An Asynchronous Algorithm on NetSolve Global Computing System ⋆

Nahid Emad [a], S.-A. Shahzadeh-Fazeli [a], Jack Dongarra [b,*]

[a] *PRiSM laboratory, University of Versailles, 45, av. des États-Unis, 78035 Versailles Cedex, France*

[b] *Computer Science Department, University of Tennessee, Knoxville, U.S.A.*

Abstract

The explicitly restarted Arnoldi method (ERAM) allows to find a few eigenpairs of a large sparse matrix. The multiple explicitly restarted Arnoldi method (MERAM) is a technique based upon a multiple projection of ERAM and accelerates its convergence [3]. MERAM allows to update the restarting vector of an ERAM by taking into account the interesting eigen-information obtained by its other ERAM processes. This method is particularly well suited to the GRID-type environments. We present an adaptation of the asynchronous version of MERAM for NetSolve global computing system. We point out some advantages and limitations of this kind of systems to implement the asynchronous hybrid algorithms. We give some results of our experiments and show that we can obtain a good acceleration of the convergence compared to ERAM. These results show also the interest of the MERAM-like hybrid methods for the GRID computing environments.

Key words: Large eigenproblem, Arnoldi method, Explicit restarting, Global computing, NetSolve.

1 Introduction

The hybrid methods were proposed 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 or several differently

⋆ The two first authors were supported in part by French ACI-GRID ANCG project.

* Corresponding author.

Email addresses: (emad, sas)@prism.uvsq.fr, dongarra@cs.utk.edu (Jack Dongarra).

parameterized copies of the same method to solve these problems efficiently [2],[7], [6], [13], [14]. The multiple explicitly restarted Arnoldi method proposed in [3] is a hybrid method which allows to approximate a few eigenpairs of a large sparse non-Hermitian matrix. This technique is based on the projection of the problem on a set of subspaces and thus creates a whole of differently parameterized ERAM processes. Each ERAM improves its subspace by taking into account all interesting intermediary eigen-information obtained by itself as well as by the other ERAM processes. In this method, the communications between ERAM processes can be completely asynchronous.

In this paper we present the application of the asynchronous version of MERAM to NetSolve global computing system. We show some advantages and limitations of this kind of systems to implement the asynchronous hybrid algorithms. We give then an adaptation of the algorithm for NetSolve and show that we can obtain a good acceleration of the convergence with respect to the explicitly restarted Arnoldi method.

Section 2 describes the basic Arnoldi algorithm and explicitly restarted Arnoldi method. Section 3 introduces MERAM and some of its algorithms. We point out the limitations of NetSolve-type systems to implement the asynchronous algorithm of MERAM and present an adaptation of this algorithm for NetSolve in section 4. This algorithm is evaluated in section 5 by a set of test matrices coming from various application problems. The concluding remarks in section 6 will contain our perspectives on the problem.

2 Explicitly restarted Arnoldi method

Let A be a large non-Hermitian matrix of dimension $n \times n$. We consider the problem of finding a few eigenpairs (λ, u) of A :

$$Au = \lambda u \text{ with } \lambda \in \mathbb{C} \text{ and } u \in \mathbb{C}^n. \tag{1}$$

Let $w_1 = v/\|v\|_2$ be an initial guess, m be an integer with $m \ll n$. A Krylov subspace method allows to project the problem (1) onto a m-dimensional subspace $\mathbb{K} = span(w_1, Aw_1, \ldots A^{m-1}w_1)$. The well-known Arnoldi process is a projection method which generates an orthogonal basis $w_1, \ldots, w_m$ of the Krylov subspace $\mathbb{K}$ by using the Gram-Schmidt orthogonalization process. Let $\texttt{AR}(input : A, m, v; output : H_m, W_m)$ be such Arnoldi reduction. The $m \times m$ matrix $H_m = (h_{i,j})$ and the $n \times m$ matrix $W_m = [w_1, \ldots, w_m]$ issued from AR algorithm and the matrix A satisfy the equation:

$$AW_m = W_m H_m + f_m e_m^H \tag{2}$$

where $f_m = h_{m+1,m} w_{m+1}$ and e_m is the mth vector of the canonical basis of $\mathbb{C}^m$. The s desired Ritz values [1] (with largest/smallest real part or largest/smallest magnitude) $\Lambda_m = (\lambda_1^{(m)}, \ldots, \lambda_s^{(m)})$ and their associate Ritz vectors $U_m = (u_1^{(m)}, \ldots, u_s^{(m)})$ can be computed as follows [2]:

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: $(\lambda_i^{(m)}, y_i^{(m)})_{i=1}^{i=s}$.
(3) Compute the s associate Ritz vectors $u_i^{(m)} = W_m y_i^{(m)}$ (for $i = 1, \ldots, s$).
(4) Compute $r_s = (\rho_1, \ldots, \rho_s)^t$ with $\rho_i = \|(A - \lambda_i^{(m)} I) u_i^{(m)}\|_2$.

If the accuracy of the computed Ritz elements is not satisfactory the projection can be restarted onto a new $\mathbb{K}$. This new subspace can be defined with the same subspace size and a new initial guess. The technique is called the explicitly restarted Arnoldi method. Starting with an initial vector v, it computes `BAA`. If the convergence does not occur, then the starting vector is updated (using appropriate methods on the computed Ritz vectors) and a `BAA` process is restarted until the accuracy of the approximated solution is satisfactory. This update is designed to force the vector in the desired invariant subspace. This goal can be reached by some polynomial restarting strategies proposed in [6], [8] and discussed in section 3.1. An algorithm of explicitly restarted Arnoldi method is the following:

ERAM algorithm: `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 tolerance value and the function g defines the stopping criterion of iterations. Some typical examples are: $g(r_s) = \|r_s\|_\infty$ and $g(r_s) = \sum_{j=1}^{s} \alpha_j \rho_j$ where α_j are scalar values.

[1] A Ritz value/vector of a matrix is an approximated eigenvalue/eigenvector.
[2] 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.

3 Multiple explicitly restarted Arnoldi method

The multiple explicitly restarted Arnoldi method is a technique based upon an ERAM with multiple projection. This method project an eigenproblem on a set of subspaces and thus creates a whole of differently parameterized ERAM processes which co-operate to efficiently compute a solution of this problem. In MERAM the restarting vector of an ERAM is updated by taking into account the interesting eigen-information obtained by the other ones. In other words, the ERAM processes of a MERAM begin with several subspaces spanned by a set of initial vectors and a set of subspace sizes. If the convergence does not occur for any of them, then the new subspaces will be defined with initial vectors updated by taking into account the intermediary solutions computed by all the ERAM processes. Each of these differently sized subspaces is defined with a new initial vector v. To overcome the storage dependent shortcoming of ERAM, a constraint on the subspace size of each ERAM is imposed. More precisely, the size of a projection subspace has to belong to the discrete interval $I_m = [m_{min}, m_{max}]$. The bounds m_{min} and m_{max} may be chosen in function of the available computation and storage resources and have to fulfill $m_{min} \leq m_{max} \ll n$. Let $m_1 \leq \ldots \leq m_\ell$ be a set of ℓ subspace sizes with $m_i \in I_m$ $(1 \leq i \leq \ell)$, $M = (m_1, \ldots, m_\ell)$ and $V^\ell = [v^1, \ldots, v^\ell]$ be the matrix of ℓ starting vectors. An algorithm of this method to compute s $(s \leq m_1)$ desired Ritz elements of A is the following:

MERAM algorithm : `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 $M = (m_1, \ldots, m_\ell)$. Let $it = 0$.
(2) `Iterate.` For $i = 1, \ldots, \ell$ do: $it = it + 1$.
  (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.
  (c) If ($it \geq \ell$ and (it mod ℓ) $\neq 0$) then use the results produced by the ℓ last `BAA` processes to update v^{i+1}.
(3) `Restart.` Use the results produced by the ℓ last `BAA` processes to update v^1 and go to 2.

---

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

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

$$v^i = f(U^{best}), \tag{3}$$

where $U^{best} = (u_1^{best}, \ldots, u_s^{best})$ and u_j^{best} is "the best" jth Ritz vector. The definition of the function f can be based onto the techniques proposed by Y. Saad in [6] and will be discussed in section 3.1.

The problem of the above algorithm is that there is no parallelism between the `BAA` processes. This is because of the existence of the synchronization points `2.(c)` and `3` in the algorithm. In the following algorithm, proposed in [3], these synchronization points are removed. That means each ERAM process, after its `BAA` step, sends its results to all other processes. Let `Send_Eigen_Info` represents the task of sending results from an ERAM process to all other ERAM processes, `Receiv_Eigen_Info` be the task of receiving results from one or more ERAM processes by the current ERAM process and finally, `Rcv_Eigen_Info` be a boolean variable that is true if the current ERAM process has received results from the other ERAM processes. A parallel asynchronous version of MERAM is the following:

Asynchronous MERAM algorithm.

---

(1) `Start.` Choose a starting matrix V^ℓ and a set of subspace sizes $M = (m_1, \ldots, m_\ell)$.
(2) `Iterate.` For $i = 1, \ldots, \ell$ 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)` Update the initial guess with
    if (`Rcv_Eigen_Info`) then hybrid restart strategy
    else simple restart strategy
- Communication process
  - `(d)` `Send_Eigen_Info`
  - `(e)` `Receiv_Eigen_Info`

The ℓ ERAM processes defined in step 2 of the above algorithm are all independents and can be run in parallel. Each of them is constituted by a computation part and a communication part. The computation and the communication can be overlapped inside of an ERAM process. The updating of the initial vector v^i can be done by taking into account the most recent results of the ERAM processes. We recall that, in the above MERAM algorithm, the ℓ last results are necessarily the results of the ℓ ERAM processes.

The above algorithm is fault tolerant. A loss of an ERAM process during MERAM execution does not interfere with its termination. It has a great potential for dynamic load balancing ; the attribution of ERAM processes of

MERAM to the available resources can be done as a function of their subspace sizes at run time. The heterogeneity of computing supports can be then an optimization factor for this method [3]. Because of all these properties, this algorithm is well suited to the GRID-type environments. In a such environment, the ℓ ERAM processes constituting a MERAM can be dedicated to ℓ different servers. Suppose that the ith ERAM process is dedicated to the server S_i. This server keeps the execution control of the ith ERAM process until the convergence which occurs, in general, by the fastest server. The figure 1 shows an execution scheme of the asynchronous MERAM with $\ell = 3$ on 3 servers. We notice that the computation and communication parts are overlapped.

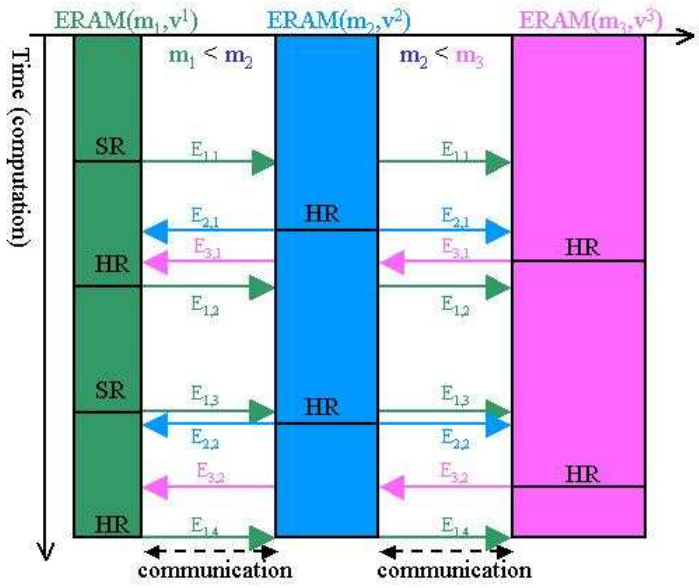

Fig. 1. Asynchronous MERAM with $\ell = 3$

3.1 Restarting Strategies

Saad [7] proposed to restart an iteration of ERAM with a vector preconditioning so that it has to be forced in the desired invariant subspace. It concerns a polynomial preconditioning applied to the starting vector of ERAM. This preconditioning aims at computing the restarting vector so that its components are nonzero in the desired invariant subspace and zero in the unwanted invariant subspace:

$$v(k) = p(A)v \tag{4}$$

where $v(k)$ is kth restarting vector of ERAM and p is a polynomial in the space of polynomials of degree $< m$. One appropriate possibility to define p is a Chebyshev polynomial determined from some knowledge on the distribution of the eigenvalues of A. This restarting strategy is very efficient to accelerate the convergence of ERAM and is discussed in detail in [7], [6]. Another possibility to define the polynomial p is to compute the restarting vector with a linear combination of s desired Ritz vectors:

$$v(k) = \sum_{i=1}^{s} \alpha_i u_i^{(m)}(k) \tag{5}$$

where $u_i^{(m)}(k)$ denotes ith Ritz vector computed at the iteration k. There are several ways to choose the scalar values α_i in (5). One choice can be α_i equal to the ith residual norm. Some other choices can be $\alpha_i = 1$, $\alpha_i = i$ or $\alpha_i = s - i + 1$ for $1 \leq i \leq s$ (see [8] for more details). We propose to make use of the following linear combination of the s wanted eigenvectors :

$$v = \sum_{k=1}^{s} l_k(\lambda) {u_k}^{(m)} \tag{6}$$

where s coefficients $l_k(\lambda)$ are defined by : $l_k(\lambda) = \prod_{\substack{j=1 \\ j \neq k}}^{s} (\frac{\lambda - {\lambda_j}^{(m)}}{{\lambda_k}^{(m)} - {\lambda_j}^{(m)}})$, with $\lambda = (\lambda_{min} + \bar{\lambda} - \frac{\lambda_{min}}{n})/2$, $\bar{\lambda} = \frac{\sum_{k=1}^{s} {\lambda_k}^{(m)}}{s}$ and λ_{min} is the eigenvalue with the smallest residual norm. In the experiments of the next section, we made use of this strategy (i.e., equation (6)) to update the initial vector of the ERAM as well as the ones of the ERAM processes of MERAM. For MERAM this equation becomes

$$v^i = \sum_{k=1}^{s} l_k^{(best)}(\lambda) u_k^{(best)} \tag{7}$$

where $u_k^{(best)}$ is "the best" kth eigenvector computed by the ERAM processes of MERAM and $l_k^{(best)}$ is its associate coefficient.

4 Asynchronous MERAM on a global computing system

4.1 NetSolve global computing system

NetSolve system is a grid middleware based on the concepts of `Remote Procedure Call (RPC)` that allows users to access both hardware and software compu-

tational resources distributed across a network. NetSolve provides an environment that monitors and manages computational resources and allocates the services they provide to NetSolve enabled client programs. NetSolve uses a load-balancing strategy to improve the use of the computational resources available [5]. Three chief components of NetSolve are clients, agents and servers. The semantics of a NetSolve client request are:

(1) Client contacts the agent for a list of capable servers.
(2) Client contacts server and sends input parameters.
(3) Server runs appropriate service.
(4) Server returns output parameters or error status to client.

There are many advantages to using a system like NetSolve which can provide access to otherwise unavailable software/harware. In cases where the software is in hand, it can make the power of supercomputers accessible from low-end machines like laptop computers. Furthermore, NetSolve adds heuristics that attempt to and the most expeditious route to a problem's solution set. NetSolve currently supports the C, FORTRAN, MATLAB, and Mathematica as languages of implementation for client programs. To solve a problem using NetSolve, a problem description file (PDF) corresponding to the problem has to be defined [15], [10–12].

4.2 Asynchronous MERAM on NetSolve system

The servers of NetSolve system can not communicate directly to each other. Consequently, contrarily to MERAM running schemes presented in figures 1 and 2, a server can't keep the control of an ERAM process until the convergence. Figure 2 shows asynchronous MERAM algorithm on 3 servers which communicate directly. Each server runs the steps 2.(a), 2.(b) and 2.(c) of an ERAM and communicates with the other servers by running the steps 2.(d) and 2.(e) of the algorithm. While figure 3 shows asynchronous MERAM algorithm on 3 servers of a system such as NetSolve where they can not communicate directly.

Indeed, to adapt the asynchronous MERAM algorithm to NetSolve system a control process centralizing the information and corresponding to a client component of NetSolve has to be defined. This process has to request to the computation servers of the system to run the step 2.(a) of ERAM processes of MERAM in RPC mode. The running of the step 2.(a) of an ERAM occurs asynchronously in respect with the execution of the same step of the other ERAMs as well as with the execution of the rest of the client algorithm. Once

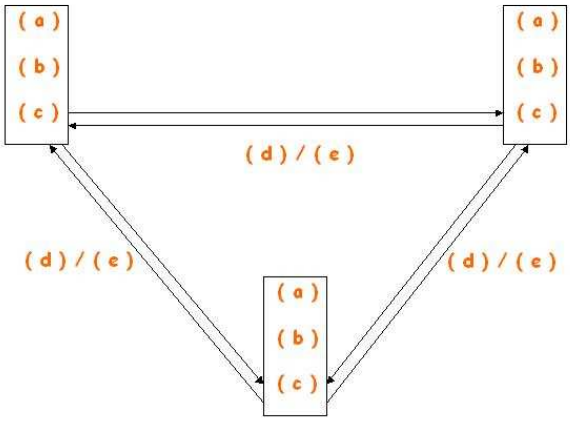

Fig. 2. Asynchronous MERAM on 3 communicating servers (with $\ell = 3$)

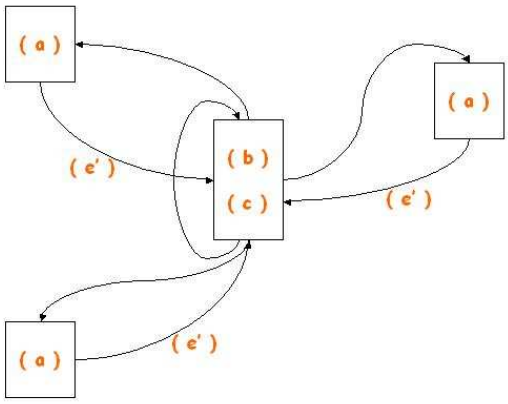

Fig. 3. Asynchronous MERAM on 3 non communicating servers (with $\ell = 3$)

the control process receives the results of an BAA step, it tests the convergence by running the step 2.(b) of the algorithm. If the convergence is not reached then it updates the initial guess with the available eigen-information on this control/client server. An adaptation of the asynchronous multiple explicitly restarted Arnoldi method for NetSolve is the following:

`MERAM-NS`$(input : A, s, M, V^\ell, tol; output : r_s, \Lambda_m, U_m)$

---

(1) `Start.` Choose a starting matrix V^ℓ and a set of subspace sizes $M = (m_1, \ldots, m_\ell)$.

Let $it_i = 0$ (for $i = 1, \ell$).

(2) For $i = 1, \ldots, \ell$ do :

`(a)` Compute a `BAA`$(input : A, s, m_i, v^i; output : r_s, \Lambda_{m_i}, U_{m_i})$ step in RPC mode.

(3) `Iterate.` For $i = 1, \ldots, \ell$ do :

- If `(ready_results)` then $it_i = it_i + 1$

  `(e')` Receive results.

  `(b)` If $g(r_s^i) \leq tol$ stop all processes.

  `(c)` Update the initial guess in function of the available eigen-information.

  `(a)` Compute a `BAA`$(input : A, s, m_i, v^i; output : r_s, \Lambda_{m_i}, U_{m_i})$ step in RPC mode.
- End if

(4) `End.` $it = \max(it_1, \ldots, it_\ell)$

Where `ready_results` is a boolean variable which is true if the outputs of the current BAA algorithm are ready. In other words, if the server computing the ith `BAA` in RPC mode is ready to send its outputs. We notice that in this implementation the step 2.(d) of the asynchronous MERAM algorithm is not necessary and the step 2.(e) is replaced by 3.(e') which consists to receive *all* eigen-information on the control process. Instead, we notice that in each computation request in RPC mode, the client program has to send all inputs to the computation server which accepts this task. That means, in MERAM-NS algorithm, for each restart (i.e., iteration) of every ERAM process, the client program has to send the n-order matrix A, and an n-size initial vector to a computation server. This engenders an intense communication between the client and computation servers. But this communication is overlapped by the running of the rest of the algorithm. We can notice that when a computational server finishes the step 2.(a) or 3.(a), it has to return $s + 2$ n-size output vectors to the client process. Figure 4 presents the implementation of `MERAM-NS` algorithm on a NetSolve system with 21 servers [3] and $\ell = 3$.

In asynchronous MERAM algorithm, at the end of an iteration each ERAM sends $s + 2$ n-size vectors to $\ell - 1$ other processes. That means, each ERAM has to communicate $(\ell-1) \times (s+2) \times n$ data to other processes. The reception of $s + 2$ n-size vectors by a process is not determinism and not quantifiable.

[3] This is an image of the system at a given instant. Indeed, since the servers are volatile, the number of the servers in NetSolve system can change at any moment.

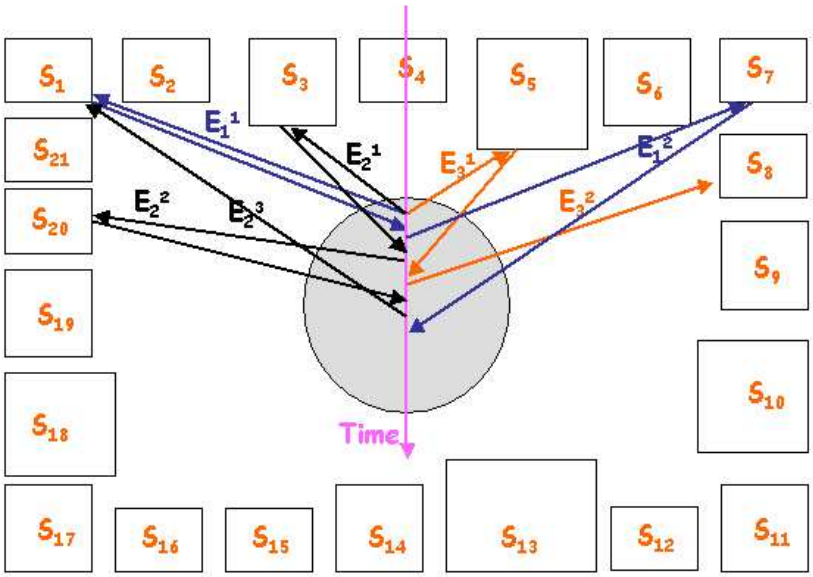

Fig. 4. MERAM-NS on NetSolve (Ex. of 3 ERAMs processes and 21 servers). S_k is the kth server of the NetSolve system and ${E_i}^j$ is the jth restart of the ith ERAM process of MERAM.

5 Numerical experiments

The experiments presented in this section have been done on a NetSolve system whose computation servers have been located in France (at the university of Versailles and the Institute of Technology of Vélizy sites) and in U.S.A. and interconnected by internet. We implemented ERAM and MERAM (i.e., MERAM-NS) algorithms using C and MATLAB for some real matrices on NetSolve system. The client applications are written in MATLAB while the programs having to run in RPC mode (i.e., ERAM processes) are written in C. The stopping criterion is $g(r_s^i) = \|r_s^i\|_\infty$ where $r_s^i = (\rho_1^i, \ldots, \rho_s^i)$ and ρ_j^i is normalized by $\rho_j^i = \rho_j^i / \|A\|_F$ for all $j \in [1, \ldots, s]$ and $i \in [1, \ldots, \ell]$. The tolerance value tol is 10^{-8} in the figures 5, 6, 8 and 10^{-14} in the figure 7. For all figures the initial vector is $v = z_n = (1, \ldots, 1)/\sqrt{n}$ and the initial matrix is $V^\ell = [v^1 = z_n, \ldots, v^\ell = z_n]$. We search a number $s = 2$ or $s = 5$ of the eigenvalues with the largest magnitude. The used matrices are taken from the matrix market [1] and presented in the table 1. The number of non zero elements of a matrix is denoted by NNZ. In our experiments, we run MERAM-NS with $\ell = 3$ ERAM processes where the steps 2 and 3.(a) are computed in RPC nonblocking mode. The efficiency of our algorithms on

Matrix	matrix size	NNZ
$af23560.mtx$	23560	484256
$mhd4800b.mtx$	4800	16160
$gre\_1107.mtx$	1107	5664
$west2021.mtx$	2021	7353

Table 1
The matrix market used matrices

NetSolve are measured in terms of the number *it* of the restarts. The number of iterations of MERAM in all of the figures is the number of iterations of the ERAM process which reaches convergence. It is generally the ERAM process with the largest subspace size.

5.1 MERAM-NS versus ERAM

In the following figures, we denote by MERAM$(m_1, \ldots, m_l)$ a MERAM with subspaces sizes $m_1, \ldots, m_l$ and by ERAM(m) an ERAM with subspace size m. The tables 2 and 3 presents the results obtained with ERAM and MERAM algorithms on NetSolve and the table 4 presents a comparison between the results obtained by ERAM and MERAM in term of the number of restarts. We show graphically in figures 5 to 9 the residual norm as a function of iteration number to reach convergence using ERAM and MERAM on NetSolve. The results of our experiments presented in the tables 2, 3 and 4 and in the figures 5 to 9 indicate that our MERAM-NS algorithm has better performance than ERAM. We notice from these tables that in term of the number of the restarts MERAM is considerably more efficient than ERAM.

Matrix	m	s	v	it	Res.Norms	Fig
$af23560.mtx$	10	2	z_n	240	No converge	5
$mhd4800b.mtx$	10	2	z_n	41	8.127003e-10	6
$mhd4800b.mtx$	20	5	z_n	19	4.089292e-15	7
$gre\_1107.mtx$	30	2	z_n	46	3.389087e-09	8
$west2021.mtx$	10	2	z_n	18	1.742610e-09	9

Table 2
ERAM results on NetSolve

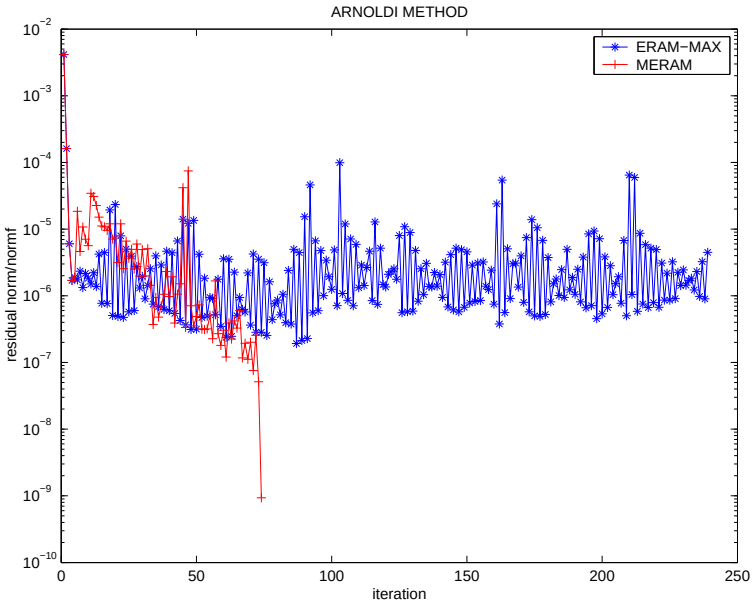

Fig. 5. MERAM$(5, 7, 10)$ versus ERAM(10) with $af23560.mtx$ matrix. MERAM converges in 74 restarts, ERAM does'nt converge after 240 restarts.

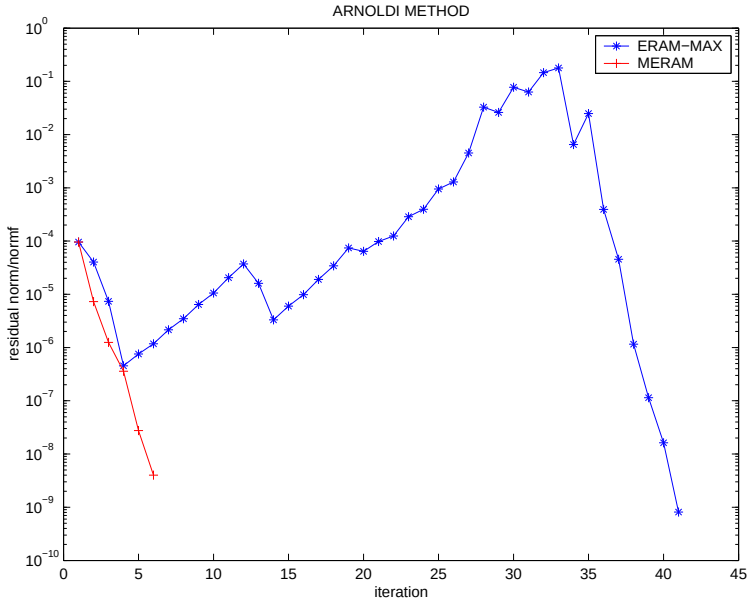

Fig. 6. MERAM$(5, 7, 10)$ versus ERAM(10) with $mhd4800b.mtx$ matrix. MERAM converges in 6 restarts, ERAM converges in 41 restarts.

6 Conclusion

The standard restarted Arnoldi algorithm and its variants may not be efficient for computing a few selected eigenpairs of large sparse non-Hermitian matrices. In order to improve the overall performance of Arnoldi type algorithm, we proposed an adaptation of the multiple explicitly restarted Arnoldi method

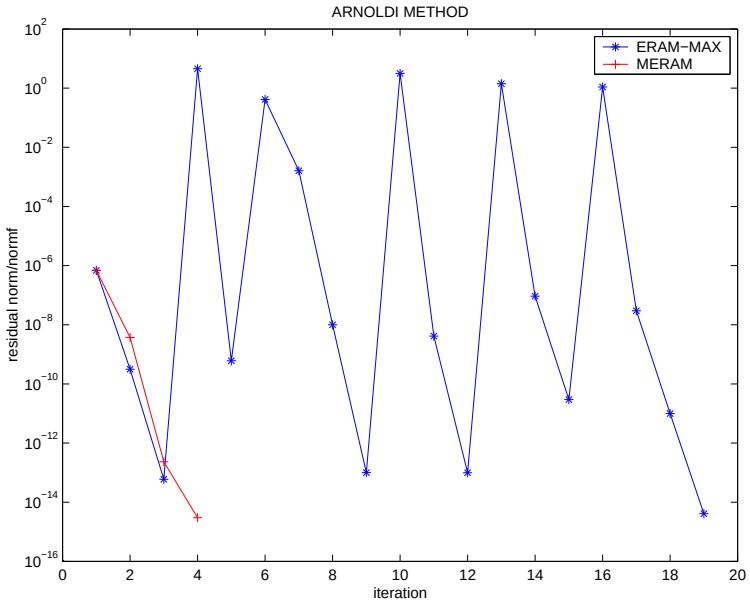

Fig. 7. MERAM$(10, 15, 20)$ versus ERAM(20) with $mhd4800b.mtx$ matrix. MERAM converges in 4 restarts, ERAM converges in 19 restarts.

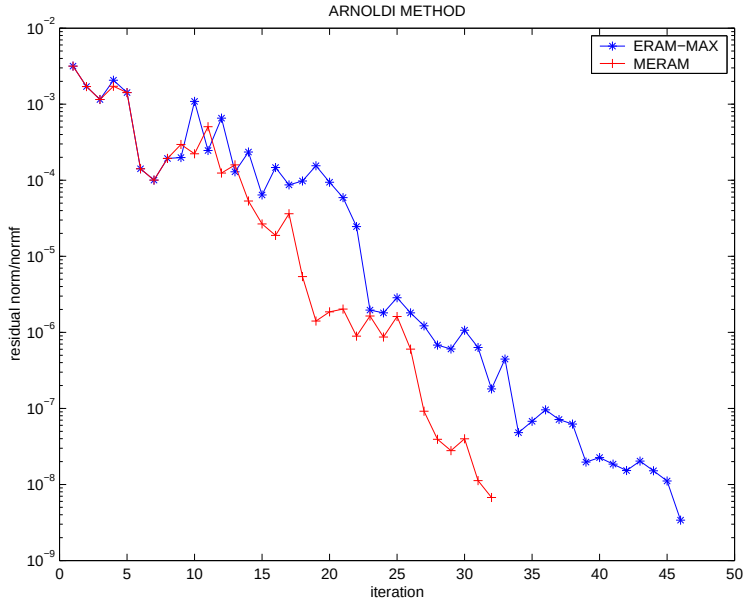

Fig. 8. MERAM$(5, 10, 30)$ versus ERAM(30) with $gre\_1107.mtx$ matrix. MERAM converges in 32 restarts, ERAM converges in 46 restarts.

for NetSolve system. We have seen that this method accelerates the convergence of explicitly restarted Arnoldi method. The numerical experiments have demonstrated that this variant of MERAM is often much more efficient than ERAM. In addition, this concept may be used in some Krylov subspace type method for the solution of large sparse non symmetric eigenproblem such as multiple implicitly restarted method based onto IRAM [4], [9].

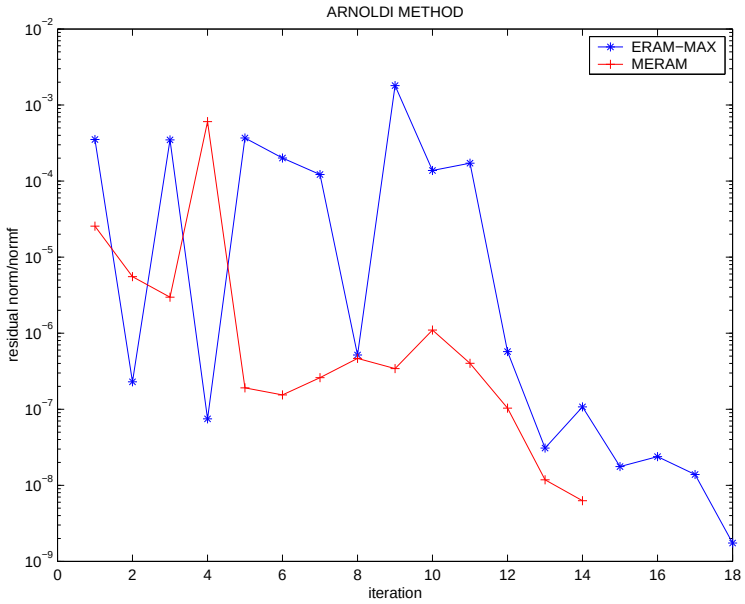

Fig. 9. MERAM$(5, 7, 10)$ versus ERAM(10) with $west2021.mtx$ matrix. MERAM converges in 14 restarts, ERAM converges in 18 restarts.

Matrix	m_1, m_2, m_3	s	$v^1, v^2, v^3,$	it	Res. Norms	Fig
$af23560.mtx$	5, 7, 10	2	z_n, z_n, z_n	74	9.329017e-10	5
$mhd4800b.mtx$	5, 7, 10	2	z_n, z_n, z_n	6	4.016027e-09	6
$mhd4800b.mtx$	10, 15, 20	5	z_n, z_n, z_n	4	2.999647e-15	7
$gre\_1107.mtx$	5, 10, 30	2	z_n, z_n, z_n	32	6.753314e-09	8
$west2021.mtx$	5, 7, 10	2	z_n, z_n, z_n	14	6.267924e-09	9

Table 3
MERAM results on NetSolve

We have shown that the MERAM-type asynchronous algorithms are very well adapted to the global computing systems such as NetSolve. Meanwhile, one of the major problems remains the transfer of the matrix from the client server towards the computation servers. For example, the order of magnitude of the transferred data between client and computation servers is $O((it_1 + \ldots + it_\ell) \times NNZ)$ for MERAM-NS algorithm. Moreover, the classical evaluation of performances is no more valid in this kind of systems. For example, the execution response time can not be a good measure of performance for MERAM nor for a comparison between MERAM and ERAM. This is for the following reasons:

`1.` the execution time is dependant to the internet load,
`2.` the servers are volatile; a server can take part in a portion of calculation and disappear afterwards, `3.` the servers are transparent; that means, we do not know the server on which a specific process (such as `2.(a)` or `3.(a)` steps of `MERAM-NS` algorithm) will be run,

			ERAM		MERAM	
Matrix	Fig.	ℓ	m	iteration	$m_1, \ldots, m_\ell$	iteration
af23560.mtx	5	3	10	*	5, 7, 10	74
mhd4800b.mtx	6	3	20	19	10, 15, 20	4
mhd4800b.mtx	7	3	10	41	5, 7, 10	6
gre_1107.mtx	8	3	30	46	5, 10, 30	32
west2021.mtx	9	3	10	18	5, 7, 10	14

Table 4
Comparison of ERAM(m) and ERAM($m_1, \ldots, m_\ell$) on NetSolve

4. the implementation of the ERAM on NetSolve introduces some artificial communications.

One could think to have a rapid response time it would be better to make use of a classical parallel supercomputer. But the supercomputers are not easily accessible and moreover, the use of a global computing system allows to take advantage of the otherwise unavailable software and/or hardware resources. In addition, the asynchronous version of the hybrid methods of type MERAM or MIRAM are interesting primarily within the framework of global computing environments.

References

[1] 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).

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

[3] N. EMAD, S. Petiton, And G. Edjlali, *Multiple explicitly restarted Arnoldi method for solving large eigenproblems*, SIAM Journal on scientific computing SJSC, Volume 27, Number 1, pp. 253-277, 2005.

[4] 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.

[5] James S. Plank, Henri Casanova, Micah Beck and Jack J. Dongarra, *Deploying fault tolerance and task migration with NetSolve*, Future Generation Computer Systems, Volume 15, Issues 5-6, pp. 745-755, 1999.

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

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

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

[9] 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.

[10] H. Casanova and J. Dongarra , *NetSolve: A Network Server for Solving Computational Science Problems.* The International Journal of Supercomputer Applications and High Performance Computing,1997.

[11] H. Casanova and J. Dongarra , *NetSolve's Network Enabled Server: Examples and Applications*, IEEE Computational Science and Engineering, 57-67, 5(3), 1998.

[12] H. Casanova and J. Dongarra , *NetSolve version 1.2: Design and Implementation*, UT Department of Computer Science Technical Report,1998.

[13] 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.

[14] 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.

[15] H., *User's Guide to NetSolve V1.4*, full reference at http://icl.cs.utk.edu/netsolve