John von Neumann Institute for Computing

A Parallel Variant of the Gram-Schmidt Process with Reorthogonalization

V. Hernández, J.E. Román, A. Tomás

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 221-228, 2006.

http://www.fz-juelich.de/nic-series/volume33

A Parallel Variant of the Gram-Schmidt Process with Reorthogonalization

V. Hernandez[a], J. E. Roman[a], A. Tomas[a]

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

1. Introduction

This work is concerned with the effective parallelization of algorithms for computing an orthonormal basis of a subspace in the context of Krylov-type eigenvalue solvers such as Arnoldi. The main motivation is to try to improve the parallel efficiency of eigensolvers implemented in SLEPc, the Scalable Library for Eigenvalue Problem Computations [4], especially when addressing large-scale eigenproblems with many processors in distributed memory platforms.

It is well known that the scalability of Krylov-type eigensolvers is limited by the synchronization points associated to vector norms and inner products, in particular those required during the orthogonalization of vectors. Typically, orthogonalization in this context is carried out by means of some variant of the Gram-Schmidt process. In this work, a new algorithm is proposed, which is a modification of classical Gram-Schmidt with reorthogonalization, that can be parallelized with fewer synchronization points thus leading to better parallel performance.

The text is organized as follows. Section 2 provides an overview of orthogonalization algorithms used in the context of eigensolvers. The proposed algorithm is described in detail in section 3. Section 4 presents timing results measured in different parallel platforms, which illustrate the benefits of the new algorithm in terms of parallel performance. Finally, a discussion is given in section 5.

2. Orthogonalization in the Context of Eigensolvers

Given a standard eigenvalue problem, $Ax = \lambda x$, or a related eigenproblem, where A is a square matrix of order n, the goal is to compute the eigenvalue, λ, or the eigenvector, x, or both.

Krylov-type eigensolvers such as Arnoldi are based on explicitly building an orthonormal basis of a Krylov subspace, $\mathcal{K}_m(A, v_1) := \mathrm{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$, and then selecting members of that subspace as approximations of the eigenvectors. These iterative eigensolvers operate with matrix-vector products, thus preserving sparsity. That makes them especially appropriate for sparse problems such as those arising after the discretization of partial differential equations. They are also well suited for addressing huge problems in parallel computers, as discussed in subsection 2.2.

Algorithm 1 illustrates the Arnoldi method schematically, denoting by V_m the $n \times m$ matrix with column vectors $v_1, \ldots, v_m$ and by H_m the $m \times m$ Hessenberg matrix whose nonzero entries are h_{ij}.

Algorithm 1 (Arnoldi)

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

```
for j = 1, 2, ..., m
    Expand the basis: w_{j+1} = A v_j
    Orthogonalize w_{j+1} with respect to V_j, obtaining w''_{j+1} and coefficients h_{1:j,j}
    Normalize: h_{j+1,j} = ||w''_{j+1}||_2,    v_{j+1} = w''_{j+1} / h_{j+1,j}
end
```

It can be shown (see for example [7]) that Algorithm 1 computes an orthonormal basis V_m of the Krylov subspace $\mathcal{K}_m(A, v_1)$. The Arnoldi algorithm 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. Arnoldi avoids this by orthogonalizing a vector that is a candidate for expanding the basis as soon as it is generated.

In the Arnoldi scheme, since only one vector at a time has to be orthogonalized, the Gram-Schmidt orthogonalization procedures come in very naturally. Orthogonalization via Householder reflectors could also be used, as proposed in [10], but at the expense of higher computational cost and more difficult implementation. This work focuses on Gram-Schmidt procedures.

In order to guarantee the numerical stability of Algorithm 1, the vector computed in the orthogonalization step, w''_{j+1}, must be orthogonal to full working precision with respect to the columns of V_j. Indeed, if the quality of orthogonality is poor then instabilities may appear. Therefore, care must be taken so that the chosen orthogonalization algorithm meets this requirement.

Several variants of Gram-Schmidt orthogonalization exist. It is well known that Modified Gram-Schmidt (MGS) provides much better quality of orthogonality than Classical Gram-Schmidt (CGS). This is the reason why MGS is preferred in linear solvers such as GMRES. However, in the context of eigensolvers the MGS scheme is not sufficient for all cases: large cancellations can still occur and in that case the computed basis fails to be orthogonal to full working accuracy. A cure for this is to resort to double orthogonalization, as described next.

2.1. Gram-Schmidt with Reorthogonalization

The simplest possibility to guard against large cancellations in Gram-Schmidt orthogonalization is to take the resulting vector and perform a second orthogonalization. This approach is usually referred to as CGS2 and MGS2 for the classical and modified variants, respectively. Algorithm 2 shows the CGS2 algorithm, where w_{j+1} denotes the initial vector, w'_{j+1} the vector resulting from the first orthogonalization, and w''_{j+1} the vector obtained after reorthogonalization. The computed coefficients h_j are referred to as $h_{1:j,j}$ in Algorithm 1.

Algorithm 2 (Classical Gram-Schmidt with Reorthogonalization, CGS2)

Input: Vector w_{j+1} to be orthogonalized against the columns of V_j
Output: Orthogonalized vector w''_{j+1} and coefficients h_j

$h_j = V_j^T w_{j+1}$
$w'_{j+1} = w_{j+1} - V_j h_j$
$c_j = V_j^T w'_{j+1}$
$w''_{j+1} = w'_{j+1} - V_j c_j$
$h_j = h_j + c_j$

Note that, in exact arithmetic, the coefficients c_j are zero and vectors w'_{j+1} and w''_{j+1} coincide. However, this is not the case in finite precision arithmetic, where c_j can be thought of as a maybe-not-so-small correction to h_j.

In cases where large cancellations have not occurred in first place, reorthogonalization is superfluous. On the other extreme, it could happen that large cancellations 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 [2], is used to determine whether the computed vector is good enough. In practical situations, typically only one reorthogonalization is necessary at most. A detailed description of iterative Gram-Schmidt procedures can be found in [5].

2.2. Parallel Implementation of the Arnoldi Method

Assuming a message-passing paradigm with standard data distribution of vectors and matrices, i.e. by blocks of rows, parallelization of Algorithm 1 amounts to carry 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, provided that the amount of data to be exchanged among processors is reduced, for example, by using a nearly optimal ordering computed via graph partitioning algorithms.
2. Orthogonalization. As exemplified in Algorithm 2, this stage consists in a series of vector operations such as inner products, additions, and multiplications by a scalar.
3. Normalization. The computation of the norm requires a global reduction operation, thus representing a synchronization point in the algorithm.

Since vector addition and scaling are trivially parallelizable operations, the efficiency of the orthogonalization stage is given by the number of required inner products. More precisely, what is important is the number of synchronization points, since the data required to be exchanged for several independent inner products can be combined in a single reduction operation, resulting in almost the same communication cost as just one inner product. Therefore, the scalability of the different variants of Gram-Schmidt orthogonalization depends on the data dependency of inner products. This is the reason why the CGS versions scale better than the MGS counterparts, since inner products associated to operations such as $h_j = V_j^T w_{j+1}$ can be grouped together in a single communication.

3. The Proposed Algorithm

The new algorithm proposed to improve the parallel efficiency of the orthogonalization stage is based on Algorithm 2. The first part of this section tries to justify this decision.

3.1. Motivation and Related Work

It should be clear from the above discussion that the orthogonalization stage is mainly the source of poor scalability of Arnoldi's method. Other authors have already attempted to enhance this stage in terms of parallel efficiency, by rearranging the computations in such a way that more than one vector at a time is ready for being orthogonalized [8], or by addressing the problem in the context of block-Krylov eigensolvers [9]. In contrast, the approach presented below is still compatible with the simpler, single-vector scheme of Algorithm 1.

As stated in subsection 2.2, parallel implementations of CGS scale better than in the case of MGS. In addition, CGS usually provides higher Mflop rates even for one processor, due to a better data access pattern. On the other hand, from the numerical point of view both CGS2 and MGS2 are equivalent in the sense that reorthogonalization repairs the effect of cancellation, as shown in [5]. For all these reasons, CGS2 has been chosen as the starting point for deriving the new algorithm.

In this work, we consider CGS2 without selective reorthogonalization, that is, reorthogonalization is carried out in all cases. As mentioned in subsection 2.1, this implies a higher computational cost compared to the variant in which a criterion is used. However, this drawback is not so serious as it may seem, since it turns out that the average number of orthogonalizations in a selective reorthogonalization variant is quite often close to 2 rather than close to 1. The next table shows the average number of orthogonalizations for several test cases taken from the NEP [1] and UF [3] collections.

AF23560	1.07	ANDREWS	1.85	BWM2000	1.90	CRY10000	1.86	CDDE1	1.81
DW8192	1.60	LIN	1.88	LOP163	1.21	OLM5000	1.87	PDE2961	1.41
QH1484	1.94	RDB2048	1.80	RW5151	1.07	TOLS4000	1.99	TUB1000	1.89

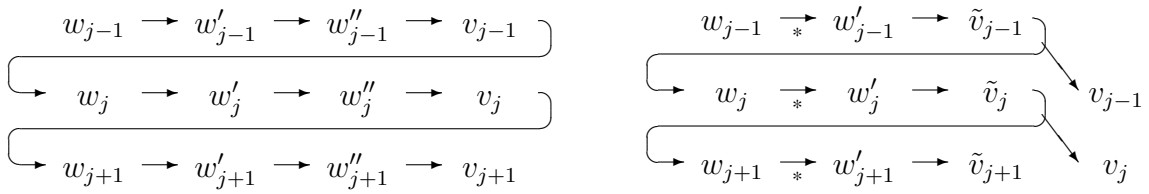

Figure 1. Computational schemes of CGS2 (left) and DCGS2 (right) algorithms.

3.2. Delayed Reorthogonalization

The key idea of the proposed algorithm is to delay the reorthogonalization (last three lines in Algorithm 2) in such a way that communication associated to it can be merged with that of the first orthogonalization of the next vector, which corresponds to the subsequent Arnoldi iteration. We refer to this algorithm as Delayed Classical Gram-Schmidt with Reorthogonalization (DCGS2).

In DCGS2, the next Arnoldi iteration is started as soon as the result of the first orthogonalization is available, w'_{j+1}, and reorthogonalization is postponed until h_{j+1} is to be computed. Figure 1 illustrates the CGS2 and DCGS2 computation schemes for several iterations. In CGS2, the sequence of steps necessary to compute v_{j+1} from v_j are followed in a conceptually linear fashion, proceeding to the next basis vector only after the current vector is completely "finished". In contrast, in the new algorithm, part of the computation associated to a vector is interleaved with the computation of the next one. Note that in DCGS2 the notation changes slightly: now the normalization is carried out over w'_{j+1}, producing $\tilde{v}_{j+1}$, which represents a vector that still requires a reorthogonalization to become the final v_{j+1}. What the right-hand side of Figure 1 represents is that the operations associated to w'_{j+1} and v_j can be done simultaneously since no data dependencies exist and, therefore, the associated communications can be combined.

The following is an algorithmic representation of the new method.

Algorithm 3 (Delayed Classical Gram-Schmidt, DCGS2)

Input: Vector w_{j+1} to be orthogonalized against the columns of $\tilde{V}_j$
Output: Partially orthogonalized vector w'_{j+1} (with coef. h_j) and corrected vector v_j (with coef. h_{j-1})

$h_j = \tilde{V}_j^T w_{j+1}$
if $j > 1$
  $c_{j-1} = V_{j-1}^T \tilde{v}_j$
  $v_j = \tilde{v}_j - V_{j-1} c_{j-1}$
  $h_{j-1} = h_{j-1} + c_{j-1}$
end
$w'_{j+1} = w_{j+1} - V_j h_j$

Note that computed quantities such as w'_{j+1} are not equal in both algorithms since they are produced by different operations. In particular, in Figure 1 the starred arrow represents the initial orthogonalization, in which the coefficients are computed with respect to $\tilde{V}_j := [V_{j-1}, \tilde{v}_j]$, as opposed to V_j in CGS2. Therefore, in DCGS2 the coefficients of the first orthogonalization are computed against a set of vectors in which the last one is not orthogonal to full working precision. However, when the second part of the orthogonalization is carried out, the corrected vector is already available and is indeed used. Although it may seem that the referred alterations of the process would

compromise the numerical stability of the algorithm, it turns out that Algorithm 3 succeeds in computing an orthogonal set of vectors in a stable way. Comparisons with different test matrices indicate that Arnoldi with DCGS2 is numerically equivalent to Arnoldi with CGS2. However, a theoretical analysis of this equivalence remains to be done as future work.

In the CGS2 algorithm, the multiple inner product operations, $h_j = V_j^T w_{j+1}$ and $c_j = V_j^T w'_{j+1}$, can be implemented in parallel with a single reduction operation each, so two global communication operations of length $j-1$ must be performed in each Arnoldi iteration. In the case of DCGS2, the two operations $h_j = \tilde{V}_j^T w_{j+1}$ and $c_{j-1} = V_{j-1}^T \tilde{v}_j$ can be joined together, so just one reduction operation of length $2j-3$ is required. The value of j is usually very small compared to the size of the problem, so the important achievement here is the latency hiding resulting from packing all the data in a single message. This is especially important in platforms with high-latency interconnection networks. On the other hand, in this way the number of synchronization points is halved with respect to CGS2. All these improvements should lead to better parallel performance.

From the practical point of view, when implementing the DCGS2 algorithm in SLEPc, the communication merging 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.

4. Performance Analysis

In order to compare the parallel efficiency of the DCGS2 orthogonalization scheme with respect to CGS2, several test cases were analyzed in different parallel platforms. The comparison was carried out in the context of an explicitly restarted Arnoldi implementation available in SLEPc. All the tests were repeated for both orthogonalization algorithms. In all cases, the solver was requested to compute 10 eigenvalues with a maximum of 50 basis vectors. In order to minimize the effect of slight variations in the iteration count, measured wall times were divided by the number of iterations.

Two types of tests were considered. On one hand, three matrices arising from real applications (see table below) were used for measuring the parallel speed-up. This speed-up is calculated in the usual way, as the ratio of elapsed time with p processors to elapsed time with one processor. 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.

Name	Order	Non-zeros	Sparsity	Reference
AF23560	23,560	484,256	0.0087 %	[1]
Andrews	60,000	760,154	0.0211 %	[3]
Lin	256,000	1,011,200	0.0015 %	[3]

The first machine used for the tests was a cluster 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 used in the tests reported in this section.

The speed-up measured in the Xeon cluster for the three real cases is shown in Figure 2. There are two plots for each matrix. The left one corresponds to the natural ordering of the matrix whereas the right one corresponds to a permuted version. This permutation was computed with Parmetis [6] in order to improve parallel efficiency of the matrix-vector product, which has a significant impact in the overall Arnoldi performance. The goal is to isolate as much as possible the performance of the orthogonalization part.

Both CGS2 and DCGS2 show overall good parallel performance. However, the speed-up curve corresponding to DCGS2 is always above that of CGS2, and the gap grows as the number of pro-

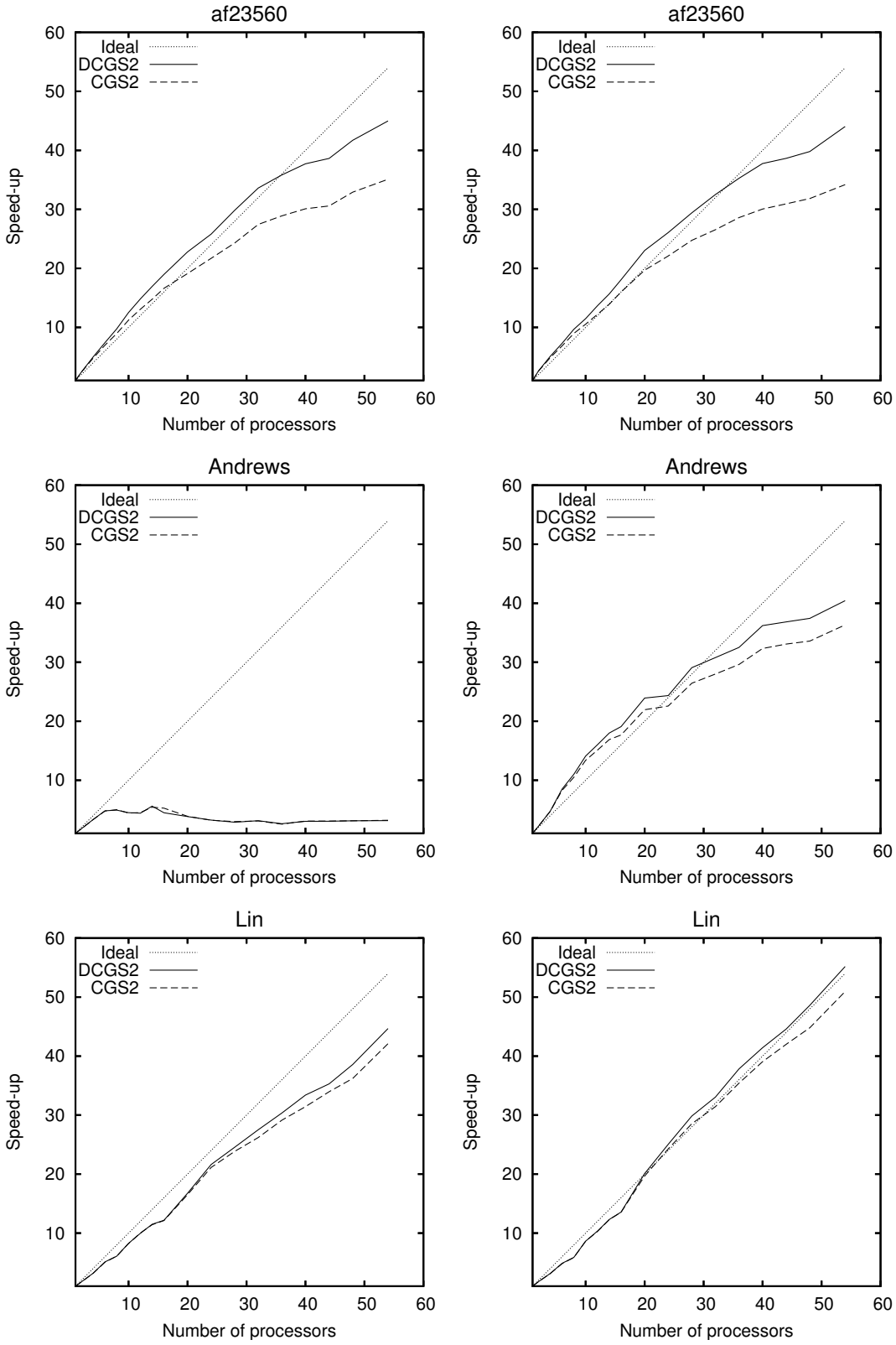

Figure 2. Measured speed-up for fixed size matrices in Xeon cluster with natural ordering (left) and with reordering (right).

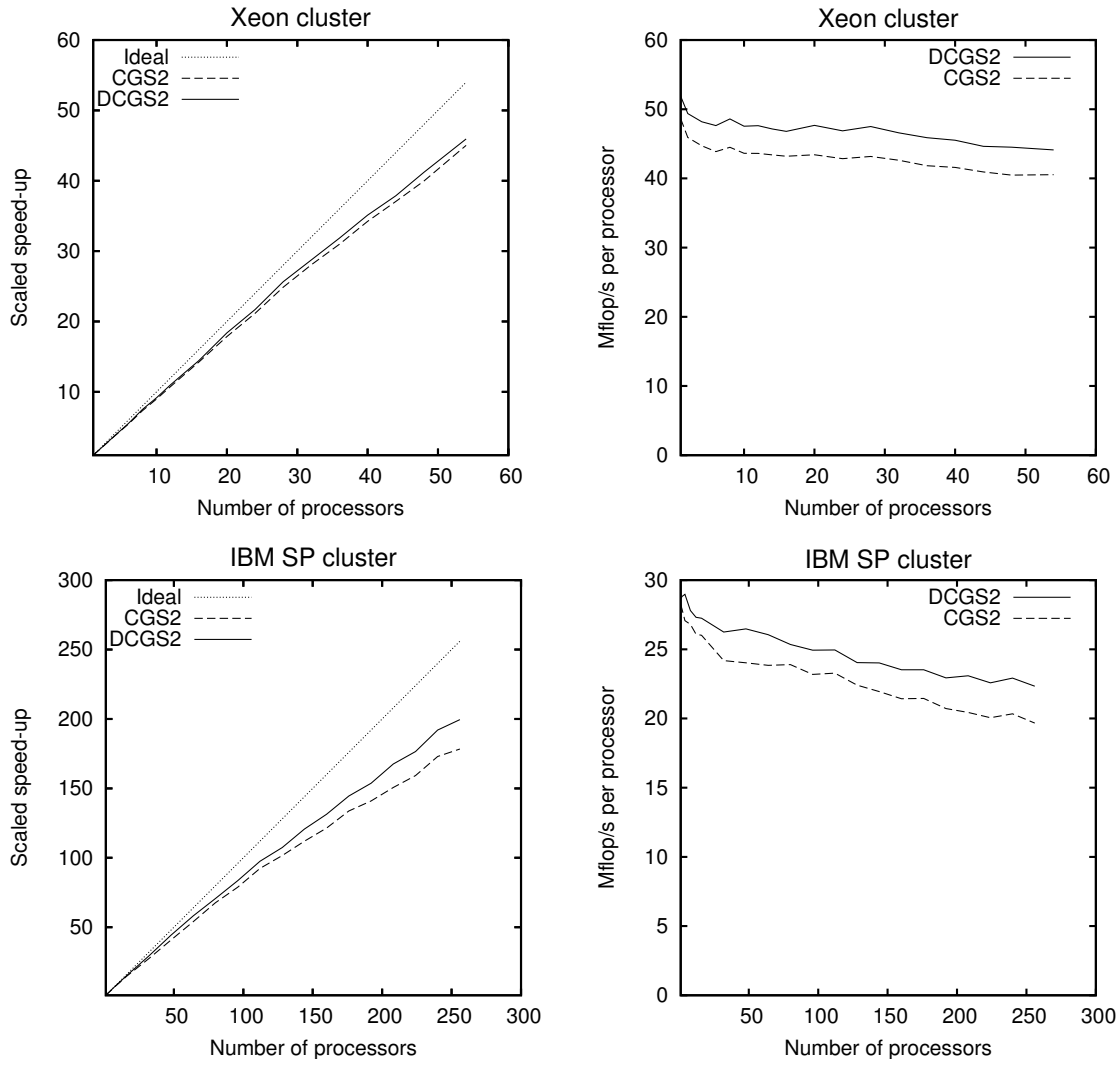

Figure 3. Scaled speed-up and Mflop/s per processor for the synthetic test case in Xeon cluster (top) and IBM SP (bottom).

cessors increases. 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 to the number of processors, p. For this analysis, a tridiagonal matrix with random entries was used, with a dimension of $10,000 \times p$. The results in the Xeon cluster (see Figure 3) show overall good parallel performance in both alternatives, with DGCS2 having higher Mflop rate and a marginally better speed-up than CGS2.

In order to extend the analysis to more processors, this last test was repeated in an IBM SP RS/6000 computer with 380 nodes and 16 processors per node. The results up to 256 processors

are included in Figure 3 as well. As before, both variants have good parallel performance, but the DCGS2 line is significantly closer to the ideal speed-up, especially for large number of processors.

5. Discussion

In this work, a new orthogonalization algorithm called DCGS2 has been proposed in order to improve parallel efficiency in the context of Arnoldi eigensolvers. This algorithm is based on Classical Gram-Schmidt with reorthogonalization and its main goal is to reduce the number of synchronization points in parallel implementation. The performance results presented in section 4 show that the proposed algorithm achieves better efficiency in all the test cases analyzed, and scales better when increasing the number of processors.

The proposed orthogonalization technique has been tested in the context of the Arnoldi method, but it could possibly be adapted to other eigensolvers as well, such as Lanczos with full reorthogonalization or Jacobi-Davidson.

Although the DCGS2 algorithm has displayed numerical robustness in our tests, the algorithm deserves further refinement and work. In particular, a theoretical rounding error analysis should be carried out. Also, a theoretical study would be required in order to establish whether a selective reorthogonalization criterion could be applied.

Finally, as a future work, there is the possibility of further refining the parallelization of the Arnoldi method by applying the same latency hiding technique to the normalization stage in order to eliminate another synchronization point.

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

References

[1] Z. Bai, D. Day, J. Demmel, and 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, 1997. Available at http://math.nist.gov/MatrixMarket.

[2] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Math. Comp.*, 30(136):772–795, October 1976.

[3] T. Davis. University of Florida Sparse Matrix Collection. NA Digest, 1992. Available at http://www.cise.ufl.edu/research/sparse/matrices.

[4] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3), September 2005.

[5] Walter Hoffmann. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41(4):335–348, 1989.

[6] G. Karypis, K. Schloegel, and V. Kumar. *ParMeTis: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 2.0*. University of Minnesota, Dept. of Computer Science, September 1999.

[7] Y. Saad. *Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms*. John Wiley, New York, 1992.

[8] Roger B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Numerical Linear Algebra with Applications*, 4(4):305–331, 1997.

[9] Andreas Stathopoulos and Kesheng Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23(6):2165–2182, 2002.

[10] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.