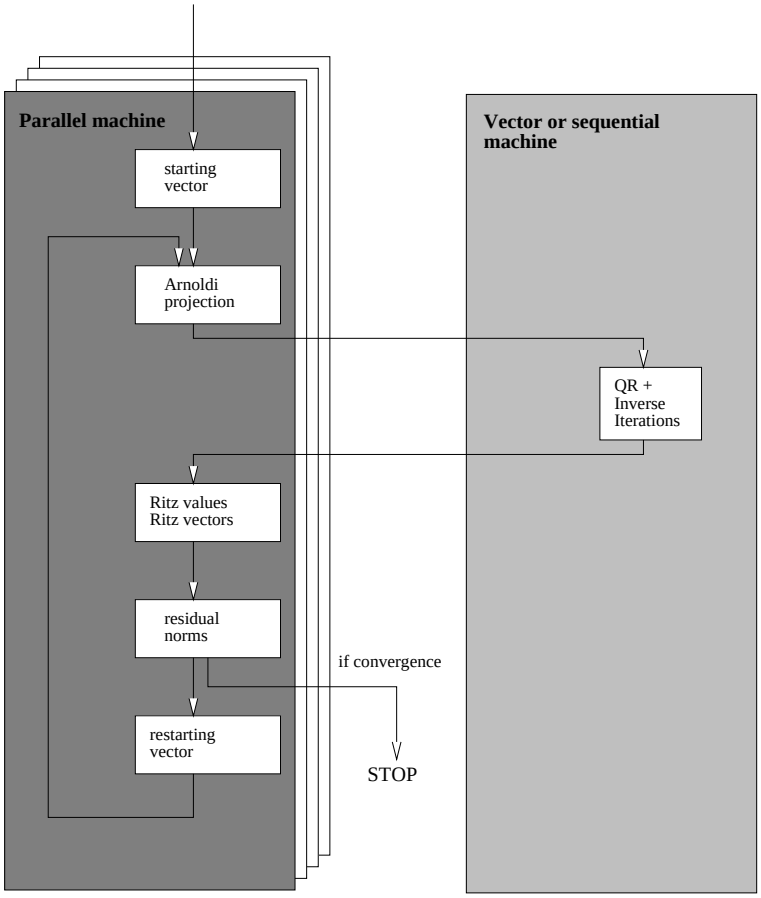

FIG. 1. *ERAM on a distributed architecture.*

by the time needed to resolve the same problem with ERAM (on CM5 alone). The execution time of MERAM is the one of the converged node (the fastest node which is CM5).

We want to find $s = 4$ algebraically largest eigenvalues and their corresponding eigenvectors. The tolerance value is $tol = 5E - 10$.

We expect that MERAM converges faster than ERAM. In fact, for the worst case scenario MERAM behaves like ERAM applied to the fastest processor (i.e., CM5). This is the case when no other processes yield information that is better than the original ERAM process. In other words, since the CM5 is faster than the CM200, the CM200 ERAM process can be viewed as an accelerator for the CM5 ERAM process. Consequently, the number of iterations (restarts) of MERAM on the CM5 ought to be less than the number of iterations of the ERAM on this machine. By the number of iterations of MERAM, we mean the number of iterations of its process on the CM5.

The figures 3 to 7 illustrate the sum of the residual norms (SRN), (4.9) with $\alpha_i = 1$ for $i = 1, s$, with respect to the number of restarts for ERAM and MERAM for $C-$diagonal matrices. They present the number of iterations required by ERAM and MERAM to converge. Table 1 brings together the parameters and the curves of performances presented in figures 3 to 7. This table and all these figures illustrate

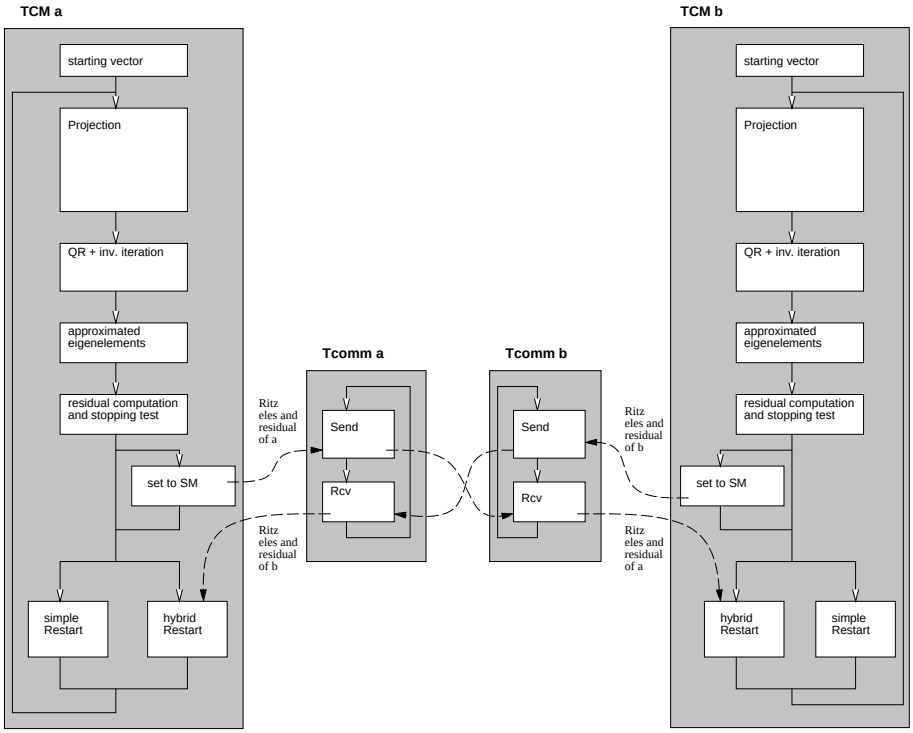

FIG. 2. *The asynchronous MERAM in a heterogeneous environment.*

the influence of the CM200 eigen-information on the improvement of the restarting vector, and consequently the convergence of ERAM process of MERAM on the CM5. We notice that the oscillations of the convergence curves of MERAM are qualitatively and quantitatively less important than the ones of ERAM. According to our experiments, MERAM always converges more rapidly than the ERAM to the desired invariant subspace. These results also show that we can obtain a good acceleration of convergence compared to ERAM.

		MERAM			ERAM			
Fig.	C	m_1, m_2	v^1, v^2	iteration	$m = m_1$	$v = v^1$	iteration	S
3	21	28,15	z, r	80	28	z	120	0.80
4	63	26,26	z, r	42	26	z	54	0.84
5	21	32,32	z, r	50	32	z	80	0.72
6	21	32,20	z, z	56	32	z	80	0.81
7	63	40,40	z, z	63	40	z	148	0.51

TABLE 1
The vector z is defined by $z = \frac{1}{\sqrt{n}}(1, \cdots, 1)^t$ and r *is a normalized random vector.*

In figure 6 there is a section of convergence history where ERAM is better than MERAM. This seems to be due to the distribution of the eigenvalues. Indeed, the s Ritz values computed by two ERAM processes with subspace sizes m_1 and m_2 do not always correspond to s wanted eigenvalues. Suppose for example that the wanted

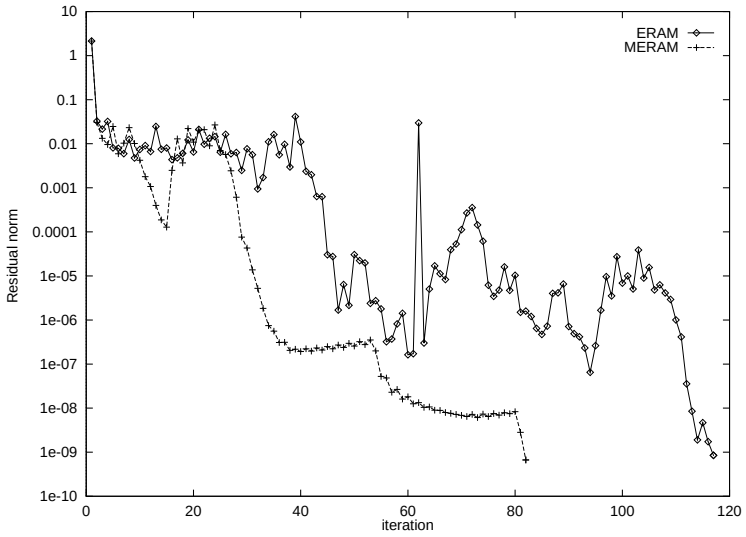

FIG. 3. *MERAM*$((28, 15), [z, \mathrm{r}])$ *versus* *ERAM*$(28, z)$ *with* $21-$*diagonal matrix.*

eigenvalues are $(s = 4)$ algebraically the largest: $\lambda_1 > \lambda_2 > \lambda_3 > \lambda_4 > \lambda_5 > \cdots$ and λ_4, λ_5 are very close to each other. At some iteration, ERAM process with m_1 can compute Ritz values corresponding to λ_1, λ_2, λ_3 and λ_4 while the ERAM process with m_2 can approach Ritz values corresponding to λ_1, λ_2, λ_3 and λ_5. Now, if the residual norm of Ritz vector corresponding to λ_5 is less than the one corresponding to λ_4, then the Ritz vector corresponding to λ_5 will be chosen as "the best". Thus, hybrid restart takes into account a "bad" information to update the restarting vector in MERAM. Consequently, the restarting vector moves away from the wanted invariant subspace. An approach to remedy this issue seems to be to use a mixture of all Ritz vectors instead of "the best" vector. The hybrid restart strategy should make use of *all* Ritz vectors and no computed information will be lost. But this approach brings together all "good" and "bad" Ritz vectors to compute a new restart guess. Thus, the convergence of MERAM with this restarting strategy would be slower than with the restarting strategy integrating just "the best" eigen-information.

6.2. MERAM and other methods. To evaluate MERAM with respect to some other methods, we compared it with ERBAM and IRAM under similar constraints. That means, we tried to know which of MERAM and ERBAM/IRAM will be faster to produce the wanted Ritz elements on a similar amount of memory and processor.

Hardware Platform. Numerical experiments were done on several workstations Sun Ultra Sparc linked by a 10 Mbit/s and 100 Mbit/s Ethernet networks for the experimentation of sections 6.2.1 and 6.2.2 respectively. We made use of two or three Sparc Ultra-1 workstations for the experiments of section 6.2.1. A maximum number of six workstations - three Sparc Ultra-1 and three Sparc Ultra-5-10 - are used for the experiments of section 6.2.2. All of these workstations function with Sun Operating System.

Implementation. We made use of LAKe and MPI message passing libraries to implement ERBAM and MERAM. LAKe (`L`inear `A`lgebra `Ke`rnel) is a linear algebra class library implemented in C++ which uses MPI through OOMPI [16] for the par-

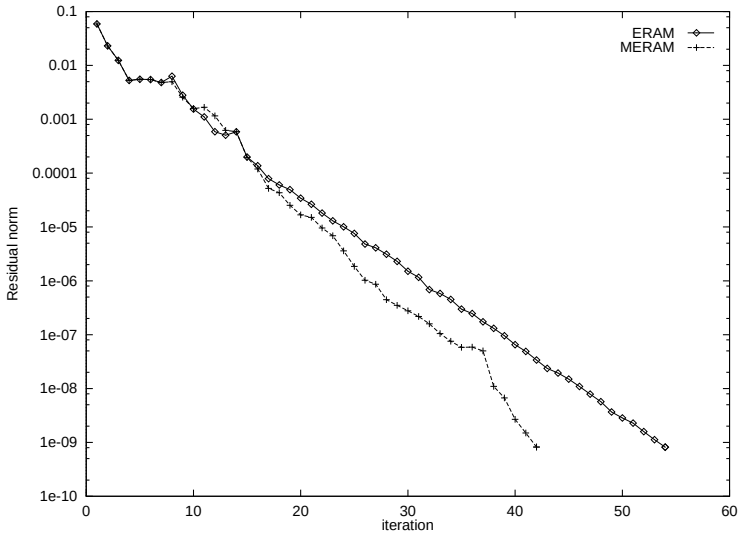

FIG. 4. *MERAM*$((26,26),[z,\mathrm{r}])$ *versus* *ERAM*$(26,z)$ *with* $63-$*diagonal matrix.*

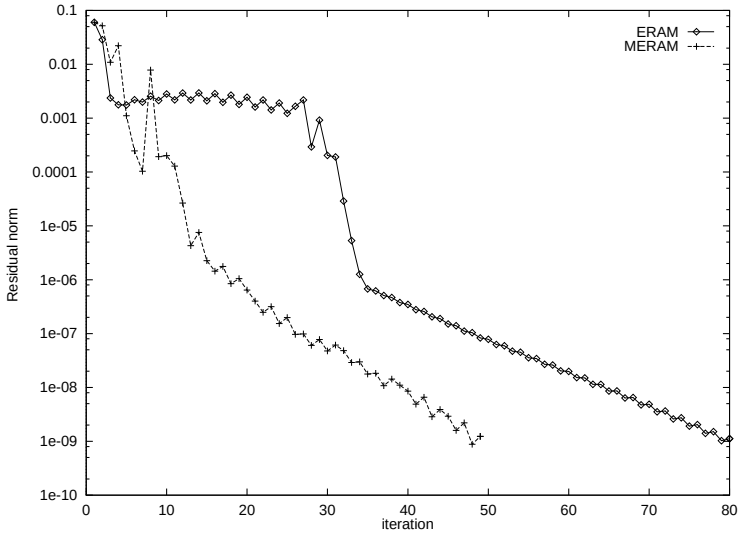

FIG. 5. *MERAM*$((32,32),[z,\mathrm{r}])$ *versus* *ERAM*$(32,z)$ *with* $21-$*diagonal matrix.*

allel classes [14, 15]. We used the PARPACK package [23] to run a parallel version of IRAM. PARPACK is a parallel version of the ARPACK software (a package of Fortran 77 subroutines) which implements IRAM. This package is targeted for distributed memory message passing systems. The message passing layers supported are BLACS and MPI. The reverse communication interface of PARPACK allows a simplified SPMD[10] parallelization strategy.

[10]Single Program Multiple Data

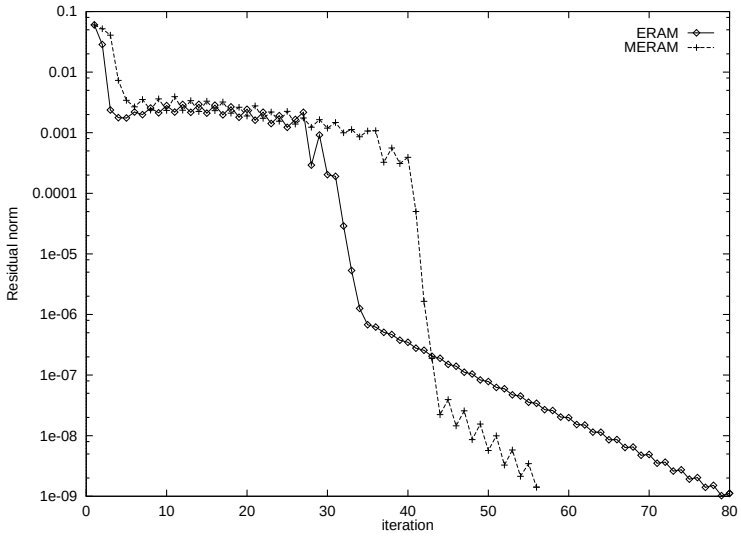

FIG. 6. *MERAM*$((32,20),[z,z])$ *versus* *ERAM*$(32,z)$ *with* $21-$*diagonal matrix.*

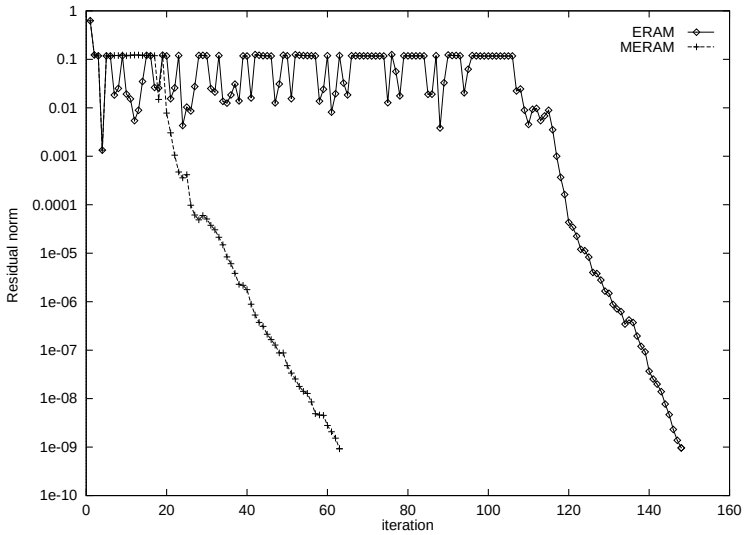

FIG. 7. *MERAM*$((40,40),[z,z])$ *versus* *ERAM*$(40,z)$ *with* $63-$*diagonal matrix.*

Test Matrices. The matrices are taken from the matrix market [2] and the University of Florida sparse matrix collection[11]. We present results obtained with two matrices $AF23560$ and $BFW782A$ of the first collection and the matrix $VENKAT01$ of the second one. The size of the first matrix is 23560 and the number of its nonzero elements is 484256. The size of the second one is 782 and the number of its nonzero elements is 5982. The matrix $VENKAT01$ has 62424 size and the number of its non-zero entrees is 1717792.

[11] http://www.cise.ufl.edu/research/sparse/matrices/

6.2.1. MERAM versus ERBAM. For our experiments we have exploited only very high level parallelism of asynchronous MERAM. We did not take advantage of other types of parallelism that can be found in each ERAM process of MERAM. This means that each ERAM process of MERAM runs in serial mode. The parallelization of ERBAM is done according to the scheme presented in section 5.3. Each of the tasks 1, 3 and 4 of `BBAA` is distributed between the workstations and step 2 runs redundantly on all processors.

We used these methods in order to find the $s = 4$ eigenvalues of largest modulus. The initial vector v^i is taken as the vector e_i (for $i = 1, \ell$). Let $\mathcal{M} = \sum_{i=1}^{\ell} m_i$ be the sum of the subspace sizes of the ℓ ERAM processes of MERAM. The subspace size of ERBAM is chosen as $\ell \times m$, where m is defined by $m = \frac{\mathcal{M}}{\ell} + mod(\ell, mod(\mathcal{M}, \ell))$. The tolerance value is $tol = 1E-6$. The number of iterations of MERAM in figures 8 to 12 is the number of iterations of the ERAM process which reaches convergence (the others ERAM processes are stopped when the first one converges). It is, generally, the ERAM process with the largest subspace size.

We run experimentations with 2 or 3 ERAM in parallel (i.e., $\ell = 2$ or 3). The figures 8 to 12 illustrate Log_{10} of the sum of the residual norms versus the number of restarts and the execution time for ERBAM and MERAM on ℓ workstations. Table 2 brings together the parameters and the curves of performances presented in figures 8 to 12. The figures 8 and 11 show approximately the same behavior for both MERAM and ERBAM. We notice in Figures 9, 10 and 12 an important reduction of MERAM response time with respect to ERBAM. The convergence of MERAM can be seen as the convergence of its ERAM process with the largest subspace size (i.e., ERAM(m_ℓ)). According to our experiments, this ERAM process accelerated by the other ERAM processes of MERAM, is competitive with ERBAM. This is due to different factors: the small subspace size of ERAM (m_ℓ) in comparison to the subspace size of ERBAM (i.e., $\ell \times m$), the mixing of restarting strategies in MERAM, and the use of some additional resources in order to reduce ERAM (m_ℓ) process response time.

			MERAM			ERBAM		
Matrix	Fig.	ℓ	$m_1, \cdots, m_\ell$	iteration	time	$\ell \times m$	iteration	time
AF23560	8	2	10, 30	6	554	2×20	9	487
	9	3	20, 25, 35	5	602	3×27	5	1053
	10	3	27, 27, 27	5	574	3×27	5	1053
BFW782	11	2	13, 37	10	35	2×25	8	36
	12	2	5, 45	4	20	2×25	8	36

TABLE 2
MERAM versus ERBAM on ℓ interconnected workstations

6.2.2. MERAM versus IRAM. The Implicitly Restarted Arnoldi is a more robust method than ERAM. This because of the powerful restarting strategy used in IRAM [22]. As we have seen before MERAM improves the convergence of ERAM. This is because the restarting vector of an ERAM process of MERAM is computed by taking into account the eigen-information obtained by the other ERAM processes. To evaluate the restating strategy of MERAM with respect to IRAM, we have compared these methods under similar constraints.

The PARPACK package[23] implements a parallel version of IRAM according to the message passing model. We run experimentations with this package on a set of p

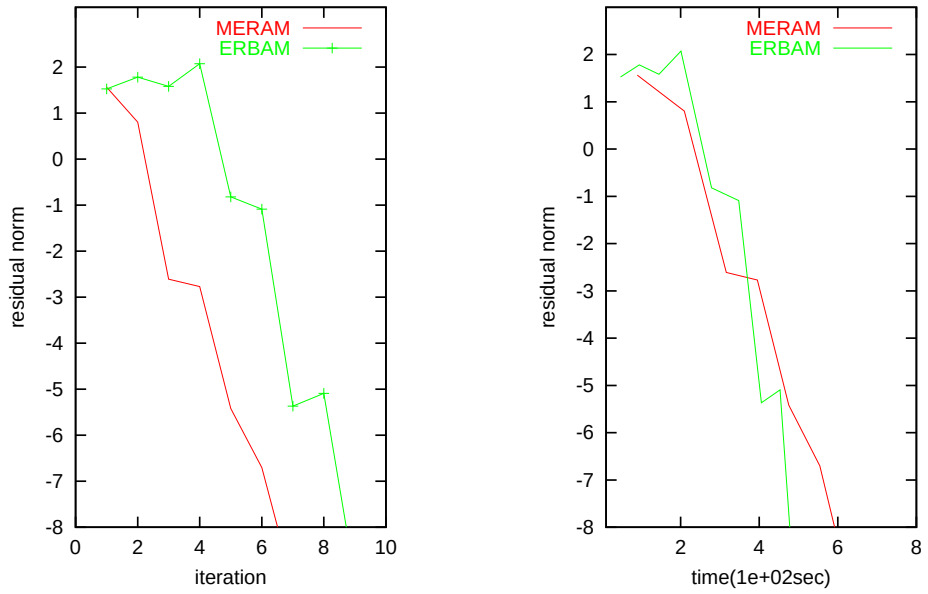

FIG. 8. *MERAM*$((10, 30), I_2)$ *versus* *ERBAM*$(40, I_2)$ *with* *AF*23560 *matrix.*

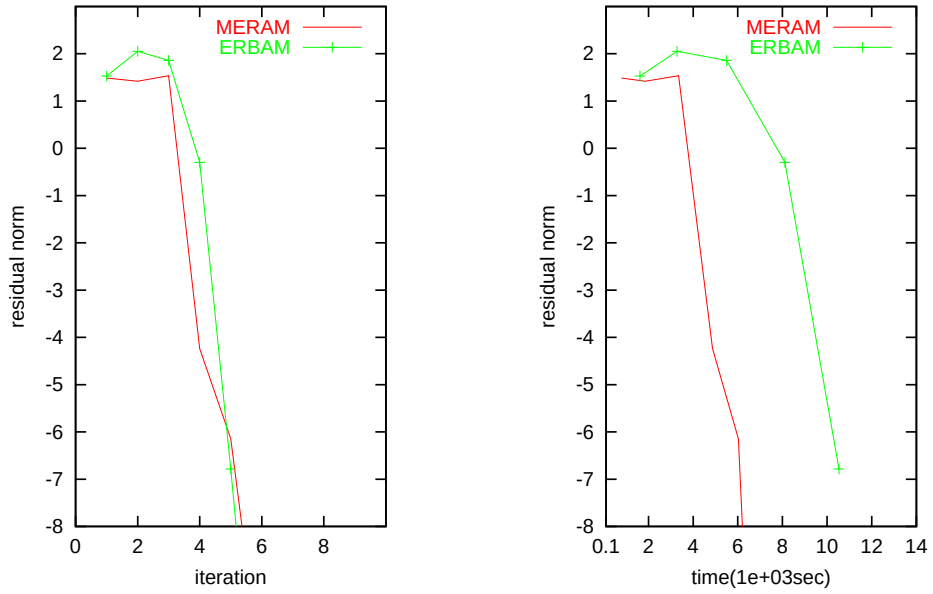

FIG. 9. *MERAM*$((20, 25, 35), I_3)$ *versus* *ERBAM*$(81, I_3)$ *with* *AF*23560 *matrix.*

interconnected workstations.

For our experiments, we have exploited the high level parallelism of asynchronous MERAM (i.e.; that one between its ERAM processes) as well as the parallelism inside each ERAM process. The Multiple Explicitly Restarted Method is implemented according to the model described in section 6.1. That means, we define ℓ compu-

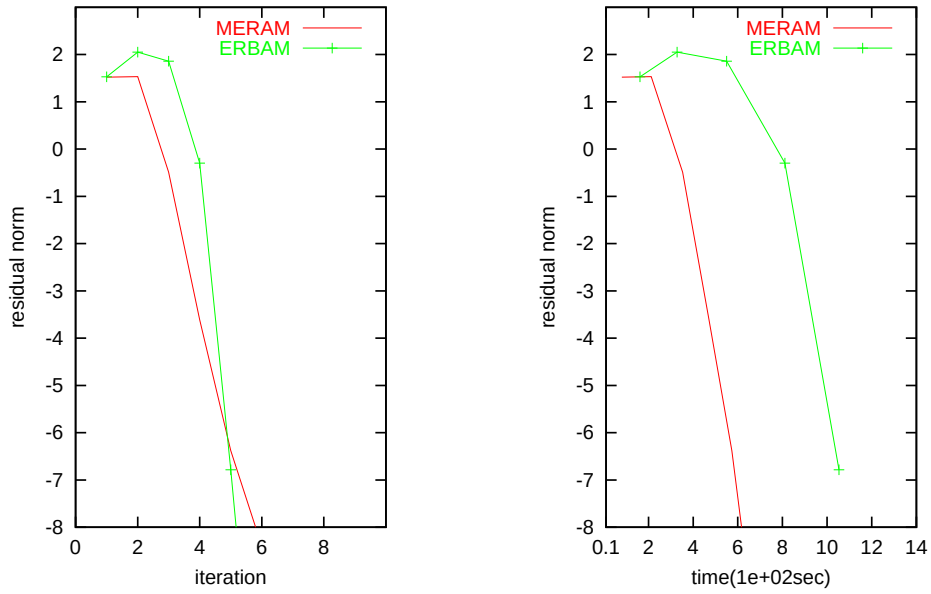

FIG. 10. *MERAM*$((27, 27, 27), I_3)$ *versus* *ERBAM*$(81, I_3)$ *with* $AF23560$ *matrix.*

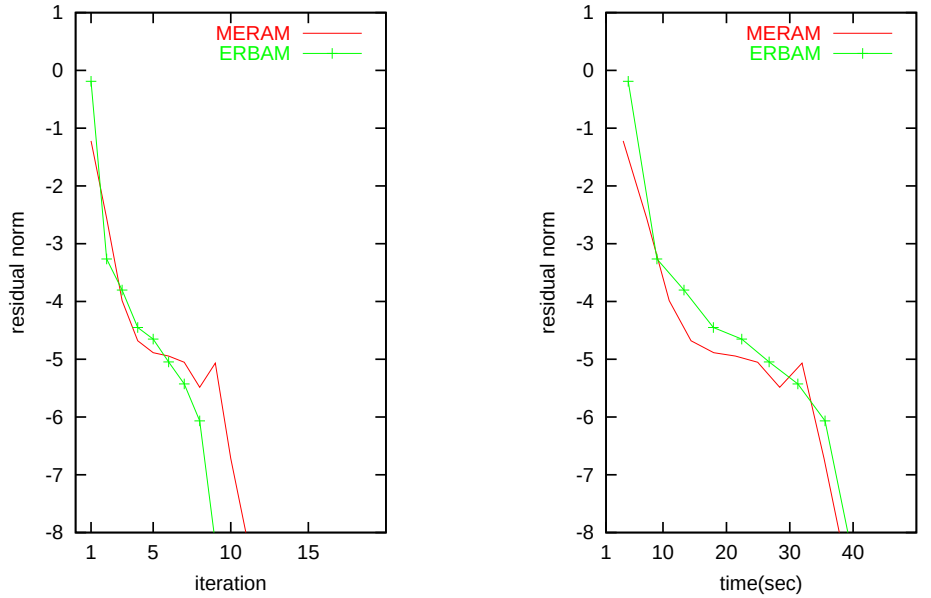

FIG. 11. *MERAM*$((13, 37), I_2)$ *versus* *ERBAM*$(50, I_3)$ *with* $BFW782A$ *matrix.*

tation tasks TCM_i and ℓ communication tasks Tcomm_i (for $i = 1, \ell$). Here TCM_i communicates with Tcomm_i using message passing model. TCM_i implements the ith ERAM process of MERAM. After each iteration, if the convergence is not reached, the eigen-information is sent to Tcomm_i which sends it in its turn to the Tcomm_j tasks (for all $j \neq i$). Moreover, Tcomm_i receives the eigen-information coming from

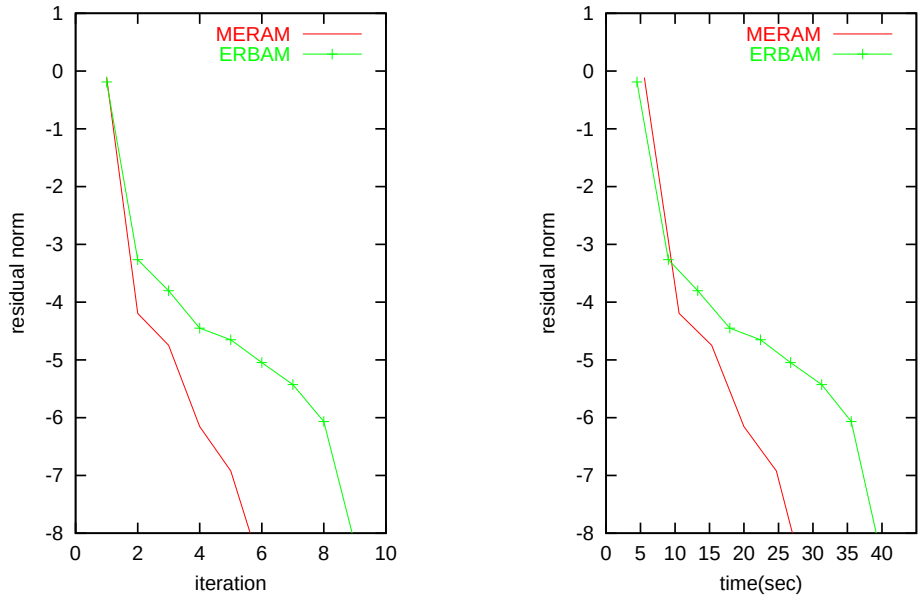

FIG. 12. *MERAM*$((5,45),I_2)$ *versus* *ERBAM*$(50,I_2)$ *with* $BFW782A$ *matrix.*

Tcomm_j $(j \neq i)$ and communicates it to TCM_i.

MERAM is implemented on a set of p interconnected workstations. The TCM_i task is implemented on p_i workstations ; $p_i = 1$ corresponds to a serial run. The subspace computation of TCM_i is replicated to all of the p_i workstations. The Tcomm_i task is implemented on a workstation. We have then $p = (\sum_{i=1}^{\ell} p_i) + \ell$. The MPI library is used to implement the communications between Tcomm_i (for $i = 1, \ell$).

We used these methods in order to find the $s = 2$ eigenvalues of largest modulus. We run experimentations with $p = 5$ or 6 workstations. The maximum number of iterations is taken to be 100 and $tol = 1.e-8$. The figures 13 to 16 illustrate Log_{10} of the sum of the residual norms versus the number of restarts and the execution time for IRAM and MERAM. Table 3 brings together the parameters and the curves of performances presented in figures 13 to 16. We notice in Figures 13, 14 and 15 an relevant reduction of MERAM response time with respect to IRAM. The figure 16 shows that PARPAK is faster than MERAM. This seems to be due to the increment of the charge of the workstation supporting the ERAM process with the smallest subspace size when the matrix size increases. Indeed, this process can be considered as the accelerator of the other ones. As we see in table 3, this ERAM process has run in serial mode on a workstation. When the matrix size is too large with respect to the resources of the workstation, this process can become very slow and would not play its role of accelerator.

7. Conclusion. We have presented the Multiple Explicitly Restarted Arnoldi Method and some of its characteristics : restarting strategy and its relationship to the block version of ERAM. We also presented an asynchronous parallel version of this method and performance results of its implementation on heterogeneous and homogeneous environments. MERAM allows us to restart each of its ERAM processes with a vector preconditioning so that it can be forced in the desired direction. This preconditioning can be seen as a particular application of an ERAM process to the

			MERAM			IRAM		
Matrix	Fig.	p, p_1, p_2	$m_1, m_{\ell=2}$	iteration	time	m	iteration	time
AF23560	13	6, 3, 1	13, 23	9	380	23	100*	900
BFW782	14	5, 2, 1	10, 20	8	22	20	74	63
	15	5, 2, 1	10, 30	5	29	30	42	60
VENKAT	16	6, 3, 1	10, 30	3	399	30	4	191

TABLE 3
MERAM versus IRAM on p workstations.

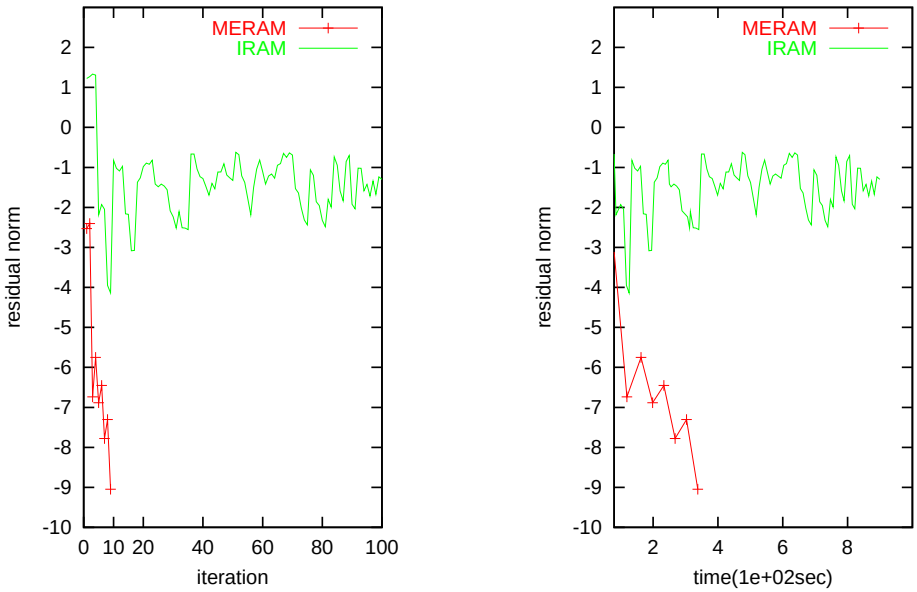

FIG. 13. *MERAM*$((13, 23), I_2)$ *versus IRAM*$(23, e_1)$ *with AF*23560 *matrix.*

starting vector of the other ones.

The restarting strategy is a critical part of MERAM. For an ERAM process of this method, a hybrid restart is defined to be a restarting strategy that takes into account all Ritz vectors including those received from the other ERAM processes. Thus, the restarting vector of this ERAM process is a linear combination of "the best" Ritz vectors of interest. We define "the best" Ritz vector to be the vector that has the smallest residual norm. The approach consisting of mixing of all Ritz vectors instead of choosing only "the best" can seem better. Thus, the updated restarting vector would be a linear combination of all Ritz vectors. But this approach brings together all "good" and "bad" Ritz vectors to update the start guess. Thus, the convergence of MERAM with this restarting strategy would be slower than with the one combining just "the best" wanted Ritz vectors.

Asynchronous MERAM is characterised by the fact that the communications between its subtasks (i.e., ERAM processes) are totally asynchronous. As a consequence, this parallel algorithm is not the parallel version of a unique serial one. *Each run* of this dynamic algorithm corresponds to one parallel static version. MERAM is a fault tolerant method. A loss of an ERAM process during MERAM execution does not

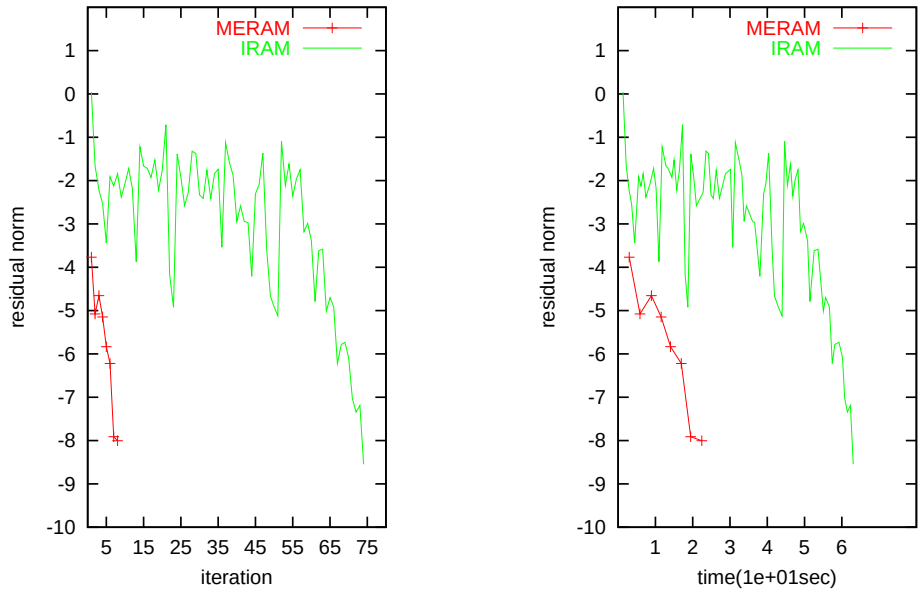

FIG. 14. *MERAM*$((10,20),I_2)$ *versus* *IRAM*$(20,e_1)$ *with* $BFW782A$ *matrix.*

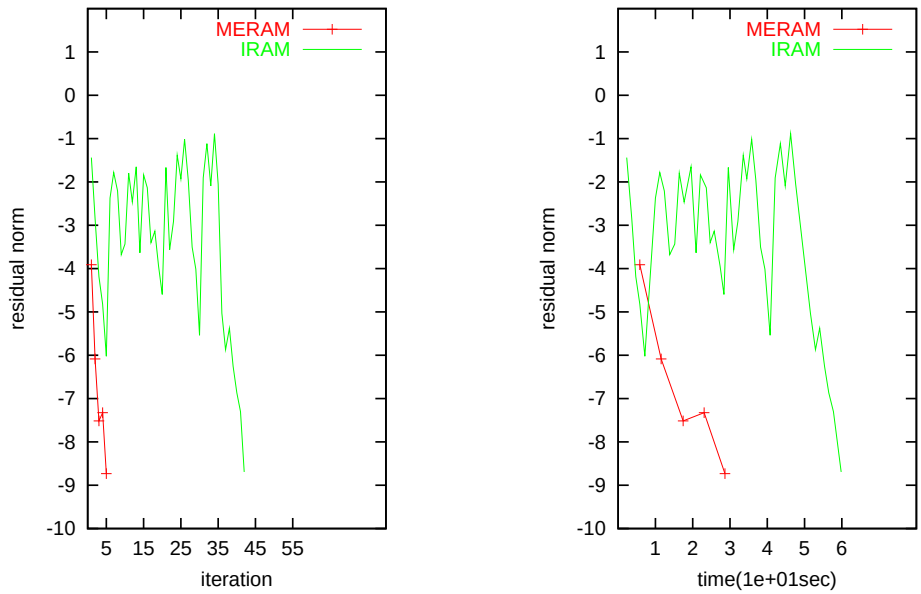

FIG. 15. *MERAM*$((10,30),I_2)$ *versus* *IRAM*$(30,e_1)$ *with* $BFW782A$ *matrix.*

interfere with its termination. Furthermore, this algorithm has a great potential for dynamic load balancing. Indeed, the attribution of ERAM processes of MERAM to the available resources can be done as a function of their subspace size at run time. The heterogeneity of computing supports can be then an optimization factor for this method. All these properties show that MERAM is well suited to the GRID

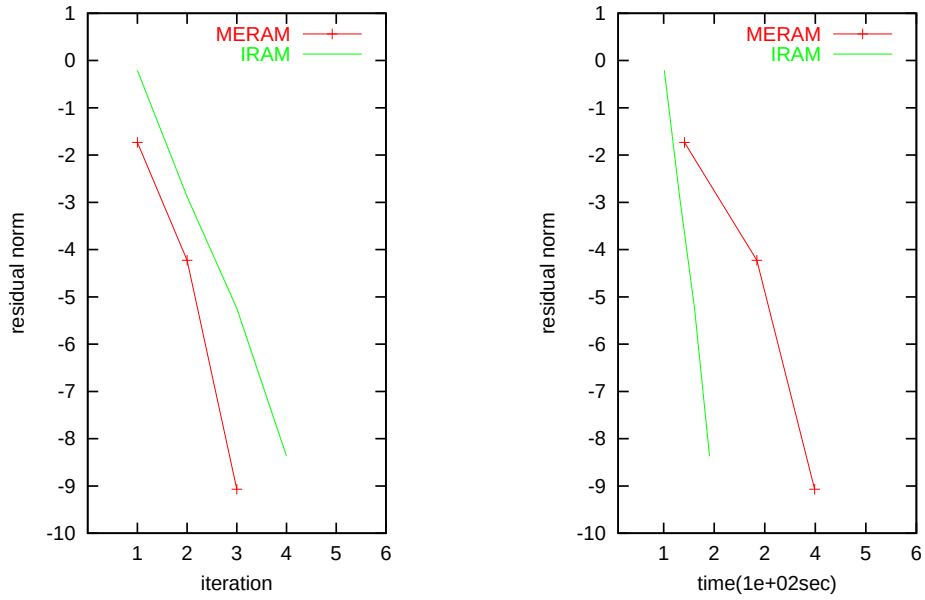

FIG. 16. *MERAM*$((10,30), I_2)$ *versus* *IRAM*$(30, e_1)$ *with* $VENKAT01$ *matrix.*

computational environments.

As we have seen, MERAM accelerates the convergence of ERAM. Furthermore, it can cooperate with convergence acceleration methods similar to the iterative Chebyshev polynomials. The presented results of our experiments on a cluster of workstations pointed out that MERAM has a good convergence behavior compared to Explicitly Restarted Block Arnoldi method. Moreover, for some matrices, one have a difficult convergence (slow or impossible) with Implicitly Restarted Arnoldi Method and a fast convergence with MERAM.

We showed that the multiple-use of ERAM is more interesting than its simple-use. This concept is simple and can be extended to some other restarted projection methods like IRAM, GMRES or Hermitian/non-Hermitian Lanczos methods.

There are still many open problems in these methods which involve interesting work from theoretical point of view as well as practical computation. Some potential improvements to restarting techniques should be considered in future works. Approaches based on sophisticated restart such as augmented Krylov or implicit restarting are also under investigation.

REFERENCES

[1] W. E. ARNOLDI, *The Principle of Minimized Iteration in the Solution of Matrix Eigenvalue Problems*, Quart. Appl. Math., 9:pp. 17-29, 1951.

[2] Z. BAI, D. DAY, J. DEMMEL AND J.J. DONGARRA, *A Test Matrix Collection for Non-Hermitian Eigenvalue Problems*, http://math.nist.gov/MatrixMarket.

[3] F. CHATELIN, *Valeurs Propres de Matrices*, Masson, Paris, 1988.

[4] C. BRÉZINSKI AND M. R. ZAGLIA, *A hybrid procedure for solving linear systems*, Numerische Mathematik, 67, pp. 1-19, 1994.

[5] J.J. DONGARRA, I.S. DUFF, D.C. SORENSEN AND H.A. VAN DER VORST, *Numerical Linear Algebra for High-Performance Computers*, Siam, Philadelphia, 1998.

[6] G. Edjlali, N. Emad and S. Petiton, *Hybrid Methods on Network of Heterogeneous Parallel Computers*. Proceedings of the 14th IMACS International Symposiom on Iterative Methods in Linear Algebra World Congress, IMACS World Congress, Atlanta, USA July 1994.

[7] G. Edjlali, S. Petiton and N. Emad, *Interleaved Parallel Hybrid Arnoldi Method for a Parallel Machine and a Network of Workstations.* Conference on Information, Systems, Analysis and Synthesis (ISAS'96), 22-26 July 1996, Orlando, Florida, USA.

[8] G. Edjlali, *Contribution à la parallélisation des méthodes itératives hybrides pour matrice creuse sur architectures hétérogènes*, PhD of the University of Pierre et Marie Curie (Paris VI) in french, 1994.

[9] G. Golub and C. Van Loan, *Matrix Computation*, The John Hopkins University Press, 2nd ed., 1989.

[10] The GRID Blueprint for a New Computing Infrastructure. Edited by Ian Foster and Carl Kesselman. Morgan Kaufmann Publishers, Inc., 1999.

[11] R. Lehoucq, D.C. Sorensen and P.A Vu, *ARPACK: Fortran subroutines for solving large scale eigenvalue problems, Release 2.1*, available form `netlibornl.gov` in the `scalapack` directory, 1994.

[12] R. B. Morgan, *On restarting the Arnoldi method for large nonsymmetric eigenvalue problems*, Math. Comput., 65, pp. 1213-1230, 1996.

[13] R. B. Morgan and M. Zeng, *A Harmonic Restarted Arnoldi Algorithm for Calculating Eigenvalues and Outlining the Spectrum of a Large Matrix*, Technical Report (http://www.baylor.edu/Ronald_Morgan/reports#spec)

[14] E. Noulard, *Object-Oriented Parallel Programming and Reutilisability applied to Linear Algebra*, PhD thesis, Versailles University, France, 2000.

[15] E. Noulard and N. Emad , *A Key for Reusable Parallel Linear Algebra Software*, Parallel Computing Journal, Elsevier Science B.V. Vol. 27 (10), 1-4, pp. 1299–1319, 2001.

[16] P.W. Rijks, J.M. Squyres and A. Lumsdaine, *Object-Oriented MPI (OOMPI) version 1.0.2g*, Tech. Rep. TR-99-14, University of Notre Dame - Department of Computer Science, 1999.

[17] S. Petiton, *Parallel Subspace Method for non-Hermitian Eigenproblems on the Connection Machine*, Applied Numerical Mathematics Journal 10, North-Holland, 1992.

[18] S. Petiton and N. Emad, *A Data Parallel Scientific Computing Introduction*, The Data Parallel Programming Model, Springer Verlag, eds: G.-R. Perrin et A. Darte, Vol. LNCS 1132, 1996.

[19] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, 1993.

[20] ———, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Math. Com., 42, 567-588, 1994.

[21] ———, *Variations on Arnoldi's Method for Computing Eigenelements of Large Unsymmetric Matrices*, Linear Algebra Applications, 34, 269-295, 1980.

[22] D.C. Sorensen, *Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations*, In D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds. *Parallel Numerical Algorithms*, pages 119-166, Dordrecht, 1997, Kluwer.

[23] K. J. Maschhoff and D. C. Sorensen, *P_ARPACK: An Efficient Portable Large Scale Eigenvalue Package for Distributed Memory Parallel Architectures*, In the Proceeding of PARA'96, 1996.

[24] G.W. Stewart, *An Arnoldi-Schur Algorithm for Large Eigenproblems*, Technical Report UMIACS TR-2000-21, University of Maryland, College Park, MD, 2000.

[25] H. D. Simon and L. Dagum, *Experince in using SIMD and MIMD Parallelism for Computational Fluid Dynamics*, Applied Numerical Mathematics 12, 431-442, North-Holland, 1993.

[26] Thinking Machine Corperation, *Connection Machine 5*, 1992.

[27] Thinking Machine Corperation, *CM Fortran Programming Guide*, 1992.

[28] G. L. G. Sleijpen, H. A. Van der Vorst and Z. Bai, *Jacobi-Davidson Algorithms for Various Eigenproblems*, Preprint 1114, Dept. of Math., Utrecht University, August 1999 (http://www.math.ruu.nl/people/vorst/publ.html#jacobi).

[29] K. Wu and H. Simon, *Thick Restart Lanczos method for Symmetric Eigenvalue Problems*, Technical Report, 41412, Lawrence Berkeley National Laboratory, Berkeley, CA, 1988.