

Evaluation of Several Variants of Explicitly Restarted Lanczos Eigensolvers and their Parallel Implementations^{*}

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

D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia,
Camino de Vera, s/n, E-46022 Valencia, Spain.
Tel. +34-963877356, Fax +34-963877359
{vhernand,jroman,atomas}@itaca.upv.es

Abstract. It is well known that the Lanczos process suffers from loss of orthogonality in the case of finite-precision arithmetic. Several approaches have been proposed in order to address this issue, thus enabling the successful computation of approximate eigensolutions. However, these techniques have been studied mainly in the context of long Lanczos runs, but not for restarted Lanczos eigensolvers. Several variants of the explicitly restarted Lanczos algorithm employing different reorthogonalization strategies have been implemented in SLEPc, the Scalable Library for Eigenvalue Computations. The aim of this work is to assess the numerical robustness of the proposed implementations as well as to study the impact of reorthogonalization in parallel efficiency.

Topics. Numerical methods, parallel and distributed computing.

1 Introduction

The Lanczos method [1] is one of the most successful methods for approximating a few eigenvalues of a large real symmetric (or complex Hermitian) matrix, A . It computes a sequence of Lanczos vectors, v_j , and scalars α_j , β_j as follows

```
Choose a unit-norm vector  $v_1$  and set  $\beta_1 = 0$ 
For  $j = 1, 2, \dots$ 
   $u_{j+1} = Av_j - \beta_j v_{j-1}$ 
   $\alpha_j = v_j^* u_{j+1}$ 
   $u_{j+1} = u_{j+1} - \alpha_j v_j$ 
   $\beta_{j+1} = \|u_{j+1}\|_2$  (if  $\beta_{j+1} = 0$ , stop)
   $v_{j+1} = u_{j+1}/\beta_{j+1}$ 
end
```

Every iteration of the loop computes the following three-term recurrence

$$\beta_{j+1}v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}. \quad (1)$$

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

The first m iterations can be summarized in matrix notation as follows

$$AV_m - V_m T_m = \beta_{m+1} v_{m+1} e_m^*, \quad (2)$$

where $V_m = [v_1, v_2, \dots, v_m]$, $e_m^* = [0, 0, \dots, 1]$, and

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & & \beta_m & \alpha_m \end{bmatrix} \quad (3)$$

It can be shown that the Lanczos vectors are mutually orthonormal, i.e. $V_m^* V_m = I_m$, where I_m is the $m \times m$ identity matrix. As described in section 2, the above procedure can be used as a basis for implementing a solver for symmetric eigenvalue problems, because eigenvalues of T_m approximate eigenvalues of A . However, practical implementations have to deal with many technical difficulties. The two major issues to be addressed are:

1. The loss of orthogonality of Lanczos vectors in finite-precision arithmetic.
2. The convenience of eventually restarting the recurrence.

If loss of orthogonality is not treated appropriately, then duplicate as well as spurious eigenvalues appear in the spectrum of T_j as the iteration progresses. Loss of orthogonality is well understood since the work by Paige [2], who showed that orthogonality is lost as soon as an eigenvalue has converged. This helped researchers devise effective reorthogonalization strategies for preserving (semi-) orthogonality, as described in section 2. These techniques make use of previously computed Lanczos vectors, thus increasing the storage needs and computational cost with respect to the original algorithm, growing as the iteration proceeds. For this reason, practical implementations of Lanczos must generally be restarted, especially in the case of very large-scale sparse problems. In a restarted version, the number of Lanczos steps is limited to a maximum allowed value, after which a new recurrence begins.

The simplest form of restart, usually called explicit restart, consists in computing a new starting vector, v_1 , from the spectral information available before the restart. Although this strategy is typically less effective than other techniques such as implicit restart [3], it can still be competitive in some cases.

The motivation of this work is to provide a robust and efficient parallel implementation of a Lanczos eigensolver in SLEPc, the Scalable Library for Eigenvalue Problem Computations [4]. Currently, only explicitly restarted symmetric Lanczos versions are available in SLEPc, so the main objective of this work is to analyze how different reorthogonalization techniques behave in this context, both from the stability and efficiency viewpoints.

The text is organized as follows. In section 2, the Lanczos method is described in more detail, including the different strategies for coping with loss of orthogonality. Implementation details such as how to efficiently parallelize the

orthogonalization operation are discussed as well. In section 3, the particular implementations available in SLEPc are described. Sections 4 and 5 show the analysis results with respect to numerical stability and parallel performance, respectively. Finally, in section 6 some conclusions are given.

2 Description of the Method

This section provides an overview of the Lanczos method and some of its variations, including techniques for avoiding loss of orthogonality. For more detailed background material the reader is referred to [5] and references therein.

2.1 Basic Lanczos Algorithm

Apart from viewing the Lanczos process from the perspective of the three-term recurrence described in the previous section, it can also be seen as the computation of the orthogonal projection of matrix A onto the Krylov subspace $\mathcal{K}_m(A, v_1) \equiv \text{span}\{v_1, Av_1, \dots, A^{m-1}v_1\}$. From this perspective, the Lanczos method is equivalent to the Arnoldi method, and can be described as follows.

Algorithm 1. Basic Lanczos

Input: Matrix A , number of steps m , and initial vector v_1 of norm 1

Output: $(V_m, T_m, v_{m+1}, \beta_{m+1})$ so that $AV_m - V_m T_m = \beta_{m+1} v_{m+1} e_m^*$

For $j = 1, 2, \dots, m$

$u_{j+1} = Av_j$

Orthogonalize u_{j+1} with respect to V_j (obtaining α_j)

$\beta_{j+1} = \|u_{j+1}\|_2$

$v_{j+1} = u_{j+1}/\beta_{j+1}$

end

In the above algorithm, the second line in the loop performs a Gram-Schmidt process in order to orthogonalize vector u_{j+1} with respect to the columns of V_j , that is, the vectors v_1, v_2, \dots, v_j (see subsection 2.4 for details about Gram-Schmidt). In this operation, j Fourier coefficients are computed. In exact arithmetic, the first $j - 2$ coefficients are zero, and therefore the corresponding operations need not be carried out (orthogonality with respect to the first $j - 2$ vectors is automatic). The other two coefficients are β_j and α_j . According to Paige [6], the β_j computed in this operation should be discarded and, instead, use the value $\|u_j\|_2$ computed in the previous iteration. As we will see in subsection 2.2, orthogonalization will be a key aspect of robust Lanczos variants that cope with loss of orthogonality.

Since $V_m^* v_{m+1} = 0$ by construction, then by premultiplying Eq. 2 by V_m^*

$$V_m^* A V_m = T_m, \tag{4}$$

that is, matrix T_m represents the orthogonal projection of A onto the Krylov subspace spanned by the columns of V_m , and this fact allows us to compute

Rayleigh-Ritz approximations of the eigenpairs of A . Let (λ_i, y_i) be an eigenpair of matrix T_m , then the Ritz value, λ_i , and the Ritz vector, $x_i = V_m y_i$, can be taken as approximations of an eigenpair of A . Typically, only a small percentage of the m approximations are good. This can be assessed by means of the residual norm for the Ritz pair, which turns out to be very easy to compute:

$$\|Ax_i - \lambda_i x_i\|_2 = \|AV_m y_i - \lambda_i V_m y_i\|_2 = \|(AV_m - V_m T_m)y_i\|_2 = \beta_{m+1} |e_m^* y_i|. \quad (5)$$

2.2 Lanczos in Finite Precision Arithmetic

When implemented in finite precision arithmetic, the Lanczos algorithm does not behave as expected. The eigenvalues of the tridiagonal matrix T_j (the Ritz values) converge very rapidly to well-separated eigenvalues of matrix A , typically those in the extreme of the spectrum. However, if enough iterations of the algorithm are carried out, then multiple copies of these Ritz values appear, beyond the multiplicity of the corresponding eigenvalue in A . In addition, the process gives wrong Ritz values as converged, which are usually called spurious eigenvalues. It can be observed that this unwanted behavior appears at the same time that the Lanczos vectors start to lose mutual orthogonality. Lanczos was already aware of this problem and suggested to explicitly reorthogonalize the new Lanczos vector with respect to all the previous ones at each step. Although effective, this costly operation seems to invalidate all the appealing properties of the algorithm. Other alternatives, discussed below, have been proposed in order to be able to deal with loss of orthogonality at less cost.

Full Orthogonalization. The simplest cure to loss of orthogonality is to orthogonalize vector u_{j+1} explicitly with respect to all the previously computed Lanczos vectors. That is, performing the computation for all vectors, including the first $j - 2$ ones for which the Fourier coefficient is zero in exact arithmetic.

The main advantage of full orthogonalization is its robustness, since orthogonality is maintained to full machine precision. (Note that for this to be true it may be necessary to resort to double orthogonalization, see subsection 2.4.) The main drawback of this technique is that the cost of orthogonalization is high and grows as more Lanczos steps are carried out. This recommends a restarted version, in which the number of Lanczos vectors is bounded, see section 2.3.

Local Orthogonalization. The quest for more efficient solutions to the problem of loss of orthogonality started with a better theoretical understanding of the Lanczos process in finite precision arithmetic, unveiled by Paige's work [6, 2, 7]. One key aspect of Paige's analysis is that Lanczos vectors start to lose orthogonality as soon as an eigenvalue of T_j stabilizes or, in other words, when a Ritz value is close to convergence, causing the subsequent Lanczos vectors to contain a non-negligible component in the direction of the corresponding Ritz vector. Until this situation occurs, the Lanczos algorithm with local orthogonalization (that is, if vector u_{j+1} is orthogonalized only with respect to v_j and v_{j-1}) computes the same quantities as the variant with full orthogonalization. This fact suggests

that an algorithm could proceed with local orthogonalization until an eigenvalue of T_j has stabilized, then either start a new Lanczos process with a different initial vector, or continue the Lanczos process with the introduction of some kind of reorthogonalization. The latter approach gave way to the development of semiorthogonal Lanczos methods, discussed below.

A completely different approach is to simply ignore loss of orthogonality and perform only local orthogonalization at every Lanczos step. This technique is obviously the cheapest one, but has several important drawbacks. For one thing, convergence of new Ritz values is much slower since multiple copies of already converged ones keep on appearing again and again. This makes it necessary to carry out many Lanczos steps to obtain the desired eigenvalues. On the other hand, there is the problem of determining the correct multiplicity of the computed eigenvalues as well as discarding those which are spurious. A clever technique for doing this was proposed in [8]. An eigenvalue of T_j is identified as being spurious if it is also an eigenvalue of the matrix T'_j , which is constructed by deleting the first row and column of T_j . Furthermore, good eigenvalues are accepted only after they have been replicated at least once.

Semiorthogonal Techniques. As mentioned above, the idea of these techniques is to perform explicit orthogonalization only when loss of orthogonality is detected. Two aspects are basic in this context:

1. How to carry out the orthogonalization so that the overall cost is small.
2. How to determine when an eigenvalue has stabilized or, in other words, how to monitor loss of orthogonality, without incurring a high cost.

With respect to the first aspect, several different approaches have been proposed: selective [9], periodic [10], and partial [11] reorthogonalization. In brief, they consist, respectively, in: orthogonalizing every Lanczos vectors with respect to all nearly converged Ritz vectors; orthogonalizing u_{j+1} and u_{j+2} with respect to all the Lanczos vectors; and orthogonalizing u_{j+1} and u_{j+2} with respect to a subset of the Lanczos vectors.

The second aspect can be addressed in two ways, basically. One is to compute the error bounds associated to the Ritz pairs at each iteration, and the other is to use a recurrence for estimating a bound of the level of orthogonality, such as the one proposed by Simon in [11]. If we define the level of orthogonality at the j -th Lanczos step as

$$\omega_j \equiv \max_{1 \leq k < j} |\omega_{j,k}|, \quad \text{with} \quad \omega_{j,k} \equiv v_j^* v_k, \quad (6)$$

then the full reorthogonalization technique keeps it at roundoff level in each step, $\omega_j \approx \epsilon_M$. However, all that effort is not necessary since, as shown in [11, 12], maintaining semiorthogonality, i.e. $\omega_j \approx \sqrt{\epsilon_M}$, is sufficient so that properties of the Rayleigh-Ritz projection are still valid.

2.3 Explicit Restart

As mentioned above, restarting is intended for reducing the storage requirements and, more importantly, reducing the computational cost of orthogonalization,

which grows as more Lanczos vectors become available. Restart can be accomplished in several ways. The idea of explicit restart is to iteratively compute different m -step Lanczos factorizations (Eq. 2) with successively “better” initial vectors. The initial vector for the next Lanczos run is computed from the information available in the most recent factorization. The simplest way to select the new initial vector is to take the Ritz vector associated to the first wanted, non-converged Ritz value.

In order for a restarted method to be effective, it is necessary to keep track of already converged eigenpairs and perform a deflation, by *locking* converged Ritz vectors. Suppose that after a Lanczos run, the first k eigenpairs have already converged to the desired accuracy, and write V_m as

$$V_m = \left[V_{1:k}^{(l)} \mid V_{k+1:m}^{(a)} \right], \quad (7)$$

where the (l) superscript indicates locked vectors and the (a) superscript indicates active vectors. In the next Lanczos run, only $m - k$ Lanczos vectors must be computed, the active ones, and in doing this the first k vectors have to be deflated. This can be done simply by orthogonalizing every new Lanczos vector also with respect to the locked ones. Therefore, deflation can be incorporated to Algorithm 1 simply by explicitly including the locked vectors in the orthogonalization operation. With this change, a restarted Lanczos method can be described as in Algorithm 2.

Algorithm 2. Explicitly Restarted Lanczos

Input: Matrix A , initial vector v_1 of norm 1, and dimension of the subspace m

Output: A partial eigendecomposition $AV_k = V_k\Theta_k$, with $\Theta_k = \text{diag}(\theta_1, \dots, \theta_k)$

Initialize $k = 0$

Restart loop

 Perform $m - k$ steps of Lanczos (Algorithm 1) with initial vector v_{k+1}

 Compute eigenpairs of T_m , $T_m y_i = y_i \theta_i$

 Compute residual norm estimates, $\tau_i = \beta_{m+1} |e_m^* y_i|$

 Lock converged eigenpairs

$V_m = V_m Y$

end

In a restarted Lanczos method, it is also necessary to deal with loss of orthogonality. In the case of the simple explicit restart scheme, it is safe to use any of the techniques described in the previous subsection, since full orthogonality of the Lanczos vectors is not required for the restart to work correctly. Only in the case of local orthogonalization, the following considerations should be made:

- The restart vector has to be orthogonalized with respect to locked vectors.
- Since the value of m (the largest allowable subspace dimension) is usually very small compared to n (the matrix dimension), then the heuristics suggested in [8] cannot be applied. Therefore, another technique should be used in order to discard spurious eigenvalues as well as redundant duplicates.
- The computed eigenvectors are not orthogonal, although they satisfy the eigenvalue-eigenvector relation to the specified precision.

2.4 Gram-Schmidt Orthogonalization

Gram-Schmidt procedures are used for orthogonalizing a vector u_{j+1} with respect to a set of vectors V_j . In finite precision arithmetic, simple versions of Gram-Schmidt such as CGS or MGS, will not be reliable enough in some cases, which may produce numerical instability in the context of Lanczos eigensolvers. This problem can be solved by introducing reorthogonalization, that is, to take the resulting vector and perform a second orthogonalization.

In exact arithmetic, the Fourier coefficients of the second orthogonalization ($c_{1:j,j}$) are zero and therefore it has no effect. However, this is not the case in finite precision arithmetic, where those coefficients can be thought of as a correction to coefficients of the first orthogonalization ($h_{1:j,j}$), which is not necessarily small.

In cases where large rounding errors have not occurred in first place, reorthogonalization is superfluous and could be avoided. In order to determine whether the computed vector is good enough or requires a refinement, it is possible to use a simple criterion such as

$$\beta_{j+1} < \eta \rho \tag{8}$$

for some constant parameter $\eta < 1$ (a safe value is $\eta = 1/\sqrt{2}$). This criterion compares the norm of u_{j+1} before (ρ) and after (β_{j+1}) orthogonalization. An orthogonalization procedure based on this scheme is illustrated in Algorithm 3. For further details about iterative Gram-Schmidt procedures, see [13, 14].

Algorithm 3. CGS with selective refinement and estimated norm

```

 $h_{1:j,j} = V_j^* u_{j+1}$ 
 $\rho = \|u_{j+1}\|_2$ 
 $u_{j+1} = u_{j+1} - V_j h_{1:j,j}$ 
 $\beta_{j+1} = \sqrt{\rho^2 - \sum_{i=1}^j h_{i,j}^2}$ 
if  $\beta_{j+1} < \eta \rho$ 
     $c_{1:j,j} = V_j^* u_{j+1}$ 
     $\sigma = \|u_{j+1}\|_2$ 
     $u_{j+1} = u_{j+1} - V_j c_{1:j,j}$ 
     $\beta_{j+1} = \sqrt{\sigma^2 - \sum_{i=1}^j c_{i,j}^2}$ 
end

```

A similar orthogonalization scheme might be considered for the Modified Gram-Schmidt variant. However, the resulting numerical quality is about the same, as pointed out in [13]. In this work, we do not consider MGS variants since they lead to poor parallel performance.

In the context of parallel implementations, orthogonalization is usually the operation that introduces more performance penalty. In order to reduce this effect, the number of synchronizations should be reduced whenever possible. One possibility for this is to defer the normalization of the vector to the next Lanczos step, as proposed in [15]. Algorithm 3 tries to optimize parallel performance by means of estimation of the norm. The main objective of this technique is to avoid the explicit computation of the Euclidean norm of u_{j+1} and, instead, use an

estimation based on the original norm (prior to the orthogonalization), by simply applying the Pythagorean theorem. The original norm is already available, since it is required for the selective reorthogonalization criterion, and can be computed more efficiently in parallel since its associated reduction is susceptible of being integrated in a previous reduction (that is, with one synchronization less). More details about these techniques can be found in [16].

3 Lanczos Methods in SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [4], is a software library for the solution of large, sparse eigenvalue problems on parallel computers. It 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 such as Arnoldi, Lanczos, Subspace Iteration and Power/RQI. It also provides built-in support for different types of problems and spectral transformations such as the shift-and-invert technique.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [17]), a parallel framework for the numerical solution of partial differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. In PETSc, the application programmer works directly with objects such as vectors and matrices, rather than concentrating on the underlying data structures. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers. SLEPc extends PETSc with all the functionality necessary for the solution of eigenvalue problems, and thus inherits all the good properties of PETSc, including portability, scalability, efficiency and flexibility. SLEPc also leverages well-established eigensolver packages such as ARPACK, integrating them seamlessly.

As of version 2.3.0, SLEPc provides a symmetric Lanczos eigensolver, which is based on explicit restart and allows the user to select among several types of reorthogonalization strategies. In SLEPc, these strategies are referred to as **local**, **full**, **selective**, **periodic**, and **partial**, and are related to the techniques described in section 2. However, the implementations slightly differ from the originally proposed techniques due to the implications of having a restarted algorithm.

In the case of local reorthogonalization, the post-processing technique proposed in [8] cannot be used in the context of a restarted method, because of the relatively small size of matrix T_m . The approach taken in SLEPc is to explicitly compute the residual norm for every converged eigenpair, then from the correct values accept only the first replica. Although this may seem a very costly strategy, results show that the incurred overhead is small (see section 5).

In the case of selective orthogonalization, there are two possible approaches in the context of a restarted method. One approach, used in [18], is to exit

the Lanczos loop as soon as an eigenvalue has converged. This provokes the computation of the corresponding Ritz vector which will be used for deflation in subsequent restarts. The other approach is to run the Lanczos process completely up to the maximum subspace dimension, managing the loss of orthogonality with the selective orthogonalization technique. This is the approach implemented in SLEPc since, to our experience, it is faster in terms of overall convergence.

With respect to periodic and partial reorthogonalization, in both cases we use Simon's recurrence for monitoring loss of orthogonality [11]. The difference with a non-restarted method is that in our implementation in all Lanczos steps the current vector is orthogonalized explicitly with respect to locked vectors, that is, the vectors used for deflation are not considered in the recurrence for monitoring loss of orthogonality. A better approach would be to do this explicit deflation only when necessary, as in [19]. This issue is proposed as future work.

4 Numerical Results

In this section, we consider an empirical test with a battery of real-problem matrices using the implementation referred to in section 3 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 symmetric matrix from the Harwell-Boeing collection [20]. These 67 matrices come from a variety of real problems and some of them are particularly challenging for eigenvalue computation. For this test, the solvers are configured with tolerance equal to 10^{-7} and a maximum of 50 basis vectors.

The level of orthogonality is defined as the maximum value of $\|I - V_m^* V_m\|_F$ at each restart, and the residual norm is computed as the maximum of $\|Ax - \lambda x\|_2 / \|\lambda x\|_2$ for every converged eigenvalue.

The results of these tests are shown in Figures 1 and 2, where each dot corresponds to one matrix from the collection. As expected, the algorithm with full reorthogonalization maintains the orthogonality level close to full machine precision and the algorithm with local reorthogonalization does not guarantee the orthogonality among Lanczos vectors. The semiorthogonal methods (selective, periodic and partial) have a good level of orthogonality, in all cases between full and half machine precision. Another remarkable conclusion that can be drawn from these results is that the Gram-Schmidt procedure with iterative refinement and estimated norm described in subsection 2.4 is a well-suited orthogonalization scheme for these algorithms.

5 Performance analysis

In order to compare the parallel efficiency of the proposed Lanczos variants, several test cases were analyzed in a cluster platform. This machine consists of 55 biprocessor 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

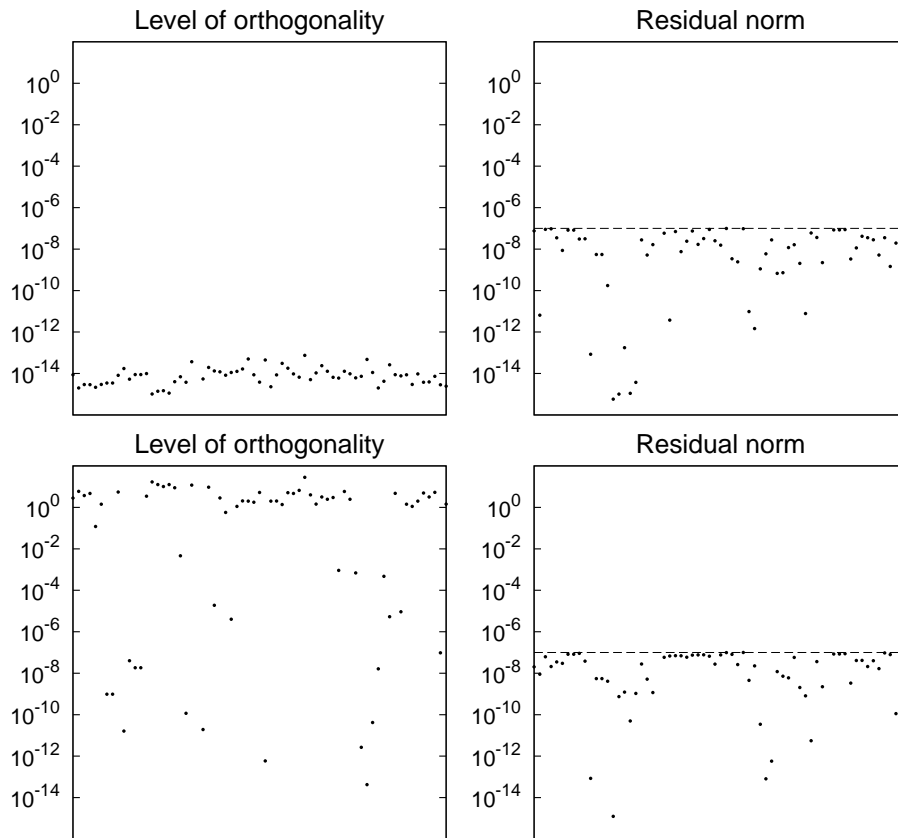


Fig. 1. Level of orthogonality and residual norm for the Lanczos algorithm with full (top) and local (bottom) orthogonalization.

used in the tests reported in this section. The solver was requested to compute 10 eigenvalues with tolerance equal to 10^{-7} using a maximum of 50 basis vectors.

Two types of tests were considered. On one hand, matrices arising from real applications were used for measuring the parallel speed-up. These matrices are listed in Table 1 and are taken from the University of Florida Sparse Matrix Collection [21]. 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 local reorthogonalization variant (including the post-process) as Table 1 shows. On the other hand, a synthetic test case was used for analyzing the scalability of the algorithms, measuring the scaled speed-up (with variable problem size) and Mflop/s per processor. For this analysis, a tridiagonal matrix was used, with a dimension of $10,000 \times p$, where p is the number of processors.

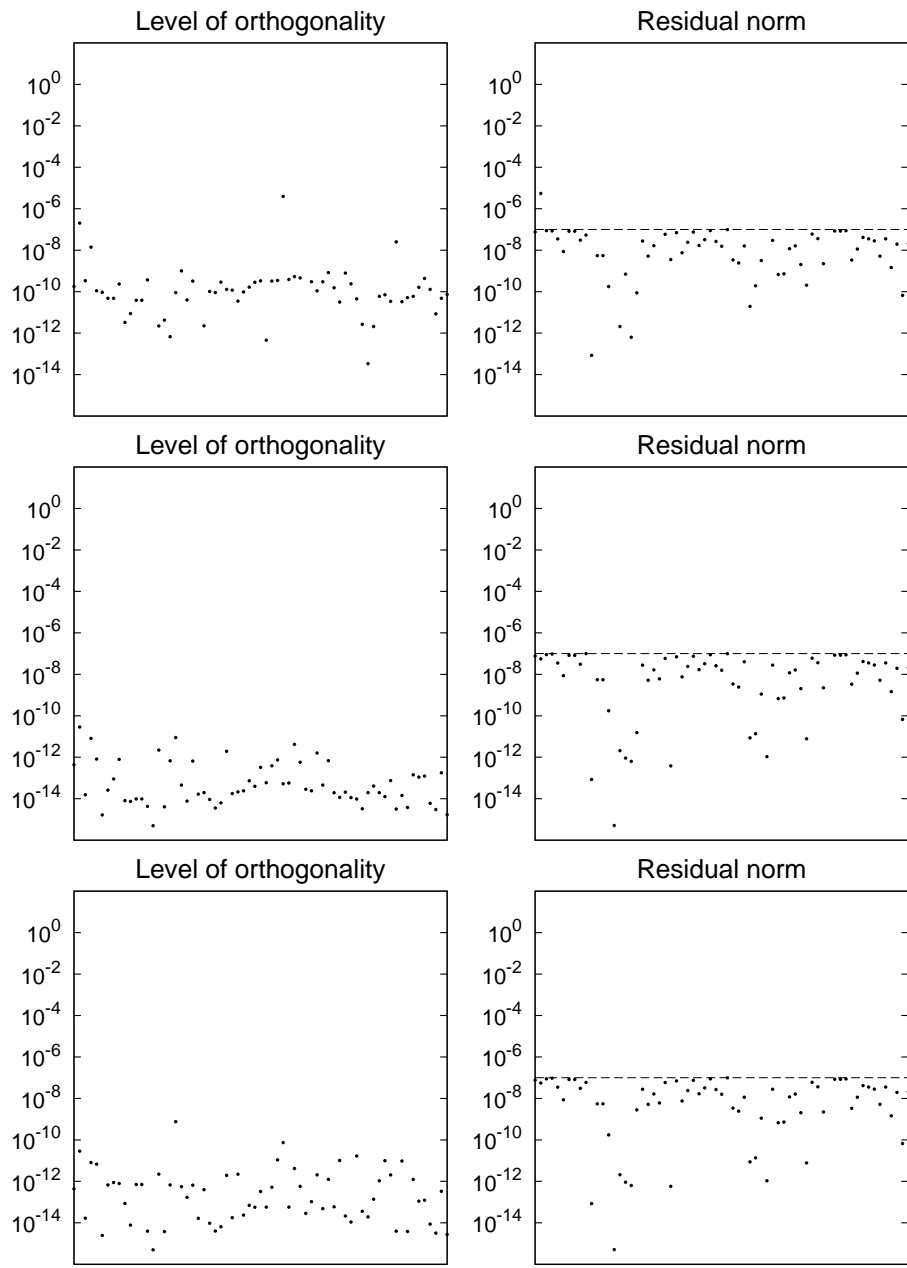


Fig. 2. Level of orthogonality and residual norm for the Lanczos algorithm with selective (top), periodic (center) and partial (bottom) reorthogonalization.

Table 1. Properties of test matrices and elapsed time in seconds with one processor.

Test matrix			Elapsed time				
Name	Order	Non-zeros	Full	Local	Selective	Periodic	Partial
NASASRB	54,870	2,677,324	29.19	17.49	18.64	27.52	26.85
SHIPSEC8	114,919	3,303,553	6.79	4.28	5.49	5.63	5.58
AF_SHELL1	504,855	17,562,051	126.89	76.03	82.08	117.18	116.97
AUDIKW_1	943,695	77,651,847	55.87	43.90	61.29	51.21	51.03

As Figure 3 illustrates, all algorithms show overall good parallel performance. However, the selective reorthogonalization algorithm has poorer performance with a large number of processor than the rest. This is due to the extra synchronizations needed to perform the deflation against converged Ritz vectors in each iteration, which cannot be combined with the other orthogonalizations. Also, because of the deflation against locked vectors needed by the explicit restart scheme, the partial and periodic algorithms have no practical advantage over the full reorthogonalization variant. The local reorthogonalization scheme has the best performance in these tests in spite of having a post-processing phase and an additional reorthogonalization at each restart.

The results in Figure 4 show overall good speed-up in all alternatives. The selective reorthogonalization algorithm has the lowest Mflops/s rate due to the extra synchronizations needed. In the rest of algorithms, the Mflops/s rate improves as the average number of vectors involved in the orthogonalization grows.

6 Conclusions

In this work, an explicit restarting scheme has been applied to different Lanczos variants. The orthogonalization of vectors in these algorithms is done with an optimized version of Classical Gram-Schmidt with selective refinement. All the implemented algorithms are numerically robust for the considered test cases.

The performance results presented in section 5 show that the algorithms achieve good parallel efficiency in all the test cases analyzed, and scale well when increasing the number of processors. The best algorithm will depend on the application, so testing different alternatives is often useful. Thanks to the object-oriented structure of SLEPc, the user can try different Lanczos variants without even having to recompile the application.

References

1. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards* **45** (1950) 255–282
2. Paige, C.C.: Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl.* **18**(3) (1976) 341–349

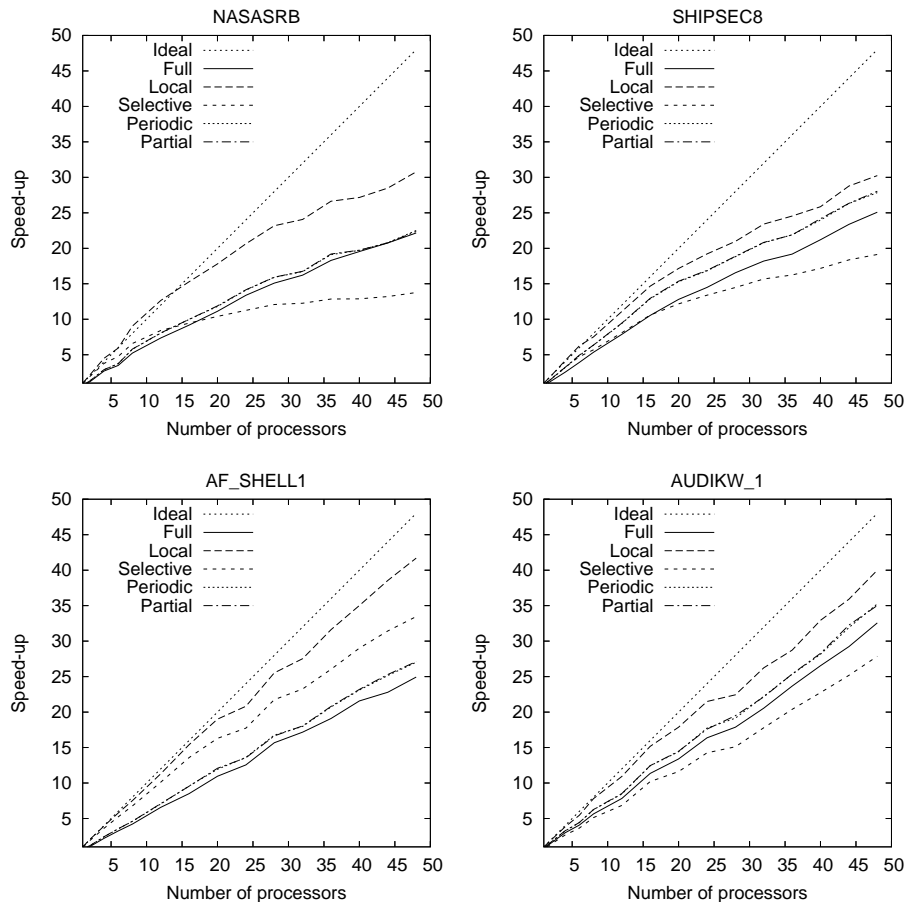


Fig. 3. Measured speed-up for test matrices.

3. Sorensen, D.C.: Implicit application of polynomial filters in a k -step Arnoldi method. *SIAM J. Matrix Anal. Appl.* **13** (1992) 357–385
4. Hernandez, V., Roman, J.E., Vidal, V.: SLEPC: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software* **31**(3) (2005) 351–362
5. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H., eds.: *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000)
6. Paige, C.C.: Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.* **10** (1972) 373–381
7. Paige, C.C.: Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra Appl.* **34** (1980) 235–258
8. Cullum, J.K., Willoughby, R.A.: *Lanczos Algorithms for Large Symmetric Eigenvalue Computations. Vol. 1: Theory*. Birkhäuser, Boston, MA (1985) Reissued by

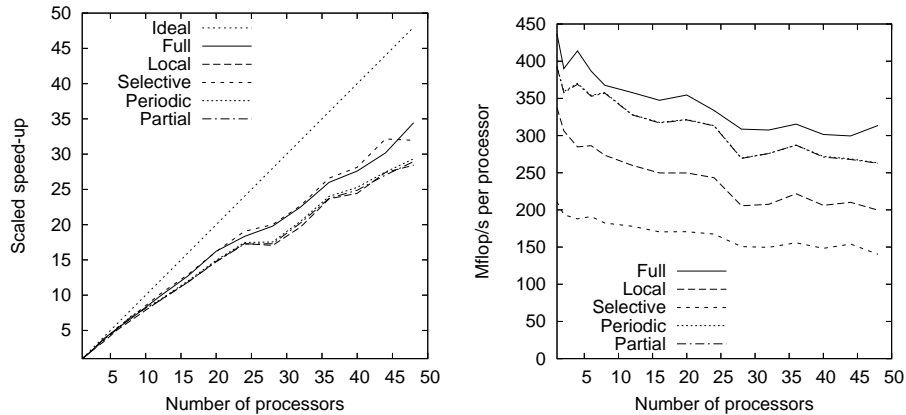


Fig. 4. Measured scaled speed-up and Mflop/s for synthetic matrix.

- SIAM, Philadelphia, 2002.
9. Parlett, B.N., Scott, D.S.: The Lanczos algorithm with selective orthogonalization. *Math. Comp.* **33** (1979) 217–238
 10. Grcar, J.F.: Analyses of the Lanczos algorithm and of the approximation problem in Richardson’s method. Technical Report 1074, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois (1981)
 11. Simon, H.D.: The Lanczos algorithm with partial reorthogonalization. *Math. Comp.* **42**(165) (1984) 115–142
 12. Simon, H.D.: Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra Appl.* **61** (1984) 101–132
 13. Hoffmann, W.: Iterative algorithms for Gram-Schmidt orthogonalization. *Computing* **41**(4) (1989) 335–348
 14. Björck, Å.: *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Philadelphia (1996)
 15. Kim, S.K., Chronopoulos, A.T.: A class of Lanczos-like algorithms implemented on parallel computers. *Parallel Comput.* **17**(6–7) (1991) 763–778
 16. Hernandez, V., Roman, J.E., Tomas, A.: Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. submitted (2006)
 17. Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B., Zhang, H.: *PETSc users manual*. Technical Report ANL-95/11 - Revision 2.3.1, Argonne National Laboratory (2006)
 18. Szularz, M., Weston, J., Clint, M.: Explicitly restarted Lanczos algorithms in an MPP environment. *Parallel Comput.* **25**(5) (1999) 613–631
 19. Cooper, A., Szularz, M., Weston, J.: External selective orthogonalization for the Lanczos algorithm in distributed memory environments. *Parallel Comput.* **27**(7) (2001) 913–923
 20. Duff, I.S., Grimes, R.G., Lewis, J.G.: Sparse matrix test problems. *ACM Trans. Math. Software* **15**(1) (1989) 1–14
 21. Davis, T.: University of Florida Sparse Matrix Collection. *NA Digest* (1992) Available at <http://www.cise.ufl.edu/research/sparse/matrices>.