LBNL-41284

A Parallel Lanczos Method for Symmetric Generalized Eigenvalue Problems[†]

Kesheng Wu[‡] and Horst Simon[‡]

December, 1997

Abstract

Lanczos algorithm is a very effective method for finding extreme eigenvalues of symmetric matrices. It requires less arithmetic operations than similar algorithms, such as, the Arnoldi method. In this paper, we present our parallel version of the Lanczos method for symmetric generalized eigenvalue problem, PLANSO. PLANSO is based on a sequential package called LANSO which implements the Lanczos algorithm with partial re-orthogonalization. It is portable to all parallel machines that support MPI and easy to interface with most parallel computing packages. Through numerical experiments, we demonstrate that it achieves similar parallel efficiency as PARPACK, but uses considerably less time.

The Lanczos algorithm is one of the most commonly used methods for finding extreme eigenvalues of large sparse symmetric matrices [2, 3, 13, 15]. A number of sequential implementations of this algorithm are freely available from various sources, for example, NETLIB (`http://www.netlib.org/`) and ACM TOMS (`http://www.acm.org/toms/`). These programs have been successfully used in many scientific and engineering applications. However, a robust parallel Lanczos code is still missing. At this time, 1997, the only widely available parallel package for large eigenvalue problems is PARPACK[1], which implements the implicitly restarted Arnoldi algorithm. The Arnoldi algorithm and the Lanczos algorithm can be applied to many forms of eigenvalue problems. Applied on symmetric eigenvalue problems, these two algorithms are mathematically equivalent. However, the Lanczos algorithm explicitly takes advantage of the symmetry of the matrix, and uses significantly less arithmetic operations than the Arnoldi algorithm. Thus, it is worthwhile to implement the Lanczos algorithm for symmetric eigenvalue problems.

[†]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.

[‡]Lawrence Berkeley National Laboratory, NERSC, Berkeley, CA 94720. Email addresses: {`kewu,simon`}-`@nersc.gov`.

[1]PARPACK is a parallel version of ARPACK which implements a implicit restarted Arnoldi algorithm. The source code is available at `http://www.caam.rice.edu/~kristyn/parpack_home.html`. Its sequential counterpart is documented in [18].

Since there are a number of sequential implementations of the Lanczos algorithm, we decided to build our parallel Lanczos code based on an existing package. The major differences between different implementations of the symmetric Lanczos algorithm include: whether and how to perform re-orthogonalization, whether to solve the generalized eigenvalue problem, where to store the Lanczos vectors, what block size to use, and so on. The choice on re-orthogonalization can have major impact on most aspects of the program. If no re-orthogonalization is performed, the Lanczos vectors may lose orthogonality after a number of steps. In this case, the computed eigenvalues may have extra multiplicities. These extra eigenvalues are known as spurious eigenvalues. Identifying and eliminating spurious eigenvalues may be a significant amount of work. The time spent in computing the spurious eigenvalues could potentially be used to compute other eigenvalues. The main advantage of not performing re-orthogonalization is that it can be implemented with a very small amount of computer memory, since only two previous Lanczos vectors need to be saved [3, 4, 19, 20]. However, on many parallel computers, memory is not a limiting factor for the majority of applications. For these applications, it is appropriate to save all Lanczos vectors, perform re-orthogonalization and eliminate the spurious eigenvalues. Among the various re-orthogonalization techniques [3, 12, 13, 17], it was shown that the ω-recurrence can predict the loss of orthogonality better than others, such as selective re-orthogonalization [17]. Given the same tolerance on orthogonality, partial re-orthogonalization scheme, which uses ω-recurrence, performs less re-orthogonalization. In [17], it was also shown that the Lanczos algorithm can produce correct eigenvalues with the loss of orthogonality as large as $\sqrt{\epsilon}$, where ϵ is the unit round-off error. The sequential Lanczos package we have chosen to use, LANSO, implements the ω-recurrence and performs re-orthogonalization only when loss of orthogonality is larger than $\sqrt{\epsilon}$. This scheme reduces the re-orthogonalization to the minimal amount without sacrificing the reliability of the solutions.

The LANSO package was designed to solve symmetric generalized eigenvalue problems. It is more flexible than a package that was designed just for symmetric simple eigenvalue problems. It also lets users choose their own schemes of storing the Lanczos vectors. These characteristics make it a very good starting point to build a parallel Lanczos algorithm. The bulk of this paper is devoted to evaluating the performance of our parallel Lanczos code and comparing it with existing parallel eigenvalue code. We are also interested in defining possible enhancements to the parallel Lanczos method by analyzing the overall performance relative to its major components. For example, LANSO only expands the basis one vector at a time, while a block version could be more efficient on parallel machines. The third goal of this paper is to evaluate the performance of the Lanczos code on different parallel platforms. Two different platforms, a tight-coupled MPP and cluster of SMPs, are used. MPP systems are currently the most effective high performance computing systems. Clusters of SMPs are low cost alternatives to the MPP systems. We want to ensure that our program is efficient on both platforms.

Given a sequential version of the Lanczos code, there are many ways to develop a parallel code from it. The approach taken here turns the sequential code into a Single-Program-Multiple-Data (SPMD) program. The converted program can easily interface with most existing parallel sparse matrix packages such as AZTEC [10], BLOCKSOLVE [11], PETSc

[1], P_SPARSLIB[2], and so on. The details on porting LANSO are discussed in section 1 of this report. Section 2 describes how to use the parallel LANSO program, and the performance of a few examples are shown in section 3 and 4. Section 3 shows the results from a massively parallel machine (MPP) and section 4 shows the results from a cluster of symmetric multiprocessors (SMP). A short summary is provided following the testing results. The user interface of PLANSO is documented in the appendix of this report.

ALGORITHM **1** *Symmetric Lanczos iterations starting with a user-supplied starting vector* r_0.

1. *Initialization.* $\beta_0 = \|r_0\|$, $q_0 = 0$.
2. *Iterate. For* $i = 1, 2, \ldots,$
   (a) $q_i = r_{i-1}/\beta_{i-1}$,
   (b) $p = Aq_i$,
   (c) $\alpha_i = q_i^T p$,
   (d) $r_i = p - \alpha_i q_i - \beta_{i-1} q_{i-1}$,
   (e) $\beta_i = \|r_i\|$.

1 Parallelizing LANSO

The Lanczos algorithm for symmetric matrices, see algorithm 1, can be found in many standard textbooks, e.g., [7, section 9.1.2], [13], and [16, section 6.6]. In this algorithm, the main operations are matrix-vector multiplication, SAXPY, dot-product, and computing the 2-norm of r. The above algorithm builds an orthogonal basis $Q_i = [q_1, \ldots, q_i]$ and computes a tridiagonal matrix T_i as follows,

$$T_i \equiv Q_i^T A Q_i = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{i-2} & \alpha_{i-1} & \beta_{i-1} \\ & & & \beta_{i-1} & \alpha_i \end{pmatrix}.$$

After any i steps of the Lanczos algorithm, the Rayleigh-Ritz projection can be used to extract an approximate solution to the original eigenvalue problem. Let (λ, y) be an eigenpair of T_i, i.e., $T_i y = \lambda y$, then λ is an approximate eigenvalue or a Ritz value of A, the corresponding approximate eigenvector or Ritz vector is $x = Q_i y$. In the Lanczos eigenvalue method, the accuracy of this approximate solution is known without explicitly computing the eigenvector x. Thus, we only need to compute x after we have found all desired eigenpairs. In practice, the Lanczos vectors may lose orthogonality when the above algorithm in carried out in floating-point arithmetic. Occasionally, the Gram-Schmidt procedure is invoked to

[2]The latest P_SPARSLIB source code is available at `http://www.cs.umn.edu/~saad/`.

explicitly orthogonalize a residual vector against all previous Lanczos vectors. The Gram-Schmidt procedure can be broken up into a series of dot-product and SAXPY operations. However, we will not break it down because it can be implemented more efficiently than calling dot-product and SAXPY functions. All in all, the major computation steps in a Lanczos algorithm are: matrix-vector multiplication, SAXPY, dot-product, norm computation, re-orthogonalization, solving the small eigenvalue problem, and computing the eigenvectors. In most cases, the floating-point operations involved in solving the small eigenvalue problem are negligible compared to the rest of the algorithm. We replicate the operation of solving this small eigenvalue problem on each processor to simplify the task of parallelizing the Lanczos algorithm.

ALGORITHM **2** *Lanczos iterations for generalized eigenvalue problem $Kx = \lambda Mx$.*

1. *Initialization.* $\beta_0 = \sqrt{r_0^T M r_0}$, $q_0 = 0$.
2. *Iterate. For* $i = 1, 2, \ldots$,
   - (a) $q_i = r_{i-1}/\beta_{i-1}$,
   - (b) $p = M^{-1}Kq_i$,
   - (c) $\alpha_i = q_i^T Mp$,
   - (d) $r_i = p - \alpha_i q_i - \beta_{i-1}q_{i-1}$,
   - (e) $\beta_i = \sqrt{r_i^T M r_i}$.

The eigenvalue package LANSO is maintained by professor Beresford Parlett of the University of California at Berkeley. It implements the Lanczos algorithm with partial re-orthogonalization for symmetric generalized eigenvalue problem, $Kx = \lambda Mx$, see Algorithm 2 [13]. The LANSO code does not directly access the sparse matrices involved in the eigenvalue problem, instead, it requires the user to provide routines with prescribed interface to perform the matrix-vector multiplications. This greatly simplifies the work needed to parallelize LANSO, since all operations to be parallelized are dense linear algebra operations. The communication operations required by these dense linear algebra functions are limited. This allows us to use any of the common communication libraries. In fact, all operations that require communication can be constructed as a series of inner-product options which can be implemented with only one global sum operation that adds up one number from each PE and returns the result to all. Since many of the sparse linear algebra packages use MPI to exchange data, we have chosen to follow the precedence and use MPI as well.

The dense linear algebra operations involved in the Lanczos iterations, dot-product, SAXPY and orthogonalization operations, can be effectively parallelized if the Lanczos vectors are uniformly and conformally mapped onto all PEs. In other words, if the matrix size is $n \times n$ and p PEs are used, the same n/p elements of each Lanczos vector will reside on one PE. In this case, the SAXPY operation is perfectly parallel. So is the operation of computing Ritz vectors after the small eigenvalue problem is solved. As long as the vectors are conformally mapped and no element resides on more than one processor, a dot-product can be computed with a local dot-product operation followed a global sum operation. When there are an equal number of elements on each processor, these parallel operations have perfect load-balance.

Since the LANSO package requires the user to provide matrix-vector multiplication routines, the data mapping should allow the user to easily interface with other packages to use their matrix-vector multiplication routines or linear system solution routines. Many of the sparse matrix packages for parallel machines have matrix-vector multiplication and linear system solution routines which produce output vectors that are conformal to the input vectors, for example, AZTEC, BLOCKSOLVE, P_SPARSLIB, PETSc. Thus, requiring the Lanczos vectors to be mapped conformally makes the parallel Lanczos routine easy to interface the parallel LANSO code with these packages.

Using MPI, the parallel dot-product routine can be constructed by computing the partial sum on each PE and adding up the partial sums using MPI_ALLREDUCE function. When computing norm of a vector, one can scale the vector first so that its largest absolute value is one. This increases the reliability of the norm computing routine. On distributed environments, this means that an extra global reduction is required to find the largest absolute value of the array and at least twice the floating-point multiplications are required compared to the case without scaling. In the regular Lanczos algorithm, we need to compute the norm of the following vectors: the initial guess, a random vector, or the result of matrix-vector multiplication. Random vectors generated in LANSO have values in between -1 and 1. The norms of vectors generated in the matrix-vector multiplications are limited by the norm of the matrix, since the input vectors to matrix-vector multiplications are always normalized. The only possible source of badly scaled vector is the user-supplied initial guess. Since an eigenvector should be unitary, we expect the user to provide an initial guess that is well scaled. If the user-supplied initial guess is badly scaled, i.e., the sum of the square of the elements is too large for the floating-point representation, a random vector is used as the initial guess. Thus, if the norm of the operator (A, $M^{-1}K$, or $(K-\delta M)^{-1}M$) is not very large, say less than 10^{100} when the Lanczos vectors are stored as 64-bit IEEE floating-point numbers, there is no benefit in scaling the vector when computing its norm. There is a slight chance that a shift-and-invert operator may have an exceedingly large norm. However, this special case probably deserves separate treatment. Therefore we have decided not to perform scaling when computing the norm of a vector.

When the Lanczos routine has detected an invariant subspace, and still more eigenvalues are wanted, it will continue with a random vector. We do not anticipate that the random vectors are needed frequently, nor do we anticipate that the quality of the random number generation will dominate the overall solution time. Thus we are satisfied with the simple random number generator used in LANSO. The only change made is to have each PE generate a different seed before the random number generator is used.

In our current implementation, the small eigenvalue problem is replicated on each PE. The size of the small eigenvalue problem is equal to the number of Lanczos steps taken. If the number of Lanczos steps required to solve the given eigenvalue problem is small, than replicating it is an acceptable option. If the number of steps is large, it might be more efficient to let the processors solve this problem in collaboration. We believe the number of Lanczos steps will be relatively small in most cases, so we have chosen not to parallelize the operation at this time.

For most of the dense linear algebra operations, LANSO only needs to know the local problem size, i.e., the number of elements of a Lanczos vector on the current processor. There are two operations where the global problem size is needed. One, when computing the estimate of loss of orthogonality, we need to use $\epsilon\sqrt{n}$ where n must be the global problem size. The other use of global problem size is to ensure the maximum number of Lanczos steps do not exceed the global problem size.

When converting LANSO to a parallel program, we also need to change how the program prints diagnostic messages. Most of the messages would be the same if printed from different processors. In this case, only processor 0 will print them. If different PE will print different messages, they will be printed with the PE number associated with the message.

Overall, a fairly small number of changes are required to parallelize LANSO. Our experience is representative of many parallelization efforts involving Krylov-based projection methods for linear systems and eigenvalue problems [10, 11, 14].

2 Using the parallel LANSO

The details of the user interface are described in the appendix of this report. Another good source of information is the documentation that comes with the source code. Here we provide a brief introduction to basic elements needed in order to use the parallel LANSO code. To use the program, the user should provide the following functions.

1. The operator of the eigenvalue problem, `OP` and `OPM`. To solve a standard symmetric eigenvalue problem, $Ax = \lambda x$, the function `OP` performs the matrix-vector multiplication with A, the function `OPM` should simply copy its input vector to the output vector. To solve a symmetric generalized eigenvalue problem, $Kx = \lambda Mx$, the function `OPM` should be a matrix-vector multiplication routine that multiplies M, the function `OP` should perform the matrix-vector multiplication with the matrix either in the standard form, $M^{-1}K$, or the shift-and-invert form, $(K - \sigma M)^{-1}M$. As mentioned before, the matrix-vector multiplication routines should map the input and output vectors conformally, i.e., if the ith (global index) element of the input vector is on processor j, the ith element of the output vector should be also on processor j. The local index of ith element on processor j should be the same for both input and output vectors. The conformal mapping is used by most existing parallel sparse matrix computation packages, e.g., AZTEC, BLOCKSOLVE, P_SPARSLIB, PETSc.

   Allowing a user-supplied matrix-vector multiplication routine to be used makes it possible for a user to select different schemes to store the matrix and to use their favorite linear system solver in the generalized eigenvalue case. For most applications, there is usually a more specific sparse matrix storage format that is more convenient for that particular application. On parallel machines, there is usually a custom parallel matrix-vector multiplication that is efficient for the particular application as well. No operations inside PLANSO depend on the global index of a vector element, thus as long as the matrix vector multiplications produce output that are conformal to the input vector, PLANSO can function correctly and find the appropriate eigenvalues and their corresponding eigenvectors.

2. A function to access Lanczos vectors, `STORE`. It is used by the program to store the Lanczos vectors no longer needed for regular steps of Lanczos algorithm, see Algorithm 2. The Lanczos vectors may be needed again to compute the Ritz vectors or to perform re-orthogonalization. The users have the flexibility of either storing the Lanczos vectors in-core or out-of-core by simply modifying this single function.

3. A parallel orthogonalization routine `PPURGE`. The user may decide whether or not to replace this routine. It performs re-orthogonalization when the loss of orthogonality has become severe. The user may replace it to enhance the performance of the default re-orthogonalization procedure which performs the modified Gram-Schmidt procedure and retrieves the old Lanczos vectors one after the other. A series of global sum operations are needed to compute the dot-products between the old Lanczos vectors and the new one. If the Lanczos vectors are stored in-core, the classic Gram-Schmidt procedure is usually more efficient on parallel machines. A version of `PPURGE` that implements the classic Gram-Schmidt procedure is presented with every example that stores the Lanczos vectors in-core. Should the user decide to have a customized `PPURGE` function, the examples can be used as starting points.

The size of workspace required by the parallel LANSO on each PE is completely defined by the local problem size and the maximum number of Lanczos steps allowed. If the local problem size is `Nloc` and the maximum number of Lanczos steps desired is `LANMAX`, the workspace size needs to be at least $5 * \mathtt{Nloc} + 4 * \mathtt{LANMAX} + \max(\mathtt{Nloc}, \mathtt{LANMAX} + 1) + 1$. If eigenvectors are needed, additional $\mathtt{LANMAX} \times \mathtt{LANMAX}$ elements are needed.

With PLANSO, the user may choose to compute a few extreme eigenvalues from both ends of the spectrum or one end of the spectrum. The eigenvalue routine will stop when the desired number of eigenvalues are found or the maximum allowed number of Lanczos steps is reached. After the program returns, the approximate eigenvalues and their corresponding error estimates are returned to the caller. The approximate eigenvectors may be written to a file or the user may choose to compute the approximate eigenvectors outside of the LANSO program using the Lanczos vectors.

3 Performance on MPP

This section demonstrates the effectiveness of the parallel LANSO code by examples. In order for a parallel program to be usable in a massively parallel environment like Cray T3E, it is important that the algorithm is scalable. There are two generally accepted approaches to measure scalability. One fixes the global problem size and the other fixes the local problem size on each PE. In this section we will show both types of test cases.

At the time of this test, the Cray T3E used was using operating system UNICOS/mk version 1.5. The Fortran compiler was CF90 version 3.0. The C compiler was Cray's C compiler Version 6.0. The Fortran programs are compiled with aggressive optimization flag `-Oaggress,scalar3,unroll2`. The C functions are compiled with standard optimization flag `-O`. The peak floating-point operation rate was 900 MFLOPS, and the peak data rate was 7200MB/sec for loading data from cache and 3600MB/sec for storing to cache.

For fixed global problem size test cases, we have chosen to find the eigenvalues of two large sparse matrices in Harwell-Boeing format [6]. The first matrix is a structural model of a space shuttle rocket booster from NASA Langley called NASASRB. It has 54,870 rows and 2,677,324 nonzero elements. The matrix is stored in Harwell-Boeing format which stores only half of the nonzero elements. The file size is about 36MB. The second test matrix is CPC stiffness matrix (CPC.K). It has 637,244 rows and 30,817,002 nonzero elements. The file size is about 600MB. The matrix CPC.K is reordered using reverse Cuthill-McKee ordering before it was used in our tests. The reverse Cuthill-Mckee ordering is one of the common ordering schemes used to reduce the bandwidth of a sparse matrix [5]. It is a generic algorithm to reduce the data communication required during sparse matrix-vector multiplication. We did not use better reordering schemes because the reordered matrices show reasonable performance in the tests. In addition, we only need the performance of the matrix-vector multiplication as a reference to measure the overall performance of PLANSO.

3.1 Fixed size tests

The first test performed is to see scalability of the major components of the LANSO program. Figure 1 shows the floating-point performance per PE of matrix-vector multiplication, dot-product, SAXPY and orthogonalization. They are not directly measured from the LANSO routine, but the test uses the same matrices and same size vectors as in LANSO. The performance of the orthogonalization is measured from orthogonalizing one vector against 32 orthonormal vectors. Given the maximum data rate, the peak rate for SAXPY on 8-byte floating-point arrays is 450MFLOPS. We are able to achieve almost 85% of this peak rate when the arrays can fit into cache. The dot-product operation and norm computation have a peak rate of 900MFLOPS, but the best measured rate is about 11% of this peak rate. There are two main reasons for this. One is that the global reduction takes significant amount of time in dot-product operation. The other reason is that most of the time the data is loaded from main memory which has a much slower rate than 3600MB/sec. The parallel efficiency for dot-product operation is the lowest among all major components of PLANSO. For the NASASRB test problem, the floating-point operation rate of dot-product operation and 2-norm computation on 32-PE partition is only about 40% of the rate on 1 PE. When more PE is used, the performance degradation of dot-product is even more pronounced. In the CPC.K example, the performance of dot-product on 512 PE is only 25% of that of 16-PE case.

The efficiencies shown in Figure 1 are measured as follows. In the NASASRB case, the efficiency of using p-PE is the ratio of the performance on p-PE versus that of the 1-PE case. For example, the speed of the matrix-vector operation is 51.6MFLOPS per PE on 16 PEs, and 64.1MFLOPS on 1 PE. Thus the efficiency of the matrix-vector operation on 16-PE is 80%. The efficiency of the CPC.K example is measured against the performance of 16-PE case. The speed of the matrix-vector multiplication is 35.8MFLOPS per PE on 512 PEs and 57.5 on 16-PE. Thus, the efficiency of multiplying CPC.K on 512 PE is 62%.

The orthogonalization operations run at about the same rate on different size partitions. The performance of the orthogonalization procedure could be different if the number of orthonormal vectors is different. Typically, the performance increases as the number of

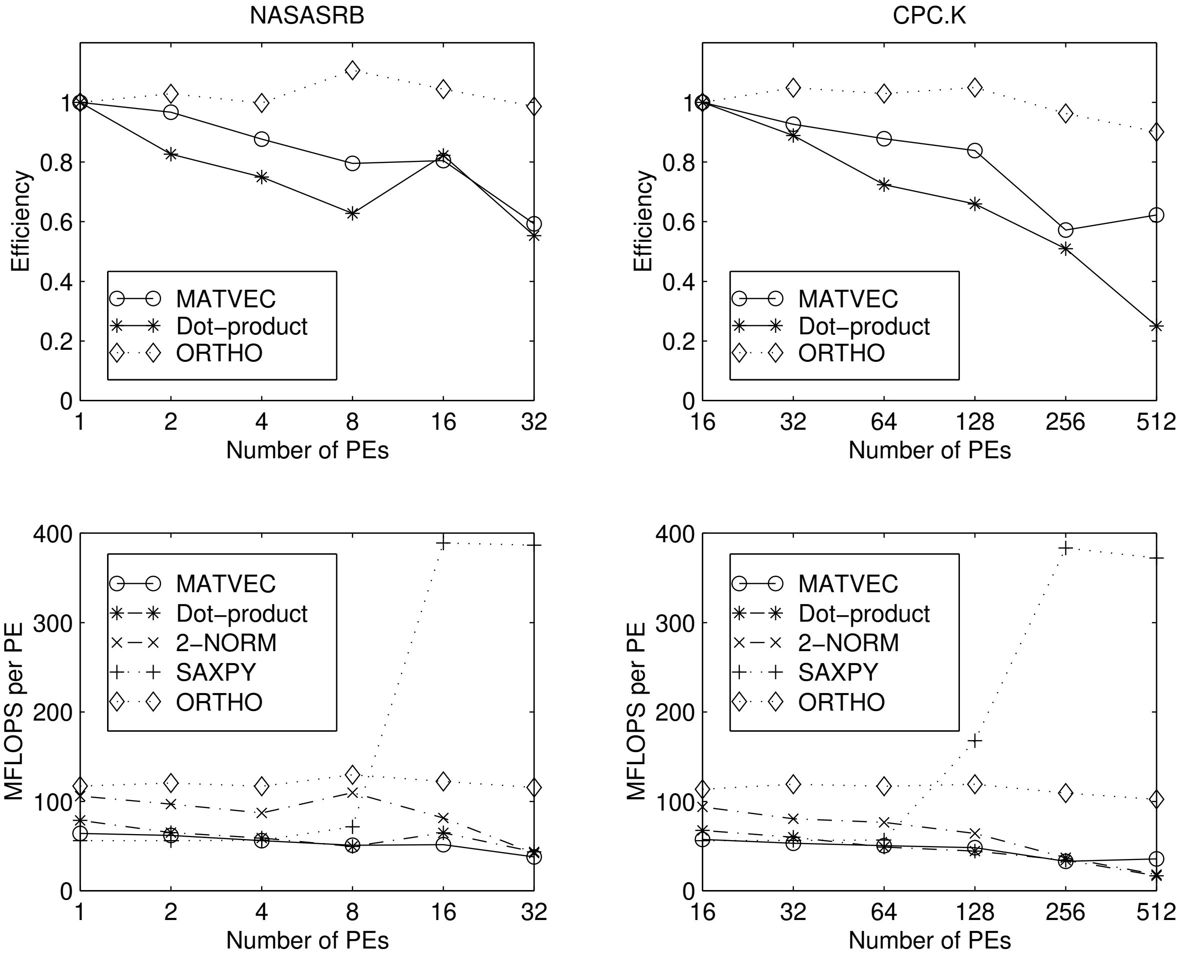

Figure 1: Scalability of core operations of PLANSO on T3E with fixed global problem size.

orthonormal vectors increases. In the case tested, the number of orthonormal vectors is 32. In many cases, the first re-orthogonalization occurs after forty or fifty steps of Lanczos iterations. The performance of the orthogonalization operation measured could be regarded as typical. In both NASASRB and CPC.K cases the per PE performance of orthogonalization operation is fairly constant as the number of PE changes. Its floating-point operation rate is much higher than most other operations.

The bulk of the floating-point operations of algorithm 1 is in applying the operator (A, $M^{-1}K$, or $(K - \delta M)^{-1}M$). LANSO routine requires the user to supply this operation. The matrix-vector multiplication routine we use in this test is from the P_SPARSLIB. It does a thorough job of analyzing the communication pattern and only passes the minimum amount of data among the processors. It also takes advantage of MPI persistent communication request and overlaps the communication and computation inside the matrix-vector multiplication routine. The intention of this test is to provide a reference point to measure how well the parallel LANSO works. The exact performance of matrix-vector multiplications, which could be highly problem-dependent, is not the main interest of this paper.

On 1-PE, the sparse matrix-vector multiplication with NASASRB runs at about 64 MFLOPS. The efficiency of matrix-vector multiplication with NASASRB is above 80% when less than 16 PEs are used. When the matrix is mapped onto 32 PEs, the distributed matrix-vector multiplication routine runs at about 38MFLOPS per PE which is nearly 60% of the single PE performance. This significant performance decrease from 16-PE case to 32-PE case can be partly explained by the difference in communication versus computation ratio. In both 16-PE and 32-PE cases, the number of bytes exchanged during matrix-vector multiplication is about the same, however the amount of floating-point operation is essentially reduced by half on each PE. More importantly, the part of the arithmetic operation that can be overlapped with communication reduced very significantly from 16-PE case to 32-PE case. The communication latency can no longer be effectively hidden in 32-PE case.

The efficiency of the matrix-vector multiplication with CPC.K is better than 80% when partition size is less than 128. Going from a 128-PE partition to a 256-PE partition, the number of neighbors a PE sends messages to changes from 4 to 8. This increases the likelihood of congestion during communication which slows down the communication. In addition, there are a significant number of PEs where there is very little overlapping between communication and computation because of the decrease in the amount of arithmetic operations on each PE. The efficiency of the matrix-vector multiplication with CPC.K is about 60% on 256 PE and 512 PE of T3E. The efficiency on 512-PE partition is slightly better than the 256-PE case because there is more communication load imbalance in the 256-PE case. The aggregate performance of the matrix-vector multiplication on 512 PE is about 18 GigaFLOPS. This is a decent rate since the problem size on each PE is very small, about 1200, and the amount of data to be communicated is large, a total of 2500 8-byte words to a maximum of 12 neighboring processors. Computing dot-product with 1200 numbers should only take 1 or 2 microseconds on a PE of T3E, while the communication latency of MPI is on about 10 microseconds. The communication time clearly dominates the arithmetic time in this case.

In one Lanczos iteration, there are two SAXPY operations, one dot-product and one norm computation in addition to the matrix-vector multiplication. The operations that are least scalable are the dot-product and 2-norm computation. As the number of processors

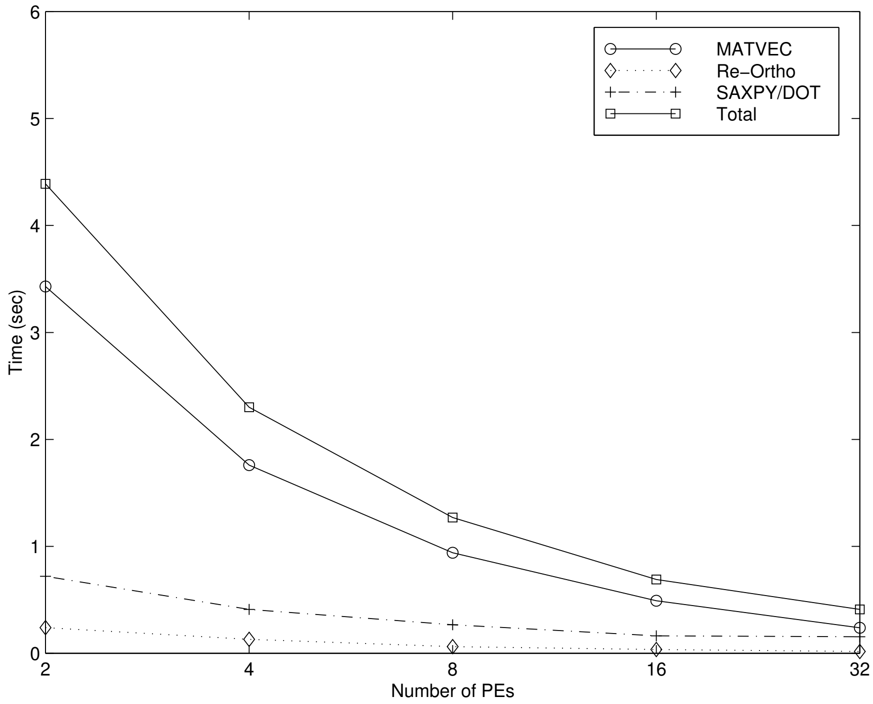

Figure 2: Time to compute the five largest eigenvalues of NASASRB with different number of processors.

increases, the dot-product and 2-norm computation could eventually dominate the overall performance. However, if we maintain reasonable minimum local problem size, the dominant operation should be the matrix-vector multiplication, i.e., step *2(b)* of algorithm 1 and 2.

Figure 2 shows the time used to find the five largest eigenvalues of NASASRB using the parallel LANSO with P_SPARSLIB for matrix-vector multiplication. In this test, the LANSO routine with the interface for computing one side of extreme eigenvalues is used. The convergence tolerance was set to $\kappa = 10^{-8}$, i.e., an eigenvalue is considered converged if the relative error in the computed eigenvalue is less than 10^{-8}. It took 75 steps to find the five largest eigenvalues. The total elapsed time and the time used by matrix-vector multiplication and re-orthogonalization are shown. The matrix-vector multiplication time obviously is the most significant part of the total time. The overall efficiency is better than the efficiency of matrix-vector multiplication routine. In 32-PE case, the overall efficiency is about 72% for the NASASRB test problem, which is better than the efficiency of matrix-vector multiplication (59%) and dot-product (55%), see Figure 3.

To put the above performance numbers in prospective, we solved the same test problem with PARPACK. The PARPACK code used in this test is version 2.1 dated 3/19/97. It implements the implicitly restarted Arnoldi method for eigenvalue problems of various kinds. The one we use is for simple symmetric eigenvalue problems. If we don't restart the Arnoldi method, it is mathematically equivalent to the Lanczos algorithm. The main difference between the Lanczos algorithm and the Arnoldi algorithm is in the orthogonalization of Aq_i against existing basis vectors. In theory, the Gram-Schmidt procedure is

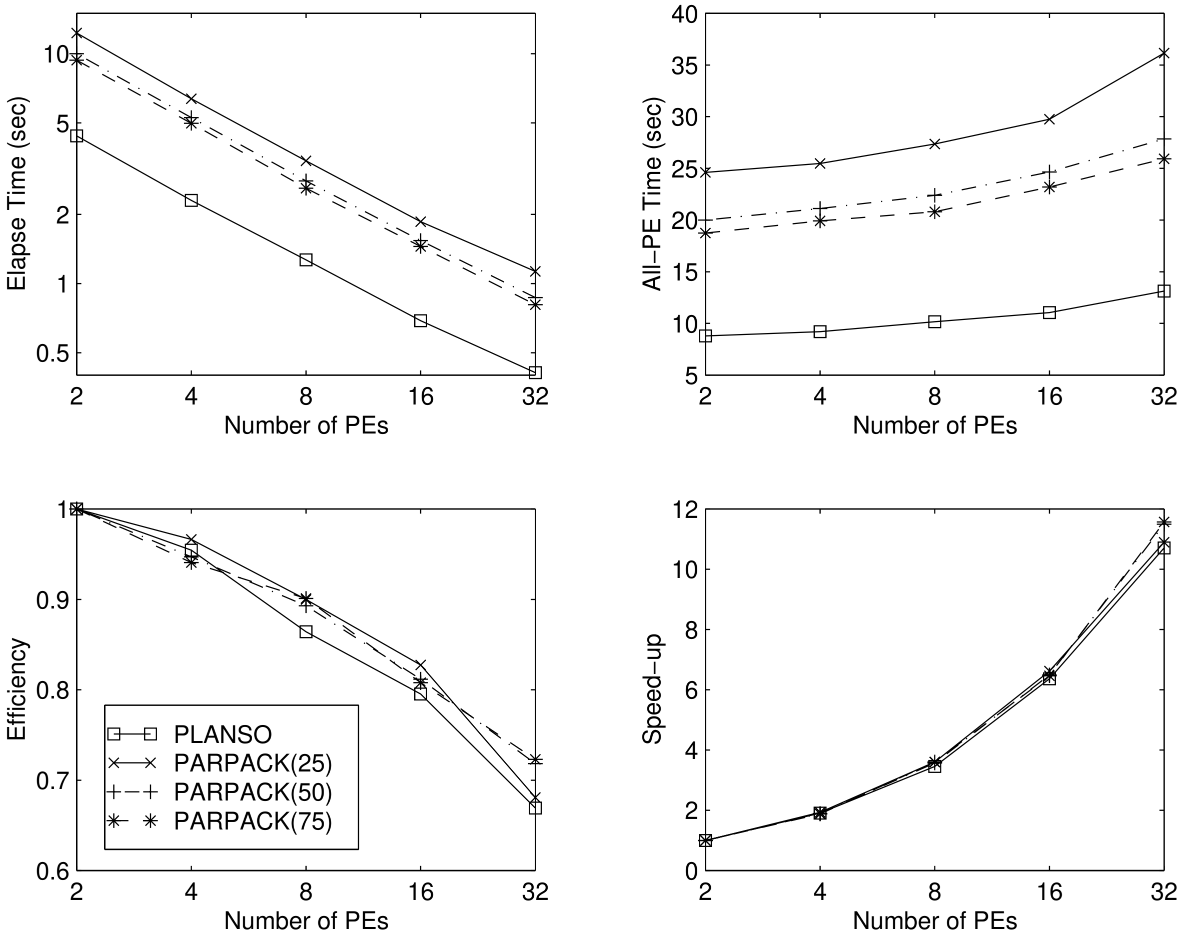

Figure 3: Comparison between PLANSO and PARPACK on NASASRB test problem.

used to accomplish orthogonalization, i.e., $Aq_i - Q_i Q_i^T A q_i$ where $Q_i = [q_1, q_2, \ldots, q_i]$. The Arnoldi method explicitly carries out this computation. The Lanczos algorithm as shown in algorithm 1 takes advantage of the fact that most of the dot-products between q_j and Aq_i are zero because the matrix A is symmetric. In fact, only two of the dot-products are not zero. Thus the Lanczos algorithm uses less arithmetic operations than the Arnoldi method. In computer arithmetic, both PLANSO and PARPACK may repeat the orthogonalization procedure in order to maintain good orthogonality. When performing re-orthogonalization, both PARPACK and PLANSO explicitly carry out the Gram-Schmidt procedure since there is no special property that could be used to reduce the amount of arithmetic operations. At step i, the re-orthogonalization operation of PLANSO is as expensive as the same operation in PARPACK. An important characteristic of PARPACK is that it limits the maximum amount of memory required by restarting the Arnoldi method. In the above test for parallel LANSO method, we allocated all unused memory on each processor to store the Lanczos vectors. For PARPACK we have chosen a few different maximum basis sizes. In Figure 3, PARPACK(25), PARPACK(50), and PARPACK(75) are used. Their maximum basis sizes are 25, 50, and 75 respectively. The same convergence tolerance is used in PARPACK as in PLANSO. The three PARPACK schemes used 140, 95 and 75 matrix-vector multiplications respectively to reach convergence. The various PARPACK schemes and PLANSO converge to the same eigenvalues with roughly the same accuracy. The maximum basis size of 75 is used because it is the basis size used by PLANSO to reach convergence. Since there is no restart in PARPACK(75), it is theoretically equivalent to PLANSO. The main difference between the two algorithms is the orthogonalization procedure.

In Figure 3, the "Elapse Time" refers to the average wall-clock time required by each PE to find the five largest eigenvalues and their corresponding eigenvectors. The "All-PE Time" is the sum of the elapse time on all PEs. We have chosen to show this total time because it is how the computer center accounts for the usage of machine. The speed-up and efficiency are computed with 2-PE case as the baseline. Figure 3 shows that the efficiencies of the four methods are about the same but the parallel LANSO program uses much less time. They have similar parallel efficiency because they both depend on the same basic computational blocks for their operation. The time used by PARPACK is considerably more than that of PLANSO. Part of the difference is because the restarted algorithms use more matrix-vector multiplications. PLANSO still uses less time than PARPACK(75) because PLANSO uses much less time in orthogonalization. Given that the re-orthogonalization time shown in Figure 2 is only for 2 re-orthogonalizations, and PARPACK almost always performs Gram-Schmidt procedure twice for each matrix-vector multiplication, the time differences between PLANSO and PARPACK can be all attributed to the difference in orthogonalization. For example, in 2-PE case, 0.24 seconds is needed by PLANSO to perform 2 re-orthogonalization, PARPACK(75) performs 74 re-orthogonalization which could take up to 9 seconds if the time per re-orthogonalization for PARPACK(75) is the same as PLANSO. Of course, on average each re-orthogonalization in PARPACK(75) orthogonalizes against less vectors than in PLANSO. Therefore the extra time spent in re-orthogonalization is less than 9 seconds. However, it is clear that the extra re-orthogonalization in PARPACK takes a significant amount of time.

Figure 4 shows the time spent by PLANSO to find the five largest eigenvalues and the

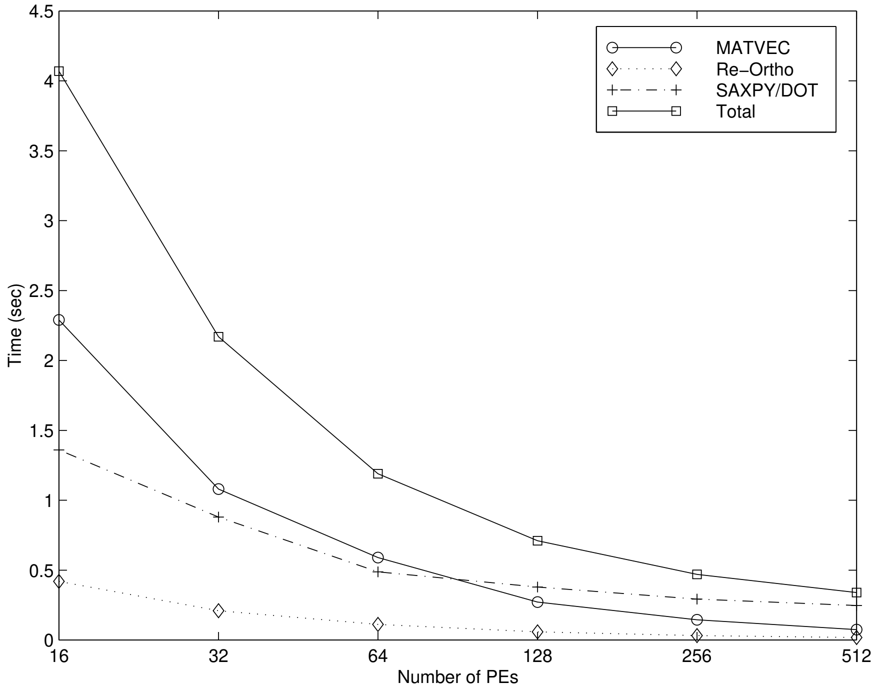

Figure 4: Time to compute the five largest eigenvalues of CPC.K with different number of processors.

corresponding eigenvectors of CPC.K. In this case, only 40 Lanczos iterations are required to find the solution during which six re-orthogonalizations are performed. From Figure 1 we know that the dot-product operation in this case is considerably less efficient than in the NASASRB test case. In Figure 4 we can see that the time spent in dot-product, SAXPY, etc., can be considerably more than the time spent in matrix-vector multiplication as the number of processors increases. In the 512-PE case, out of the 0.34 seconds total time, only 0.075 seconds are spent in matrix-vector multiplication, more than 70% of the total time is used to perform dot-products. Obviously, the efficiency of the overall parallel Lanczos routine is low in this case, see Figure 5, because the dominate operation has low parallel efficiency. The efficiency of PLANSO on 512-PE is 37%, which is above the efficiency of the dot-product (25%), but below the efficiency of the matrix-vector multiplication routine (62%).

As in the NASASRB case, we also tested PARPACK on the CPC.K test problem. The timing results are shown in Figure 5. Similar to the NASASRB case, the time taken by PLANSO to find the five largest eigenvalues is less than that of PARPACK. Part of this is because more matrix-vector multiplications are used by the various version of PARPACK, PARPACK(10): 49, PARPACK(20): 43, PARPACK(40): 40. The number of matrix-vector multiplications used by PLANSO is 41. The time difference between PLANSO and PARPACK(40) is attributable to the difference in the orthogonalization process. When more processors are used, the matrix-vector multiplication time becomes a small portion of the total time for both PLANSO and PARPACK. In the PARPACK program, almost all the rest

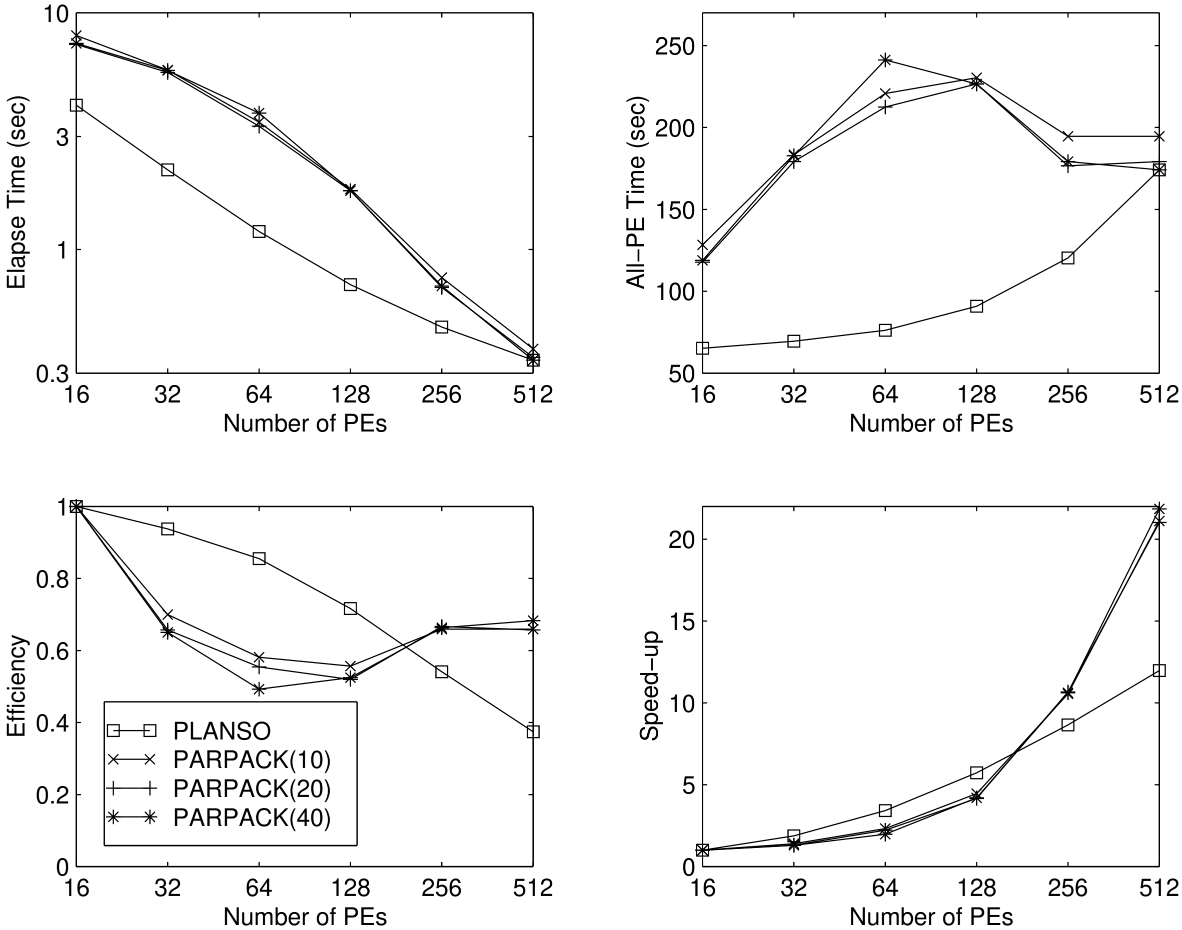

Figure 5: Comparison between PLANSO and PARPACK on CPC.K test problem.

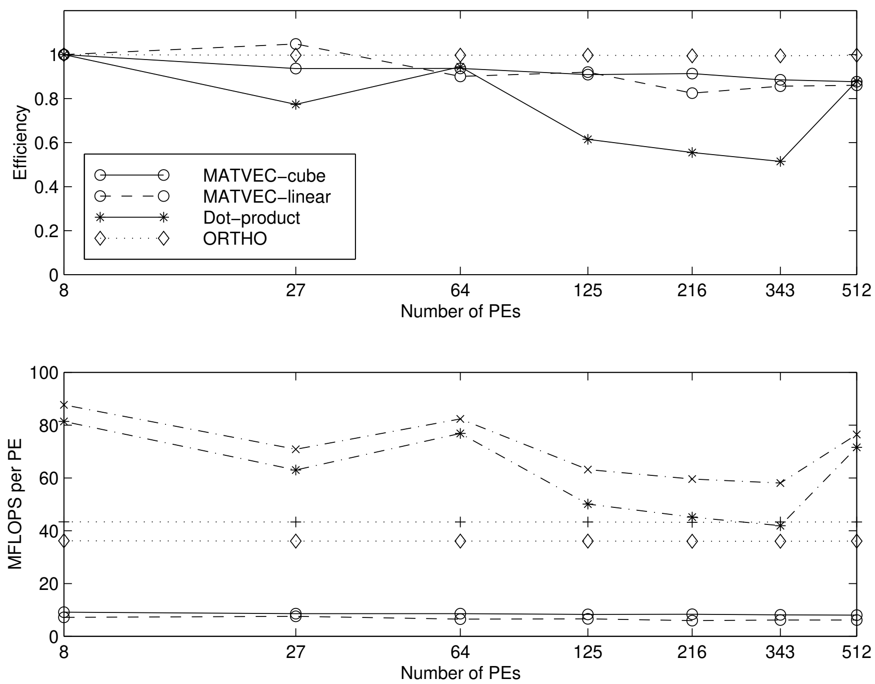

Figure 6: Per PE performances on T3E with fixed local problem size.

of the time is spent in Gram-Schmidt orthogonalization which has good parallel efficiency, see Figure 1. This explains the significant rebound in parallel efficiency of PARPACK as the number of processors becomes large.

Overall, PLANSO can maintain above 80% efficiency on the CPC.K test problem when it is mapped to 64 PEs or less. Similarly, 80% efficiency can be maintained for the NASASRB test problem if it is distributed to 16 or less PEs. Mapping CPC.K onto 64 PEs, the problem size on each PE is about 10,000. Mapping NASASRB onto 16 PEs, the problem size on each PE is about 3,400. These two examples give us two reference points on what we can expect the parallel efficiency to be when mapping a fixed size problem onto different size partitions of T3E. A moderate local problem size must be maintained in order to maintain reasonable communication to computation ratio, therefore to maintain good parallel efficiency.

3.2 Scaled test case

The intuition on parallel efficiency is that ideal efficiency may be achieved if the communication versus computation ratio is maintained constant, or if the problem size on each PE is maintained constant. This is what is shown in Figures 6 and 7. In this case, the parallel LANSO routine is used to find the five largest eigenvalues and corresponding eigenvectors of finite difference discretization of the Laplacian operator on a 3-D domain. The local problem size is maintained to be 32,768. As more processors are used the global problem size increases proportionally. Two different schemes are used to map the global problem onto each PE, one simply divides the matrix as row blocks (linear), the other divides the 3-D domain in to sub-cubes and maps the problem on each sub-cube onto a processor. It is clear that the

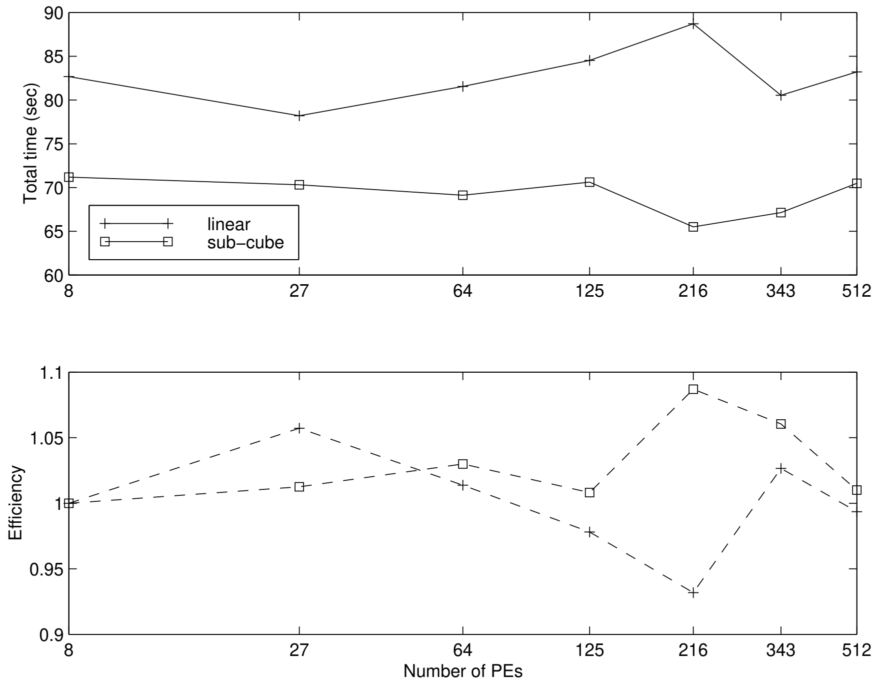

Figure 7: Scaled performance of PLANSO on 3-D Laplacian test problem.

linear mapping requires more data to be exchanged among the processors but each processor needs to communicate with fewer neighboring processors. Because the linear mapping leads to large messages being exchanged during matrix-vector multiplication, the matrix-vector multiplication takes more time than in the sub-cube mapping case. The differences in the local matrix structure also contribute to the performance difference as well. A small part of the performance difference also comes from the fact that different packages are used to perform the matrix-vector multiplications. The linear mapping case is done with AZTEC and the sub-cube mapping case is done with BLOCKSOLVE95. The intention of doing this is to show that the parallel LANSO can easily work with different software packages.

The performance of the major components of the parallel Lanczos routine is shown in Figure 6. Similar to Figure 1, the performance of the following operations are shown: matrix-vector multiplication, dot-product, SAXPY, and orthogonalization. The efficiencies of matrix-vector multiplication, dot-product computation and orthogonalization are shown as well. In the bottom plot of Figure 6, the legends are the same as in the plot above. The three lines that do not appear in the top plot are:

- dash-dot line with cross symbols denotes the performance of computing 2-norm,
- dash-dot line with asterisk symbol denotes the performance of computing dot-product,
- the dotted line with plus symbols denotes the performance of SAXPY.

Most of the lines in Figure 6 are almost straight horizontal lines which indicates that the per PE performance is constant as the number of processors change. The only exception is the

	in-core		out-of-core		
# of PE	CGS (sec)	MGS (sec)	total (sec)	I/O time (sec)	rate (MB/s)
2	4.39	4.75	10.94	5.63	18.1
4	2.30	2.50	6.82	3.29	29.3
8	1.27	1.40	5.39	2.70	35.5
16	0.69	0.77	6.56	3.38	28.5
32	0.43	0.46	7.60	3.80	25.3

Table 1: Performance of an out-of-core versions of parallel LANSO.

performance of dot-product and 2-norm computing routines. If the number of processors is not a power of 2, than their efficiency can reduce to below 80%.

Since the largest eigenvalues of the Laplacian matrix change as matrix sizes change, we have chosen to run PLANSO for a fixed number of iterations. The time shown in Figure 7 is collected from running for 602 steps of Lanczos iterations. This number of steps is determined by allowing the program to take up all the memory of each processing element on the T3E. The 602 Lanczos vectors take up about 160MB of memory which is the amount of memory left after the problem is setup on a PE with 256MB of physical memory. Going from 8 PE to 512 PE, the matrix-vector multiplication slows down about 15% for both mapping schemes. The total time spent by LANSO is almost exactly the same. This is because there is a reduction in time spent in re-orthogonalization as the problem size increases and the number of converged eigenvalues decreases. The amount of re-orthogonalization required in a Lanczos algorithm with partial re-orthogonalization is strongly related to the number of eigenvalues converged [9, 17]. As more eigenvalues converge, more re-orthogonalization is needed. As problem size increases, the eigenvalues of Laplacian operator require more Lanczos steps to compute. Thus, within the first 602 iterations, less eigenvalue will converge. In the 8-PE case, problem size 262,144, 46 eigenvalues converged within 602 iterations. In the 512-PE case, problem size 16,777,216, no eigenvalue converged within 602 iterations.

Overall, in this test case, with 64-fold change in the number of processors, the parallel efficiency of PLANSO is better than 90%.

3.3 Out-of-core case

Table 1 shows the time used by two versions of parallel LANSO to find the five largest eigenvalues of NASASRB, one stores the Lanczos vectors in-core, the other out-of-core. The in-core version also has two different orthogonalization schemes, one uses the classic Gram-Schmidt (CGS) procedure, the other uses the modified Gram-Schmidt (MGS) procedure. The time shown is the average wall-clock time measured on different processors. This test matrix is small enough that the 75 Lanczos vectors can be store in the main memory of two T3E processors. In the out-of-core version, each PE stores its own portion of the Lanczos vectors in an unformatted direct access file using Cray's asynchronous memory cached I/O, `cachea`, library to interact with the disk system. A large buffer is used to enhance the I/O performance. Because the Lanczos vectors are slow to retrieve, we have chosen to use the

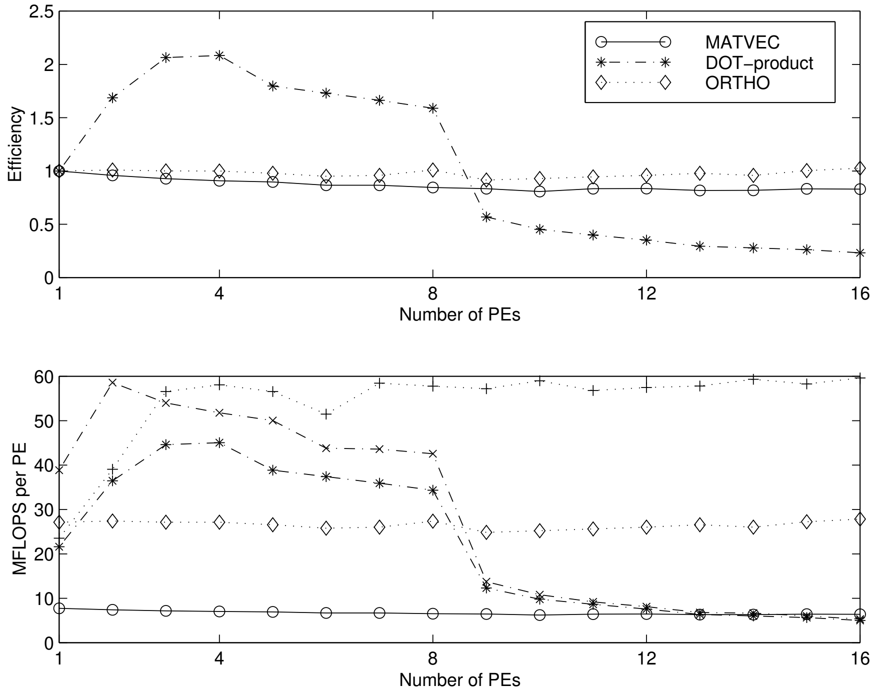

Figure 8: Per PE performance of major components of PLANSO.

modified Gram-Schmidt procedure to perform re-orthogonalization. This also increases the number of communications required during a re-orthogonalization. The in-core version of PLANSO uses less time than the out-of-core version. The in-core version with CGS uses slightly less than the one with MGS orthogonalization procedure. The difference between them could be larger if more re-orthogonalization were performed. From Table 1, we see that the I/O time dominates the total time used by the out-of-core LANSO. The I/O rate is to total number of byte read/write versus the time used. As the number of processors increases, more I/O processors can be utilized to perform independent read/write operations. There are a number of factors that limit the I/O performance. For example, there are only a small number of I/O processors, a disk can only support a fixed number of strides. Typically, the number of simultaneous read/write operations is small compared to the number of computing processors. This is true for many massively parallel machines.

On a massively parallel machine like T3E, out-of-core storage for Lanczos vectors is often not necessary. The simple-minded scheme tested here is mainly to show that it is easy to construct an out-of-core version of PLANSO. Since the performance of the out-of-core Lanczos routine is poor, we should consider limiting the maximum memory usage so that all Lanczos vectors can fit in-core.

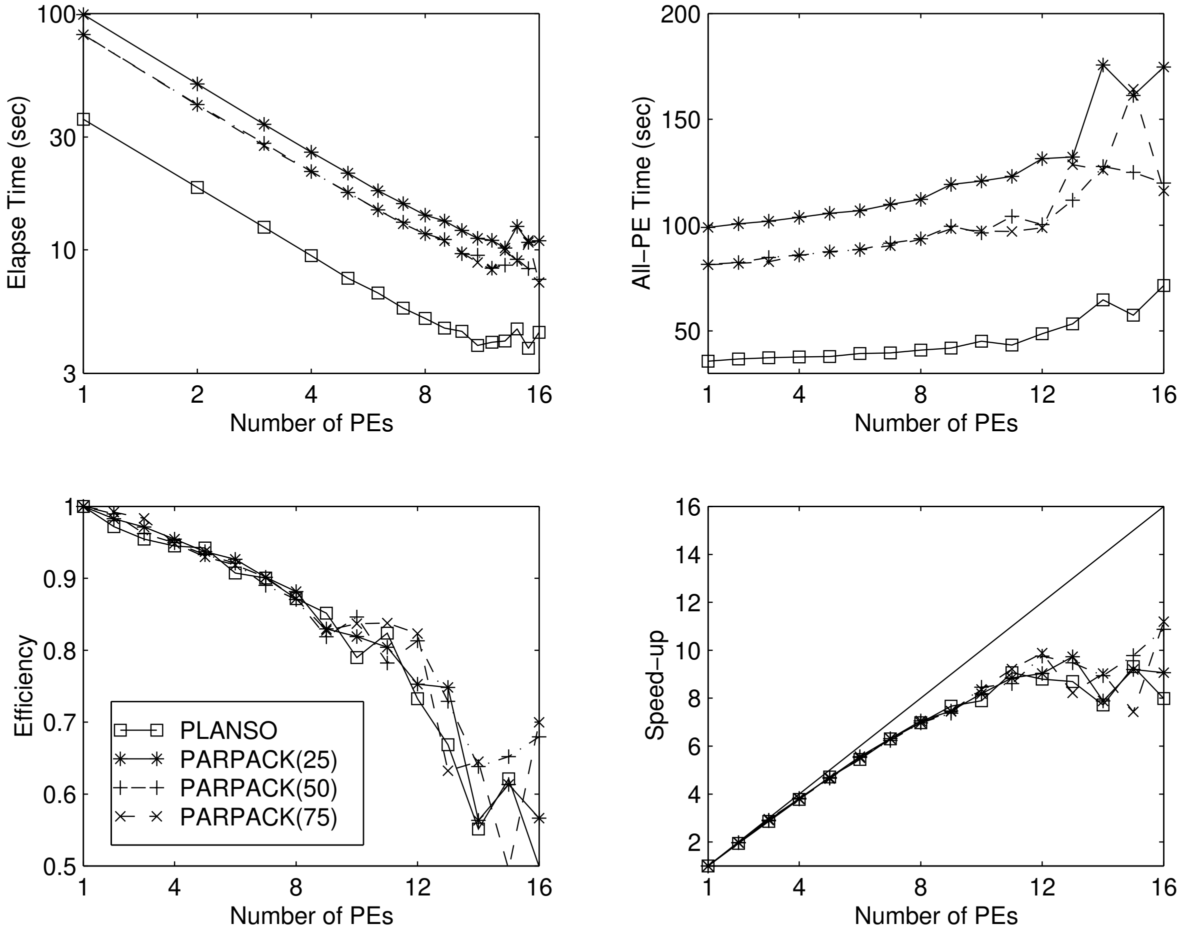

Figure 9: Performance of PLANSO and PARPACK on a cluster of 2 8-PE SPARC Ultra workstations.

4 Performance on cluster of SMP

Clusters of symmetric multiprocessors (SMP) are used for many different scientific and engineering applications because they are more economical than massively parallel machines (MPP). We are interested in knowing the performance characteristics of PLANSO on these types of machines. The tests reported in this section are performed on a cluster of 2 8-PE SPARC Ultra workstations. The SMP workstations are connected by multiple 622 Mbit/s ATM networks using Sun ATM adaptor cards to support parallel programming. At the time of this test, November 1997, the CPUs of the workstations are 167 Megahertz UltraSPARC I processors with 16 KByte on-board cache and 512 KByte external cache. They were running a beta version of the Sun HPC clustering software. The tests run here use each individual CPU as an independent processor. No effort is made to take advantage of the fact that some of them are actually connected on the same bus. We rely on the clustering software to handle the non-uniformity in data access. Data communication among the processors is done using Sun's MPI library. The same Fortran programs used in the T3E test are used in this test. The BLAS routine from Sun's high performance S3L library is used. All user programs are compiled with `-fast` option for the best performance. For more information about the cluster, visit `http://www.nersc.gov/research/COMPS`.

The test matrix is NASASRB. As in the MPP case, we show the per PE performance of the major components of PLANSO, see Figure 8, and the overall performance of PLANSO on the test problem, see Figure 9. The data shown in Figure 8 is not collected from inside the Lanczos routine, but from special testing routines. For dot-product and SAXPY operations, only two vectors are involved. In this test, these vectors can fit into the secondary caches of 2 or more PEs. This causes a noticeable performance difference going from the 1-PE case to the 2-PE case. Otherwise, most performance curves in Figure 8 are fairly flat as the number of processors change. The exceptions are the lines for computing dot-product and 2-norm. When the number of PE is smaller than nine, all PEs involved are on one bus, the data exchange among them is significantly faster than communicating between the workstations. The performance of dot-product operation decreases slightly when the number of PEs involved changes from nine to sixteen. However, the change between the 8-PE case and the 9-PE case is much larger.

Figure 9 shows a number of different measures of the effectiveness of the parallel Lanczos routine, and PARPACK with three different basis sizes. Similar to figure 3, we show the wall-clock time, the sum of time used by all PEs, the efficiency and the speed-up of the parallel programs. Comparing the overall efficiency of PLANSO with the matrix-vector multiplication routine used, when the number of PEs is less than nine, the efficiency of the Lanczos routine is better than that of the matrix-vector multiplications routine. As more PEs are used, the dot-product operation eventually drags the overall performance down to below 80%. On 16 PEs, the speed-up of the Lanczos routine is about 12 (75% efficiency) and the speed-up of the matrix-vector multiplication routine is about 13 (81% efficiency). This again shows that PLANSO can achieve good parallel efficiency when dot-product routine has reasonable efficiency.

Comparing the performance of PLANSO with PARPACK, we see that both can achieve very similar parallel efficiency and speed-up on the test problem. However, because the

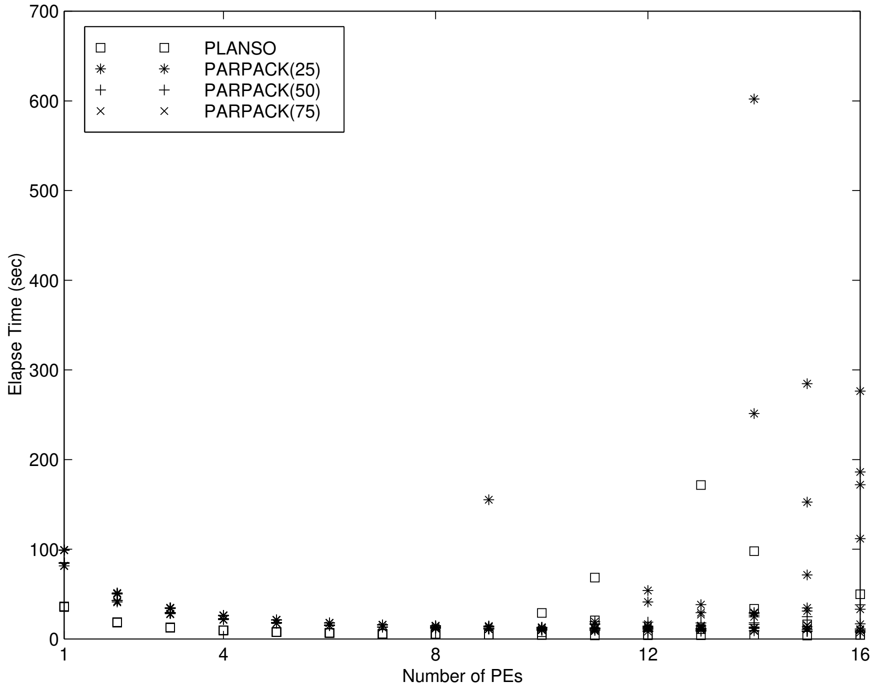

Figure 10: Time used by PLANSO on a cluster of 2 8-PE SPARC Ultra workstations.

Lanczos routine uses significantly less work in orthogonalization, it uses much less time to solve the same problem.

Figure 10 records the time measured in our tests. The two previous figures, Figures 8 and 9, use only the minimum time measured. The variance in wall-clock time is fairly small, $< 5\%$, when all the processors are on one bus. However, when processors from both SMP are used, the variance can be fairly large. The longest time measured is about an order of magnitude larger than the shortest time measured in some cases. This reveals that it is not easy to achieve consistent high performance on the cluster of SMPs. Some of the reasons include: a parallel job is scheduled as a number of independent tasks on the processors involved, interference from other users both on communication medium and CPU resource, and so on. These types of problems are common to most clusters of SMPs and clusters of workstations. There is a significant amount of research on improving job scheduling and other issues. Clusters of SMPs could eventually be able to consistently deliver the performance of the MPP systems.

5 Summary

This report presents our parallel version of LANSO (PLANSO) and measures its performance on two different platforms. As we have shown in numerical tests, as long as the local problem size is not very small, say more than a few thousand, the parallel LANSO can maintain the parallel efficiency above that of the matrix-vector multiplication routine. Because the Lanczos algorithm uses significantly less arithmetic operations than other similar algorithms, such as the Arnoldi algorithm, PLANSO solves symmetric eigenvalue problems in less time than PARPACK. The main reasons are: PLANSO implements the Lanczos algorithm with partial re-orthogonalization which reduces the work spent on orthogonalization to the minimum, and it uses less matrix-vector multiplications because it does not restart.

The two platforms on which we performed our tests, the MPP system and Cray T3E, can deliver more consistent performance results. The best speed-up result from solving the NASASRB problem on the cluster of Sun's is very close to the result from the Cray T3E tests, 11.2 versus 11.5. However, the difference between the worst measured time and the best measured time is very large on the cluster of SMPs.

The dot-product operation is the one with the lowest parallel efficiency in the tests presented. This indicates to us that enhancing its efficiency could significantly enhance overall efficiency. One commonly used scheme is a block version of the Lanczos algorithm. Other alternatives include rearranging the algorithm so that the two dot-product computations can be done with only one global reduction operation, and so on. These alternatives will be considered in the future.

On some parallel environments, such as Cray T3E, the I/O performance does not grow proportional to the number of processors used for computing. In this case, considerable effort may be required to develop a scalable out-of-core Lanczos method. The simple out-of-core PLANSO tested does not scale well on T3E. If a large number of Lanczos steps are needed to compute a solution, PARPACK could be more effective because the user can limit the maximum memory usage by restarting the Arnoldi algorithm. The restarting scheme lets PARPACK use more matrix-vector multiplications but smaller amounts of memory to find the solution. This is a common trade-off that could benefit the Lanczos algorithm as well.

From the tests, we also identified that the dot-production operation can be a bottleneck for scalability of the parallel Lanczos algorithm. Reducing the cost of dot-product operation could significantly enhance the overall performance of the Lanczos algorithm.

PLANSO can also be easily used with different parallel sparse matrix packages. In this report we have shown examples of using PLANSO with P_SPARSLIB, AZTEC and BLOCKSOLVE. The example programs used in this report are also available with the distribution of the source code of PLANSO at `http://www.nersc.gov/research/SIMON/planso.tar.gz`.

6 Acknowledgments

The MPP tests were performed on the Cray T3E at National Energy Research Scientific Computing Center (NERSC). The cluster of SMPs used was established by a joint research project between NERSC, UC Berkeley, Sun Microsystems, and the Berkeley Lab Information and Computing Sciences Division (ICSD) and Materials Sciences Division (MSD).

Appendix Interfaces of parallel LANSO

The interface to the main access points of the parallel LANSO package are similar to their sequential counterparts. The two main entries are named `PLANDR` and `PLANSO`, they are the parallel counterparts of `LANDR` and `LANSO` of the LANSO package. The difference is that a new argument named `MPICOM` is attached to the end of the calling sequences, see below.

```
      SUBROUTINE PLANDR(N,LANMAX,MAXPRS,CONDM,ENDL,ENDR,EV,KAPPA,
     &                  J,NEIG,RITZ,BND,W,NW,IERR,MSGLVL,MPICOM)
      INTEGER N,LANMAX,MAXPRS,EV,J,NEIG,NW,IERR,MSGLVL,MPICOM
      REAL*8  CONDM,ENDL,ENDR,KAPPA,
     &        RITZ(LANMAX),BND(LANMAX),W(NW)

      SUBROUTINE LANSO(N,LANMAX,MAXPRS,ENDL,ENDR,J,NEIG,RITZ,BND,
     &                 R,WRK,ALF,BET,ETA,OLDETA,NQ,IERR,MSGLVL,MPICOM)
      INTEGER N,LANMAX,MAXPRS,J,NEIG,NQ(4),IERR,MSGLVL,MPICOM
      REAL*8  ENDL,ENDR,R(5*N),WRK(*), ETA(LANMAX),OLDETA(LANMAX)
     &        ALF(LANMAX),BET(LANMAX+1),RITZ(LANMAX),BND(LANMAX)
```

The arguments are as follows,

- `N` (INPUT) The local problem size.
- `LANMAX` (INPUT) Maximum Lanczos steps. If the LANSO routine does not find `MAXPRS` number of eigenvalues, it will stop after `LANMAX` steps of Lanczos iterations.
- `MAXPRS` (INPUT) Number of eigenvalues wanted.
- `CONDM` (INPUT) Estimated effective condition number of operator M.
- `ENDL` (INPUT) The left end of interval containing unwanted eigenvalues.
- `ENDR` (INPUT) The right end of interval containing unwanted eigenvalues.
- `EV` (INPUT) Eigenvector flag. Set `EV` to be less or equal to 0 to indicate that no eigenvector is wanted. Set it to a natural number to indicate that both eigenvalues and eigenvectors are wanted. When eigenvectors are wanted, they are written to Fortran I/O unit number `EV` without formatting.
- `KAPPA` (INPUT) Relative accuracy of the Ritz values. It is only used when eigenvectors are computed, in which case an eigenpair is declared converged if the error estimate is less than $\kappa|\lambda|$. If eigenvector is not wanted, an eigenvalue is declared converged when its error is less than $16\epsilon|\lambda|$, where ϵ the machine's unit round-off error. In fact, the subroutine `PLANSO` always use this as convergence test. `KAPPA` is only used in subroutine `RITVEC` which computes Ritz vectors when the user requested the eigenvectors, `EV` > 0.
- `J` (OUTPUT) Number of Lanczos steps taken.
- `NEIG` (OUTPUT) Number of eigenvalues converged.
- `RITZ` (OUTPUT) Computed Ritz values.
- `BND` (OUTPUT) Error bounds of the computed Ritz values.

- W (INPUT) Workspace. On input, the first N elements is assumed to contain the initial guess r_0 for the Lanczos iterations.
- NW (INPUT) Number of elements in W. It must be at least $5 * \mathtt{N} + 4 * \mathtt{LANMAX} + \max(\mathtt{N}, \mathtt{LANMAX} + 1) + 1$ when no eigenvector is desired. Otherwise an additional $\mathtt{LANMAX} \times \mathtt{LANMAX}$ elements is needed.
- IERR (OUTPUT) Error code from TQLB, a modified version of TQL2 from EISPACK [8].
- MSGLVL (INPUT) Message level. The larger it is, the more diagnostic message is printed by LANSO. Set it to zero or smaller to disable printing from LANSO, set it to 20 or larger to enable all printing.
- MPICOM (INPUT) MPI communicator used. All members of this MPI group will participate in the solution of the eigenvalue problem.

The following items are in the argument list of PLANSO but not in the argument list of PLANDR. They are part of workspace W. PLANDR checks the size of W and splits it correctly before calling PLANSO. It also provides the extra functionality of computing the eigenvectors after returning from PLANSO. We encourage the user to use PLANDR rather than directly calling PLANSO.

- R holds 5 vectors of length N.
- NQ contains the pointers to the beginning of each vector in R.
- ALF holds diagonal of the tridiagonal T, α_i.
- BET holds off-diagonal of T, β_i.
- ETA holds orthogonality estimate of Lanczos vectors at step i.
- OLDETA holds orthogonality estimate of Lanczos vectors at step $i-1$.
- WRK miscellaneous usage, size $\max(\mathtt{N}, \mathtt{LANMAX} + 1)$.

As mentioned before, the matrix operations are performed through two fixed-interface functions. They are as follows:

```
SUBROUTINE OP(N,S,Q,P)
SUBROUTINE OPM(N,Q,S)
INTEGER N
REAL*8 P(N), Q(N), S(N)
```

The subroutine OPM computes $s = Mq$, where Q and S are local components of the input vector q and output vector s. The subroutine OP computes $p = M^{-1}Kq$, where $s = Mq$ is passed along in case it is needed. The subroutine OP may alternatively compute $p = (K - \sigma M)^{-1}Mq$ in order to compute eigenvalues near σ. To find the eigenvalues of a symmetric matrix A, OPM simply copies Q to S, OP computes $p = Aq$.

LANSO stores and retrieves old Lanczos vectors through the STORE function which is assumed to have the following interface.

```
SUBROUTINE STORE(N,ISW,J,S)
INTEGER N,ISW,J
REAL*8 S(N)
```

The argument N is the local problem size, ISW is a switch that can take on one of the following four values: 1, 2, 3, or 4 which means store $q_j(1)$, retrieve $q_j(2)$, store $Mq_j(3)$, and retrieve $Mq_j(4)$, argument J is the aforementioned index j, and local components of input/output vector are stored in S. Through this routine, LANSO may store the Lanczos vectors in-core or out-of-core. Normally, storing the Lanczos vectors in-core means that they can be retrieved much faster than storing them out-of-core. However, since the Lanczos iteration may extend many steps before the wanted eigenvalues are computed to satisfactory accuracy, it may not be possible to store the Lanczos vectors in-core. The user is required to provide an effective routine to store the Lanczos vectors.

The partial re-orthogonalization scheme implemented in LANSO orthogonalizes the last two Lanczos vectors against all previous Lanczos vectors when a significant loss of orthogonality is detected. It does so by retrieving each of the previous Lanczos vectors and orthogonalize the two vectors against the one retrieved. If the norm of the vector after orthogonalization is small, the orthogonalization may be repeated once. The parallel LANSO package provides a default re-orthogonalization that uses the STORE function to access each Lanczos vector and perform a modified Gram-Schmidt procedure. A faster orthogonalization routine can be constructed by the user since the user knows where the Lanczos vectors are stored. For example, on parallel machines, if the Lanczos vectors are stored in-core, the user may use the classic Grid-Schmidt procedure rather than modified Grid-Schmidt procedure. This allows the global sum operations of all the inner-products to be performed at once, which reduces the communication overhead in orthogonalization. The user may not have to replace this routine if the re-orthogonalization is rear and a relatively small percentage of total time is spent in it. However, in many cases, re-orthogonalization occurs every 5 to 10 Lanczos iterations near the end, the time spent in re-orthogonalization may be significant part of the overall computation. In these cases, the user should consider replacing the orthogonalization routine PPURGE to enhance the performance of the eigenvalue routine. Examples of the modified PPURGE functions are provided along with various examples of using the parallel LANSO code.

LANSO was designed to compute a few eigenvalues outside of a given range which is suitable to find eigenvalues near σ using the shift-and-invert operator $(K - \sigma M)^{-1}M$. Often one only wants a few smallest or largest eigenvalues and corresponding eigenvectors. For this purpose, an alternative interface to LANSO is provided.

```
      SUBROUTINE PLANDR2(N,LANMAX,MAXPRS,LOHI,CONDM,KAPPA,
     &                   J,NEIG,RITZ,BND,W,NW,IERR,MSGLVL,MPICOM)
      INTEGER N,LANMAX,MAXPRS,LOHI,J,NEIG,NW,IERR,MSGLVL,MPICOM
      REAL*8  CONDM,KAPPA,
     &        RITZ(LANMAX),BND(LANMAX),W(NW)
```

When LOHI is greater than 0, it computes MAXPRS largest eigenvalues, otherwise it attempts to compute MAXPRS smallest eigenvalues. It does not attempt to compute eigenvectors, but instead it places α_i and β_i at the front of `W` before returns to caller. This allows easy access to α_i and β_i which is needed to compute the eigenvectors. This version of LANSO always uses `KAPPA` in convergence test in the same fashion as in ARPACK, i.e., an eigenvalue λ_i is considered converged if its error bound is less than $\kappa|\lambda_i|$ or $\kappa\epsilon^{2/3}$. This alternative interface can provide exactly the same functionality as PARPACK's routine for symmetric eigenvalue problems.

References

[1] S. Balay, W. Gropp, L. C. McInnes, and B. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11, Mathematics and Computer Science Division, Argonne National Laboratory, 1995. Lastest source code available at URL `http://www.mcs.anl.gov/petsc`.

[2] F. Chatelin. *Eigenvalues of Matrices.* Wiley, 1993.

[3] J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Theory*, volume 3 of *Progress in Scientific Computing.* Birkhauser, Boston, 1985.

[4] J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Programs*, volume 4 of *Progress in Scientific Computing.* Birkhauser, Boston, 1985.

[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.*, pages 1–14, 1989.

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

[8] B. S. Grabow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, editors. *Matrix Eigensystem Routines — EISPACK Guide Extension.* Number 51 in Lecture Notes in Computer Science. Springer, New York, 2nd edition, 1977.

[9] R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM J. Matrix Anal. Appl.*, 15(1):228–272, 1994.

[10] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro. AZTEC user's guide. Technical Report SAND95-1559, Massively parallel computing research laboratory, Sandia National Laboratories, Albuquerque, NM, 1995.

[11] M. T. Jones and P. E. Plassmann. Blocksolve95 users manual: scalable library software for parallel solution of sparse linear systems. Technical Report ANL-95/48, Mathematics and Computer Science Division, Argonne national alboratory, Argonne, IL, 1995.

[12] B. N. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. *Math. Comp.*, 33:217–2388, 1979.

[13] Beresford N. Parlett. *The symmetric eigenvalue problem.* Prentice-Hall, Englewood Cliffs, NJ, 1980.

[14] Y. Saad and K. Wu. Design of an iterative solution module for a parallel matrix library (P_SPARSLIB). *Applied Numerical Mathematics*, 19:343–357, 1995. Was TR 94-59, Department of Computer Science, University of Minnesota.

[15] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems.* Manchester University Press, 1993.

[16] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, Boston, MA, 1996.

[17] Horst D. Simon. *The Lanczos algorithm for solving symmetric linear systems.* PhD thesis, University of California, Berkeley, 1982.

[18] D. Sorensen, R. Lehoucq, P. Vu, and C. Yang. ARPACK: an implementation of the Implicitly Restarted Arnoldi iteration that computes some of the eigenvalues and eigenvectors of a large sparse matrix. Available from `ftp.caam.rice.edu`, directory `pub/people/sorensen/ARPACK`, 1995.

[19] L.-W. Wang and A. Zunger. Large scale electronic structure calculations using the Lanczos method. *Computational Materials Science*, 2:326–340, 1994.

[20] F. Webster and G.-C. Lo. Projective block Lanczos algorithm for dense, Hermitian eigensystems. *J. Comput. Phys.*, pages 146–161, 1996.