Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 33 (2007) 521–540
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement

V. Hernandez, J.E. Roman, A. Tomas *

D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

Received 28 July 2006; received in revised form 19 December 2006; accepted 20 April 2007
Available online 4 May 2007

Abstract

This paper presents several new variants of the single-vector Arnoldi algorithm for computing approximations to eigenvalues and eigenvectors of a non-symmetric matrix. The context of this work is the efficient implementation of industrial-strength, parallel, sparse eigensolvers, in which robustness is of paramount importance, as well as efficiency. For this reason, Arnoldi variants that employ Gram-Schmidt with iterative reorthogonalization are considered. The proposed algorithms aim at improving the scalability when running in massively parallel platforms with many processors. The main goal is to reduce the performance penalty induced by global communications required in vector inner products and norms. In the proposed algorithms, this is achieved by reorganizing the stages that involve these operations, particularly the orthogonalization and normalization of vectors, in such a way that several global communications are grouped together while guaranteeing that the numerical stability of the process is maintained. The numerical properties of the new algorithms are assessed by means of a large set of test matrices. Also, scalability analyses show a significant improvement in parallel performance.

Keywords: Arnoldi eigensolvers; Iterative Gram-Schmidt orthogonalization; Distributed-memory programming

1. Introduction

Krylov subspace methods are a widely-used class of algorithms for approximating a small subset of the eigenvalues (and corresponding eigenvectors) of a large, sparse matrix A. In the case of non-symmetric matrices, Krylov eigensolvers can be classified in two families, depending on whether they rely on the Arnoldi or the two-sided Lanczos processes. The Arnoldi process [1–3] computes a set of so-called Arnoldi vectors, which constitute an orthonormal basis of the Krylov subspace associated with the matrix A and a given initial vector x_1. As a byproduct, the Arnoldi process also computes an upper Hessenberg matrix, which can be shown to be the matrix associated with an orthogonal projection onto the Krylov subspace, thus enabling the computation of approximate eigenpairs of A by a Rayleigh-Ritz procedure. In the two-sided Lanczos process [4–6] two sets

* Corresponding author.
E-mail addresses: vhernand@dsic.upv.es (V. Hernandez), jroman@dsic.upv.es (J.E. Roman), atomas@itaca.upv.es (A. Tomas).

0167-8191/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2007.04.004

of bi-orthogonal vectors are computed and the byproduct is a tridiagonal matrix, associated in this case with an oblique projection. Computing approximations of the eigenpairs by the two-sided Lanczos process potentially involves many more numerical complications than in the case of Arnoldi. On the other hand, a remarkable advantage is that Lanczos vectors are computed via short recurrences as opposed to Arnoldi vectors. In this work, we focus on Arnoldi methods, i.e. eigensolvers based on the Arnoldi process.

The fact that Arnoldi vectors cannot be defined via short recurrences means in practice that the computation of every new vector involves all the previously computed ones, in particular, the $(j+1)$th vector has to be orthogonalized with respect to the first j vectors. This has two main implications when implementing an Arnoldi-based eigensolver: (a) the storage and computational requirements are much higher than in the case of Lanczos, growing as the iteration proceeds and more vectors become available, and (b) the required vector inner product operations are much more numerous than in the case of Lanczos and this has a potential impact on the performance of parallel implementations.

The problem of high storage and computational requirements in the Arnoldi process is solved by so-called restarted Arnoldi methods. These methods allow the Krylov subspace to grow up to a certain dimension, say m, thus limiting the amount of memory required to store the basis to just m vectors. As a consequence, these methods usually need to perform several Arnoldi processes of length m at most, and each of these processes is built upon the results of the previous one by means of some restarting strategy. The simplest restarting scheme, usually called explicit restart, consists in taking an approximate eigenvector as the initial vector for the next Arnoldi process. Other restarting strategies, such as the implicit restart, have been proposed in order to make the whole iteration more effective in terms of convergence, see [7] and references therein. In this paper, we will focus primarily on the simplest explicitly restarted Arnoldi method, although the results could also be extended to solvers with more sophisticated restarting strategies.

The problem of potentially poor parallel performance is addressed in this paper. Our main goal is to improve the scalability of Arnoldi-based eigensolvers, that is, to try to maintain a good speed-up as the number of processors increase. This work is framed in the context of the SLEPc project. SLEPc, the Scalable Library for Eigenvalue Problem Computations [8,9], is a software library for the solution of large, sparse eigenvalue problems on parallel computers, which can be used for the solution of problems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic. SLEPc provides a collection of eigensolvers, including Arnoldi and Lanczos solvers, and also has built-in support for spectral transformations such as shift-and-invert or spectrum folding. SLEPc builds on top of PETSc, the Portable, Extensible Toolkit for Scientific Computation [10], and extends it with all the functionality necessary for the solution of eigenvalue problems.

The scalability of eigensolvers is especially important in challenging applications that require the solution of a sequence of very large-scale eigenvalue problems. One such application can be found for instance in electronic structure calculations, where for large semiconductor nanostructure systems of about one million atoms it is necessary to make use of hundreds of processors [11].

In this paper, we will assume that the algorithms are implemented in a distributed-memory platform with a message-passing paradigm and that matrices and vectors are distributed in the usual way, that is, assigning a contiguous block of rows to each processor. If we further assume that the matrix–vector product operation can be implemented efficiently in parallel (e.g. in finite-element applications with an appropriate partitioning of the discretization mesh), then improving scalability amounts to alleviating the performance penalty induced by global communication operations. It is well known that these operations imply a global synchronization point for all processors and, in addition, in distributed-memory platforms its cost grows with the number of processors. In the case of Arnoldi eigensolvers, global communications are required in vector norms and inner products, in particular those required during the orthogonalization and normalization of Arnoldi vectors. Typically, orthogonalization in this context is carried out by means of some variant of the Gram-Schmidt process. As discussed in Section 2, the variant of choice for our requirements will be Classical Gram-Schmidt with iterative refinement.

The Arnoldi process described so far is inherently sequential in the sense that Arnoldi vectors are produced one at a time. In order to obtain a new vector, the method has to multiply the previous vector by matrix A, then orthogonalize the result with respect to all the previous vectors, and finally normalize it. Other authors have already attempted to enhance Arnoldi eigensolvers in terms of parallel efficiency, with approaches that

try to coarsen the grain of computation by working with several vectors at a time. This can be achieved by either precomputing in advance a small number of vectors so that they are available for the orthogonalization stage [12–15], or by addressing the problem in the context of block-Krylov eigensolvers [16,17]. In contrast, the approach presented in this paper is still compatible with the simpler, single-vector scheme of the Arnoldi process. Our approach consists in rearranging the computations, in some cases interleaving the three basic steps of the process (basis expansion, orthogonalization and normalization), and even deferring some of the operations for the next iteration. This rearrangement attempts to being able to group together several global communication operations in order to reduce the number of synchronization points in the algorithm. Note that this reordering of operations may compromise the numerical stability of the Arnoldi process. For this reason, great emphasis is placed on maintaining the numerical properties of the new algorithms.

The text is organized as follows. Section 2 reviews the Arnoldi process, setting the notation that will be used throughout the paper. This section also provides an overview of orthogonalization algorithms used in this context. The new proposed algorithms are described in detail in Section 3. Some particular details of the implementation in SLEPc are discussed in Section 4. The numerical properties of the new algorithms are analyzed in Section 5. Section 6 presents timing results measured on different parallel platforms, which illustrate the benefits of the new algorithms in terms of parallel performance. We conclude with a discussion in Section 7.

2. Review of the Arnoldi process and Gram-Schmidt variants

We center our discussion on the standard matrix eigenvalue problem, $Ax = \lambda x$, where A is a real non-symmetric (or complex non-Hermitian) square matrix of order n. The goal is to compute a small subset of the n possible eigensolutions (λ_i, x_i), $i = 1, \ldots, k$, $k \ll n$, where the scalar λ_i is called the eigenvalue and the n-vector x_i is called the eigenvector.

2.1. Arnoldi eigensolvers

Arnoldi methods compute approximations to the eigenvalues and eigenvectors by means of a Rayleigh-Ritz procedure that realizes an orthogonal projection onto the Krylov subspace. More precisely, given an initial vector v_1 of norm 1, the method performs m steps of the Arnoldi process in order to build an orthonormal basis of the mth Krylov subspace associated with A and v_1,

$$\mathscr{K}_m(A, v_1) := \operatorname{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}. \tag{1}$$

The Arnoldi process is illustrated schematically in Algorithm 1, where V_m denotes the $n \times m$ matrix whose columns are the vectors $v_1, \ldots, v_m$, and H_m denotes the $m \times m$ upper Hessenberg matrix whose non-zero entries are $h_{i,j}$. Lines (1.1)–(1.3) represent the three main operations required to build a new Arnoldi vector v_{j+1}. In addition to the Arnoldi vector, in iteration j the jth column of matrix H_m is also computed, being $h_{i,j}$, $i = 1, \ldots, j$, the Fourier coefficients calculated during the orthogonalization and $h_{j+1,j}$ the norm computed in line (1.3). The working vector w_{j+1} represents the candidate new Arnoldi vector both before and after orthogonalization. The orthogonalization stage will be described in detail later in this section, where the notation will be made more precise. Note that the distinctness of these three steps will remain clear throughout this section but that separation will become somewhat blurred in the new algorithms proposed in Section 3.

Algorithm 1. Arnoldi Process

Input: Matrix A, number of steps m, and initial vector v_1 of norm 1
Output: $(V_m, H_m, v_{m+1}, h_{m+1,m})$ so that $AV_m - V_mH_m = h_{m+1,m}v_{m+1}e_m^{\mathrm{T}}$

for $j = 1, 2, \ldots, m$

Expand the basis : $w_{j+1} = Av_j$ (1.1)

Orthogonalize w_{j+1} with respect to $\{v_1, v_2, \ldots, v_j\}$ (1.2)

Normalize : $h_{j+1,j} = \|w_{j+1}\|_2, \quad v_{j+1} = w_{j+1}/h_{j+1,j}$ (1.3)

end

It can be shown that the columns of V_m computed by Algorithm 1 constitute an orthonormal basis of the Krylov subspace $\mathscr{K}_m(A, v_1)$. The Arnoldi process is numerically superior to the straightforward approach of computing all the basis vectors and then orthogonalizing them. The reason for this is that the trivial basis of the Krylov subspace, $\{v_1, Av_1, \ldots, A^{m-1}v_1\}$, is very ill-conditioned and linear independence is lost very rapidly in finite precision. The Arnoldi process avoids this by orthogonalizing a vector that is a candidate for expanding the basis as soon as it is generated.

If vector w_{j+1} vanishes after orthogonalization, then it means that the new vector is linearly dependent with respect to the previous Arnoldi vectors or, in other words, that the Krylov subspace is exhausted. In practical implementations, this situation should be detected in order to avoid a division by zero in line (1.3), and a new Arnoldi process should then be started. This implementation detail is omitted from all the algorithms presented in this paper. In finite precision arithmetic, the question of detecting linear dependence is more subtle and will be further discussed below in the context of orthogonalization methods.

From Algorithm 1, the following two relations can be derived,

$$AV_m - V_m H_m = h_{m+1,m} v_{m+1} e_m^{\mathrm{T}}, \tag{2}$$

$$V_m^{\mathrm{T}} A V_m = H_m, \tag{3}$$

where e_m denotes the mth vector of the canonical basis. Eq. (3) follows from Eq. (2) by pre-multiplying both sides of the equation by V_m^{T} and making use of the orthonormality of the Arnoldi vectors. The approximate eigenvalues λ_i provided by the projection process onto the Krylov subspace are the eigenvalues of H_m, which are called Ritz values. The approximate eigenvector (or Ritz vector) associated with λ_i is defined by $z_i = V_m y_i$, where y_i is the corresponding eigenvector of H_m. With these definitions, the residual norm associated with a Ritz pair can be computed as

$$\|(A - \lambda_i I) z_i\|_2 = h_{m+1,m} |e_m^{\mathrm{T}} y_i|. \tag{4}$$

In practical implementations, this relation is typically used for checking convergence of the eigenpairs, since $h_{m+1,m}|e_m^{\mathrm{T}} y_i|$ can be computed very cheaply. However, in finite precision arithmetic, Eq. (4) is satisfied only approximately and, in some cases, this may result in a wrong value being accepted as converged.

2.2. Gram-Schmidt orthogonalization

We now turn our attention to the orthogonalization stage. In the Arnoldi process, since only one vector at a time has to be orthogonalized, Gram-Schmidt orthogonalization procedures come in very naturally. However, simple versions of Gram-Schmidt will not be reliable enough from a numerical point of view, as will be discussed shortly. This problem can be solved by introducing reorthogonalization, which requires about twice as many operations. Another alternative would be to perform the orthogonalization via Householder reflectors as proposed in [18], but also at the expense of doubling the computational cost along with an increase in the complexity of the implementation. Since both Gram-Schmidt with reorthogonalization and the Householder method have roughly the same computational cost, and all of them perform similarly in parallel (see [19] for a comparison), we have opted for the Gram-Schmidt procedures for devising the new algorithms.

In general terms, Gram-Schmidt orthogonalization procedures operate in the following way. Given an orthonormal set of vectors $\{v_1, v_2, \ldots, v_j\}$, which will

Algorithm 2. Arnoldi Process based on CGS

for $j = 1, 2, \ldots, m$

$$w_{j+1} = Av_j$$

$$h_{1:j,j} = V_j^{\mathrm{T}} w_{j+1} \tag{2.1}$$

$$w_{j+1} = w_{j+1} - V_j h_{1:j,j} \tag{2.2}$$

$$h_{j+1,j} = \|w_{j+1}\|_2$$

$$v_{j+1} = w_{j+1}/h_{j+1,j}$$

end

Algorithm 3. Arnoldi Process based on MGS

for $j = 1, 2, \ldots, m$

$w_{j+1} = Av_j$

for $i = 1, \ldots, j$

$h_{i,j} = v_i^{\mathrm{T}} w_{j+1}$ (3.1)

$w_{j+1} = w_{j+1} - h_{i,j} v_i$ (3.2)

end

$h_{j+1,j} = \|w_{j+1}\|_2$

$v_{j+1} = w_{j+1}/h_{j+1,j}$

end

be denoted in matrix form as V_j, and another vector w_{j+1}, the objective is to compute a new vector v_{j+1} so that $\|v_{j+1}\|_2 = 1$, $v_i^{\mathrm{T}} v_{j+1} = 0$ if $i \leqslant j$ and $\mathrm{span}\{v_1, v_2, \ldots, v_j, v_{j+1}\} = \mathrm{span}\{v_1, v_2, \ldots, v_j, w_{j+1}\}$. This is achieved by computing the orthogonal projection of w_{j+1} onto $\mathrm{span}\{v_1, v_2, \ldots, v_j\}$. This projection is then subtracted from the original vector and the result is normalized to obtain v_{j+1}. Note that for our purposes we will consider the normalization a separate step, as made explicit in Algorithm 1.

The above procedure is equivalent to computing $(I - V_j V_j^{\mathrm{T}}) w_{j+1}$ and normalizing the result, that is, applying the orthogonal projector onto the orthogonal complement of $\mathrm{span}\{v_1, v_2, \ldots, v_j\}$. Note that this can also be written as $(I - v_1 v_1^{\mathrm{T}})(I - v_2 v_2^{\mathrm{T}}) \cdots (I - v_j v_j^{\mathrm{T}}) w_{j+1}$, that is, the projection can be carried out with respect to each vector at a time. The two formulations give rise to the Classical Gram-Schmidt (CGS) and Modified Gram-Schmidt (MGS) algorithms, respectively. The two basic Arnoldi variants resulting from them are shown in Algorithms 2 and 3. In Algorithm 2 we introduce the notation $h_{1:j,j}$ to denote the column vector whose components are the first j entries of column j of matrix H.

CGS and MGS are mathematically equivalent. However, when implemented in finite precision arithmetic they behave rather differently (the interested reader is referred to [20] for background material on Gram-Schmidt error analysis). In particular, the level of orthogonality of the computed vectors, $\|I - V_j^{\mathrm{T}} V_j\|$, may be large. It is well known that MGS provides a much better quality of orthogonality than CGS. This is the reason why MGS is usually preferred in linear solvers such as GMRES. In the context of Arnoldi-based eigensolvers, the loss of orthogonality of the computed basis can affect the reliability of the computed eigenpairs, [21]. Therefore, a robust implementation of the Arnoldi process must guarantee that the new vector v_{j+1} is orthogonal to full working precision with respect to the columns of V_j. In this scenario, even the MGS scheme is not sufficient in many cases and large rounding errors can still occur. A cure for this is to resort to double orthogonalization, as described next.

Algorithm 4. Arnoldi with Reorthogonalization (AR)

for $j = 1, 2, \ldots, m$

$w_{j+1} = Av_j$

$h_{1:j,j} = V_j^{\mathrm{T}} w_{j+1}$ (4.1)

$w_{j+1} = w_{j+1} - V_j h_{1:j,j}$ (4.2)

$c_{1:j,j} = V_j^{\mathrm{T}} w_{j+1}$ (4.3)

$w_{j+1} = w_{j+1} - V_j c_{1:j,j}$ (4.4)

$h_{1:j,j} = h_{1:j,j} + c_{1:j,j}$ (4.5)

$h_{j+1,j} = \|w_{j+1}\|_2$ (4.6)

$v_{j+1} = w_{j+1}/h_{j+1,j}$

end

2.3. Gram-Schmidt with reorthogonalization

The simplest possibility to guard against large rounding errors in Gram-Schmidt orthogonalization is to take the resulting vector and perform a second orthogonalization. Algorithm 4 shows the Arnoldi process based on CGS with reorthogonalization.

Algorithm 4 is obtained from Algorithm 2 by adding lines (4.3)–(4.5). Note that, in exact arithmetic, the coefficients $c_{1:j,j}$ are zero and therefore these lines have no effect. However, this is not the case in finite precision arithmetic, where $c_{1:j,j}$ can be thought of as a correction to $h_{1:j,j}$, which is not necessarily small.

In cases where large rounding errors have not occurred in first place, reorthogonalization is superfluous and could be avoided. On the other hand, it could happen that large rounding errors occur also in the reorthogonalization stage, thus requiring yet another reorthogonalization. In general, an iterative scheme can be defined in which a simple criterion, such as the one described in [22], is used to determine whether the computed vector is good enough or requires further refinement. A detailed description of iterative Gram-Schmidt procedures can be found in [23,24]. For simplicity, in the algorithms presented in this paper only one reorthogonalization is shown, although in a practical implementation a second reorthogonalization should be allowed if necessary as explained later in this section.

Algorithm 5. Arnoldi with Selective Reorthogonalization (ASR)

for $j = 1, 2, \ldots, m$

$w_{j+1} = Av_j$

$h_{1:j,j} = V_j^{\mathrm{T}} w_{j+1}$ (5.1)

$\rho = \|w_{j+1}\|_2$ (5.2)

$w_{j+1} = w_{j+1} - V_j h_{1:j,j}$

$h_{j+1,j} = \|w_{j+1}\|_2$ (5.3)

if $h_{j+1,j} < \eta\rho$ (5.4)

$c_{1:j,j} = V_j^{\mathrm{T}} w_{j+1}$ (5.5)

$w_{j+1} = w_{j+1} - V_j c_{1:j,j}$

$h_{1:j,j} = h_{1:j,j} + c_{1:j,j}$

$h_{j+1,j} = \|w_{j+1}\|_2$ (5.6)

end if

$v_{j+1} = w_{j+1}/h_{j+1,j}$ (5.7)

end

An Arnoldi variant with selective reorthogonalization is shown in Algorithm 5, where the reorthogonalization is carried out whenever the criterion

$$h_{j+1,j} < \eta\rho \tag{5}$$

is satisfied for some constant parameter $\eta < 1$. This criterion compares the norm of w_{j+1} before (ρ) and after ($h_{j+1,j}$) orthogonalization. Experience has shown that a safe value for η is $1/\sqrt{2}$ as proposed in [22,25]. A smaller value will result in less reorthogonalizations but more risk of numerical instability.

This same criterion can also be used to detect Krylov subspace exhaustion. If it is satisfied after reorthogonalization, a second reorthogonalization step should be made, making the whole process more robust when the Krylov subspace is nearly exhausted. If it is satisfied again after two reorthogonalizations, then this indicates that vector w_{j+1} (numerically) lies in the span of the columns of V_j so that the Arnoldi process cannot be continued any further. Most of the time, only one reorthogonalization is necessary at most, and these checks do not require extra computations.

A similar reorthogonalization scheme might be considered for the Modified Gram-Schmidt variant. However, the resulting numerical quality is about the same, as pointed out in [23]. In this work, we do not consider MGS variants since they are less scalable, as discussed below. Indeed, practical implementations of Arnoldi for parallel platforms are usually based on CGS with reorthogonalization, even in the case of linear solvers such as GMRES [26].

2.4. Parallel implementation of the Arnoldi process

Assuming a message-passing paradigm with standard data distribution of vectors and matrices, i.e. by blocks of rows, parallelization of the Arnoldi process amounts to carrying out the three stages in parallel:

(1) *Basis expansion*. In the simplest scenario, this stage consists in a sparse matrix–vector product. In most applications, this operation can be parallelized quite efficiently. This is the case in mesh-based computations, in which data needs to be exchanged only among neighboring processes if the mesh is correctly partitioned.
(2) *Orthogonalization*. This stage consists of a series of vector operations such as inner products, additions, and multiplications by a scalar.
(3) *Normalization*. From the parallelization viewpoint, the computation of the norm is equivalent to a vector inner product.

We will center our discussion on the orthogonalization and normalization stages, and assume that the implementation of the basis expansion is scalable. The test cases used for the performance analysis presented in Section 6 have been chosen so that basis expansion has a negligible impact on scalability.

Since vector addition and scaling are trivially parallelizable operations with no associated communication, the efficiency of the orthogonalization and normalization stages depends solely on how the required inner products are performed. The computation of a vector inner product requires a global reduction operation (in particular, an all-reduce addition), which may pose a significant burden to scalability because it represents a global synchronization point in the algorithm and, in addition, in distributed-memory platforms its cost grows with the number of processes.

In order to pursue scalability, the number of required global reduction operations should be reduced as much as possible. A general strategy for achieving this goal is to group together several inner products in a single reduction. In this way, it is possible to accomplish all the individual inner products with just one synchronization point and roughly the same communication cost as just one inner product. As a consequence, the scalability of the different variants of Gram-Schmidt orthogonalization depends on the data dependencies of inner products. This is the reason why the CGS versions scale better than the MGS counterparts. In Algorithm 2, line (2.1) represents multiple inner products that can be grouped together in a single reduction, whereas in Algorithm 3, the inner products in line (3.1) have to be done separately due to a data dependency with respect to the result of line (3.2). On the other hand, CGS is also preferred to MGS because it usually provides higher Mflop rates even for one processor, due to a better data access pattern. In the sequel, only CGS variants are considered.

In Algorithm 4, three global reduction operations are required in lines (4.1), (4.3), and (4.6). In Algorithm 5, two global reduction operations are required (on one hand, lines (5.1) and (5.2), which can be grouped together, and, on the other hand, line (5.3)), plus two additional ones in the case that the reorthogonalization criterion is satisfied (lines (5.5) and (5.6)). In this work, we consider modifications of Algorithms 4 and 5 in such a way that the number of global reduction operations are minimized. In the case of Algorithm 4 it will be possible to accomplish the computation with just one synchronization point, whereas for Algorithm 5 two synchronization points will be necessary at least. The minimization of global reduction operations will be achieved by applying the techniques described below. The new algorithms are described in detail in Section 3.

2.5. Techniques for scalability enhancement

The following three techniques have been used for devising the new Arnoldi algorithms presented in Section 3. Note that these techniques consist in applying some transformations to the algorithms, which are equivalent in exact arithmetic but, when implemented in finite precision, they may have a significant impact on numerical stability. This aspect is analyzed in Section 5.

2.5.1. Estimation of the norm

The main objective of this technique is to avoid the explicit computation of the Euclidean norm of w_{j+1} and, instead, use an estimation based on other quantities easily computable or already available in the algorithm.

The basic idea, which will be fully developed in Section 3, is to estimate the norm starting from the original norm (prior to the orthogonalization), by simply applying the Pythagorean theorem. The original norm is either already available, as in Algorithm 5 (line (5.2)), since it is required for the selective reorthogonalization criterion, or can be computed more efficiently since its associated reduction is susceptible of being integrated in a previous reduction (e.g. line (4.3) in Algorithm 4).

This technique was already used by Frank and Vuik [19] for improving the parallel performance of a GCR linear solver. A similar strategy has been used in the context of the preconditioned conjugate gradient method for linear systems, in order to reduce the number of synchronizations from 2 to 1 per iteration, see [27,28].

2.5.2. Delayed normalization

This technique was originally proposed by Kim and Chronopoulos [29] in the context of an Arnoldi eigensolver with no reorthogonalization. The basic idea of this technique is to defer the normalization (including the computation of the norm), moving it from the end of step j to the first global reduction operation in step $j+1$. This makes it possible to eliminate a synchronization point by merging the two global communications. The side effect is that now the basis expansion has to be carried out with a vector that has not been normalized yet. This is not a problem provided that all the computed quantities are corrected as soon as the norm is available.

Note that this technique is not compatible with the estimation of the norm mentioned above.

2.5.3. Delayed reorthogonalization

This technique was introduced by the authors in [30]. The basic idea is quite similar to delayed normalization. In this case, the reorthogonalization stage is what is deferred from step j to step $j+1$. In this way, the global reductions associated with the first orthogonalization of vector v_{j+1} and the reorthogonalization of vector v_j are combined together, resulting in a single synchronization. Note that this concept is only applicable to variants with unconditional reorthogonalization (without selective criterion). Although the idea is rather simple, its implementation is far more difficult than the other techniques because each step is working with data from three consecutive Arnoldi iterations. This potentially involves more numerical complications because some approximations are used in place of values not available until the next iteration. The algorithms presented in [30] displayed good parallel performance but are less robust numerically than the ones proposed in this paper.

3. Proposed algorithms

In this section, three new algorithms are proposed that make use of some of the techniques hinted above in order to combine the reduction operations of the Arnoldi process. These algorithms are based on the reference AR (Algorithm 4) and ASR (Algorithm 5) algorithms. Although, in some cases, the floating point operation count is increased, they can be implemented with fewer synchronization points. As a consequence, the new algorithms will be competitive when the number of processors is sufficiently large.

Algorithm 6. Arnoldi with Reorthogonalization and Estimated Norm (AREN), synchronizations are shown enclosed in frames.

for $j = 1, 2, \ldots, m$
$w_{j+1} = Av_j$
$\boxed{h_{1:j,j} = V_j^T w_{j+1}}$ (6.1)
$w_{j+1} = w_{j+1} - V_j h_{1:j,j}$
$\boxed{\begin{array}{l} c_{1:j,j} = V_j^T w_{j+1} \\ \rho = \|w_{j+1}\|_2 \end{array}}$ (6.1) (6.2)
$w_{j+1} = w_{j+1} - V_j c_{1:j,j}$
$h_{1:j,j} = h_{1:j,j} + c_{1:j,j}$
$h_{j+1,j} = \sqrt{\rho^2 - \sum_{i=1}^{j} c_{i,j}^2}$ (6.3)
$v_{j+1} = w_{j+1}/h_{j+1,j}$
end

3.1. Arnoldi with reorthogonalization and estimated norm

Algorithm 6, Arnoldi with Reorthogonalization and Estimated Norm (AREN), uses the first technique mentioned in the previous section for estimating the norm of w_{j+1} after reorthogonalization. This algorithm is based on the AR algorithm (Algorithm 4) and it addresses the synchronization point present in its line (4.6). The estimation of the norm is based on the following relation due to line (4.4),

$$w_{j+1} = w'_{j+1} + V_j c_{1:j,j}, \tag{6}$$

where w_{j+1} and w'_{j+1} denote the vector before and after reorthogonalization, respectively, and $V_j c_{1:j,j}$ is the vector resulting from projecting w_{j+1} onto span$\{v_1, v_2, \ldots, v_j\}$. In exact arithmetic, w'_{j+1} is orthogonal to this subspace and it is possible to apply the Pythagorean theorem to the right-angled triangle formed by these three vectors,

$$\|w_{j+1}\|_2^2 = \|w'_{j+1}\|_2^2 + \|V_j c_{1:j,j}\|_2^2. \tag{7}$$

Since the columns of V_j are orthonormal, the wanted norm can be computed as

$$\|w'_{j+1}\|_2 = \sqrt{\|w_{j+1}\|_2^2 - \sum_{i=1}^{j} c_{i,j}^2}. \tag{8}$$

In deriving Eq. (8), we have assumed that w'_{j+1} is orthogonal to span$\{v_1, v_2, \ldots, v_j\}$ and that the columns of V_j are orthonormal. These assumptions are not necessarily satisfied in finite precision arithmetic, and for this reason we state that it is an estimation of the norm. Usually, this estimation is very accurate.

Algorithm 7. Arnoldi with Selective Reorthogonalization and Estimated Norm (ASREN), synchronizations are shown enclosed in frames.

for $j = 1, 2, \ldots, m$
$w_{j+1} = Av_j$
$\boxed{h_{1:j,j} = V_j^T w_{j+1}}$ (7.1)
$\boxed{\rho = \|w_{j+1}\|_2}$ (7.2)
$w_{j+1} = w_{j+1} - V_j h_{1:j,j}$
$h_{j+1,j} = \sqrt{\rho^2 - \sum_{i=1}^{j} h_{i,j}^2}$ (7.3)
if $h_{j+1,j} < \eta\rho$
$\boxed{\begin{array}{l} c_{1:j,j} = V_j^T w_{j+1} \\ \sigma = \|w_{j+1}\|_2 \end{array}}$ (7.4) (7.5)
$w_{j+1} = w_{j+1} - V_j c_{1:j,j}$
$h_{1:j,j} = h_{1:j,j} + c_{1:j,j}$
$h_{j+1,j} = \sqrt{\sigma^2 - \sum_{i=1}^{j} c_{i,j}^2}$ (7.6)
end
$v_{j+1} = w_{j+1}/h_{j+1,j}$ (7.7)
end

because the $c_{1:j,j}$ coefficients are very small compared to $\|w_{j+1}\|_2^2$, that is, w_{j+1} and w'_{j+1} have roughly the same norm. In very exceptional cases, $\|w_{j+1}\|_2^2$ is as small as $c_{1:j,j}$ and it is safer to discard the estimation and to compute the norm explicitly. This estimation is also successfully used for the linear dependence check explained in Section 2.3. These implementation details are omitted from Algorithms 6 and 7 in order to improve their readability.

In this algorithm, the communications associated with the second multiple vector dot product (line 6.1) and the computation of the norm (line 6.2) can be joined together. Therefore, it can be implemented with two synchronization points per iteration, one less than the AR algorithm. With respect to the computational cost, line 6.3 represents an overall flop count increase of $O(m^2)$.

3.2. Arnoldi with selective reorthogonalization and estimated norm

Algorithm 7, Arnoldi with Selective Reorthogonalization and Estimated Norm (ASREN), is based on the ASR algorithm (Algorithm 5) and uses the same technique as the previous algorithm to estimate two norms of w_{j+1}: the norm after the initial orthogonalization (line 7.3) and the norm after the reorthogonalization if the criterion is satisfied (line 7.6).

The same discussion as above can be applied to these norm estimations. In this algorithm, the first estimation (line 7.3) may be inaccurate if w_{j+1} is not fully orthogonal after the first orthogonalization. This does not represent a problem for the normalization, because in that case the criterion would force a reorthogonalization step and then a new norm estimation would be computed. On the other hand, the criterion may seem less trustworthy due to the use of estimations instead of true norms. However, numerical experiments described in Section 5 reveal that this algorithm is as robust as ASR.

This algorithm is computationally cheaper than ASR because the estimation of $h_{j+1,j}$ requires fewer operations than the explicit computation of the norm since $m \ll n$. The parallel implementation of this algorithm can be built with two all-reduce additions, joining the multiple dot product operations 7.1 and 7.4 with the computation of the norms 7.2 and 7.5, respectively. Therefore, this algorithm requires one synchronization to compute the orthogonalization and another one if a reorthogonalization is needed. This should lead to a significant parallel performance improvement on the ASR algorithm, which needs two synchronizations to compute the orthogonalization and evaluate the criterion, and another two if a reorthogonalization is required.

Another advantage over the ASR algorithm is that there is no parallel performance penalty for using selective reorthogonalization instead of unconditional reorthogonalization, because this algorithm does not need the extra synchronization required by the ASR algorithm to evaluate the criterion.

3.3. Arnoldi with delayed reorthogonalization

The selective reorthogonalization scheme in the Arnoldi algorithm reduces the number of floating point operations with respect to variants with unconditional reorthogonalization. However, in practice the benefit is rather small because the reorthogonalization step is very frequent. For example, in the test cases used in Section 5, the average reorthogonalization ratio is around 86%, that is, the criterion is almost always satisfied.

Since the number of synchronizations using the selective reorthogonalization scheme can not be reduced further, an algorithm with unconditional reorthogonalization and only one synchronization should have better parallel performance.

This is the key idea of Algorithm 8, Arnoldi with Delayed Reorthogonalization (ADR), which is based on the AR algorithm (Algorithm 4) and combines the last two techniques from Section 2.5. In this algorithm, the basis expansion is performed as soon as a vector in the "right" direction is available, and the reorthogonalization and normalization steps are delayed to the next iterations. Because of this, the ADR algorithm has to perform computations with approximate vectors whenever the final value is not available, as explained in the next paragraphs.

Algorithm 8. Arnoldi with Delayed Reorthogonalization (ADR), framed communications are grouped in only one synchronization.

for $j = 1, 2, \ldots, m$
  $w_{j+1} = Aw_j$
  $\boxed{h_{1:j,j} = [V_{j-2}, u_{j-1}, w_j]^T w_{j+1}}$ (8.1)
  if $j > 1$
    $\boxed{\rho = \|w_j\|_2}$ (8.2)
    $u_j = \rho^{-1} w_j$
    $w_{j+1} = \rho^{-1} w_{j+1}$
    $h_{1:j-1,j} = \rho^{-1} h_{1:j-1,j}$
    $h_{j,j} = \rho^{-2} h_{j,j}$
  end
  $w_{j+1} = w_{j+1} - [V_{j-2}, u_{j-1}, u_j] h_{1:j,j}$
  if $j > 1$
    $\boxed{c_{1:j-1,j-1} = [V_{j-2}, u_{j-1}]^T w_j}$ (8.3)
    $w_j = w_j - [V_{j-2}, u_{j-1}] c_{1:j-1,j-1}$
    $h_{1:j-1,j-1} = h_{1:j-1,j-1} + c_{1:j-1,j-1}$
  end
  if $j > 2$
    $\boxed{h_{j-1,j-2} = \|w_{j-1}\|_2}$ (8.4)
    $v_{j-1} = w_{j-1} / h_{j-1,j-2}$
  end
end
$h_{m,m-1} = \|w_m\|_2$
$v_m = w_m / h_{m,m-1}$
$c_{1:m,m} = V_m^T w_{m+1}$
$w_{m+1} = w_{m+1} - V_m c_{1:m,m}$
$h_{1:m,m} = h_{1:m,m} + c_{1:m,m}$
$h_{m+1,m} = \|w_{m+1}\|_2$
$v_{m+1} = w_{m+1} / h_{m+1,m}$

At a given iteration j, this algorithm works with the following vectors: V_{j-2}, whose columns are the final Arnoldi vectors, that is, they have been reorthogonalized and normalized; w_{j-1}, which only needs to be normalized in order to become the $(j-1)$th Arnoldi vector; w_j, which needs both reorthogonalization and normalization; u_j, which is a normalized w_j and therefore an approximation to v_j; and w_{j+1}, which is the new direction of the Arnoldi basis, requiring all the processing.

In order to delay the reorthogonalization, the u_j approximation should be used in the basis expansion and the reorthogonalization of w_{j-1}. This approximation is computed by normalizing w_j just after the first orthogonalization, so it may be not fully orthogonal to V_{j-1}. Therefore, the level of orthogonality among the columns of V_j may not be as good as in previous algorithms, as will be discussed further with the results in Section 5. In addition, the normalization of u_j is also delayed and the unnormalized vector w_j is used instead to compute both w_{j+1} and $h_{1:j,j}$, so these values must be corrected as soon as the norm of w_j is available. If $\rho = \|w_j\|_2$ and $u_j = \rho^{-1} w_j$, then the corrected w_{j+1} is

$$w'_{j+1} = Au_j = A\rho^{-1} w_j = \rho^{-1} A w_j = \rho^{-1} w_{j+1}, \tag{9}$$

and the corrected $h_{1:j,j}$ are

$$h'_{1:j,j} = [V_{j-2}, u_{j-1}, u_j]^{\mathrm{T}} w'_{j+1}. \tag{10}$$

Expanding the last equation yields the following corrections,

$$h'_{1:j-2,j} = V_{j-2}^{\mathrm{T}} w'_{j+1} = V_{j-2}^{\mathrm{T}} \rho^{-1} w_{j+1} = \rho^{-1} V_{j-2}^{\mathrm{T}} w_{j+1} = \rho^{-1} h_{1:j-2,j}, \tag{11}$$

$$h'_{j-1,j} = u_{j-1}^{\mathrm{T}} w'_{j+1} = u_{j-1}^{\mathrm{T}} \rho^{-1} w_{j+1} = \rho^{-1} u_{j-1}^{\mathrm{T}} w_{j+1} = \rho^{-1} h_{j-1,j}, \tag{12}$$

$$h'_{j,j} = u_j^{\mathrm{T}} w'_{j+1} = [\rho^{-1} w_j]^{\mathrm{T}} \rho^{-1} w_{j+1} = \rho^{-2} w_j^{\mathrm{T}} w_{j+1} = \rho^{-2} h_{j,j}. \tag{13}$$

In this algorithm, operations 8.1–8.4 can be implemented in one reduction operation, that is, together with line 8.1 the first line of all three conditional clauses is also executed. This creates a sort of pipeline where several multiple vector dot products and vector norms corresponding to different Arnoldi vectors are computed simultaneously without data dependencies among them. In this pipeline, w_{j+1} is orthogonalized, w_j is reorthogonalized and w_{j-1} is normalized simultaneously, where the approximation u_j is used instead of v_j until it is finally available. The operations that appear after the main loop are intended for flushing this pipeline.

Although the number of floating point operations in this algorithm is increased with respect to the AR algorithm, the number of synchronization points per iteration is reduced to only one. This should give the smallest possible parallel overhead per iteration of all these algorithms.

4. Implementation in SLEPc

In order to check the numerical stability and parallel performance of the proposed algorithms, they have been implemented in the SLEPc library in the context of an explicitly restarted Arnoldi eigensolver. These implementations are available starting from version 2.3.1 [9].

The matrix–vector product at the core of the Arnoldi process is implemented with PETSc's MatMult operation. This operation is optimized for parallel sparse storage and even allows for matrix-free, user-defined matrix–vector product operations. The communication merging needed by the proposed algorithms can be easily accomplished by making use of PETSc's VecDotBegin/VecDotEnd mechanism, which gives the high-level programmer some flexibility for specifying when communications must take place. See [10] for additional details.

As explained in Section 2.3, the selective reorthogonalization criterion is also used to detect the linear dependence of the current Arnoldi vector. The norms required by this verification are already available in all the presented algorithms except in the case of AR. A practical implementation of the AR algorithm (Algorithm 4) must compute the norm of w_{j+1} between lines (4.3) and (4.4), thus adding some floating point operations but no extra synchronization points.

Two more implementation details are related to the case when the Krylov subspace is almost exhausted. One is to allow a third step in the selective reorthogonalization mechanism. The other one is to check the expression inside the square root in Eq. (8), and, if negative, compute the norm explicitly. These two improvements make the whole Arnoldi process slightly more stable in this exceptional case.

The ADR algorithm is not able to compute an orthogonal basis with sufficient accuracy in some cases. This loss of orthogonality affects numerical stability, resulting in convergence stagnation and computation of spurious eigenvalues. In order to prevent this effect, a restart is forced whenever the magnitude of $v_{j-2}^{\mathrm{T}} v_{j-1}$ is above a certain threshold. Our experiments show that a value of 10^{-14} is appropriate in the case of double precision arithmetic. This test is computed simultaneously with operations 8.1–8.4, adding some floating point operations but no extra synchronization.

All these details have been incorporated in the SLEPc implementation of the proposed algorithms.

5. Numerical results

The algorithms proposed in Section 3 and the reference algorithms in Section 2 are not equivalent when using finite precision arithmetic. Therefore, their numerical behavior must be analyzed. According to Braconnier et al. [21], the stability of the Arnoldi process depends on the quality of orthogonality among the Arnoldi vectors. This is measured with the level of orthogonality, which we define as the maximum value of

Table 1
Average number of converged eigenvalues, residual norm and orthogonality level for each of the analyzed algorithms

Algorithm	Converged eigenvalues	Residual	Orthogonality
AR	13.0	2.67×10^{-7}	1.23×10^{-14}
ASR	12.9	5.60×10^{-7}	1.58×10^{-14}
AREN	12.9	6.91×10^{-7}	1.26×10^{-14}
ASREN	13.0	2.60×10^{-7}	2.39×10^{-14}
ADR	9.3	5.46×10^{-8}	3.65×10^{-13}

$\|I - V_m^{\mathrm{T}} V_m\|_F$ at each restart. Additionally, in order to check the accuracy of the computed eigensolutions, the associated residual norms are computed explicitly after the finalization of the Arnoldi process. If this norm is greater than the required tolerance then the estimation in Eq. (4) was incorrect. The residual norm shown in the results below is computed as the maximum of $\|Ax - \lambda x\|_2 / \|\lambda x\|_2$ for every converged eigenvalue.

In this section, we consider an empirical test with a battery of real-problem matrices using the implementation referred to in Section 4 with standard double precision arithmetic. The analysis consists in measuring the level of orthogonality and the residual norm when computing the 10 largest eigenvalues of every matrix from the Harwell-Boeing [31] and NEP [32] collections. Also the number of converged eigenvalues is measured. These 210 matrices come from a variety of real problems and those from the NEP collection are particularly challenging for eigenvalue computations. For this test, the solvers are configured with a tolerance equal to 10^{-7} and a maximum of 50 basis vectors.

The results of this test are summarized in Table 1, which contains the average values for each algorithm. The estimated norm algorithms, AREN and ASREN, seem to have as good numerical properties as the reference algorithms AR and ASR. The level of orthogonality of the basis vectors computed by the ADR algorithm is slightly worse than the other algorithms with reorthogonalization, and the number of converged eigenvalues is clearly inferior. Actually, this algorithm is unable to compute an orthogonal basis with sufficient accuracy in some problems from the test battery, and in these cases the restarted Arnoldi process is unable to converge at all. This effect can be observed more clearly in the histogram plots of Fig. 1, which show the distribution of tests according to the number of converged eigenvalues. The left plot shows the results from the ASREN variant, which computes the 10 required eigenvalues in almost all cases. The rest of the algorithms have equivalent behavior (plots not shown), with the exception of the ADR variant (right plot) which has around 40 cases where the Arnoldi process fails to provide any converged eigenvalues. However, when convergence occurs (81% of the cases) the ADR algorithm behaves similarly to the other variants: it is able to compute all required eigenvalues with an average orthogonality of 1.68×10^{-14}.

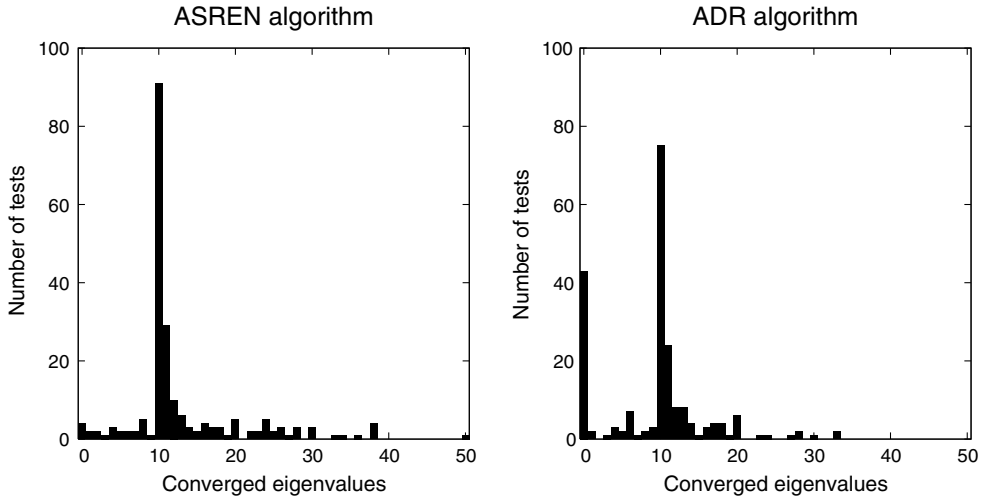

Fig. 1. Distribution of tests classified by number of converged eigenvalues.

6. Performance analysis

In order to compare the parallel efficiency of the proposed Arnoldi variants, several test cases have been analyzed on different parallel platforms. The comparison is carried out in the context of the implementation referred to in Section 4. In all cases, the solver is requested to compute the 10 largest eigenvalues with tolerance equal to 10^{-7} using a maximum of 50 basis vectors. In order to minimize the effect of slight variations in the iteration count, measured wall times are divided by the total number of iterations (including restarts).

Two types of tests are considered. On one hand, matrices arising from real applications are used for measuring the parallel speed-up. This speed-up is calculated as the ratio of elapsed time with p processors to the elapsed time with one processor corresponding to the fastest algorithm. This latter time always corresponds to the ASREN variant, due to the implementation details described in Section 4. On the other hand, a synthetic test case is used for analyzing the scalability of the algorithms, measuring the scaled speed-up (with variable problem size) and Mflop/s per processor.

The first machine used for the tests is a cluster of 55 dual processor nodes with Pentium Xeon processors at 2.8 GHz interconnected with an SCI network in a 2-D torus configuration. Only one processor per node was used in the tests reported in this section.

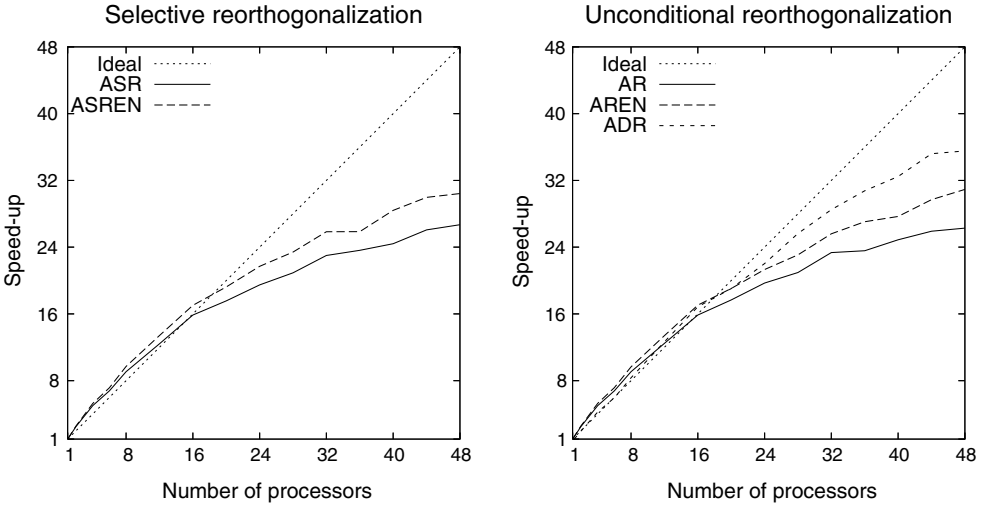

Fig. 2. Measured speed-up for AF23560 matrix in Xeon cluster.

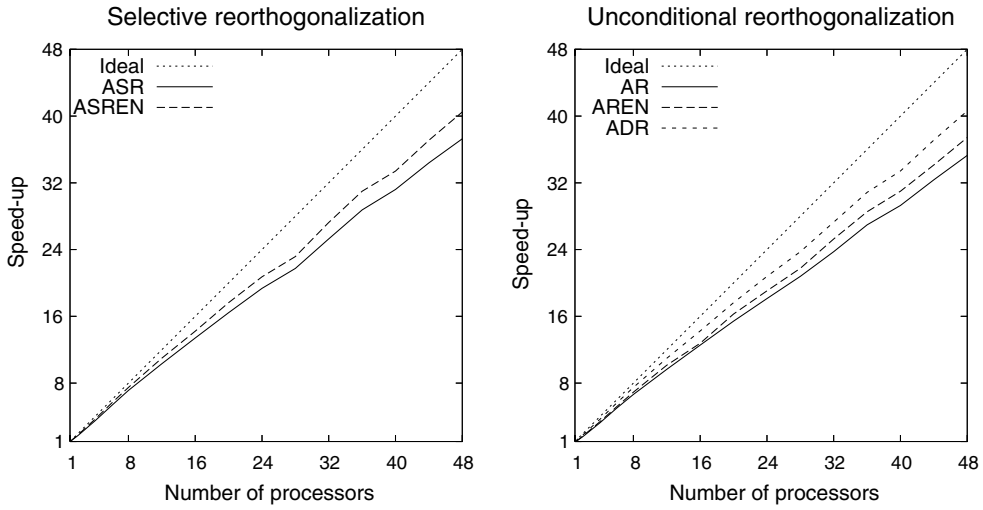

Fig. 3. Measured scaled speed-up for tridiagonal matrix in Xeon cluster.

The speed-up measured in the Xeon cluster for the AF23560 matrix is shown in Fig. 2. This matrix is real non-symmetric of order 23,560 and it is the largest one from the NEP collection [32]. The left plot corresponds to the selective reorthogonalization algorithms ASR and ASREN, whereas the right one corresponds to the algorithms AR, AREN and ADR.

All algorithms show overall good parallel performance with a remarkable performance improvement in the case of the delayed reorthogonalization scheme. Rather surprisingly, the speed-up improvement with the estimated norm variant is not very significant in this case. Although the ADR algorithm carries out more operations than ASREN, it can be the fastest algorithm when the number of processors is sufficiently increased.

Another observation is that there is superlinear speed-up with an intermediate number of processors. This is due to having a fixed problem size, thus decreasing the amount of data assigned to each processor and leading to better cache memory performance as the number of processors increases. This performance gain compensates for the communications cost until the local problem size fits entirely in cache memory.

To minimize this cache effect, the size of local data must be kept constant, that is, the matrix dimension must grow proportionally with the number of processors, p. For this analysis, a tridiagonal matrix with random entries has been used, with a dimension of $10{,}000 \times p$. The results in the Xeon cluster (see Figs. 3 and 4)

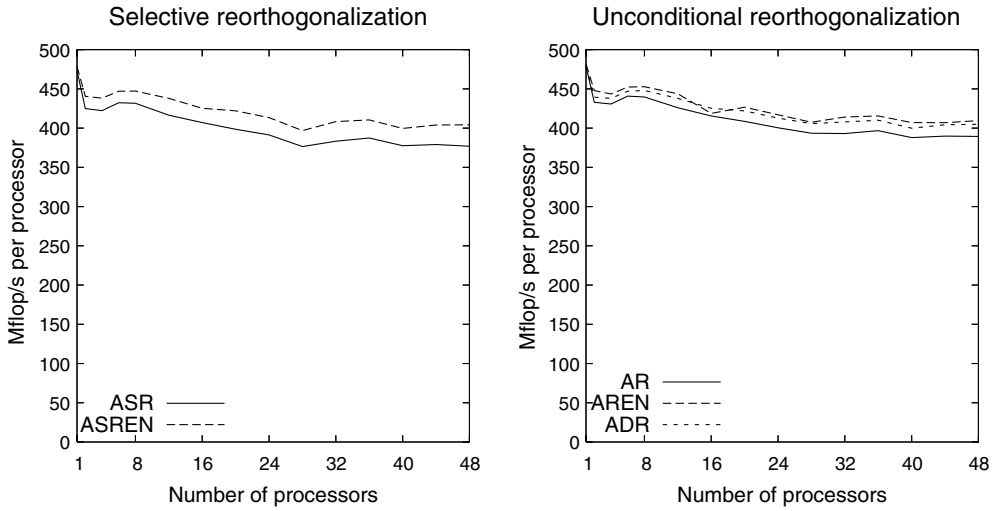

Fig. 4. Measured Mflop/s for tridiagonal matrix in Xeon cluster.

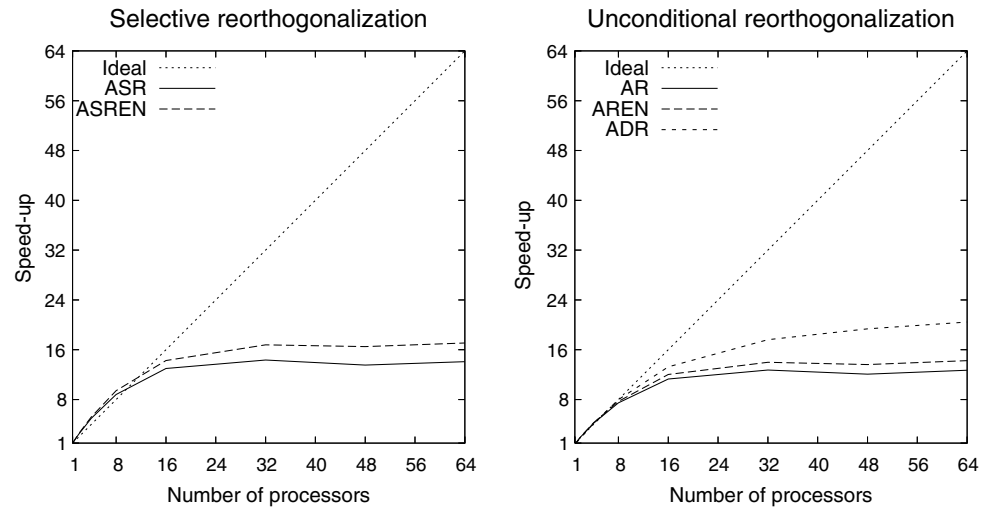

Fig. 5. Measured speed-up for AF23560 matrix in IBM SP.

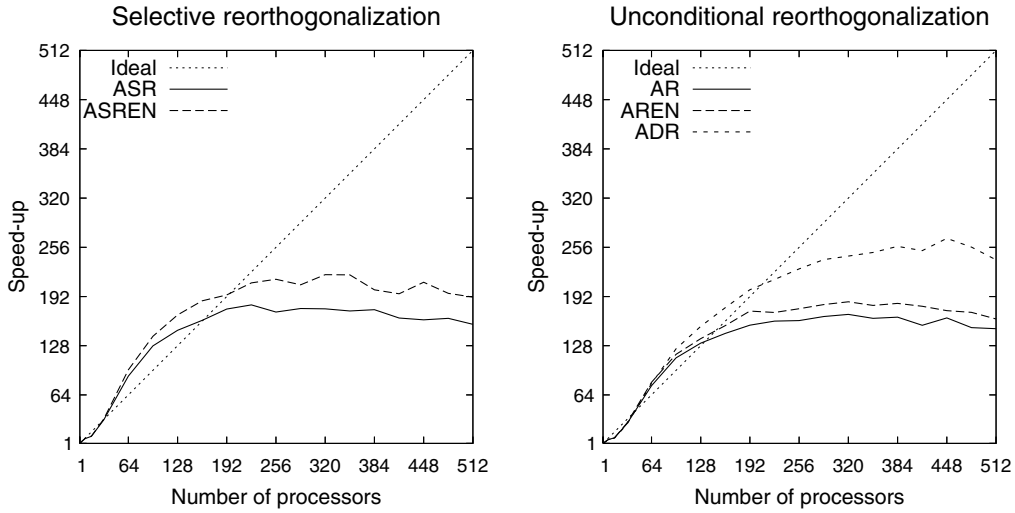

Fig. 6. Measured speed-up for PRE2 matrix in IBM SP.

show overall good parallel performance in all alternatives, with the proposed algorithms having better speed-up and slightly higher Mflop/s rate.

In order to extend the analysis to more processors, the tests were repeated in an IBM SP RS/6000 computer with 380 nodes and 16 processors per node. In this case, although the proposed algorithms show better performance with the AF23560 matrix than the reference versions (see Fig. 5), the overall speed-up is poor because of the relatively small problem size compared with the number of processors.

Fig. 6 shows the performance results using the larger matrix PRE2 of order 659,033 from the University of Florida Sparse Matrix Collection [33]. As before, the ADR variant has a huge performance improvement as the number of processors increases, giving an almost ideal speed-up around 200 processors. The results with the tridiagonal synthetic problem (Figs. 7 and 8) demonstrate that all these algorithms scale well and the proposed algorithms give better performance with a large number of processors.

Also, the tests were repeated in an IBM JS20 cluster of 2406 dual processor nodes with PowerPC 970FX processors at 2.2 GHz interconnected with a Myrinet network. As before, only one processor per node has

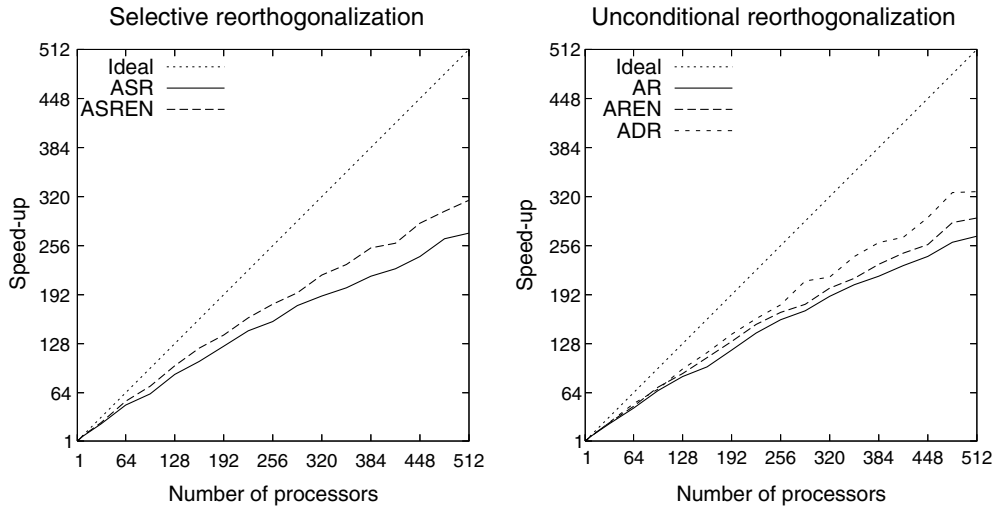

Fig. 7. Measured speed-up for tridiagonal matrix in IBM SP.

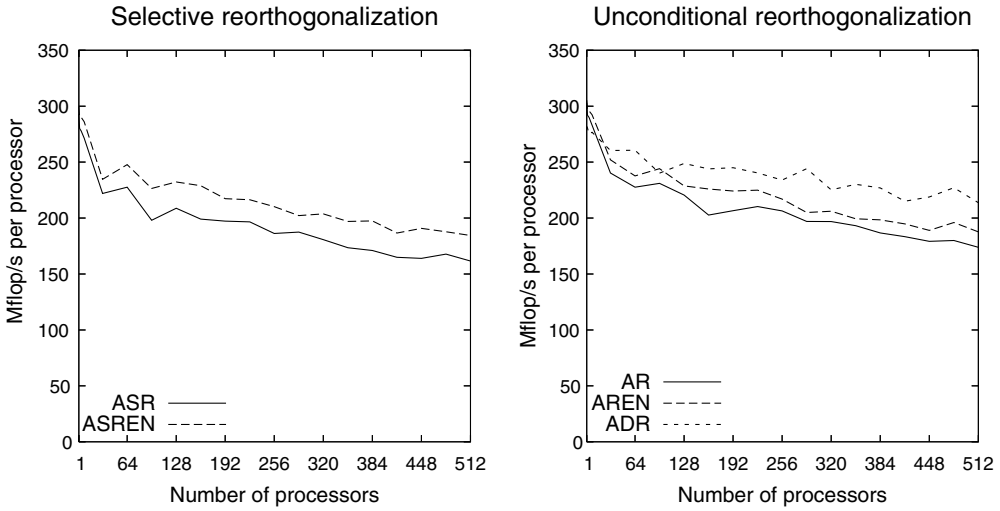

Fig. 8. Measured Mflop/s for tridiagonal matrix in IBM SP.

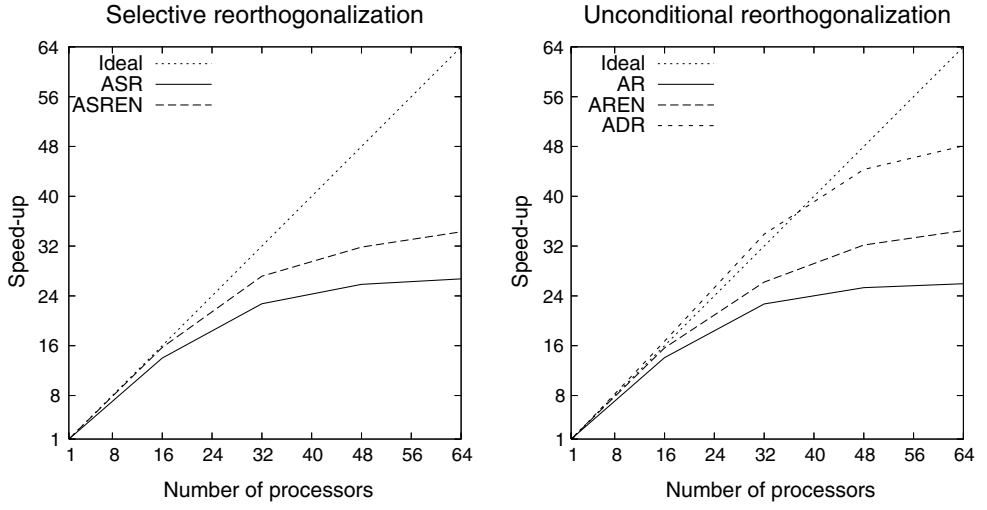

Fig. 9. Measured speed-up for AF23560 matrix in IBM cluster.

been used in the tests. The results with the AF23560 matrix (Fig. 9) show a huge advantage in performance for the ADR algorithm. However, Fig. 10 shows similar performance for all Arnoldi variants with the larger PRE2 matrix. Figs. 11 and 12 show good overall parallel performance for all these algorithms, with no significant differences.

7. Discussion

In this work, three new variants of the Arnoldi algorithm have been proposed in order to improve parallel efficiency in the context of eigensolvers. These algorithms are based on Classical Gram-Schmidt with reorthogonalization and their main goal is to reduce the number of synchronization points in their parallel implementation via a reordering of operations.

The performance results presented in Section 6 show that the proposed algorithms achieve better parallel efficiency in all the test cases analyzed, and scale better when increasing the number of processors.

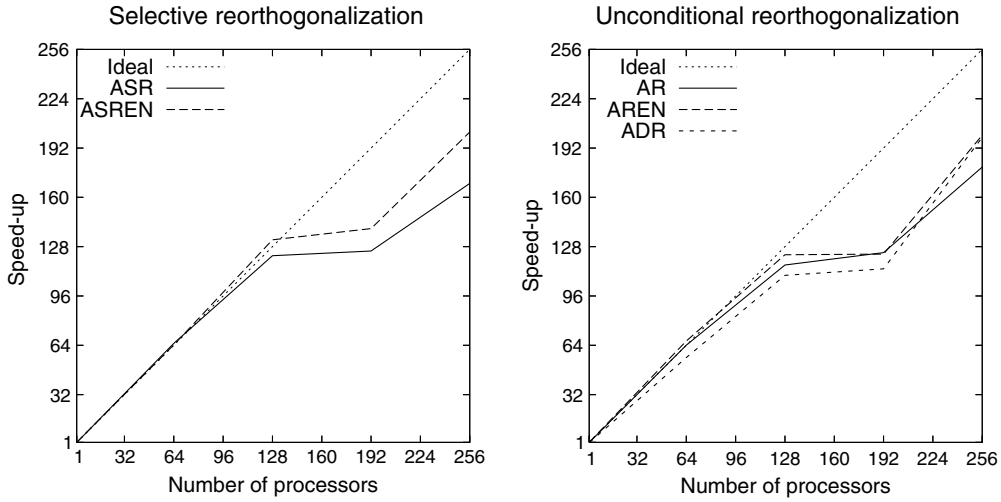

Fig. 10. Measured speed-up for PRE2 matrix in IBM cluster.

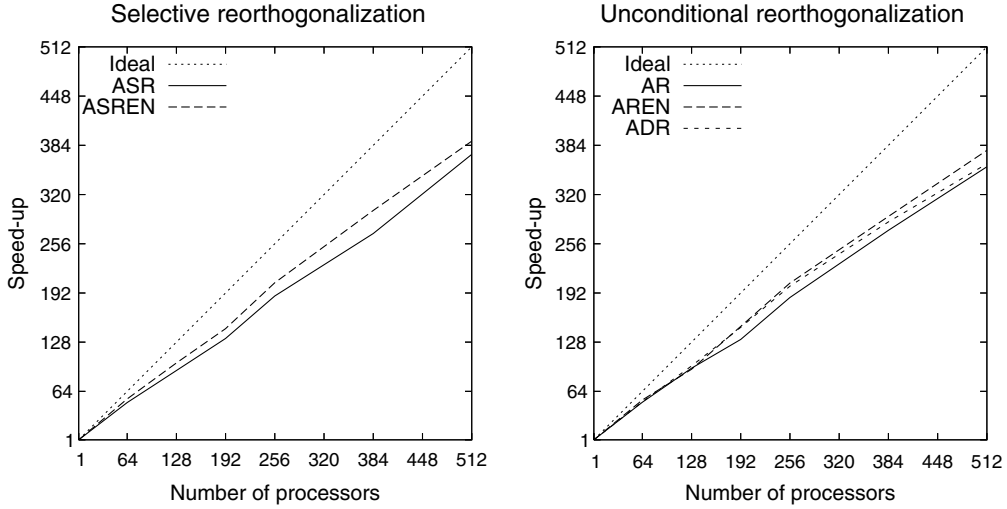

Fig. 11. Measured speed-up for tridiagonal matrix in IBM cluster.

The ASREN algorithm has the best sequential performance, it is as robust as the reference Arnoldi method with reorthogonalization and it has slightly better parallel performance. The ADR algorithm has far better parallel efficiency with large numbers of processors but in some problems can fail to converge. So there is a trade-off between speed and robustness, where the ASREN algorithm is the best choice for the general case and the ADR algorithm is better suited when solving a well-behaved problem with a large number of processors. This is the policy implemented in the SLEPc library, where the default Arnoldi variant is ASREN and ADR must be explicitly requested by the user.

The proposed operation reordering techniques could also be adapted in other contexts such as the symmetric Lanczos method, in particular the variant with full reorthogonalization, or other eigensolvers that make extensive use of orthogonalization. In the case of Lanczos, the authors have implemented the corresponding equivalents with good results. In particular, the ADR variant is more robust in the case of Lanczos than in Arnoldi. However, Lanczos with full reorthogonalization is usually beaten by other reorthogonalization strategies, see [34] for details. On the other hand, it would also be possible to apply them to GMRES-type linear solvers.

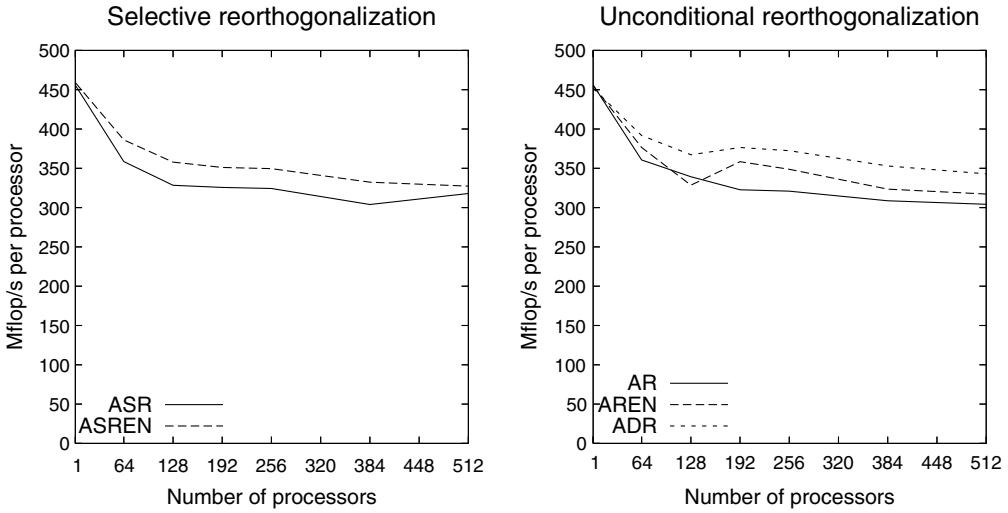

Fig. 12. Measured Mflop/s for tridiagonal matrix in IBM cluster.

Finally, we discuss the applicability of the proposed algorithms in the context of the generalized eigenvalue problem, $Ax = \lambda Bx$. If B is non-singular, then the problem can be addressed with the Arnoldi process applied to matrix $C = B^{-1}A$, where the inverse of B is not computed explicitly but applied implicitly via linear solves whenever a matrix–vector product by C is required. Alternatively, if B is symmetric positive definite, the process can be run on $C = L^{-\mathrm{T}}AL^{-1}$, where L is the Cholesky factor of B. In both cases, the variants proposed in this paper carry over without changes. A more interesting case is when B is singular and a spectral transformation such as shift-and-invert, $C = (A - \sigma B)^{-1}B$, is necessary. In order to avoid corruption of eigenvectors (see [35] for details), the B-Arnoldi process has to be used, which is obtained by replacing the standard inner product, $\langle x,y\rangle = x^{\mathrm{T}}y$, by the B-inner product, $\langle x,y\rangle_B = x^{\mathrm{T}}By$. A practical implementation of this modified process with, for instance, the ASREN variant could be accomplished by replacing lines 7.1 and 7.2 in Algorithm 7 (and similarly lines 7.4 and 7.5) with the following:

$$
\begin{aligned}
z_{j+1} &= Bw_{j+1} \\
h_{1:j,j} &= V_j^{\mathrm{T}} z_{j+1} \\
\rho &= \sqrt{w_{j+1}^{\mathrm{T}} z_{j+1}} (= \|w_{j+1}\|_B)
\end{aligned}
$$

Note that the above fragment of the algorithm can still be implemented in parallel with a single synchronization. These algorithmic variants are implemented in the general SLEPc framework for spectral transformations.

Acknowledgements

This work was supported in part by the Valencian Regional Administration, Directorate of Research and Technology Transfer, under grant number GV06/091.

Part of this research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC03-76SF00098.

The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center – Centro Nacional de Supercomputación.

References

[1] W.E. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem, Quart. Appl. Math. 9 (1951) 17–29.
[2] Y. Saad, Variations of Arnoldi's method for computing eigenelements of large unsymmetric matrices, Linear Algebra Appl. 34 (1980) 269–295.
[3] Y. Saad, Numerical solution of large nonsymmetric eigenvalue problems, Comput. Phys. Commun. 53 (1989) 71–90.
[4] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Stand. 45 (1950) 255–282.
[5] B.N. Parlett, D.R. Taylor, Z.A. Liu, A look-ahead Lánczos algorithm for unsymmetric matrices, Math. Comp. 44 (169) (1985) 105–124.
[6] D. Day, An efficient implementation of the nonsymmetric Lanczos algorithm, SIAM J. Matrix Anal. Appl. 18 (3) (1997) 566–589.
[7] R.B. Morgan, On restarting the Arnoldi method for large nonsymmetric eigenvalue problems, Math. Comp. 65 (1996) 1213–1230.
[8] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems, ACM Trans. Math. Software 31 (3) (2005) 351–362.
[9] V. Hernandez, J.E. Roman, A. Tomas, V. Vidal, SLEPc users manual, Tech. Rep. DSIC-II/24/02 – Revision 2.3.1, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, available from: http://www.grycap.upv.es/slepc, 2006.
[10] S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L.C. McInnes, B. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 – Revision 2.3.1, Argonne National Laboratory, 2006.
[11] A. Canning, L.W. Wang, A. Williamson, A. Zunger, Parallel empirical pseudopotential electronic structure calculations for million atom systems, J. Comput. Phys. 160 (1) (2000) 29–41.
[12] Z. Bai, D. Hu, L. Reichel, A Newton basis GMRES implementation, IMA J. Numer. Anal. 14 (1994) 563–581.
[13] R.B. Sidje, Alternatives for parallel Krylov subspace basis computation, Numer. Linear Algebra Appl. 4 (4) (1997) 305–331.
[14] E. de Sturler, H.A. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, App. Numer. Math. 18 (4) (1995) 441–459.
[15] J. Erhel, A parallel GMRES version for general sparse matrices, Electron. Trans. Numer. Anal. 3 (1995) 160–176.
[16] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput. 23 (6) (2002) 2165–2182.
[17] G. Li, A block variant of the GMRES method on massively parallel processors, Parallel Comput. 23 (8) (1997) 1005–1019.
[18] H.F. Walker, Implementation of the GMRES method using Householder transformations, SIAM J. Sci. Statist. Comput. 9 (1988) 152–163.
[19] J. Frank, C. Vuik, Parallel implementation of a multiblock method with approximate subdomain solution, App. Numer. Math. 30 (4) (1999) 403–423.
[20] V. Hernandez, J.E. Roman, A. Tomas, V. Vidal, Orthogonalization routines in SLEPc, Tech. Rep. STR-1, available from: http://www.grycap.upv.es/slepc/documentation/reports/str1.pdf, 2006.
[21] T. Braconnier, P. Langlois, J.C. Rioual, The influence of orthogonality on the Arnoldi method, Linear Algebra Appl. 309 (1–3) (2000) 307–323.
[22] J.W. Daniel, W.B. Gragg, L. Kaufman, G.W. Stewart, Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization, Math. Comp. 30 (136) (1976) 772–795.
[23] W. Hoffmann, Iterative algorithms for Gram-Schmidt orthogonalization, Computing 41 (4) (1989) 335–348.
[24] Å. Björck, Numerical Methods for Least Squares Problems, Society for Industrial and Applied Mathematics, Philadelphia, 1996.
[25] L. Reichel, W.B. Gragg, FORTRAN subroutines for updating the QR decomposition, ACM Trans. Math. Software 16 (1990) 369–377.
[26] V. Frayssé, L. Giraud, S. Gratton, J. Langou, Algorithm 842: a set of GMRES routines for real and complex arithmetics on high performance computers, ACM Trans. Math. Software 31 (2) (2005) 228–238.
[27] Y. Saad, Krylov subspace methods on supercomputers, SIAM J. Sci. Statist. Comput. 10 (6) (1989) 1200–1232.
[28] G. Meurant, Multitasking the conjugate gradient method on the CRAY X-MP/48, Parallel Comput. 5 (3) (1987) 267–280.
[29] S.K. Kim, A.T. Chronopoulos, An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices, Int. J. Supercomp. Appl. 6 (1) (1992) 98–111.
[30] V. Hernandez, J.E. Roman, A. Tomas, A parallel variant of the Gram-Schmidt process with reorthogonalization, in: G.R. Joubert, W.E. Nagel, F.J. Peters, O.G. Plata, P. Tirado, E.L. Zapata (Eds.), Proceedings of the International Conference on Parallel Computing (ParCo 2005), vol. 33, Central Institute for Applied Mathematics, Jülich, Germany, 2006, pp. 221–228.
[31] I.S. Duff, R.G. Grimes, J.G. Lewis, Sparse matrix test problems, ACM Trans. Math. Software 15 (1) (1989) 1–14.
[32] Z. Bai, D. Day, J. Demmel, J. Dongarra, A test matrix collection for non-Hermitian eigenvalue problems (release 1.0), Technical Report CS-97-355, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, available from: http://math.nist.gov/MatrixMarket, 1997.
[33] T. Davis, University of Florida Sparse Matrix Collection, NA Digest, available from: http://www.cise.ufl.edu/research/sparse/matrices, 1992.
[34] V. Hernandez, J.E. Roman, A. Tomas, Evaluation of several variants of explicitly restarted Lanczos eigensolvers and their parallel implementations, in: High Performance Computing for Computational Science – VECPAR 2006, LNCS vol. 4395, 2007, pp. 403–416.
[35] B. Nour-Omid, B.N. Parlett, T. Ericsson, P.S. Jensen, How to implement the spectral transformation, Math. Comp. 48 (178) (1987) 663–673.