Reducing Synchronization on the Parallel Davidson method for the Large, Sparse, Eigenvalue Problem

Andreas Stathopoulos and Charlotte F. Fischer
Computer Science Department
Vanderbilt University
Nashville, TN 37235

Abstract

The Davidson method is extensively used in quantum chemistry and atomic physics for finding a few extreme eigenpairs of a large, sparse, symmetric matrix. It can be viewed as a preconditioned version of the Lanczos method which reduces the number of iterations at the expense of a more complicated step. Frequently, the problem sizes involved demand the use of large multicomputers with hundreds or thousands of processors. The difficulties occurring in parallelizing the Davidson step are dealt with and results on a smaller scale machine are reported. The new version improves the parallel characteristics of the Davidson algorithm and holds promise for a large number of processors. Its stability and reliability is similar to that of the original method.

1 Introduction

The eigenvalue problem, $Ax = \lambda x$, is central to many scientific applications. In these applications it is common for A to be real, symmetric, and frequently very large and sparse. Examples abound both in engineering and in science [15, 7, 8, 4]. Following the recent advances in High Performance Computing technology, the demands for even higher order matrices have increased. Numerous large matrix methods have been developed that solve only for a few extreme eigenpairs, and do not modify the matrix since usually it cannot be stored in memory or on disk in full.

The Lanczos [16] and the Davidson [3] methods are amongst the most widely used methods. The Lanczos iteration builds an orthogonal basis for the Krylov subspace, $\mathcal{K}(A, g, m) = \text{span}\{g, Ag, \ldots, A^m g\}$, from which the required eigenvectors are approximated through a Rayleigh-Ritz procedure. The attraction of the Lanczos method is that the orthogonal basis is built through an easy-to-compute three-term recurrence and the projection of A onto $\mathcal{K}(A, g, m)$ is a tridiagonal matrix of order m. The Davidson method builds a subspace that deviates from the Krylov subspace. In each iteration the Rayleigh-Ritz procedure is solved and the residual of the current approximation is preconditioned ($(M - \lambda I)^{-1}(A - \lambda I)x$) before it enters the basis. Therefore, the new vector should be explicitly orthogonalized to all previous basis vectors. The Davidson method can be considered a preconditioned version of the Lanczos method [12, 13, 18]. The convergence is much faster but the work per iteration is increased. In quantum chemistry and atomic physics calculations the size and structure of the matrix justifies the extra work per step, even with a preconditioner as $M = Diag$, where $Diag$ is the diagonal of A.

In the last ten years, considerable effort has been put into transporting the Lanczos method to parallel computers. This effort has been fruitful in machines that offer vectorization and coarse grain parallelism. For medium and fine grain parallelism more difficulties had to be faced because of the iterative nature of the method. However, due to the simplicity of the recurrence, good performance was achieved on these architectures by reducing the synchronization overheads. The Davidson method has been optimized for vector and coarse grain machines [1, 18, 21, 22], but there are no previous attempts to transport the Davidson to a medium grain machine (tens or a few hundreds of processors). A reason for this is the more complicated nature of the algorithm and the increased synchronization requirements.

In this paper the original algorithm is restructured so that it requires only one synchronization per iteration. In some cases the numerical stability of the restructured algorithm is relaxed but in quantum chemistry and atomic physics applications shifting the matrix with the extreme diagonal element restores most of the lost numerical accuracy. Moreover, when used with frequent restarting, the two algorithms are practically equivalent. The gain in time and speedup is significant when the number of processors is large.

In Section 2 the original Davidson algorithm is outlined. In Section 3, the distribution issues and the synchronization needs of a parallel Davidson are addressed. The restructured algorithm is presented in Section 4. Sections 5 and 6 cope respectively with time and numerical behavior of the new algorithm. Concluding remarks and future directions are given in Section 7.

2 The Original Davidson Method

The Davidson method is similar to the Lanczos method. If no preconditioning is used, the space created by Davidson is identical with the Lanczos space. In this case the Davidson method becomes an expensive way of implementing the Lanczos procedure; explicit orthogonalization is used, the Rayleigh-Ritz is applied in each step, and the projection matrix is full [2, 18, 12, 10, 22].

If preconditioning is used instead of the residual as the next basis vector, a better estimate can be chosen from perturbation theory: $b = (M - \lambda I)^{-1} Res(x)$, where M is a good approximation to matrix A, λ very close to the eigenvalue and $Res(x)$ the residual of the current approximation x [5]. Davidson proposed the diagonal of A as the preconditioner M. In applications where the eigenvectors have only one dominant component this choice is sufficient to provide extremely rapid convergence. Applications in quantum chemistry and atomic physics demonstrate the benefits [5, 22]. The same preconditioner is considered hereafter because of its excellent parallel properties.

Assuming that the K lowest eigenpairs are required, a brief description of the original algorithm follows:

The Original Algorithm

Step 0: Set $m = K$. Compute initial Basis $B = \{b_1, \ldots, b_m\} \in \Re^{N\times m}$, also $D = AB = \{d_1, \ldots, d_m\}$, and the projection of size $m \times m$, $S = B^T AB = B^T D$.

Repeat until converged steps 1 through 8:

1. Solve $SC = C\Lambda$, with $C^T C = I$, and Λ diagonal.
2. Target one of the K sought eigenpairs, say (λ, c).
3. If the basis size is maximum truncate: $D \leftarrow DC,\ B \leftarrow BC,\ C = I_K,\ S = \Lambda,\ m = K$.
4. Compute $R = (Diag - \lambda I)^{-1}(Dc - \lambda Bc)$.
5. Orthogonalize: $b_{new} = R - \sum b_i b_i^T R$, normalize: $b_{new} \leftarrow b_{new}/||b_{new}||$.
6. Matrix vector multiply: $d_{new} = Ab_{new}$
7. Compute the new column of S: $S_{i,m+1} = b_i^T d_{new},\ i = 1, \ldots, m+1$.
8. Increase m.

The above algorithm can be extended in several useful ways that improve its functionality and its run-time behavior [22, 5].

3 Transporting to a Parallel Computer

Previous results have shown that the Davidson algorithm can be implemented efficiently on parallel-vector computers (both shared and distributed memory) when the number of processors is small [18, 21]. The necessity of using more processors is evident, not because of the intolerably long execution of the algorithm but because of the increasing storage demands. Large applications can easily saturate the memory of hundreds of processors with the auxiliary vectors, without even storing the matrix. Care should be taken when this transition is made, so that the algorithm does scale up with the processors. Specifically, considerations about the distribution of the work arrays and the matrix, and about the synchronization-communication bottlenecks of the algorithm are in order.

3.1 Distribution Issues

The distribution of the matrix A onto the processors is the most compelling need. The matrix A affects the algorithm only through the user provided matrix-vector multiply. The distribution of A can be also left to the user so as to suit the specific multiplication routine. In many applications the matrix is large enough to be stored only on disc or recomputed each time it is needed [14]. The only assumption used in this paper about the matrix vector multiply is that it accepts and returns vectors in the following specified format.

The arrays D and B are the only "long" arrays needed in the algorithm, and their distribution can be performed in either of their two dimensions. The Davidson method was designed to cope with problems of large order. Thus, the number of columns in D and B is small, because of resource limitations. In parallel computers with several tens or hundreds of processors the number of vectors is not large enough to allow every vector to reside on one processor, hence distribution along the long dimension is considered.

Distribution can be performed in a wrap-around row fashion, or in terms of blocks of contiguous rows. The second option is followed in this implementation. If $nodes$ is the number of processors, each processor stores $\lfloor \frac{N}{nodes} \rfloor$ rows, except the first $\mathrm{mod}(N, nodes)$ ones that store $\lfloor \frac{N}{nodes} \rfloor + 1$ rows. The same distribution is also assumed for the elements of the diagonal of the matrix, $Diag$, used in the preconditioning step. The load imbalancing ratio is then limited to $\frac{min(\mathrm{mod}(N,nodes),1)}{N/nodes} \leq \frac{nodes}{N}$, which is negligible for large matrices. The two arrays S and C are very small (size $m \times m$) and they are duplicated in all processors.

3.2 Synchronization Needs

The above choice of distribution facilitates the parallel execution of vector updates like addition and scaling (daxpy and dscal operations). Each processor simply updates the local piece of the vector. Reduction operations (ddot) require two steps. First, a parallel computation of the local dot-products and second, a global addition of the partial results ($\log nodes$ step). Besides the communication costs, the second step introduces a synchronization point that serializes the processors preventing them from exploiting later parallelism.

The Lanczos algorithm can be given in a form where only one synchronization point is necessary per iteration. The same can not be easily achieved with the Davidson algorithm. Synchronization stemming from dot products appears in the following points in the Davidson algorithm:

1. The inner products $b_i^T R,\ i = 1, \ldots, m$ in orthogonalization.

2. The computation of the norm $||b_{new}||^2 = b_{new}^T b_{new}$.

3. The computation of the new column: $S_{i,m+1} = b_i^T d_{new},\ i = 1, \ldots, m+1$.

Data dependencies prohibit any postponement of global addition of inner products to some common synchronization point: Point (3) depends on Point (2) which depends on Point (1). The existence of three well separated synchronization points in one iteration, places strict limits on the efficient parallelization of the algorithm.

The solution of the Rayleigh-Ritz small system (step 1) could introduce additional synchronization if a parallel algorithm was used. Because of the small size of this system, the benefits from a parallel execution do not account for the additional overheads. Consequently, step 1 of the algorithm is executed identically by all processors.

4 The Restructured Davidson Method

In each iteration a new basis vector is computed from the current information. Since the information is distributed through the processors, the need of at least one synchronization point is obvious if the new vector is to contain the new Davidson direction. An alternative way of proceeding is the s-step methodology which is not considered in this paper [11]. To achieve only one synchronization point in each iteration, the above dependencies must be removed.

4.1 Removing the dependencies

The objective is to carry out all necessary inner products independently. After a single global addition, their values can be used to compute the new basis vector, the new column of S, and d_{new}. Since b_{new} is not available before the synchronization, the computation of $||b_{new}||$, S and D without an additional synchronization requires the use of the equivalent form:

$$b_{new} = R - \sum_{i=1}^{m} b_i b_i^T R. \qquad (1)$$

By substituting (1) for b_{new} whenever b_{new} is needed, the dependencies disappear. The following formulae present this approach.

$$\begin{array}{lrcll} \text{If} & V & = & AR, & (2) \\ \text{and} & rar & = & R^T V & (3) \\ & rr & = & R^T R & (4) \\ & t_i & = & b_i^T R & (5) \\ & g_i & = & b_i^T V, \quad i = 1, \ldots, m & (6) \end{array}$$

then the new vectors are given by:

$$||b_{new}|| = \sqrt{rr - \sum_{i=1}^{m} t_i^2} \qquad (7)$$

$$S_{i,m+1} = \frac{g_i - \sum_{j=1}^{m} S_{ij} t_j}{||b_{new}||}, \quad i = 1, \ldots, m \qquad (8)$$

$$S_{m+1,m+1} = \frac{rar - \sum_{i=1}^{m} t_i(g_i + S_{i,m+1}||b_{new}||)}{||b_{new}||^2} \quad (9)$$

$$b_{new} = (R - \sum_{i=1}^{m} b_i t_i)/||b_{new}|| \quad (10)$$

$$d_{new} = (V - \sum_{i=1}^{m} d_i t_i)/||b_{new}|| \quad (11)$$

Since the four inner products (3–6) are independent, one synchronization point is enough to compute their values. These values can be used later for the updates (7–11). The "long" updates (10–11) can be computed in parallel.

4.2 The Restructured Algorithm

The following algorithm is a restructured version of the Davidson algorithm that uses the above formulae to compute the elements of the next iteration. It requires only one synchronization point per iteration.

Step 0: Set $m = K$. Compute initial Basis $B \in \Re^{N \times m}$, also $D = AB$, and the projection of size $m \times m$, $S = B^T AB = B^T D$.

Repeat until converged steps 1 through 14:

1. Solve $SC = C\Lambda$, with $C^T C = I$, and Λ diagonal.
2. Target one of the K sought eigenpairs, say (λ, c).
3. If the basis size is maximum truncate: $D \leftarrow DC,\ B \leftarrow BC,\ C = I_K,\ S = \Lambda,\ m = K$.
4. Compute $R = (Diag - \lambda I)^{-1}(Dc - \lambda Bc)$.
5. Matrix vector multiply: $V = AR$.
6. Locally compute $t_i = b_i^T R,\ g_i = b_i^T V,\ i = 1, \ldots, m$ and $rar = R^T V,\ rr = R^T R$.
7. Synchronize: Globally sum the corresponding t_i, g_i, rar, rr.
8. Set $||b_{new}|| = \sqrt{rr - \sum_{i=1}^{m} t_i^2}$.
9. Set $S_{i,m+1} = (g_i - \sum_{j=1}^{m} S_{ij} t_j)$.
10. Set $S_{m+1,m+1} = rar - \sum_{i=1}^{m} t_i(g_i + S_{i,m+1})$.
11. Scale $S_{i,m+1} \leftarrow S_{i,m+1}/||b_{new}||,\ i = 1, \ldots, m+1$.
12. Set $b_{new} = (R - \sum_{i=1}^{m} b_i t_i)/||b_{new}||$.
13. Set $d_{new} = (V - \sum_{i=1}^{m} d_i t_i)/||b_{new}||$.
14. Increase m.

The original and the restructured algorithm have the same first four steps. The matrix-vector multiplication is also common although it appears in different steps. Table 1 gives a comparison of the arithmetic involved in each iteration for the non-common steps. The amount of work per iteration is slightly increased compared to the work of the original algorithm. The restructured algorithm requires two less dot products but m more daxpy operations. Since the daxpy is an update operation, it can be performed fully in parallel and for a large number of processors the extra work is insignificant.

	Original	Restructured
dot	$2m + 4$	$2m + 2$
daxpy	m	$2m$
other (flops)	-	$\mathcal{O}(2m^2)$

Table 1: Comparison of the iteration work for the non-common steps of the two algorithms.

5 Timing Results

The two versions of the algorithm have been implemented on a iPSC/860 hypercube, with 8Mb of memory per node. The large network latency and low computation-to-communication ratio of the iPSC/860 suggest the use of coarse grain parallelism [6]. However, the high processor speed is achievable in optimal situations, thus medium grain applications may perform equally well on the iPSC/860. The major advantage of using a hypercube architecture for the above iterative methods is the node topology which permits the global addition-synchronization in $\log nodes$ steps without channel contention.

The matrix used in the timing experiments is a single diagonal matrix with elements $A_{ii} = i$, except the initial block $A_{ij} = -1,\ i, j \leq 30$ and $i \neq j$. This form facilitates a fully parallel execution of the matrix vector multiply and it is economical to compute. It requires a vector update operation and a small matrix-vector multiply, operations that consist a small percentage of the iteration-work of the algorithm. In this way the algorithm timings are affected neither by extra communication on the multiplication routine nor by the dominating parallelizing properties of a large matrix. The sizes of matrices tested are varied from small cases, $N = 5000$, up to a moderately large case $N = 10^6$. In the latter case, the memory requirements impose the use of at least 16 processors. For all the experiments the lowest eigenpair is required, the algorithm is restarted every 7 iterations, and it is run from twenty to forty iterations depending on the size of the matrix. Results are reported in node configurations from 1 to 64 nodes.

The objective is to reduce synchronization on the Davidson algorithm and hence improve the speedup for a large number of processors. This is important when a bigger number of processors is needed to compensate for the increased memory needs of large applications. Figures 1 and 2 illustrate the speedup curves for both versions applied on seven different sizes of the above test matrix and for all node configurations.

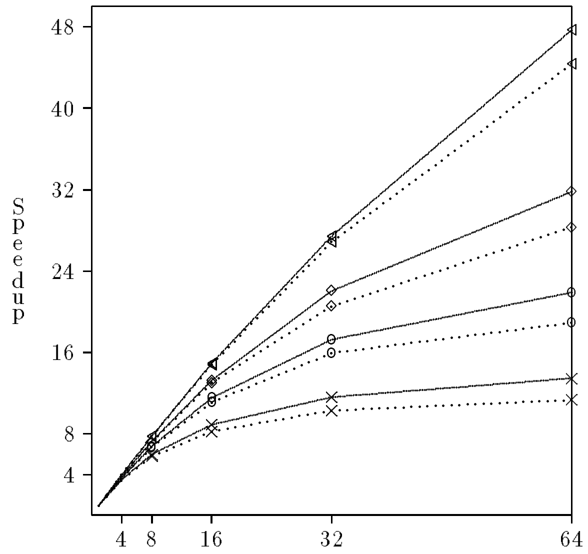

Figure 1: Speedup curves for the test matrix and sizes:($\times$) : $N = 5000$, ($\circ$) : $N = 10000$, ($\diamond$) : $N = 20000$, ($\triangleleft$) : $N = 50000$. The original version is depicted by dotted lines.

In both figures the speedup increases with the size of the cube and the size of the matrix. For some small cases in Figure 1 the speedup levels off but does not decline, at least for the cube-sizes used, and for the large cases in Figure 2 it is close to linear. In addition, most of the small cases demonstrate an ascending speedup character which is promising for execution in multiprocessors with hundreds of nodes. On the other hand, the "leveling off" is justified by the fact that all of the small cases in Figure 1 have execution times close or much lower than 1 sec.

The comparison of the restructured with the original method verifies what is theoretically expected. There is a consistent improvement over the older version that increases with the number of processors. This is the consequence of the reduction on synchronization. In the original algorithm the more processors there are, the more time is wasted in synchronization and communication. The new version allows the processors to run independently until the following iteration, exploiting more of the available parallelism. The speedup increase over the older version varies from 5% to 18%. Projecting the experience from 64 nodes onto hundreds of nodes, the restructured algorithm should present significant improvements on the Davidson method. As the size of the matrix grows, there is an slight decrease in the difference between the two versions because each processor is assigned more work and this diminishes the effect of the synchronization overheads. However, the superiority of the restructured algorithm is obvious even for the case $N = 10^6$.

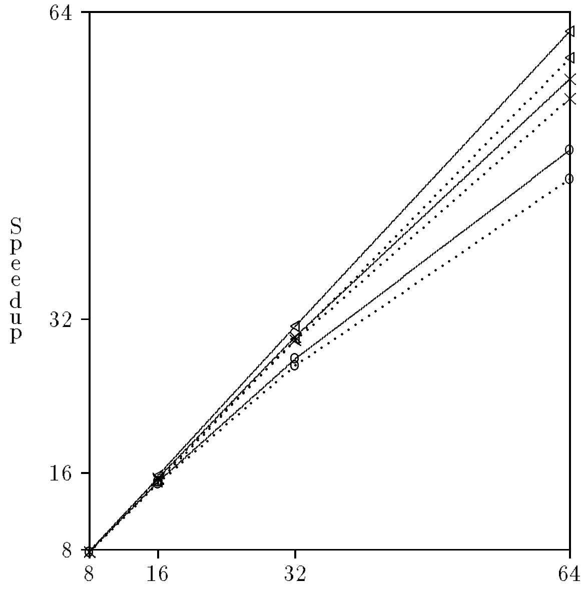

Figure 2: Speedup curves for the test matrix and sizes:($\circ$) : $N = 60000$, ($\times$) : $N = 200000$, and ($\triangleleft$) : $N = 10^6$. The original version is depicted by dotted lines.

In view of the increased arithmetic on the restructured algorithm, execution timings are also important to show the actual benefits. The restructured algorithm is slower on one node by 0.7% to 2.6%. This is due to the additional vector updates and the extra operations for calculating $S_{i,m+1}$. The better parallel properties offset the overhead after 8 or 16 processors and the new version steadily reduces the time. With 64 nodes the new algorithm is faster by 2.6% to 9%

and execution times show descendent tendencies while the original ones level off. For large cases, the break even point can increase to 16 or 32 processors, but this is only a small percentage of the processors required by these sizes. Figure 3 shows the differences in these execution times for some of the test cases.

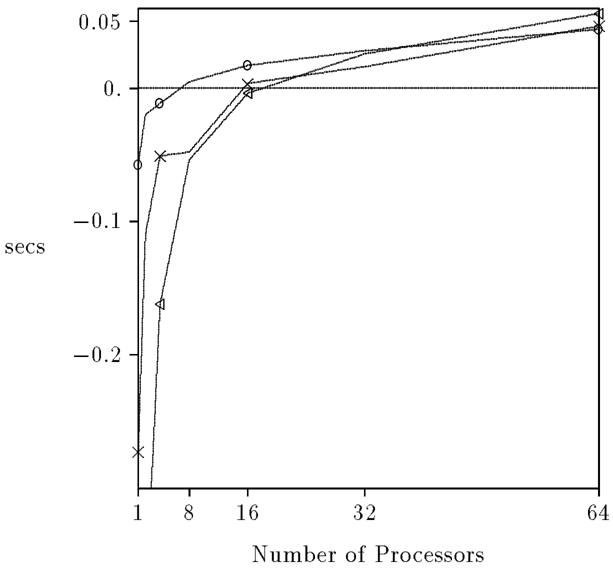

Figure 3: Time differecnce between the restructured and the original algorithm ($T_{orig} - T_{rest}$) for various node configurations and for the sizes: ($\circ$): N=10000, ($\times$): N=50000, ($\triangleleft$): N=60000.

Finally, it should be mentioned that since communication consists a $\log nodes$ step, synchronization is the bigger bottleneck for a large number of processors. When more complicated matrix vector multiplies are introduced, the larger load imbalances and delays favor the version with fewer synchronization points. In addition, the restructured version is also favored if smaller dimension topologies (as the mesh) are used.

6 Numerical Behavior

The restructured algorithm makes extensive use of values computed in previous steps to remove the dependencies from the same step. The results of this methodology have generally been regarded as numerically unstable [19]. Numerical instabilities are also observed for the restructured method when run without restarting, i.e., without step 3. However, these instabilities are not severe and the algorithm converges to a low residual threshold (see Figure 4) for the atomic physics test problems appearing on Table 2, [17, 7]. The original method improves the residual about two orders of magnitude. For the same test problems, the accuracy can be improved significantly if the matrix is shifted by the extreme diagonal element corresponding to the extreme required eigenvalue (see Figure 5). When restarting is used (usually every 15-20 iterations), the original and the restructured methods perform in a similar way (Figure 6). A brief explanation of the observed behavior appears below.

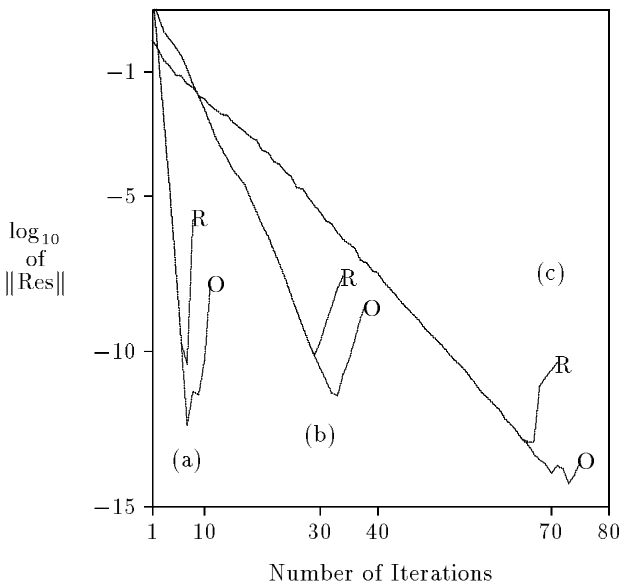

Figure 4: Convergence of the residuals of the restructured (R) and original (O) versions for the three cases in Table 2 with no restarting.

The numerical errors introduced from the use of old entries of S and D for computing the new columns in equations (8) and (11) are first considered. For comparison purposes and without loss of generality it is assumed that g_i and t_i are accurate since they are computed in the current step by simple dot products and matrix vector multiplies. These operations appear in the original Davidson as well. Assume that for every (i,j) S-element, the computed value, $S_{i,j}$, and the theoretically correct value $S^{real}_{i,j}$, satisfy:

$$S^{real}_{i,j} = S_{i,j} + \delta S_{i,j} \qquad (12)$$

$$S^{real}_{i,j} = g_i - \sum_{k=1}^{j-1} S^{real}_{i,k} t_k \qquad (13)$$

$$S_{i,j} = g_i - \sum_{k=1}^{j-1} S_{i,k} t_k \qquad (14)$$

Case	Program	Description	Size	Nonzero
(a)	(MCDF)	Lithium-like Uranium.	410	31087
(b)	(MCDF)	A complete active space calculation with n up to 4 for Beryllium-like Xenon.	2149	335416
(c)	(MCHF)	Li, 2S; by n method; n=6.	862	120735

Table 2: Description of the atomic physics test cases.

Substituting (13) and (14) to (12) yields:

$$\begin{aligned} \delta S_{i,j} &= -\sum_{k=1}^{j-1} \delta S_{i,k} t_k && (15) \\ \Rightarrow \quad |\delta S_{i,j}| &\leq \sum_{k=1}^{j-1} |\delta S_{i,k}||t_k| && (16) \end{aligned}$$

Under the initial assumptions a similar formula for the error of the D vectors can be found:

$$||\delta D_j|| \leq \sum_{k=1}^{j-1} ||\delta D_k|||t_k| \qquad (17)$$

Formulae (16) and (17) suggest the intuitive result that the errors from previous steps are accumulated to create unstable later steps. If the crude upper bound $|t_k| \leq ||R|| < 1$ is assumed, the error on both S and D is bound by the exponential form $\mathcal{O}(u2^j)$, where u is the error in the first step. Since t_k are the overlaps of the new vector with the old basis vectors, t_k are usually several orders of magnitude less than one in early iterations. Without the preconditioning, t_k would be exactly zero. As the algorithm proceeds, the basis fills up with meaningful vectors. When preconditioning is applied to the new residual, the improvement in the direction causes unavoidable larger overlaps with existing vectors. If the t_k are bounded by some very small number T, $|T| \ll 1$, the error grows like $\mathcal{O}(u(1+T)^j)$. The bound is still exponential but it requires hundreds of steps to become large.

When the basis is truncated after a number of steps, the restarting vectors for D and B as well as the new projection S, are computed from the step 3 in both algorithms. As a result, the errors introduced in the previous iterations in calculating S, C and D are now propagated to the new restarting vectors and matrices. If the restarting takes place before the errors in formulae (16) and (17) become large, the two versions of the algorithm become effectively equivalent and this explains the convergence behavior in Figures 4 and 6. In Figure 6 the test case (a) is not included because it converges before restarting occurs. In some ill-conditioned problems the restructured algorithm could demonstrate numerical instability. Fortunately, encountering such an instability would also imply a good basis and signify a quite close convergence.

Ideally, to avoid error propagation from restarting, step 3 of both algorithms should be performed explicitly, by the alternative steps:

(3.1) $B \leftarrow BC$

(3.2) reorthonormalize B

(3.3) $D = AB$

(3.4) $S = B^T D$

(3.5) solve $SC = C\Lambda$ for C.

This is an expensive procedure that introduces extra synchronization and additional matrix-vector multiplies. Practically, this procedure should be performed only near convergence or when the monitored error reaches a specified threshold [22].

Shifting the spectrum of the matrix is a widely used technique to improve numerical stability. Applications that benefit from the Davidson algorithm have large diagonal-dominance ratio [1], i.e., very small off-diagonal elements compared with the changes in magnitude between diagonal elements [10, 12, 22]. It is expected that some digits will be lost if arithmetic involves both the large and the small elements. The purpose of shifting is to reduce the norm of A which is an important factor in error bounds. For example, the matrices S and D are not normalized and their norm is bounded by $||A||$. Reducing $||S||$ and $||D||$ can substantially improve the method's numerical stability. The appropriate shift for optimally reducing $||A||_2$ is $(Diag_{min} + Diag_{max})/2$. However, this choice is not necessarily beneficial for the Davidson algorithm.

The shift to be chosen is one that brings the required eigenvalue very close to zero. In atomic

[1] The term is used to refer to the ratio $d = \min_{i,j} |(A_{ii} - A_{jj})/A_{ij}|$

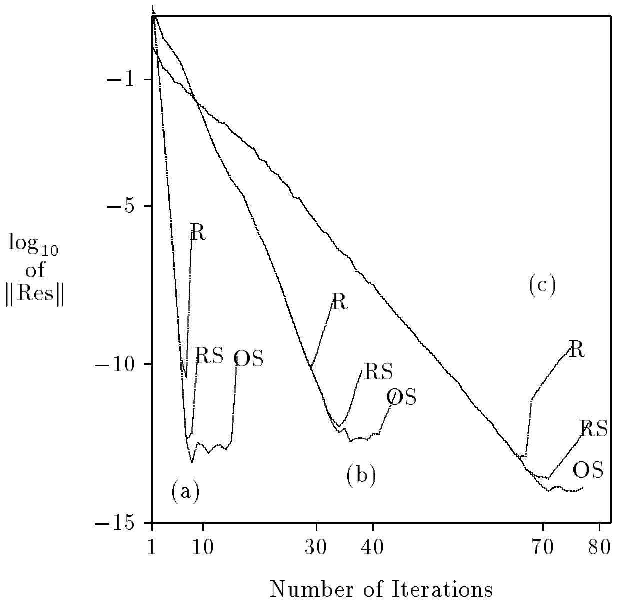

Figure 5: Convergence of the residuals of the restructured (R), the restructured with shift (RS), and the original with shift (OS) versions with no restarting.

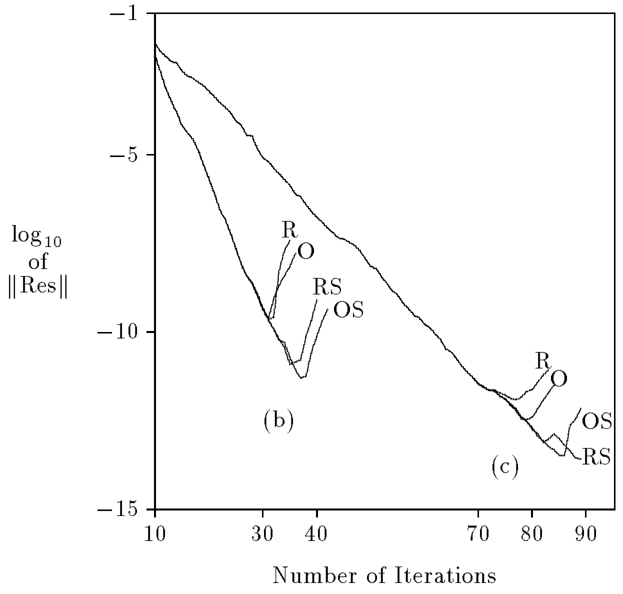

Figure 6: Convergence of the residuals of the restructured (R), the original (O), restructured with shift (RS), and original with shift (OS) with restarting.

physics and quantum chemistry applications the matrices have large diagonal-dominance ratio and therefore the eigenvectors have one dominant component. From the Gershgorin Circles theorem [9] it falls that the required eigenvalue is very close to the diagonal element corresponding to the dominant eigenvector component. This diagonal element is thus a good shift. Such a shift does not reduce $||A||$ in general. Since the goal in iterative algorithms is to find one eigenpair rather than the whole eigensystem, it is desirable to make the problem most stable near the solution. Therefore, the magnitude of the required eigenvalue and the eigenvalues close to it should be minimized. This is satisfied by the choice of the corresponding diagonal element as a shift. Figure 5 illustrates the gains of this strategy, by comparing it with the unshifted restructured and the shifted original version. The shifted version is better than the unshifted original method and similar to the shifted original version. The improvements are obvious even for the test case (c), which has the lowest diagonal dominance ratio of the three cases in Table 2.

In most of the current applications, the Davidson method is used with frequent restarting. If shifting is applied in addition, the numerical stability of the restructured algorithm is enchanced and it is proved as reliable as the original method. Results from the two test cases in Figure 6 demonstrate the similarity of the two versions.

7 Conclusions

The Davidson method is a useful tool for finding a few extreme eigenpairs of large symmetric matrices appearing in quantum chemistry and atomic physics. The large diagonal-dominance ratio and the very large size of these matrices advocate the use of the Davidson instead of the Lanczos method and the large sizes necessitate the use of hundreds of processors in todays multicomputers.

The three distinct synchronization points arising from dot products in the original Davidson algorithm are reduced to one by restructuring the algorithm. The new version demonstrates better speedups and faster execution times even on a fairly low number of processors. The experiments show that the benefits are expected to increase with the number of processors.

In some ill-conditioned problems the new version can exhibit numerical instability. However, the nature of the matrices where the Davidson is used, as well as the employment of simple and necessary techniques as restarting and shifting, alleviate the problem. Convergence close to working accuracy is observed for several atomic structure matrices.

The above results are encouraging, and further experiments have to be carried out in multicomputers with more processors and faster interconnects. Candidates include the CM-5 and the Intel Paragon. It is interesting to see if and how the absence of the hypercube topology can be counterbalanced by a fast network. Parallel implementation of the many extensions of the algorithm needs to be considered as well.

Acknowledgements

This work has been supported by a National Science Foundation grant No. ASC-9005687. The authors would like to thank the Joint Institute for Computational Science that make the 128-node iPSC/860 in Oak Ridge available for scientific experiments.

References

[1] G. Cisneros, M. Berrondo and C.F. Bunge, *DVDSON: A Subroutine to Evaluate Selected Sets of Eigenvalues and Eigenvectors of Large Symmetric Matrices*, Compu. Chem. 10 (1986) 281.

[2] M. Crouzeix, B. Philippe and M. Sadkane, *The Davidson Method*, Tech. Rep., Report TR/PA/90/45, CERFACS, Toulouse, 1990.

[3] E.R. Davidson, *The Iterative Calculation of a Few of the lowest Eigenvalues and Corresponding Eigenvectors of Large Real-Symmetric Matrices*, J. Comput. Phys. 17 (1975) 87.

[4] E.R. Davidson, in: Methods in Computational Molecular Physics, eds. G.H.F. Diercksen and S. Wilson (Reidel, Dordrecht, 1983) p. 95.

[5] E.R. Davidson, *Super-Matrix Methods*, Comput. Phys. Commun. 53 (1989) 49.

[6] T.H. Dunigan, *Performance of the Intel iPSC/860 and NCUBE 6400 Hypercubes*, Report ORNL/TM-11790, Oak Ridge National Laboratory, 1991.

[7] C.F. Fischer, *The MCHF atomic-structure package*, Comput. Phys. Commun. 64 (1991) 369.

[8] C.F. Fischer, The Hartree-Fock Method for Atoms: A Numerical approach, (J. Wiley, New York, 1977).

[9] G.H. Golub and C.F. Van Loan, Matrix Computations, 2nd ed. (Johns Hopkins Univ. Press, Baltimore, 1989).

[10] T.Z. Kalamboukis, *Davidson's algorithm with and without perturbation corrections*, J. Phys. A 13 (1980) 57.

[11] S.K. Kim and A.T. Chronopoulos, *A class of Lanczos-like algorithms implemented on parallel computers*, Parallel Computing 17 (1991) 763.

[12] R.B. Morgan and D.S. Scott, *Generalizations of Davidson's Method for Computing Eigenvalues of Sparse Symmetric Matrices*, SIAM J. Sci. Stat. Comput. 7 (1986) 817.

[13] R.B. Morgan and D.S. Scott, *Preconditioning the Lanczos Algorithm for Sparse Symmetric Eigenvalue Problems*, SIAM J. Sci. Stat. Comput. Vol. 14, No. 3 (1993).

[14] J. Olsen, P. Jørgensen and J. Simons, *Passing the One-Billion Limit in Full Configuration-Interaction (FCI) Calculations*, Chem. Phys. Lett. 169 (1990) 463.

[15] B.N. Parlett, *The Software Scene in the Extraction of Eigenvalues from Sparse Matrices*, SIAM J. Sci. Stat. Comput. 5 (1984) 590.

[16] B.N. Parlett, The Symmetric Eigenvalue Problem (Prentice-Hall, Englewood Cliffs, New Jersey, 1980).

[17] F.A. Parpia, I.P. Grant and C.F. Fischer, *GRASP2*, 1990.

[18] B. Philippe and Y. Saad, in: Proceedings of International Workshop on Parallel Algorithms and Architectures, eds. M. Cosnard at al. (North-Holland, Amsterdam, 1989) p. 33.

[19] Y. Saad, *Krylov Subspace methods on Supercomputers*, SIAM J. Sci. Stat. Comput. 10 (89) 1200.

[20] D.S. Scott, *Implementing Lanczos-like Algorithms on Hypercube Architectures*, Comput. Phys. Commun., 53 (1989) 271.

[21] A. Stathopoulos and C. F. Fischer, *A Hypercube Implementation of Davidson's Algorithm for the Large, Sparse, Symmetric Eigenvalue Problem*, Intel Supercomputer Users' Group, 1991 Annual Users' Conference, (1991) 343.

[22] A. Stathopoulos and C. F. Fischer, *A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix*, Comput. Phys. Commun., submitted.