LBNL-42828

Parallel Efficiency of the Lanczos Method for Eigenvalue Problems†

Kesheng Wu‡and Horst Simon‡

Abstract

Two of the commonly used versions of the Lanczos method for eigenvalues problems are the shift-and-invert Lanczos method and the restarted Lanczos method. In this talk, we will address two questions, is the shift-and-invert Lanczos method a viable option on massively parallel machines and which one is more appropriate for a given eigenvalue problem?

1 Introduction

This talk is on how to compute eigenvalues and eigenvectors of large sparse symmetric matrices on massively parallel machines. One of the the most commonly used algorithms for this task is the Lanczos method which projects the large eigenvalue problem onto a low dimensional Krylov subspace [7, 10]. In most cases, the Krylov subspace basis is built through a series of matrix-vector multiplications, the whole Lanczos method can be implemented with a matrix-vector multiplication routine and a few simple vector operations. This algorithm is highly efficient on parallel machines and it is effective for computing extreme and well separated eigenvalues.

To compute the interior or not well-separated eigenvalues, the shift-and-invert Lanczos method is one of the most effective methods. Since the eigenvalues of a matrix A are related to the eigenvalues of $(A-\sigma I)^{-1}$ by a simple relation $\lambda(A) = \sigma + 1/\lambda((A-\sigma I)^{-1})$ and the corresponding eigenvectors of A and $(A-\sigma I)^{-1}$ are identical, the shift-and-invert scheme computes the extreme eigenvalues of $(A-\sigma I)^{-1}$ and deduce the corresponding eigenvalues of A. With appropriate choice of σ, the extreme eigenvalues of $(A-\sigma I)^{-1}$ are well separated and can be easily computed. The shift-and-invert Lanczos method needs to build a Krylov subspace basis of $(A-\sigma I)^{-1}$ which is done by solving a series of linear systems involving the coefficient matrix $(A-\sigma I)$. The linear systems need to be solved accurately and the only reliable means to accomplish this is by using a direct method [5]. Because it is difficult to implement direct methods on distributed parallel computers, efficient parallel implementation has not been widely available until recently [1, 8]. This talk will present our study of the parallel efficiency of the shift-and-invert Lanczos method using these newly available direct solvers. The study will show that the shift-and-invert Lanczos method is

†This work was supported by the Director, Office of Energy Research, Office of Laboratory Policy and Infrastructure Management, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Energy Research of the U.S. Department of Energy.

‡Lawrence Berkeley National Laboratory/NERSC, Berkeley, CA 94720. Email: {kwu, hdsimon}@lbl.gov.

a viable option for solving large eigenvalue problems on distributed parallel machines but with significant limitations at the moment.

When the shift-and-invert Lanczos method cannot be used, the standard Lanczos method may need a large number of steps in order to reach an accurate solution. In recent years, there have been significantly improvements to the algorithm through restarting [3, 14]. These restarted methods use almost the same number of Lanczos steps as the non-restarted versions and they require much less computer memory than their non-restarted counterparts. In this talk we will also present some comparisons between the restarted Lanczos method and the shift-and-invert Lanczos method. Through these comparisons, we would like establish some guidelines on when to use the shift-and-invert scheme and when to avoid it.

2 The software packages

Before giving the comparison data, this section briefly describes the software packages used. The scopes of the different packages determine how the comparisons can be performed.

SPOOLES This parallel direct method package implements factorization procedures for symmetric and nonsymmetric, real and complex matrices. The basic algorithm is based on the fundamental supernode tree [1]. It can perform not only LDU factorization but also QR factorization. Further description about it can be found in reference [2, 1]. In our study, this package is only used to solve simple symmetric linear systems. In this case, it computes a diagonal matrix D and an upper triangular matrix U such that $U^T DU = A$. It performs pivoting if diagonal element is found to be small. The input matrix to the factorization routine needs to be in a special data structure. Only half of the off-diagonal entries are needed when the input matrix is symmetric or Hermitian.

PSPASES This parallel direct solver software package [8] implements a multifrontal Cholesky factorization for symmetric positive definite matrices, i.e., it computes an upper triangular matrix U such that $U^T U = A$. Because it assumes the input matrix to be positive definite, there is no need to perform pivoting. In practice, this also prevents it from factoring some ill-conditioned matrices. Even though the input matrix is symmetric, it requires the user to provide all nonzero entries. In addition, PSPASES can only be run on power of 2 number of processors. Compared to SPOOLES, PSPASES accepts limited type of matrices and provides less functionality. However, because of these limitations, it is able to perform its tasks more effectively in some cases.

PLANSO This is a parallel version of the Lanczos method with partial reorthogonalization (available at `http://www.nersc.gov/research/SIMON/planso.html`). It implements the standard non-restarted Lanczos algorithm for symmetric generalized eigenvalue problems. The partial reorthogonalization scheme monitors the loss of orthogonality among the Lanczos vectors and maintains a minimal orthogonality level that is necessary to compute the Ritz values accurately. This software package is used as the basis for the shift-and-invert Lanczos method with either PSPASES or SPOOLES as the linear system solver.

TRLan This software package implements a version of the thick-restart Lanczos method for eigenvalue problems [14]. In theory, the thick-restart Lanczos algorithm is equivalent to

TABLE 1
Time (seconds) used to factor a series of 3-D 27-point stencil matrices.

n	1	2	4	8	16	32	64	128	256	512
28	25.2	14.6	8.2	2.4	3.7	3.2	2.8	2.8	3.2	3.3
34	74.4	41.5	23.5	14.3	9.9	8.3	6.8	6.5	6.8	6.9
40		100.9	56.1	32.5	21.8	16.7	14.1	12.9	13.1	
48			152.3	87.1	55.5	40.3	32.4	29.2	28.1	
56				204.0	125.1	86.6	66.1	58.2		

(Column header spanning 1–512: # of PE)

TABLE 2
The size of matrix A and the triangular factor U.

n	$N(\equiv n^3)$	NNZ(A)	NNZ(U)
28	2.2×10^4	5.5×10^5	5.7×10^6
34	3.9×10^4	1.0×10^6	1.3×10^7
40	6.4×10^4	1.6×10^6	2.5×10^7
48	1.1×10^5	2.9×10^6	5.4×10^7
56	1.8×10^5	4.6×10^6	1.0×10^8

the implicitly restarted Lanczos algorithm [3] implemented in ARPACK [9]. The thick-restart Lanczos algorithm avoid some numerical instabilities of the implicit restarting scheme and the package TRLan implements the thick-restart Lanczos algorithm with more sophisticated restarting strategies, therefore TRLan is chosen as the representative of restarted Lanczos methods.

3 Performance characteristics

Since there are general purpose direct solver packages for distributed parallel machines, the shift-and-invert Lanczos method is an available option if the matrix is explicitly generated and the factors can be stored. To answer the question of whether it is an effective option, this section will show its performance characteristics by using PLANSO with SPOOLES and PSPASES.

In the shift-and-invert Lanczos method, the most time-consuming operation is solving the linear systems. When using a direct method, solving a linear system involves two steps, first factor the matrix into triangular form and then solve the resulting triangular linear systems. The factorization step is done once for each matrix and the triangular solution step is invoked for each right-hand side. Typically, it needs about 3 – 5 Lanczos steps to compute one eigenvalue. Unless a large number of eigenpairs are wanted, the factorization time dominates the whole computation. For these reasons, we will discuss the factorization time and solution time separately.

The first set of tests shown in Tables 1 and 3 uses SPOOLES to solve a set of linear systems with coefficient matrices that have the structure of 3-dimensional 27-point stencils, Table 1 shows the factorization time and Table 3 shows the solution time. The tests are conducted using a Cray T3E massively parallel computer. The test matrices are generated on a uniform $n \times n \times n$ grid. The blank cells in the lower left corner are due to memory

TABLE 3

Time (seconds) used to solve four linear systems with triangular factors of the 3-D 27-point stencil matrices.

	# of PE									
n	1	2	4	8	16	32	64	128	256	512
28	1.1	0.6	0.4	0.3	0.1	0.1	0.1	0.1	0.1	0.1
34	2.3	1.2	0.6	0.6	0.3	0.3	0.3	0.2	0.2	0.3
40		2.2	1.2	0.9	0.5	0.5	0.4	0.4	0.4	
48			2.4	1.4	1.3	0.8	0.8	0.7	0.7	
56				3.5	2.3	1.7	1.3	1.2		

TABLE 4

Information about the Harwell-Boeing test matrices.

name	N	NNZ	description
CT20	52329	1375396	an engine block
NASASRB	54870	2677324	shuttle rocket booster structure

limitations of the processors. Each processor of this T3E has only 256 MB(MegaBytes) of memory. From Table 2, it is clear that the factorized form has many more nonzero entries than the original matrix (NNZ(A) $\propto n^3$, NNZ(U) $\propto n^4$). All the test matrices shown here can be stored on one processor of the T3E, however, at least eight processors are need to store the triangular factor of the matrix based on the $56 \times 56 \times 56$ grid. When a sparse matrix is stored in memory, only a very small amount of additional memory is needed to perform a distributed matrix-vector multiplication efficiently. For simple matrices like these grid matrices, it is even possible not to explicitly store any of the nonzero entries of a matrix without sacrificing the efficiency of the sparse matrix-vector multiplication operation. Thus, the eigenvalues of a much larger matrix can be computed if only the matrix-vector multiplication is used.

The blank cells in the lower right corner of Tables 1 and 3 are due to MPI message header buffer limitation. The size of this buffer can be increased to allow more MPI messages being posted during the factorization. A limitation in the matrix generation routine prevents us from testing larger grid matrices. This limitation can be removed in the future as well.

From the data in Table 1 and 3 we see that if the minimum number of processors needed to perform the factorization is p_0, the parallel efficiency of using twice as many processors is roughly 80 percent. As more and more processors are used, the execution time quickly approaches a constant value and the parallel efficiency approaches p_0/p where p is the number of processors used. For these grid matrices, the execution time almost reaches the minimum when using $16p_0$ processors. Using more processors does not generate meaningful additional reduction in computer time. In fact, the time may actually increase.

The second set of test problems are a number of large symmetric matrices in Harwell-Boeing format [6]. Information about the matrices are listed in Table 4. The time used to factor these two test matrices by the SPOOLES and PSPASES are shown in Figure 1. It is clear that as more processors are employed for the factorization, the time does not proportionally decrease. In fact, as the number of processors change from 2 to 64, the

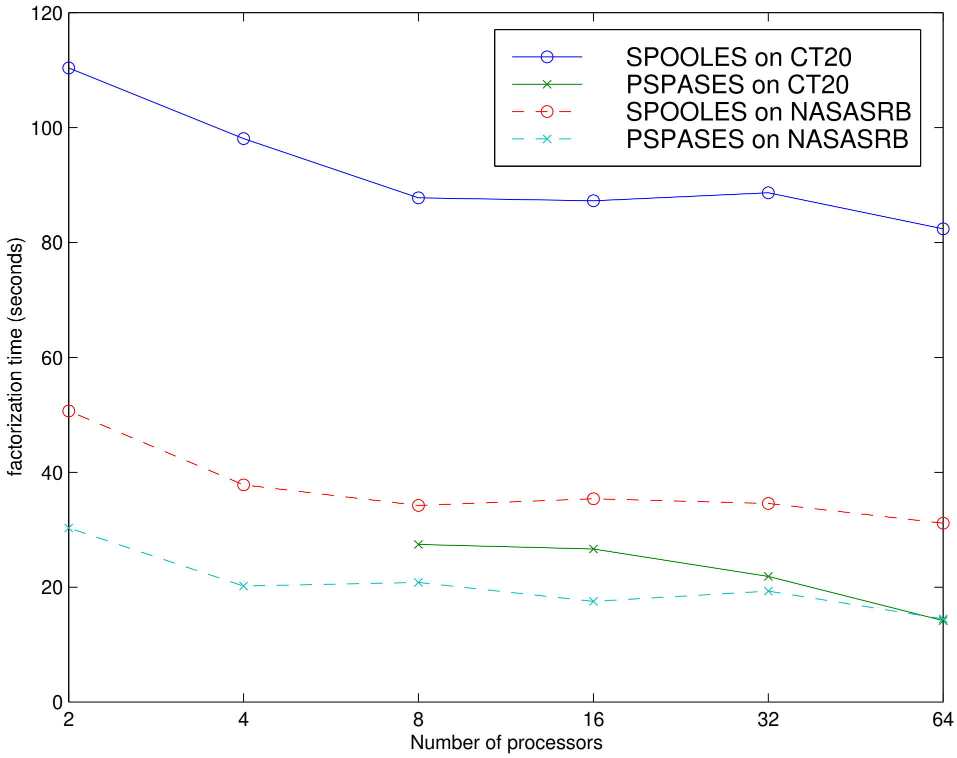

FIG. 1. *Time to factor the two Harwell-Boeing matrices.*

TABLE 5
Performance information about factorizations using 8 processors.

	CT20		NASASRB	
	SPOOLES	PSPASES	SPOOLES	PSPASES
NNZ(U)	1.0×10^7	1.4×10^7	1.0×10^7	1.3×10^7
OPS	5.2×10^9	1.3×10^{10}	2.8×10^9	5.9×10^9
MFLOPS	59.4	461	81.1	283

time decreased about 20 seconds giving a speedup of 1.3 – 2.1. Because these two matrices has more complicated nonzero patterns than the simple grid matrices used previously, the parallel efficiency of the factorization procedures are worse than before.

Some additional performance information about the factorizations are shown in Table 5. These data are collected using 8 processors. They are aggregate data, where NNZ(U) is the total number of nonzero entries in the triangular factor, OPS is the total number of floating-point operations used during the factorization and MFLOPS is the aggregate speed of all 8 processors. PSPASES generates about 30-40 percent more nonzero entries in U and uses about twice as many floating-point operations as SPOOLES. These differences are mainly due to different reordering strategies used. PSPASES employs parMETIS to perform reordering; SPOOLES performs many independent multi-section ordering and minimal degree ordering and then chooses the best one. The ordering algorithm in SPOOLES takes more time. On the two test matrices, the ordering generated by SPOOLES leads to smaller triangular factors but the ordering generated by PSPASES leads to less data

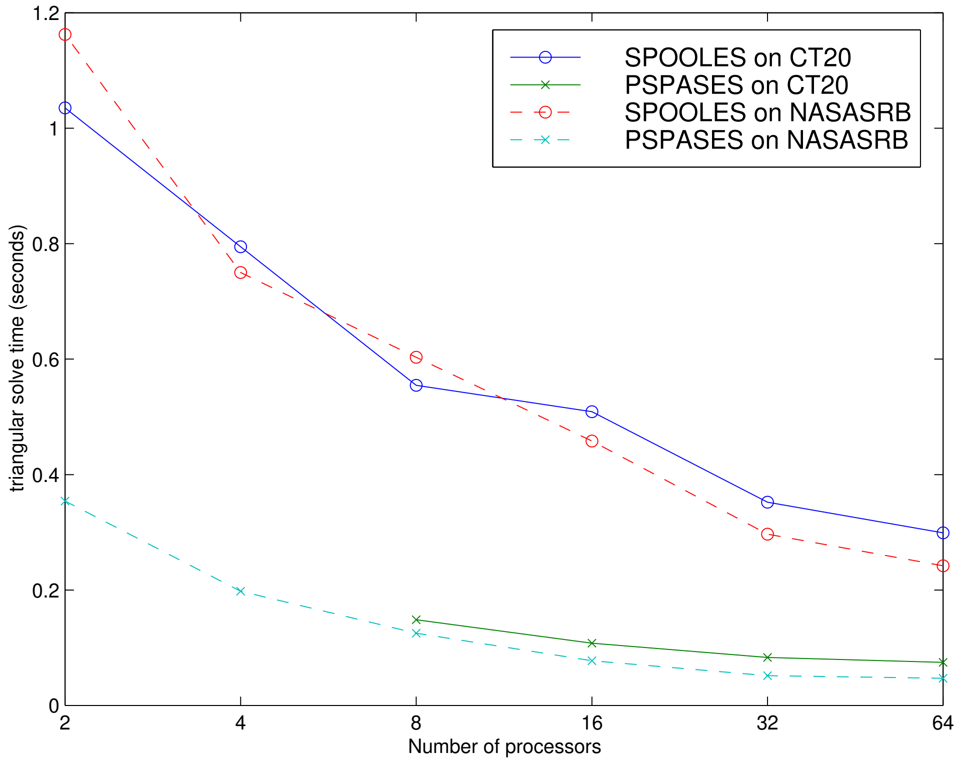

FIG. 2. *Time to solve with the triangular factors of two Harwell-Boeing matrices.*

communication. In addition, because PSPASES does not perform pivoting, its internal data structure and data communication pattern can be completely determined during the symbolic factorization phase and the numerical factorization phase takes less time than the corresponding phase in SPOOLES. The aggregate speed of PSPASES is considerably higher than that of SPOOLES.

The timing results of solving linear systems using the triangular factors of the two Harwell-Boeing matrices are shown in Figure 2. As the number of processors changes from 2 to 64, the solution time decreases by a factor of 4. The triangular solution stages of the two packages are more effective in taking advantage of more processors than the factorization stages. Because of differences in ordering, PSPASES also uses less time to perform the triangular solution than SPOOLES on the two test problems. Similar to the factorization time, the parallel efficiency for solving these two matrices are lower than solving the grid matrices test problems.

Figure 3 shows the total time used by PLANSO to take 25 Lanczos steps. The time shown here include both the factorization, the triangular solution and other operations needed by the Lanczos method such as the orthogonalization, the Rayleigh-Ritz projection and so forth. For both test matrices, 25 Lanczos steps is sufficient to compute 3 smallest eigenvalues and the corresponding eigenvectors. Comparing Figure 1 and 3, it is clear that the factorization time dominates the overall execution time. The total time used by PLANSO does not decrease significantly as the number of processors increases.

When the shift-and-invert scheme is not used, the most time-consuming operation in the Lanczos method is multiplying the matrix with a vector. Many research have shown that this operation can be parallelized effectively [4, 11, 12] and in turn the restarted Lanczos

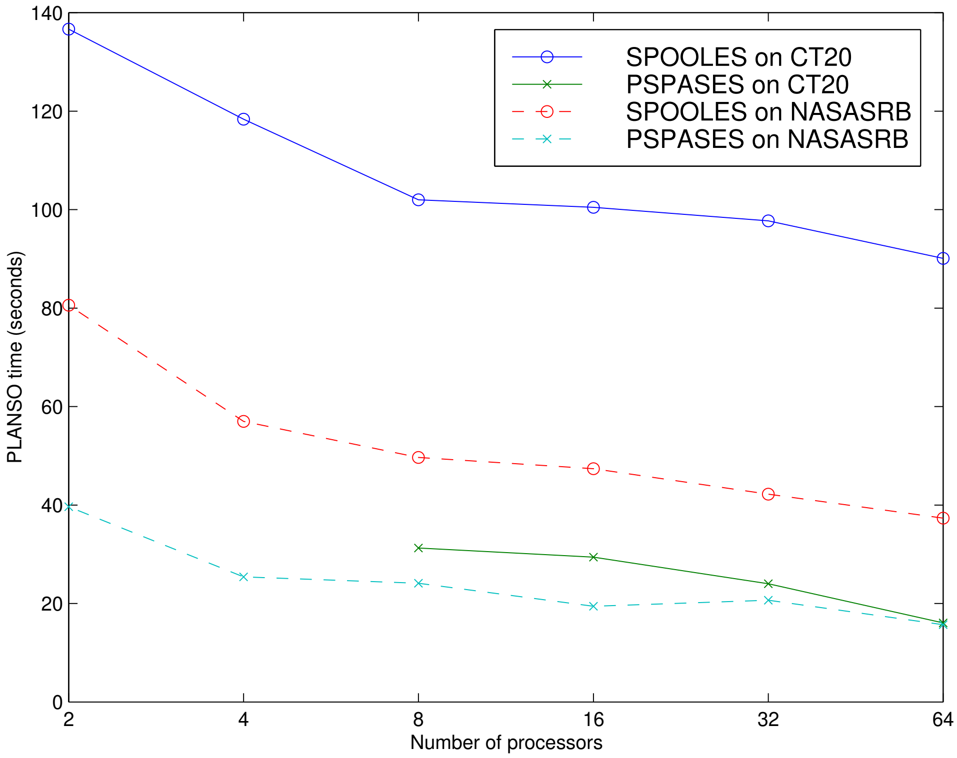

FIG. 3. *Time to run PLANSO for 25 steps.*

method can be run efficiently on distributed environments [13]. In particular, when using the uniform grid as the test problem, if the number of grid points is scaled as the number of processors increases, the perfect parallel efficiency can be achieved, i.e., the elapsed time to perform a fixed number of Lanczos steps remain constant as the problem size increases. Clearly, the direct methods are yet to achieve the same level of parallel efficiency.

4 Effectiveness of the restarted method

This section tries to answer the question when is the restarted Lanczos method or the shift-and-invert Lanczos method preferred? As long as an appropriate shift can be determined and the factorization can be computed, the shift-and-invert Lanczos method should be able to compute any eigenvalue and the associated eigenvector. If the goal is to compute some eigenvalues in the minimal number of Lanczos steps, the shift-and-invert scheme is usually preferred. However, typically the memory required to store the triangular factors of a matrix may be used to store hundreds of Lanczos vectors and the time to factor a matrix may be enough to perform thousands of matrix-vector multiplications. If the goal is to compute some eigenvalues in the least amount of time, the restarted Lanczos method may be preferred. This section will give some concrete examples to demonstrate the point.

When trying to compute the largest eigenvalues of the NASASRB matrix, 5 seconds are needed using the Lanczos method on 2 processors. Clearly, the shift-and-invert scheme is not appropriate here because the factorization takes about 30 seconds on 2 processors. On the other hand, as little as 21 seconds are needed to compute the smallest eigenvalues of NASASRB with shift-and-invert Lanczos method on 32 processors, but 2587 seconds are needed for PLANSO to compute the same eigenvalue on the same number of processors

TABLE 6
Number of Lanczos steps needed to compute five desired eigenvalues of three diagonal matrices using TRLan.

	basis size 25			basis size 50		
N	A_q	A_c	A_l	A_q	A_c	A_l
100	297	527	286	271	511	231
1000	7720	> 10000	1250	2995	> 10000	1032
10000	> 10000	> 10000	13691	> 10000	> 10000	4056

TABLE 7
The relative gap ratios.

N	A_q	A_c	A_l
100	3×10^{-4}	8×10^{-6}	2.5×10^{-3}
1000	3×10^{-6}	8×10^{-9}	1.6×10^{-4}
10000	3×10^{-8}	8×10^{-12}	1.2×10^{-5}

using only the matrix-vector multiplication. In this case, 11619 Lanczos steps are taken which means that 11619 Lanczos vectors are stored. On many computers, there is not enough memory to store this many Lanczos vectors. If 1000 Lanczos vectors are stored, TRLan needs 8546 seconds to compute the smallest eigenvalue of NASASRB on 8 processors (It uses roughly the same amount of aggregate time as the non-restarted case but uses less then one tenth of the computer memory). If less than 1000 Lanczos vectors are stored, the time needed to compute the smallest eigenvalue increases dramatically.

To further illustrate the differences between the shift-and-invert Lanczos method and the restarted Lanczos method, we compare their performance on a sequence of eigenvalue problems of varying difficulty. To limit the time required to conduct the experiment, we have chosen to use only diagonal matrices in this set of tests. Three different test matrices are used in this set of tests, they are

$$\begin{aligned} A_q &= \mathrm{diag}(1^2, 2^2, 3^2, \ldots), \\ A_c &= \mathrm{diag}(1^3, 2^3, 3^3, \ldots), \\ A_l &= \mathrm{diag}(\log 2, \log 3, \log 4, \ldots). \end{aligned}$$

The tests try to compute the five smallest eigenvalues of A_q and A_c and the five largest eigenvalues of A_l using TRLan. Table 6 shows the number of Lanczos steps needed to compute these eigenvalues. If the shift-and-invert Lanczos method were used, the wanted eigenvalues are computed within 20 Lanczos steps. If general sparse matrices with similar spectra are actually used and the restarted Lanczos method takes more than 10000 steps, the shift-and-invert Lanczos method is very likely to use less time. This is the reason why the restarted Lanczos method is stopped after 10000 steps. If the restarted Lanczos method completes the task within 10000, it is probably preferred over the shift-and-invert Lanczos method.

The relative gap ratios shown in Table 7 is a measure of difficulty of the eigenvalue problems. Let $\lambda_1, \lambda_2, \ldots, \lambda_N$ denote the eigenvalues of the test matrices in ascending order, the relative gap ratio shown in Table 7 are $(\lambda_2 - \lambda_1)/(\lambda_N - \lambda_1)$ for A_q and A_c, $(\lambda_N - \lambda_{N-1})/(\lambda_N - \lambda_1)$ for A_l. From this set of tests, we see that when the relative gap is less than 10^{-6}, TRLan cannot compute the eigenvalues in less than 10000 steps. The smallest eigenvalue of NASASRB has a relative gap ratio of 10^{-10}. The restarted Lanczos method needs more than 50000 steps using a basis of 1000 vectors. This indicates that it is possible for TRLan to compute very difficult eigenvalues, however it needs to keep a large number of Lanczos vectors and it may take a large number of Lanczos steps.

The comparison here is only on extreme eigenvalues, if an interior eigenvalue is wanted, the spectrum needs to be transformed so that the wanted eigenvector corresponds to an extreme eigenvalue. There are different ways to achieve this, but often the shift-and-invert scheme is the most effective one.

5 Summary

The availability of the general purpose sparse direct methods on distributed parallel computers makes it possible to easily implement the shift-and-invert Lanczos method for eigenvalue problems. In this paper we studied the parallel efficiency of the direct methods and the shift-and-invert Lanczos method. At the moment, the two sparse direct method packages tested do not exhibit the same level of parallel efficiency as a typical sparse matrix-vector multiplication routine.

The shift-and-invert Lanczos method spent largest portion of its execution time in factoring the sparse matrix. The efficiency of the factorization procedure dominates the overall efficiency of the shift-and-invert Lanczos method. If the minimal number of processors needed to perform the factorization is p_0, our tests shown that as more and more processors are used, the factorization time quickly approaches a minimum. It is not beneficial to use more than $16p_0$ processors. Our earlier tests shown that if p_0 processors are needed to store a matrix, the parallel efficiency is about 80 percent when performing the matrix-vector multiplication on $16p_0$ processors ($t_0/t_{16p_0} \sim 0.8*16$). At each Lanczos step, a linear system is solved by using the triangular factors generated during the factorization step. This triangular solution step exhibits higher parallel efficiency than the factorization step but its parallel efficiency is still considerably lower than that of the sparse matrix-vector multiplication. Overall, the restarted Lanczos method using only sparse matrix-vector multiplication has higher parallel efficiency than the shift-and-invert Lanczos method.

The issue of parallel efficiency is an important one on massively parallel computers. Given that the minimal elapsed time required by PLANSO with PSPASES to compute the smallest eigenvalue of NASASRB is 20 seconds. If TRLan can scale perfectly, even though it took 8546 seconds on 8 processors, it would only need 20 seconds on 3420 processors. As the direct solver packages mature, we are sure that their parallel efficiency will be enhanced to make them more effective on massively parallel computers.

On the question of when to use the shift-and-invert Lanczos method, our tests shown that if the relative gap ratio is greater than 10^{-6}, a restarted Lanczos method can compute the eigenvalues in a reasonable number of steps which indicates that the shift-and-invert Lanczos method will not be competitive. It is possible for the restarted Lanczos method (without shift-and-invert) to compute eigenvalues with relative gap ratio as small as 10^{-10}. However, because it would use a large amount of memory to store the Lanczos vectors and take a long time to compute the solutions, the shift-and-invert Lanczos method may be more effective in this case. The major limitation on the shift-and-invert Lanczos method is that the triangular factors often require significantly more memory to store than the matrices themselves and the currently available software packages can not significantly reduce their execution time by using more processors. If there is the memory to store the triangular factors, the shift-and-invert Lanczos method often compute the desired solution in a very small number of steps.

References

[1] C. C. Ashcraft, R. G. Grimes, D. J. Pierce, and D. K. Wah, *The user manual for SPOOLES, Release 2.0: An Object Oriented Software Library for solving sparse linear systems of equations*, 1998. The latest version of the software is available from NETLIB at `http://www.netlib.org/linalg/spooles/spooles.html`.

[2] C. C. Ashcraft and D. K. Wah, *The reference manual for SPOOLES, Release 2.0: An Object Oriented Software Library for solving sparse linear systems of equations*, 1998. The latest version of the software is available from NETLIB at `http://www.netlib.org/linalg/spooles/spooles.html`.

[3] D. Calvetti, L. Reichel, and D. Sorensen, *An implicitly restarted Lanczos method for large symmetric eigenvalue problems*, Electronic Transactions on Numerical Analysis, 2 (1994), pp. 1–21.

[4] U. V. Catalyurek and C. Aykanat, *Decomposing irregularly sparse matrices for parallel matrix-vector multiplication*, in Parallel Algorithms for Irregularly Structured Problems. Proceedins of Third International Workshop IRREGULAR '96 (Santa Barbara, CA, USA, 19-21 Aug. 1996), A. Ferreira, J. Rolim, Y. Saad, and T. Yang, eds., Berlin, Germany, 1996, Springer-Verlag, pp. 75–86.

[5] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse matrices*, Oxford University Press, Oxford, OX2 6DP, 1986.

[6] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Soft., 15 (1989), pp. 1–14.

[7] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD 21211, third ed., 1996.

[8] M. Joshi, G. Karypic, and V. Kumar, *PSPASES: Scalable parallel director solver library for sparse symmetric positive definite linear systems*, 1998. The latest version of the software package is available at `http://www-users.cs.umn.edu/~mjoshi/pspases/`.

[9] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK USERS GUIDE: solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods*, SIAM, Philadelphia, PA, 1998. ARPACK Software is available at `http://www.caam.rice.edu/software/ARPACK/`.

[10] B. N. Parlett, *The symmetric eigenvalue problem*, Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1998.

[11] Y. Saad, K. Wu, and S. Petiton, *Sparse matrix computations on the CM-5*, in Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. Sincovec, D. Keyes, M. Leuze, L. Petzold, and D. Reed, eds., Philadelphia, 1993, SIAM, pp. 414–420.

[12] R. S. Tuminaro, J. N. Shadid, and S. A. Hutchinson, *Parallel sparse matrix-vector multiply software for matrices with data locality*, Tech. Rep. SAND95-1540J, Sandia National Laboratories, Albuquerque, NM, 1995.

[13] K. Wu and H. Simon, *A parallel Lanczos method for symmetric generalized eigenvalue problems*, Tech. Rep. 41284, Lawrence Berkeley National Laboratory, 1997.

[14] ——, *Thick-restart Lanczos method for symmetric eigenvalue problems*, Tech. Rep. 41412, Lawrence Berkeley National Laboratory, 1998.