SIAM J. SCI. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 27, No. 1, pp. 253–277

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 the Explicitly Restarted Arnoldi method (ERAM) and improves its convergence.

This technique is implemented and tested on a distributed environment consisting of two interconnected parallel machines. The MERAM technique is compared with ERAM, and one can notice that the convergence is improved. In some cases, more than a twofold improvement can be seen in MERAM results.

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

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

AMS subject classifications. 65F15, 65F50, 65Y05

DOI. 10.1137/S1064827500366082

1. Introduction.

Hybrid methods were proposed during the last decade to accelerate the convergence and/or 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 method introduced by Brézinski and Redivo-Zagila [4], some eigenproblems with the Arnoldi–Chebyshev method proposed by Saad [20, 19, 3], and Jacobi–Davidson algorithms from Sleijpen and Van der Vorst [30, 29].

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 the 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 (ERBAM). Each ERAM invocation is done with a different set of parameter values. This approach also may be applied to some other restarted projection methods.

We propose an asynchronous parallel algorithm for MERAM and show that some of its properties such as asynchronous communications between its coarse grain subtasks, fault tolerance, and dynamic load balancing make this method well adapted

[*]Received by the editors January 13, 2000; accepted for publication (in revised form) October 26, 2004; published electronically September 12, 2005. The two first authors were supported in part by French ACI-GRID ANCG and CGP2P Projects.
http://www.siam.org/journals/sisc/27-1/36608.html

[†]Laboratoire PRiSM, Université Versailles St-Quentin, 45, av. des États-Unis, 78035 Versailles, France (emad@prism.uvsq.fr).
[‡]Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Cité Scientifique, 59655 Villeneuve d'Ascq, France (petiton@lifl.fr).
[§]Google, 1600 Amphitheater Parkway, Mountain View, CA 94043 (edjlali@google.com).

to the GRID computational environments. Our experiments ran on two different platforms—a network of parallel machines and a cluster of workstations—and showed that MERAM converges quicker than ERAM and ERBAM. Moreover, according to our experiments, for some matrices, one has a slow convergence with the Implicitly Restarted Arnoldi method (IRAM) and a faster 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 run time. For example, a dedicated parallel machine can take advantage of the availability of a cluster of nondedicated processors to speed up its convergence. We implemented our approach on a Connection Machine 5 (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 with our 32-node CM5 parallel machine. In this paper, we present results when multiple parallel machines are available for computation. In particular, we show that in some cases we can achieve more than twofold improvement in the number of iterations before convergence by combining the results from a faster and a slower parallel machine. We also will present certain results of comparison of MERAM with the Explicitly Restarted Block Arnoldi and Implicitly Restarted Arnoldi methods on a cluster of workstations.

Before defining MERAM 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. MERAM and an asynchronous algorithm of this method are presented in 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 ERBAM. 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. Notation.

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

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

and X is spanned by a set of μ vectors. The point (resp., block) Krylov subspace methods are characterized 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 $X = \mathrm{Span}(v)$. Let W_m be the matrix whose columns $w_1, \ldots, w_m$ 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 a basis, and the most used process is the Arnoldi 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 $w_1, \dots, w_m$ of Krylov subspace $\mathbb{K}_{m,v}$ by using the Gram–Schmidt orthogonalization process:

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

    For $j = 1, 2, \dots, m$ do:
      • $h_{i,j} = (Aw_j, w_i)$, for $i = 1, 2, \dots, j$
      • $\bar{w}_j = Aw_j - \sum_{i=1}^{j} h_{i,j} w_i$
      • $h_{j+1,j} = \|\bar{w}_j\|_2$
      • $w_{j+1} = \bar{w}_j / h_{j+1,j}$
```

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 Arnoldi [1] in 1951 to reduce a matrix to a Hessenberg form. The reduced matrix $H_m = (h_{i,j})$ is a 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 $m < n$.

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 mth 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 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) $\Lambda_m = (\lambda_1^{(m)}, \dots, \lambda_s^{(m)})$ and the corresponding Ritz vectors $U_m = (u_1^{(m)}, \dots, u_s^{(m)})$ can be calculated as follows:[1]

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

    1. Compute an 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 $r_s = (\rho_1, \dots, \rho_s)$ 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 reindexed 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 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 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 a BAA (Basic Arnoldi Algorithm). 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 method (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 $\Lambda_m$ and $U_m$ to update the starting vector
       $v$ and go to 2.
```

In the previous algorithm, *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 a restarting technique is proposed by Sorensen [23]. This refined version, called the 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 [23, 11].

Several other versions of the Arnoldi method based on the augmented Krylov methods exist, including variants proposed by Wu and Simon [31], Stewart [25], Morgan [12], and Morgan and Zeng [13].

4. Multiple Explicitly Restarted Arnoldi method.

The method we propose is characterized by a new restarting technique for the Arnoldi method. Our restarting technique combines the two explicit algorithms described above. 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 $I_m = [m_{min}, m_{max}]$. The lower bound m_{min} and upper bound m_{max} may be chosen as a function of the computation and storage resources. In addition, the constraint $m_{min} \leq m_{max} \ll n$ has to be fulfilled. Let $M = (m_1, \ldots, m_\ell)$ be a set of subspace sizes with $m_i \leq \cdots \leq m_\ell$ and $m_i \in I_m$ $(1 \leq i \leq \ell)$, and let $V^\ell = [v^1, \ldots, v^\ell]$ 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($input: A, s, M, V^\ell, tol; output: r_s^m, \Lambda_m, U_m$).
    1. Start.  Choose a starting matrix $V^\ell$ and a set of subspace sizes $M = (m_1, \dots, m_\ell)$.
    2. Iterate. For $i = 1, \dots, \ell$ 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 all processes.
    3. Restart. Update the vectors $v^1, \dots, v^\ell$ and go to 2.
```

Note that r_s^i is the vector of the residual norms at the ith iteration.

We suppose that

$$(\lambda_j^{(m_p)}, u_j^{(m_p)}) \text{ is better than } (\lambda_j^{(m_q)}, u_j^{(m_q)}) \text{ if } \rho_j^p \leq \rho_j^q. \tag{4.1}$$

Then an interesting updating strategy would be the choice of v^i as a function f^i of "the best" Ritz vectors:

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

where $U^{best} = (u_1^{best}, \dots, u_s^{best})$ and u_j^{best} is "the best" jth 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 $u_1^{best}, \dots, u_s^{best}$ vectors. In order to ensure that the BAA processes used in the MERAM algorithm produce ℓ different Krylov subspaces, $(f^i)_{i=1,\dots,\ell}$ 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, (4.2) may become

$$v^i = f(U^{best}), \tag{4.3}$$

where $f = f^i$ (for $i = 1, \dots, \ell$). The vectors $v^1, \dots, v^\ell$ would become identical and the Krylov subspaces of different processes would look extremely close to one another after a few iterations. Meanwhile, we will see in the next section that this restarting strategy can be useful for the asynchronous version of MERAM.

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 into account interesting eigeninformation obtained by the other ERAMs. Another advantage of this algorithm is that it can be parallelized 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 synchronize at step 3. The only drawback of this parallelization 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, the process sends its output of interest to all other processes. The computation of a restarting vector v^i on process (i) can then 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.

We refer to a dynamic algorithm as an algorithm whose one or more components are dynamic and to a dynamic component as a component whose run time behavior can depend on outside events. The parallel asynchronous MERAM can be considered 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 (4.2) are different for every ERAM process. That means that 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, to ensure 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 asynchronizing communications between the BAA processes allows MERAM to produce ℓ *different* Krylov subspaces even with the restarting strategy (4.3).[2]

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 eigeninformation of interest from an ERAM process to all other ERAM processes. Let `Receiv_Eigen_Info` be the task of receiving eigeninformation of interest from one or more ERAM processes by the current ERAM process. Finally, let `Rcv_Eigen_Info` be a boolean variable that is true if the current ERAM process has received eigeninformation from the other ERAM processes. An asynchronous parallel version of MERAM is explained in the following:

```
Asynchronous MERAM Algorithm.
    1. Start.  Choose a starting matrix $V^\ell$ and a set of subspace sizes $M = (m_1, \dots, m_\ell)$.
    2. Iterate. For $i = 1, \dots, \ell$ do in parallel (ERAM process):
        • Computation 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
            (d) Send_Eigen_Info
            (e) 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 eigeninformation computed by the current ERAM process. A `hybrid restart` is a restarting strategy taking into account the eigeninformation received by the other ERAM processes also. A hybrid restart includes data from other processes when eigeninformation has been received, while the simple restart uses just the eigeninformation computed by the current process. Let $u_j^{(m_k)}$ be the jth Ritz vector computed by the kth process (corresponding to the one with the subspace size m_k) and received by the current process. The matrix U^{best} of (4.2) is then defined by $(u_1^{best}, \dots, u_s^{best})$, where u_j^{best} is "the best" of $u_j^{(m_k)}$ and the jth Ritz vector computed by the current process, where $k \in F \subset [1, \ell]$, cardinal $(F) = np$, and np is the number of processes that sent their eigeninformation 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. Moreover,

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

when the initial vector is updated by the hybrid restart strategy (i.e., step 2(c)) the current ERAM process changes the flag `Rcv_Eigen_Info` to false. Once a new Ritz pair is received from a process, this flag changes to true by the current ERAM process. We notice that in addition to the coarse grain parallelism between ℓ ERAM processes, one can also overlap communication steps with computations. More details about the parallelization of this algorithm are 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 that the loss of an ERAM process at run time doesn't impede the other ERAM processes from continuing to run and finish the algorithm. Another important property of this algorithm is its capacity of dynamic load balancing. Indeed, the time and space complexities of an iteration of an ERAM process of MERAM is a function of its subspace size. Because these complexities are known, we can dynamically allocate ERAM processes according to the system work load. All of 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 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:

$$v(k) = p(A)v, \tag{4.4}$$

where $v(k)$ is the kth 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:

$$v(k) = \sum_{i=1}^{s} \alpha_i u_i^{(m)}(k), \tag{4.5}$$

where $u_i^{(m)}(k)$ denotes the ith 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 (4.5). 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 $v = \sum_{j=1}^{n} \gamma_j u_j$, IRAM with exact shifts [23, 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 eigeninformation of interest from large Krylov subspaces while avoiding the storage and numerical difficulties. This is done by continually compressing eigeninformation 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 (4.2). These functions with several variables have to be used as input for MERAM. Their variables are calculated dynamically in the case of the asynchronous MERAM. These functions constitute a very important 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 based on the one defined in (4.5) and we call it the *hybrid restarting strategy*. According to hypothesis (4.1) this hybrid restarting strategy is defined by

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

where α_j^i (for $j = 1, s$ and $i = 1, \ell$) are some scalar values. The hybrid restarting strategy corresponding to (4.3) is then

$$v^i = f(U^{best}) = \sum_{j=1}^{s} \alpha_j u_j^{best}, \tag{4.7}$$

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 (4.7). 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 (4.5).

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

$$\rho_i = \|(A - \lambda_i^{(m)} I) u_i^{(m)}\|_2 = h_{m+1,m} |e_m^H y_i^{(m)}|, \tag{4.8}$$

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

To stop the restarts in ERAM algorithm one has to define the function g given in step 3. Some typical examples are

$$g(r_s) = \|r_s\|_\infty \tag{4.9}$$

or a linear combination of the residual norms

$$g(r_s) = \sum_{j=1}^{s} \alpha_j \rho_j, \tag{4.10}$$

where α_j are scalar values.[3] The same technique applied to $r_s^i = (\rho_1^i, \ldots, \rho_s^i)$ 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 an eigenpair (λ, u) of A by a $(\lambda^{(m)}, u^{(m)})$ solution of (2.2) with $\mathbb{K}_{m,X}$ an $m \times \mu$-dimensional subspace defined by (2.1), where $X = \mathrm{Span}(x_1, \ldots, x_\mu)$ and $\mu > 1$. We assume that the vectors $x_1, \ldots, x_\mu$ 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 [9]. Let X_1 be the orthonormal matrix represented by the columns $x_1, \ldots, x_\mu$. The block Arnoldi process generates a set of $X_1, \ldots, X_m$ matrices. Let $\boldsymbol{W}_m$ be the $(n, m \times \mu)$-sized matrix $[X_1, \ldots, X_m]$. 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 $j = 1, 2, \ldots, m$ do:
  • $H_{i,j} = X_i^H A X_j$, for $i = 1, 2, \ldots, j$.
  • $\bar{X}_j = AX_j - \sum_{i=1}^{j} X_i H_{i,j}$.
  • 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 equation equivalent to (3.1) will be

$$A\boldsymbol{W}_m = \boldsymbol{W}_m \boldsymbol{H}_m + F_m \boldsymbol{E}_m^H, \tag{4.11}$$

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 $\mathbb{K}_{m,X}$ with $X = \mathrm{Span}(x_1, \ldots, x_\ell)$, while in MERAM the problem is projected onto ℓ subspaces $\mathbb{K}_{m,x_1}, \ldots, \mathbb{K}_{m,x_\ell}$. Now, assuming that the degree of the minimal polynomial of X_1 is greater than m, these Krylov subspaces satisfy the following relation:

$$\mathbb{K}_{m,X} = \mathbb{K}_{m,x_1} \oplus \cdots \oplus \mathbb{K}_{m,x_\ell}. \tag{4.12}$$

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 $((x_1^1, \ldots, x_\ell^1), \ldots, (x_1^m, \ldots, x_\ell^m))$ of $\mathbb{K}_{m,X_1}$ allows the construction of an orthogonal

[3]If $\alpha_j = 1$ (for $j = 1, \ldots, s$), then (4.10) defines the sum of the residual norms associated with s desired eigenelements.

basis $\tilde{X}_i = (x_i^1, \ldots, x_i^m)$ for each subspace $\mathbb{K}_{m,x_i}$, $1 \leq i \leq \ell$. 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 $\mathbb{K}_{m,x_i}$ $(1 \leq i \leq \ell)$, but the inverse is not always true.

In the first method, s desired eigenvalues are approximated by the eigenvalues of an $(\ell \times m)$-sized block Hessenberg matrix, while in the second, they are approximated by s "best" eigenvalues of ℓ Hessenberg matrices of order m. We also notice that in ERBAM the relation $m \geq s/\ell$ has to be true, while in MERAM the relation $m \geq s$ 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, Λ_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 phases deal with the n- and m-sized data sets.[4] 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 assumes 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 the 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 redundantly on all processors. This is equivalent to distributing each of the tasks 1, 3, and 4 among 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[5], 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 the message passing model for step 2, as this task would become a bottleneck in the algorithm [17]. This means, in terms of implementation, that the architecture used for step 2 ought to be different from the one used for steps 1, 3, and 4 (see Figure 1).

In the 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].

[4]This is not true for the fourth step if the right-hand side of (4.8) is used to compute the vector r_s.

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

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($input: A, s, m, v; output: r_s, \Lambda_m, U_m$)—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 the above described parallelism.

In addition to the coarse grain parallelism between ℓ ERAM processes, one can also overlap communication steps with computations. The processors would always be active and execute a nonredundant 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 parallelization 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) computing $\mathcal{M} = m \times \mu$ matrix-vector products with A, while in the 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 computational cost of ERBAM should be less than that of ERAM.

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.

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 $i = 1, \dots, \mu - 1$) 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 metacomputing platform. A metacomputer 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 metacomputer is limited. We present, in this section, some results of our experiments on homogenous and heterogeneous 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 $M = (m_1, \ldots, m_\ell)$ and $V^\ell = [v^1, \ldots, v^\ell]$. The matrix I_k denotes the one whose columns are $e_1, \ldots, e_k$.

6.1. MERAM and ERAM.

Hardware 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), a CM5 and a Sparc 20, a CM5 and two Sparc 20's, and a CM5 and a 4K Connection Machine 200 (CM200). The CM5 is a distributed memory MIMD[6] architecture [27]. Each node of CM5 consists of a Sparc processor, a 32 megabyte memory, and four floating point computation units operating at 32 megaflops each. The CM5 supports the message passing model, the 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 communication and has 0.5 gigabyte 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[7] machine consisting of thousands of bit serial data processors and a front-end for management and control [26]. 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 gigabyte memory and operates at 10 megahertz. The CM200 supports the data-parallel programming model. Both CM5 and CM200 have a workstation as front-end. The programming language used on CM5 and CM200 is CM Fortran [28]. This is an extension of Fortran 77 allowing us 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 two 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 10 Mbit/s Ethernet link for MERAM.

Implementation. As we have seen in the previous section, the BA algorithm can be divided into 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 in 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 the shared memory model. Tcomma and Tcommb are connected by a network. These communication tasks are designed to overlap

[6]Multiple instruction multiple data.

[7]Single instruction multiple data.

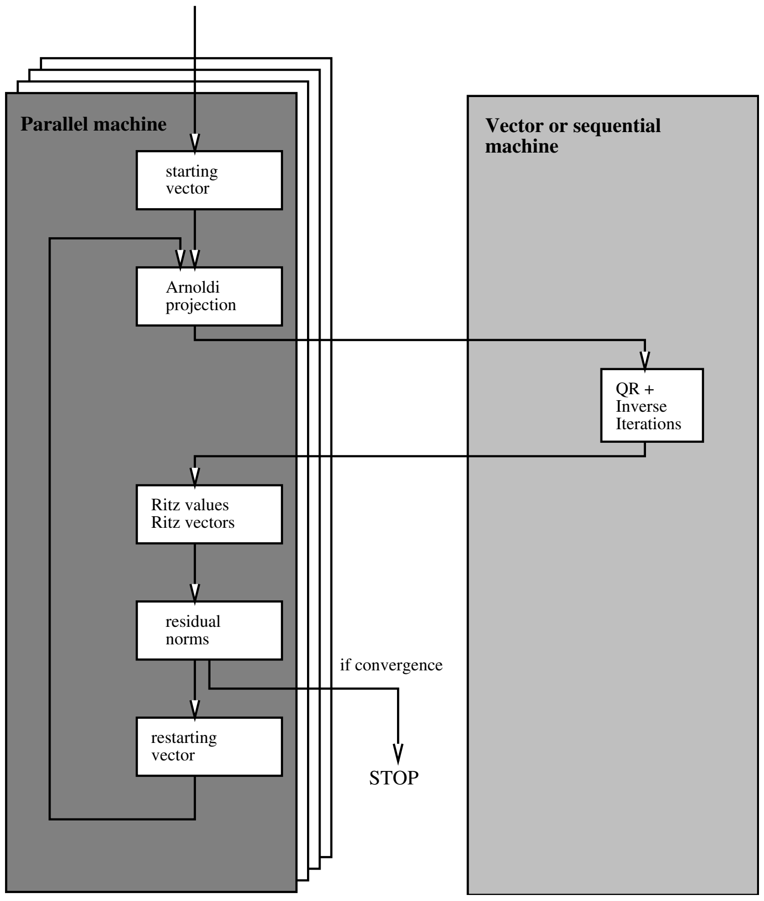

FIG. 1. *ERAM on a distributed architecture.*

communication with computation phases between parallel machines. TCMa (resp., TCMb) implements an ERAM process. After each iteration, if the convergence is not reached, the eigeninformation 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 10 Mbit/s Ethernet link. The subspace computations 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 nonzero elements of a C-diagonal matrix are on the main diagonal and the diagonals immediately around it. C is the maximum number of nonzero elements per row or per column. These sparse matrices have some interesting properties. They are easy to build and well suited for the sparse general pattern (SGP) data-parallel storage format [18]. It has also been shown in [8] that matrices with a C-diagonal pattern have good performance in sparse matrix-vector multiplication algorithms executed on data-parallel machines. The nonzero upper-diagonal and lower-diagonal elements of our nonsymmetric matrices are chosen in $[-1, +1]$ range, while the nonzero diagonal elements are in $[0, C]$. To study the convergence properties of MERAM and

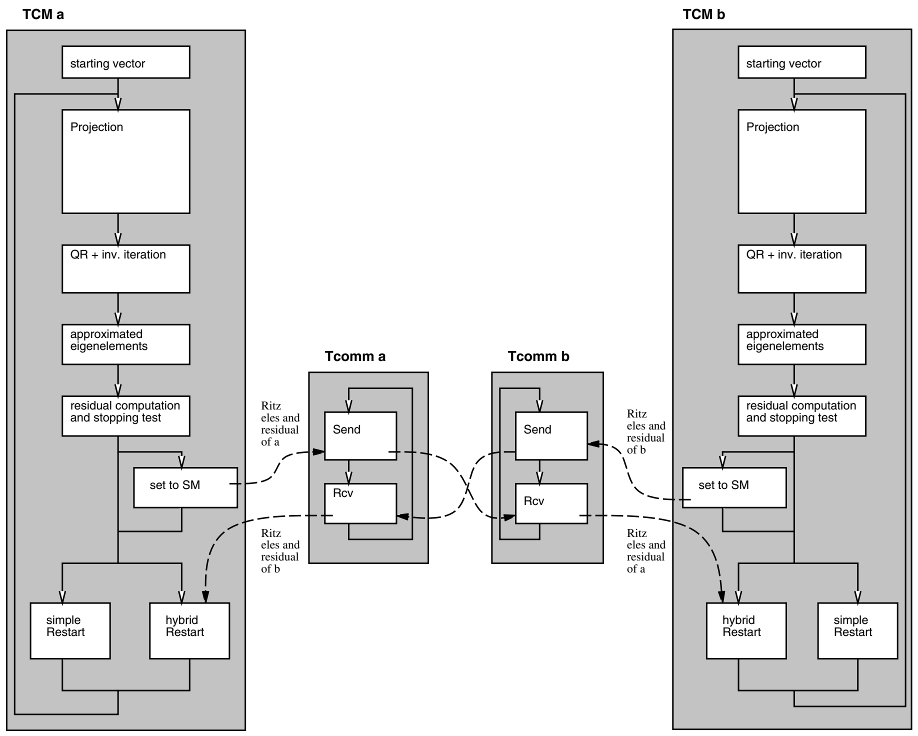
TCM a
starting vector
Projection
QR + inv. iteration
approximated eigenelements
residual computation and stopping test
set to SM
simple Restart
hybrid Restart
Tcomm a
Send
Rcv
Ritz eles and residual of a
Ritz eles and residual of b
Tcomm b
Send
Rcv
Ritz eles and residual of b
Ritz eles and residual of a
TCM b
starting vector
Projection
QR + inv. iteration
approximated eigenelements
residual computation and stopping test
set to SM
hybrid Restart
simple Restart

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

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 the starting vector equal to v, the subspace size m, and the restarting strategy (4.7) with $\alpha_i = 1$ (for $i = 1, s$). On the other hand, 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 eigeninformation $E_{m_1} = (\Lambda_{m_1}, U_{m_1}, r_s^1)$ and $E_{m_2} = (\Lambda_{m_2}, U_{m_2}, r_s^2)$. We choose "the best" to form a new set of eigeninformation: $E_{best} = (\Lambda^{best}, U^{best}, r_s^{best})$. Then, with these s Ritz vectors, we compute the restarting vector by (4.7) 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) to the time needed to resolve the same problem with ERAM (on CM5 alone). The execution time of MERAM is that 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

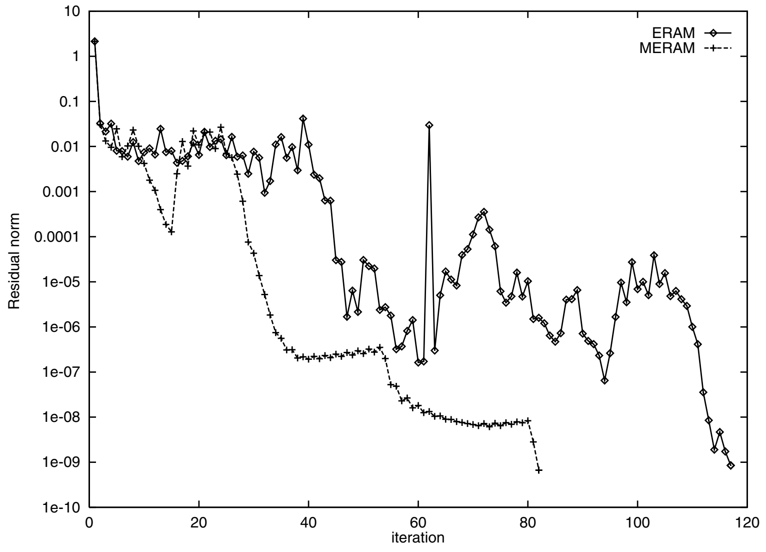

FIG. 3. *MERAM*$((28,15),[z,\mathrm{r}])$ *versus ERAM*$(28,z)$ *with* 21*-diagonal matrix.*

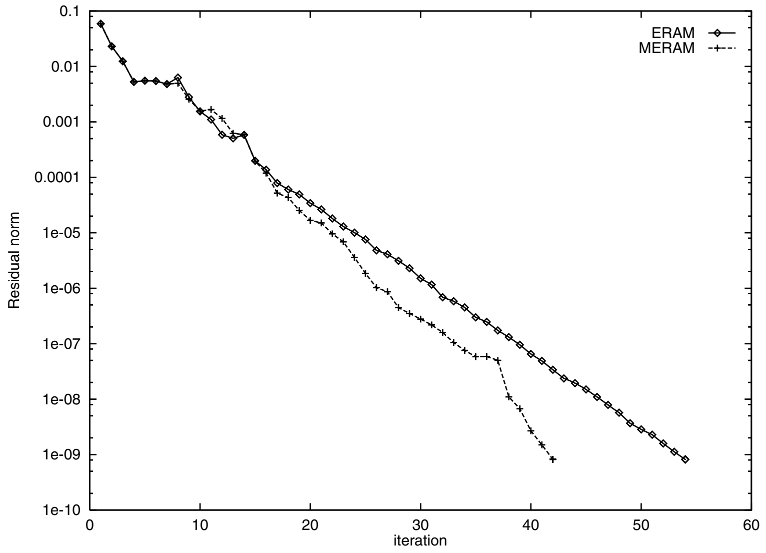

FIG. 4. *MERAM*$((26,26),[z,\mathrm{r}])$ *versus ERAM*$(26,z)$ *with* 63*-diagonal matrix.*

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 ERAM on this machine. By the number of iterations of MERAM, we mean the number of iterations of its process on the CM5.

Figures 3–7 illustrate the sum of the residual norms (SRN)[8] (4.10) with $\alpha_i = 1$

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

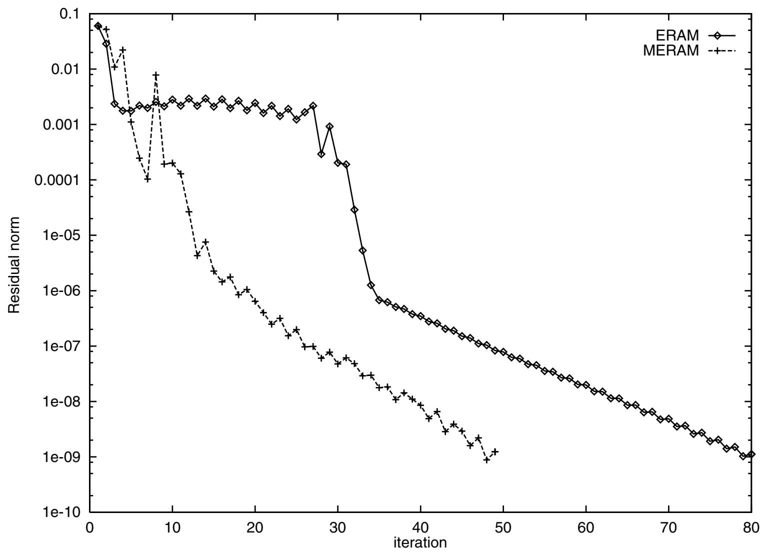

FIG. 5. *MERAM*$((32,32),[z,\mathrm{r}])$ *versus ERAM*$(32,z)$ *with* 21*-diagonal matrix.*

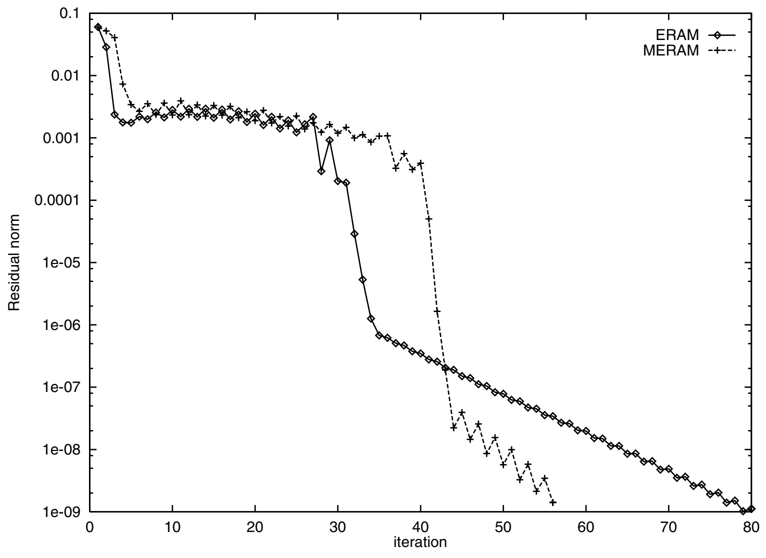

FIG. 6. *MERAM*$((32,20),[z,z])$ *versus ERAM*$(32,z)$ *with* 21*-diagonal matrix.*

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 3–7. This table and all of these figures illustrate the influence of the CM200 eigeninformation on the improvement of the restarting vector, and consequently on the convergence of the 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 ERAM to the desired

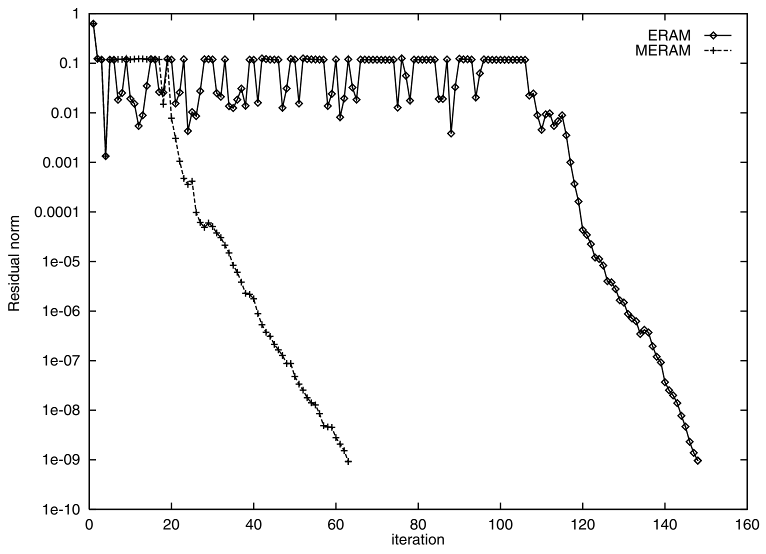

FIG. 7. *MERAM*$((40, 40), [z, z])$ *versus ERAM*$(40, z)$ *with* 63*-diagonal matrix.*

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

		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

invariant subspace. These results also show that we can obtain a good acceleration of convergence compared to ERAM.

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 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, the 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 the 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 the whole computed information will be taken into account.

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. The goal is to know which of the two schemes, MERAM and ERBAM (respectively, MERAM and IRAM), is the fastest given the same computational resources.

Hardware platform. Numerical experiments were done on several Sun Ultra Sparc workstations. They were linked by 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 of six workstations—three Sparc Ultra-1 and three Sparc Ultra-5-10—were used for the experiments of section 6.2.2. All of these workstations function with the Sun operating system.

Implementation. We made use of the 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++ that uses MPI through OOMPI [16] for the parallel classes [14, 15]. We used the PARPACK package [24] 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[9] parallelization strategy.

Test matrices. The matrices are taken from MatrixMarket [2], the University of Florida sparse matrix collection[10], and the SPARSKIT package [22]. The matrices $PDE2961$ and $PDE490000$ are from partial differential equations (PDE) and are created by the `MATPDE` matrix generator of MatrixMarket. The finite difference matrix $PDE248004$ is from a second-order elliptic operator and is produced by the generator `GENMAT` of SPARSKIT. Table 2 presents some characteristics of these matrices. The number of nonzero elements of a matrix A is denoted by NNZ and its Frobenius norm by $\|A\|_F$.

TABLE 2
List of matrices.

Collection	Matrix	Matrix size	NNZ	$\|A\|_F$
MatrixMarket	$AF23560$	23560	484256	$1.0E+4$
	$BFW782A$	782	5982	$1.3E+2$
	$PDE2961$	2961	14585	$2.2E+2$
	$PDE490000$	490000	2447200	$3.4E+3$
UF collection	$TWOTONE$	120750	1224224	$8.2E+5$
	$VENKAT01$	62424	1717792	$1.2E+2$
SPARSKIT	$PDE248004$	248004	1238028	$2.3E+3$

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

[9]Single program multiple data.
[10]http://www.cise.ufl.edu/research/sparse/matrices/.

TABLE 3
MERAM versus ERBAM on ℓ interconnected workstations.

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

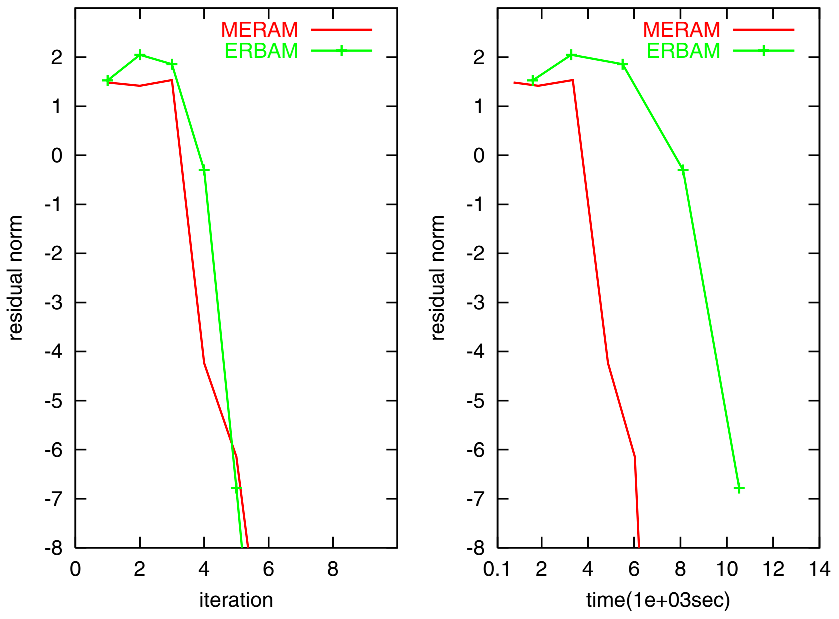

FIG. 8. *MERAM*$((20,25,35), I_3)$ *versus ERBAM*$(81, I_3)$ *with AF*23560 *matrix.*

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 $m = \frac{\mathcal{M}}{\ell} + \text{mod}(\ell, \text{mod}(\mathcal{M}, \ell))$. The tolerance value is $tol = 1E-6$. The number of iterations of MERAM is the number of iterations of the ERAM process that reaches convergence (the other ERAM processes are stopped when the first one converges). It is, generally, the ERAM process with the largest subspace size.

We run experimentations with two or three ERAMs in parallel (i.e., $\ell = 2$ or 3). Table 3 brings together the input and output parameters of some comparisons between MERAM and ERBAM. Figures 8 and 9 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. We notice in these figures and Table 3 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.

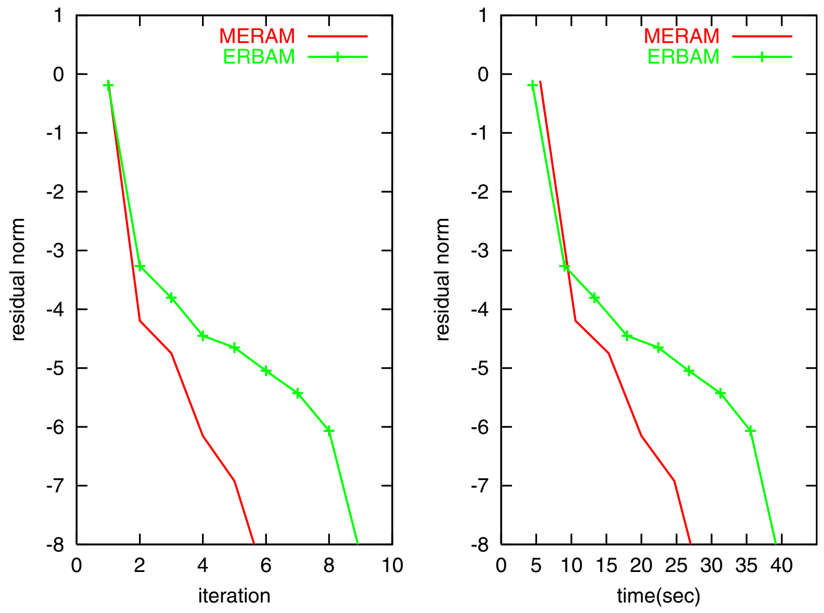

Fig. 9. *MERAM*$((5,45), I_2)$ *versus ERBAM*$(50, I_2)$ *with BFW*782*A matrix.*

6.2.2. MERAM versus IRAM. The Implicitly Restarted Arnoldi is a more robust method than ERAM. This is because of the powerful restarting strategy used in IRAM [23]. 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 eigeninformation obtained by the other ERAM processes. To evaluate the restarting strategy of MERAM with respect to IRAM, we have compared these methods under similar constraints.

The PARPACK package [24] implements a parallel version of IRAM according to the message passing model. We run experimentations with this package on a set of p interconnected workstations.

For our experiments, we have exploited the high level parallelism of asynchronous MERAM (i.e., that which is between its ERAM processes) as well as the parallelism inside each ERAM process. MERAM is implemented according to the model described in section 6.1. That is, we define ℓ computation tasks TCM_i and ℓ communication tasks Tcomm_i (for $i = 1, \ell$). Here TCM_i communicates with Tcomm_i using the message passing model. TCM_i implements the ith ERAM process of MERAM. After each iteration, if the convergence is not reached, the eigeninformation is sent to Tcomm_i which sends it in turn to the Tcomm_j tasks (for all $j \neq i$). Moreover, Tcomm_i receives the eigeninformation coming from 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 then have $p = (\sum_{i=1}^{\ell} p_i) + \ell$. 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 eigenvalues of largest modulus. We run experimentations with $p = 5$ or 6 workstations. The stopping criterion SRN is normalized by $\|A\|_F$ in both methods, and the tolerance value is $tol = 1.E - 6$.

Table 4
MERAM versus IRAM on p workstations.

				MERAM			IRAM		
Matrix	Fig.	p, p_1, p_2	s	$m_1, m_{\ell=2}$	iteration	time	m	iteration	time
AF23560	-	6, 3, 1	2	13, 23	3	119	23	8	118
BFW782	10	5, 2, 1	2	10, 20	6	17	20	23	19
VENKAT01	-	6, 3, 1	2	10, 30	3	399	30	3	161
TWOTONE	-	6, 3, 1	4	15, 25	3	550	25	4	354
PDE2961	12	5, 2, 1	2	15, 20	5	17	20	10	21
PDE248004	11	6, 3, 1	4	20, 35	20	8.87E+03	35	27	4.62E+03
PDE490000	13	5, 2, 1	4	55, 30	2	1.25E+04	55	30	4.07E+04

Table 4 brings together the input and output parameters of some comparisons between MERAM and PARPACK. For some of them, Figures 10–13 illustrate graphically Log_{10} of the normalized sum of the residual norms versus the number of restarts and the execution time for IRAM and MERAM. We notice in these figures and in Table 4 that the number of restarts of MERAM is always smaller than the number of restarts of PARPACK. On the other hand, the PARPACK response time is often faster than that of MERAM. This is due especially to the too heavy load of the workstations supporting the ERAM processes of MERAM. Let us recall that we dedicated $\ell = 2$ workstations to the communication processes Tcomm_1 and Tcomm_2. Consequently, the m_1 and m_2 matrix-vector multiplications of the first and the second ERAM processes of MERAM are distributed, respectively, on p_1 and p_2 workstations, while $(m_1 - s)$ matrix-vector multiplications of PARPACK are distributed on $p = p_1 + p_2 + \ell$ workstations. That means that each ERAM process runs in parallel on 2 or 3 processors or even in serial mode while IRAM runs on 5 or 6 processors. The load of the processors supporting IRAM is lighter than that of the one supporting ERAM processes of MERAM. Despite this unfavorable context for MERAM, the performances presented by Figures 10, 12, and 13 show that, for the corresponding matrices and parameters, MERAM has better convergence properties than IRAM.

7. Conclusion.

We have presented the MERAM and some of its characteristics: the 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 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 defined "the best" Ritz vector to be the vector that has the smallest residual norm.

Asynchronous MERAM is characterized 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 interfere with its termination. Furthermore, this algorithm has a great po-

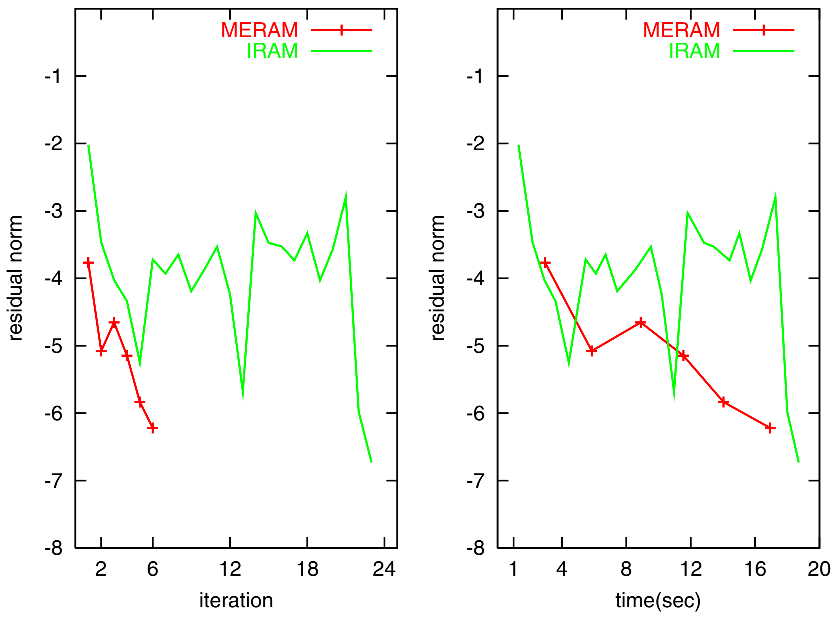

FIG. 10. *MERAM*$((10, 20), I_2)$ *versus IRAM*$(20, e_1)$ *with BFW*782*A matrix.*

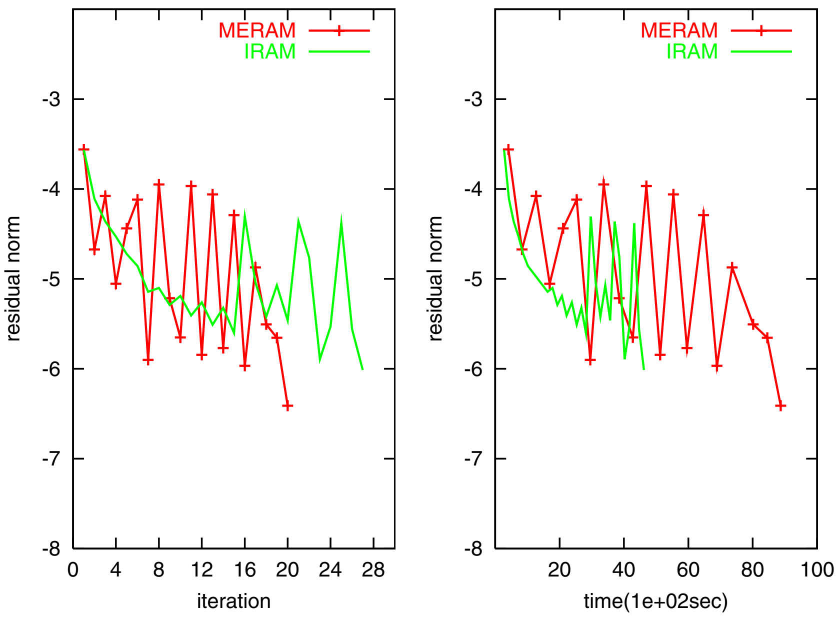

FIG. 11. *MERAM*$((20, 35), I_2)$ *versus IRAM*$(35, e_1)$ *with PDE*248004 *matrix.*

tential 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 then can be an optimization factor for this method. All of these properties show that MERAM is well suited to the GRID computational environments.

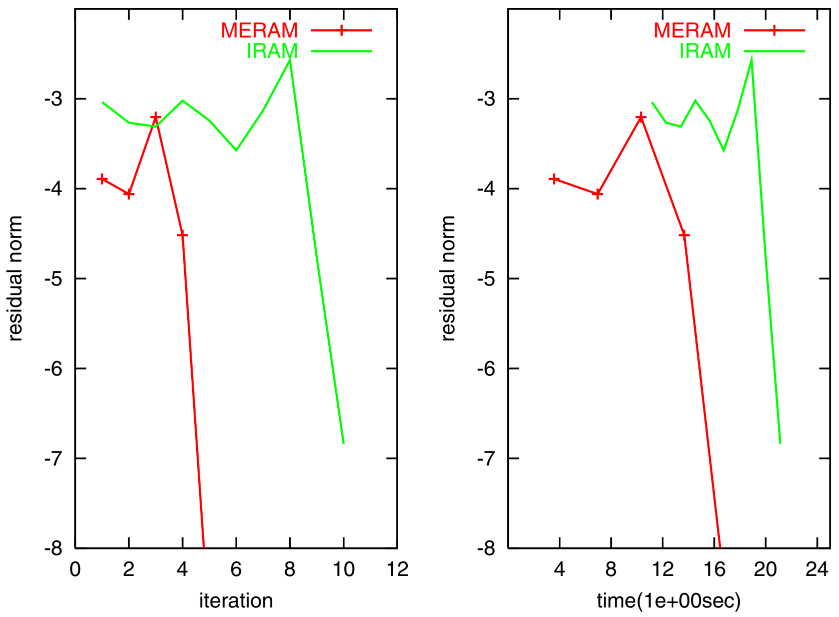

FIG. 12. *MERAM*$((15,20),I_2)$ *versus* *IRAM*$(20,e_1)$ *with* $PDE2961$ *matrix.*

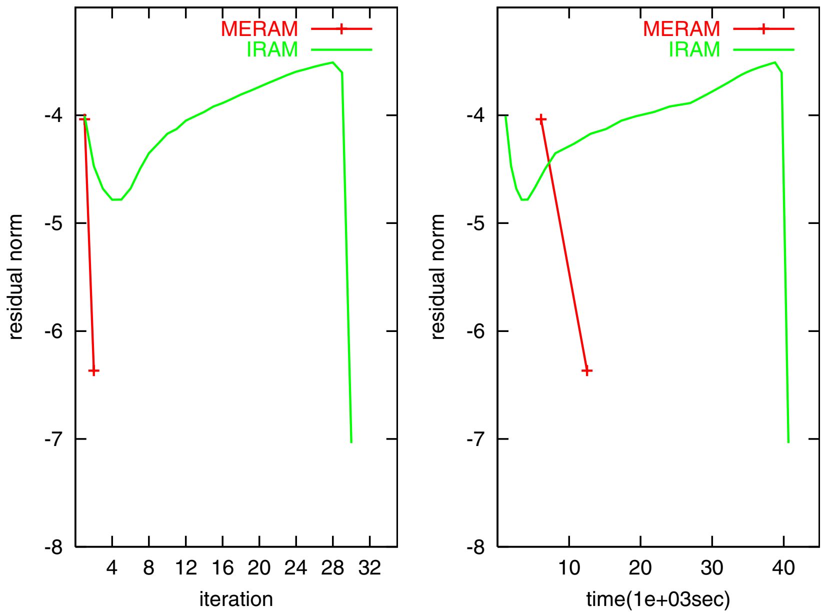

FIG. 13. *MERAM*$((55,30),I_2)$ *versus* *IRAM*$(55,e_1)$ *with* $PDE490000$ *matrix.*

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 point out that MERAM has a good convergence behavior compared with ERBAM. Moreover, according to our experiments, for some matrices, MERAM converges faster than IRAM.

We showed that to solve a large eigenproblem, the projection of ERAM on multiple subspaces and the communication of the intermediary results are more interesting than this projection on just one subspace. 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 a 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.

Acknowledgments. The authors wish to thank the SIAM referees for their numerous suggestions and Ani Sedrakian for the realization of some of the experiments.

REFERENCES

[1] W. E. Arnoldi, *The principle of minimized iteration in the solution of matrix eigenvalue problems*, Quart. Appl. Math., 9 (1951), pp. 17–29.

[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 (October 1996).

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

[4] C. Brézinski and M. Redivo-Zaglia, *A hybrid procedure for solving linear systems*, Numer. Math., 67 (1994), pp. 1–19.

[5] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van der Vorst, *Numerical Linear Algebra on High-Performance Computers*, Software Environ. Tools 7, SIAM, Philadelphia, 1998.

[6] G. Edjlali, N. Emad, and S. Petiton, *Hybrid methods on network of heterogeneous parallel computers*, in Proceedings of the 14th IMACS International Symposium on Iterative Methods in Linear Algebra, IMACS World Congress, Atlanta, 1994, pp. 1224-1228.

[7] G. Edjlali, S. Petiton, and N. Emad, *Interleaved Parallel Hybrid Arnoldi Method for a Parallel Machine and a Network of Workstations*, in Proceedings of the International Conference on Information, Systems, Analysis and Synthesis (ISAS'96), Orlando, FL, 1996.

[8] G. Edjlali, *Contribution à la parallélisation des méthodes itératives hybrides pour matrice creuse sur architectures hétérogènes*, Ph.D. thesis, University of Pierre et Marie Curie, Paris, 1994.

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

[10] I. Foster and C. Kesselman, eds., *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.

[11] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK User's Guide. Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, Software Environ. Tools, SIAM, Philadelphia, 1998.

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

[13] R. B. Morgan and M. Zeng, *A Harmonic Restarted Arnoldi Algorithm for Calculating Eigenvalues and Determining Multiplicity*, Tech. report, 2003, available online at http://www3.baylor.edu/~Ronald_Morgan/papers/hram2.pdf.

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

[15] E. Noulard and N. Emad, *A Key for Reusable Parallel Linear Algebra Software*, Parallel Comput., 27 (2001), pp. 1299–1319.

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

[17] S. Petiton, *Parallel subspace method for non-Hermitian eigenproblems on the connection machine*, Appl. Numer. Math., 10 (1992), pp. 19–35.

[18] S. Petiton and N. Emad, *A data parallel scientific computing introduction*, in The Data Parallel Programming Model, Lecture Notes in Comput. Sci. 1132, G.-R. Perrin and A. Darte, eds., Springer-Verlag, Berlin, 1996, pp. 45–64.

[19] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, UK, 1992.

[20] Y. Saad, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Math. Comp., 42 (1994), pp. 567–588.

[21] Y. Saad, *Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices*, Linear Algebra Appl., 34 (1980), pp. 269–295.

[22] Y. Saad, *SPARSKIT: A Basic Tool-Kit for Sparse Matrix Computations*, Version 2, http://www-users.cs.umn.edu/∼saad/software/SPARSKIT/sparskit.html (1994).

[23] D. C. Sorensen, *Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations*, in Parallel Numerical Algorithms, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997, pp. 119–165.

[24] K. J. Maschhoff and D. C. Sorensen, *P_ARPACK: An Efficient Portable Large Scale Eigenvalue Package for Distributed Memory Parallel Architectures*, in Applied Parallel Computing in Industrial Problems and Optimization (PARA'96), Lecture Notes in Comput. Sci. 1184, Springer-Verlag, Berlin, 1996, pp. 478–486.

[25] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614.

[26] H. D. Simon and L. Dagum, *Experince in using SIMD and MIMD parallelism for computational fluid dynamics*, Appl. Numer. Math., 12 (1993), pp. 431–442.

[27] Thinking Machines Corporation, *Connection Machine* 5, http://www.cs.berkeley.edu/∼demmel/cs267/cm5docs.html (1992).

[28] Thinking Machines Corporation, *CM Fortran Programming Guide*, http://www.cs.berkeley.edu/∼demmel/cs267/cm5docs.html (1992).

[29] G. L. G. Sleijpen and H. A. Van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM Rev., 42 (2000), pp. 267–293.

[30] G. L. G. Sleijpen and H. A. Van der Vorst, *Jacobi–Davidson methods*, in Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., Software Eviron. Tools 11, SIAM, Philadelphia, 2000, pp. 88–105.

[31] K. Wu and H. Simon, *Thick-restart Lanczos method for symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.