View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

IN-39
46453
P. 31

NASA Technical Memorandum 104108

LANCZOS EIGENSOLUTION METHOD FOR HIGH-PERFORMANCE COMPUTERS

(NASA-TM-104108) LANCZOS EIGENSOLUTION METHOD FOR HIGH-PERFORMANCE COMPUTERS (NASA) 31 p CSCL 20K
N91-32526
Unclas
G3/39 0046453

Susan W. Bostic

September 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Lanczos Eigensolution Method for High-Performance Computers

Susan W. Bostic

Structural Mechanics Division
Computational Mechanics Branch
NASA Langley Research Center
Hampton, VA 23665-5225

Introduction

One of the most computationally intensive tasks in large scale mechanics problems is the solution of the eigenproblem. Eigenproblems occur in virtually all scientific and engineering disciplines. This chapter will discuss a particular method, the Lanczos method, for the solution of this problem. A brief discussion of the theory of the method will be followed by the computational analysis of the method and the implementation on parallel-vector computers. Two structural analysis applications will be presented: the buckling of a composite panel, and the free vibration analysis of an entire aircraft.

Several efficient eigenvalue solvers are widely used in the structural analysis community, examples of which include: the QR and QL methods [1-3], the inverse power method [1-3], subspace or simultaneous iteration [1-3], determinant search [4], and the sectioning method [5]. Each of these methods has advantages for certain classes of problems and limitations for others. Many of the most popular methods, such as the QR and QL methods, solve the complete system of equations rather than a reduced set. For very large problems, these methods prove to be inefficient. In contrast, recent studies indicate a growing acceptance of the Lanczos method as a basic eigenvalue analysis procedure for large-scale problems. Compared to subspace iteration, one of the most widely-used algorithms, the Lanczos method is as accurate and more efficient and has the advantage that information previously computed is preserved throughout the computation [6-11]. The Lanczos method shares the rapid convergence property of the inverse power and subspace iteration methods but is more efficient when only a few eigenvalues of a large order system are required. The single vector Lanczos procedure is the focus of this chapter, although the block Lanczos method is presently being examined by many researchers, particularly for cases where

multiple roots are expected [12]. The block Lanczos method typically requires more storage, more computation and produces less accurate results [13].

The implementation of the Lanczos method and some techniques that optimize the solution process by exploiting the vector and parallel capabilities on today's high-performance computers are discussed in this chapter.

Application to Structural Problems

The large-scale mechanics problems to be addressed in this chapter are the free vibration problem and the buckling problem. In the example problems, the finite element method is used to discretize the structure; that is, the structure is approximated by many "finite" elements joined together at "nodes". The fact that the elements can be connected in a variety of ways means that they can represent exceedingly complex shapes. The finite character of the structural connectivity makes the analysis by algebraic (or matrix) equations possible. All material properties of the original system are retained in the individual elements. The element properties, represented by the stiffness and mass matrices, are then assembled into global matrices. Matrix equations then express the behavior of the entire structure. For detailed discussion of the finite element method as applied to structural dynamics see references 14, 15, and 16.

The finite element method was originally developed for the aerospace industry to provide a solution for extremely complex confiqurations. The simultaneous availability of high-speed digital computers permitted the application of the method to a large range of engineering problems and by the early seventies, it was the method of choice for the numerical solution of continuum problems. Today there exist many large finite element codes capable of solving large-scale problems on individual workstations as well as large mainframe computers. Vibration and buckling problems are representative of the types of problems that require efficient algorithms as well as fast computation rates for timely solutions.

To determine the dynamic structural response, a free vibration analysis is carried out to find the natural frequencies and mode shapes. The natural frequencies are those at which a system oscillates in free vibration or without any external forces. In free vibration the motion of the structure is maintained by gravitational or elastic restoring forces. The natural frequencies of a system are related to its eigenvalues and must be known to prevent resonance which occurs when the natural frequencies coincide with the frequencies in the applied dynamic loads. They are also used in aeroelastic analysis and flexible deformation control. In the buckling problem, the buckling load is related to the eigenvalue. In these problems, the response of a system is represented by a set of eigenvectors and eigenvalues.

The Lanczos method solves the standard eigenvalue problem, $Av = \lambda v$, by a recursion formula. The application of this recursion results in a set of vectors, the Lanczos vectors, and elements of a tridiagonal matrix, T. The original large eigenvalue problem is transformed into a small tridiagonal problem which can easily be solved to obtain a small subset of eigenvalues and eigenvectors. These solutions can then be used to find the eigenpairs of interest of the original problem. The vibration and buckling structural analyses discussed in this chapter result in generalized eigenvalue problems and must be transformed to the standard eigenvalue problem for the Lanczos method. This transformation from the generalized eigenvalue problem to the standard eigenvalue problem necessary for the application of the Lanczos method is a computationally-expensive operation. Because the eigenvalues of interest for this class of problems are either the smallest eigenvalues, or those eigenvalues in a given range of the spectrum, the problem undergoes an inversion transformation as well. In order to make the algorithm even more general, a shift parameter is introduced into the original problem to allow solution for the eigenvalues closest to the value of the shift.

Vibration Analysis

For the vibration problem, the transformation process from the generalized eigenvalue problem to the standard eigenvalue problem to be solved by the Lanczos method is as follows:

The generalized eigenvalue problem for free vibration is,

$$K x = \omega^2 M x \qquad (1)$$

where K is a symmetric positive semi-definite stiffness matrix and M is a symmetric positive definite matrix and represents either a banded consistent mass matrix or a diagonal mass matrix, where the mass of the elements are lumped at the nodes. The vectors x_i represent the eigenvectors, or vibration mode shapes and the ω's are the eigenvalues or the vibration frequencies. The solution of equation (1) by the Lanczos method would yield the largest eigenvalues. For the vibration and buckling problems implemented here, the smallest eigenvalues are the ones of interest. Therefore, a shift, σ, close to the eigenvalues of interest, is introduced and then the problem is inverted. The computations necessary to convert the original problem to an equivalent shifted inverse form and then transform the generalized eigenvalue problem into the standard Lanczos form, $Av = \lambda v$, for the vibration problem follow.

Introducing a shift, σ, and inverting by letting

$$\omega^2 = 1/\lambda + \sigma \qquad (2)$$

then substituting (2) in equation (1) gives

$$Kx = (1/\lambda + \sigma) M x. \qquad (3)$$

Multiplying each side by λ yields

$$K \lambda x = M x + \sigma \lambda M x \tag{4}$$

and rearranging terms gives

$$\lambda [K-\sigma M] x = M x. \tag{5}$$

Finally, multiplying both sides by $[K-\sigma M]^{-1}$ produces

$$\lambda x = [K - \sigma M]^{-1} M x. \tag{6}$$

Letting,

$$A = [K - \sigma M]^{-1} M \tag{7}$$

substituting (7) in equation 6 and rearranging terms yields

$$Ax = \lambda x, \text{ or the standard eigenvalue equation.} \tag{8}$$

The implementaiton of the Lacnzos algorithm required the computation of the vector quantity Av for a agiven v. It is important to avoid the expensive computation of finding the inverse of the matrix in equation 7, which would result in a full matrix, losing the advantages of the banded, sparse, symmetric matrix. The following procedure is thus implemented.

Let

$$\bar{K} = [K - \sigma M] \tag{9}$$

1. Factor $\bar{K}$

$$\bar{K} = LDL^T \tag{10}$$

where L is a lower triangular matrix with unit diagonal, D is a diagonal matrix and L^T, or the transpose of L , due to symmetry, is the upper triangular matrix.

2. Then rearrange terms and introduce y

$$A v = (LDL^T)^{-1} M v \tag{11}$$

or $$A v = (L^T)^{-1} (LD)^{-1} M v \tag{12}$$

$$L^T A v = (LD)^{-1} M v = y. \tag{13}$$

3. Solve for y

$$L D y = M v. \tag{14}$$

4. Then solve for A v

$$L^T (A v) = y. \tag{15}$$

Buckling Analysis

Similarly, the transformation for the bucking problem is carried out as follows the generalized buckling problem is,

$$K x = -\delta K_g x \tag{16}$$

where K is the linear stiffness matrix, K_g is the geometric stiffness matrix, x is a buckling mode shape and δ is the buckling load. Because the geometric, or differential stiffness matrix K_g may be an indefinite matrix, a different shifting and inverting strategy is required.

In this case let

$$\delta = \sigma\lambda / (1-\lambda) \tag{17}$$

then substituting (17) into equation (16) gives

$$K x = -(\sigma \lambda /(1-\lambda)) K_g x. \tag{18}$$

Multiplying each side by (1- λ) yields

$$K x - K \lambda x = - \sigma \lambda K_g x \tag{19}$$

then

$$K x = [K - \sigma K_g] \lambda x. \tag{20}$$

Multiplying each side by $(K - \sigma Kg)^{-1}$ gives

$$\lambda x = [K - \sigma K_g]^{-1} K x . \tag{21}$$

Therefore, for the buckling problem

$$A = [K - \sigma K_g]^{-1} K \tag{22}$$

or , in standard form

$$Ax=\lambda x \tag{23}$$

Each multiplication by the mass matrix, M, in the vibration case is replaced by a multiplication by the stiffness matrix in the buckling case. To form the matrix A, the geometric stiffness matrix is multiplied by the shift parameter in the buckling case in place of the mass matrix as in the vibration case. The eigenvalue problems are real symmetric problems and the matrices, which result from the finite element method, are symmetric, sparse and banded. Symmetry of the matrices, where the upper triangle of the matrix is identical to the lower triangle matrix reduces the storage and computation requirements. Sparsity refers to the number of non-zeros in the matrix. A banded matrix is one where the non-zeros are clustered close to the diagonal. For the vibration problem, the mass matrix (M) can be either a diagonal

mass matrix where the mass is taken to be at the nodes, or the consistent mass matrix containing the distributed mass associated with the elements.

Lanczos Method

The Lanczos method was first introduced in 1950 by Cornelius Lanczos [18]. When the method was applied to real problems, using finite arithmetic, the method did not behave in accordance with the theoretical properties, numerical instabilities arose and the method was not widely accepted. In recent years, due to the research of many analysts [3,13,19-24], these instabilities have been understood and eliminated. As a result, new and innovative approaches have been developed to implement the method. The basic procedure uses a recursion to produce a set of vectors, referred to as the Lanczos vectors, and scalars that form a tridiagonal matrix. This tridiagonal matrix can then be easily solved for its eigenvalues which are used to compute a few of the eigenvalues of the original problem.

The basic Lanczos algorithm solves the standard eigenvalue problem:

$$A x = \lambda x \tag{24}$$

using the basic Lanczos recursion described below which results in a reduced eigenvalue problem:

$$T q = \lambda q \tag{25}$$

where T is a tridiagonal matrix consisting of α's on the diagonal and β's on the off diagonals.:

$$T = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \beta_4 & \alpha_4 & \beta_5 \\ & & \cdots & & \end{bmatrix}$$

The steps in this transformation process to tridiagonal form are:

1. Initialization

   a. Choose a starting vector v_1 , where v_1 is normalized., $|v_1| = 1$.

   b. Set $\beta_1 = 0$ and $v_0 = 0$.

2. Iteration

   for $i = 1,2,3,\ldots m$ as follows;

$$w = A v_i - \beta_i v_{i-1} \tag{26}$$

Then,

$$\alpha_i = v^T_i w \tag{27}$$

$$c = w - \alpha_i H v_i \tag{28}$$

$$\beta_{i+1} = [c^T H c]^{1/2} \tag{29}$$

$$v_{i+1} = c / \beta_{i+1} \tag{30}$$

where for the vibration case H is M and for the buckling case H is K, w and c are temporary vectors, the α vector is the diagonal term of the resulting tridiagonal matrix T and the β vector is the off-diagonal term. The vectors $v_1, v_2, \ldots v_m$ are the set of Lanczos vectors.

The order N of A may be 10,000 or more while order m is typically equal to twice the number of eigenvalues and eigenvectors desired, usually less than 50.

For each eigenvalue, λ, of T_m, a corresponding eigenvector, q, is computed such that,

$$T_m q = \lambda q \tag{31}$$

The frequencies of the vibration problem are found by

$$\omega^2 = \sigma + 1/\lambda \tag{32}$$

and the eigenvalues of the buckling problem are found by

$$\delta = \sigma \ (\lambda / (1-\lambda)) \tag{33}$$

The m eigenvectors X_m of equation 1 are then found by

$$X_m = V_m \, Q_m . \tag{34}$$

The eigenvalues of the tridiagonal matrix can be easily obtained using readily available library routines such as the QL algorithm, a fast, efficient method for the solution of tridiagonal matrices.

Reorthogonalization of the Lanczos vectors

When the Lanczos method was first put into practice, it was found that due to finite arithmetic calculations, the vectors tend to lose their orthogonality. Extra eigenvalues, labeled "spurious" may appear, as well as redundant values of the "good" eigenvalues. One of the on-going topics of research concerning the Lanczos method involves finding robust ways to overcome this deficiency. One approach is to reorthogonalize at each step, thus eliminating the effects of the previous vectors on the succeeding ones.

This has been considered "expensive" and shortcuts, such as selective reorthogonalization or partial reorthogonalization, have been proposed by Parlett and his co-workers, among others [21-24].

Another approach is proposed by Cullum and Willoughby [13] which involves no reorthogonalization but uses an identification test to select those approximations which are to be accepted. By comparing the eigenvalues found using the complete tridiagonal matrix to the eigenvalues found using the submatrix obtained by deleting the first row and column of the tridiagonal matrix, a decision can be made as to which eigenvalues are approximate enough to be considered accurate. In the examples that follow, the effect of reorthogonalization will be shown. An example of the unacceptable eigenvalues that appear when no reorthogonalization is performed will be presented and the cost of reorthogonalization will be tabulated. No attempt will be made to compare or promote the various reorthogonalization techniques.

Computational Analysis

The preceding sections outlined the computational steps to be carried out when implementing the Lanczos method. An efficient algorithm must take into consideration the architecture of the computer on which it will be implemented. The next sections will discuss

the characteristics of high-performance computers and some of the techniques and tools available to improve the efficiency of the Lanczos method.

High-Performance Computers

The computational power of today's high-performance computers now makes it possible to solve large, complex problems which were prohibitively expensive to solve in the past. This computational power in turn requires new computational algorithms that address the present problems now of interest as well as take advantage of the capabilities of the latest generation of computers. The vector capabilities of these computers offer speedups in computation of several magnitudes over sequential computers. When this vector capability is coupled with the capability to perform computations in parallel which is now available on many different types of architectures, the potential for solving larger problems substantially increases. This increase in computational power yields a more accurate and efficient solution to the eigenproblem.

The computation rate on high performance computers is commonly measured in millions of floating point operations per second or MEGAFLOPS (Mflops). The computation rate of the most powerful supercomputers has surpassed a billion floating point operations, or GIGAFLOPS, and rates will soon be measured in TERAFLOPS, for a trillion floating point operations per second.

The example problems cited in this chapter were executed on the Convex C220, the CRAY-2, and the CRAY Y-MP multicomputers. The Convex C220 at the NASA Langley Research Center consists of two central processing units, each of which can compute at a rate of from 20 to 40 Mflops for a computationally-intensive calculation. The CRAY-2 at NASA Langley Research Center has four central processing units (CPUs), while the CRAY Y-MP at NASA Ames Research Center has eight. Optimized code running on one CPU of the CRAY computers typically generates results in the 100-200 Mflops range. Each CPU in the Convex and CRAY has multiple vector functional units which access very large main memories though eight high-speed vector registers. These functional units can operate simultaneously and the maximum performance rate is obtained when both the addition and multiplication functional units are operating simultaneously.

Vectorization Optimization

The vectorization of code must be fully optimized before considering any parallel processing on parallel-vector computers. For maximum performance, software must be tuned to best exploit the hardware capabilities. The high performance of vector computers is due to "vector units", designed to perform such computations as adds and multiplies simultaneously.

Arithmetic operations are "pipelined" into these vector units. Pipelined arithmetic units allow for operations to be overlapped as in an assembly line. Several specialized subsections work together to execute an operation. When the first section completes its processing on a set of operands, the results are passed to the next section, and a new set of operands enters the pipe. To carry out such operations there must be no data dependency. In other words, DO loops must be avoided where a result depends on completion of a previous iteration of the loop,such as in the recursion: A(I) = A(I-1) + B(I). By efficient vectorization, speedups of 10-20 can be achieved.

On vector computers, three factors that influence the vector computational rate are: the number of memory accesses per computation, the length of the vectors, and the vector stride, which is the spacing in memory between elements of the vectors involved. Long vectors reduce the ratio of the overhead and initial memory access time to the amount of computation. Vectors of stride one, or vectors whose elements are contiguous in memory, are the fastest to access. The number of memory accesses can be reduced using different techniques, the most rewarding being loop unrolling.

Loop unrolling

A useful technique to enhance vector performance is loop unrolling. An example of loop unrolling to level 6 is show in figure 1. The example is a matrix-vector multiply, C = A * B, where A is an n x n matrix and B and C are n x 1 vectors, the iterations of the inner loop are decreased by a factor of 6 by the explicit inclusion of the next five iterations. In the loop below, the vector C is accessed once for the 6 multiply and add operations. The vector multiply, vector add, and vector access from memory are,for the most part, carried out concurrently.

```
DO 10 j=1,n,6
  DO 10 i=1,n
    C(i) = C(i) + A (i,j) * B(j)
                + A(i,j+1) * B(j+1)
                + A(i,j+2) * B(j+2)
                + A(i,j+3) * B(j+3)
                + A(i,j+4) * B(j+4)
                +A(i,j+5) * B(j+5)
10  CONTINUE
```

Figure 1. Loop unrolling to level 6

Compiler directives

High-performance computers have sophisticated compilers which can detect vectorizable and sometimes parallelizable code. There are situations, however, when the compiler cannot optimize code because of unknown conditions, such as data dependencies. Compiler directives can be used by the analyst in these situations. When the analyst knows that the variable in question will never have a value that would create a dependency conflict, vectorization of a loop can be forced. The use of this directive requires an intimate knowledge of the algorithm and problem in order to maintain the integrity of the data, but can result in significant reductions in computation time.

Local memory

The latest generation of computers often have local memory, or caches, which can be used to store data which is accessed repeatedly in a computation sequence. The number of memory accesses can be decreased dramatically when this option is available. An example of the savings on the CRAY-2 using local memory storage for data is shown in the examples to follow. This option is utilized in the factorization of the matrix to store the multipliers used to update the columns.

Parallel Processing

There are many types of parallel architectures now available. An early classification of parallel architectures consisted of four types of architectures: Single-Instruction/Single Data, a scalar computer consisting of one processor working on one stream of data, Single-Instruction/Multiple Data, which defines vector machines where a single stream of instructions operate on separate elements of an array simultaneously, Multiple-Instruction/Single Data which implies that several instructions are operating on a data stream simultaneously and Multiple-Instruction/Multiple Data computers, where several processors concurrently act on multiple data streams [25]. The computers used in this study belong to the class labeled, Multiple-Instruction Multiple Data (MIMD) computers with shared memory, although some of the principles will apply when programming for other architectures. The implementations to be presented were designed for computers with a few, powerful processors, as opposed to the class of computers referred to as Massively-Parallel Processors which have thousands of processors, each capable of performing a small amount of computation.

The use of multiple processors to execute portions of a program simultaneously offer significant speedups in computation. However, these speedups can be difficult to achieve in practice. All programs have a portion of work that must be executed sequentially, or duplicated in other processors, however, and it is rare when large portions of the work can be equally divided among the number of processors available. There is also overhead

associated with parallel processing, particularly synchronization, or the process of coordinating the tasks within the parallel regions. Parallel tasks must execute independently, in any order and without regard to the number of processors available at running time. The main purpose of executing computations in parallel is to decrease wall clock time for a particular solution. The actual computation time, which is the sum of time expended by all processors, will increase. One must determine the potential wall clock time to be saved versus the programming effort involved and the increase in total CPU time to evaluate the benefits of parallel processing.

Software written for parallel processing requires more analysis, with particular attention given to data dependencies, the scope of the data, critical regions where communication must be synchronized and load balancing, which implies an equal distribution of work to each available processor. Local, or private data, such as loop indices, are only accessible to the defined task. Shared, or common data is accessible to all tasks. Shared data must be protected and the proper communication and synchronization provided.

In some cases load balancing can be determined before runtime, in which case it is referred to as static load balancing. If the work must be distributed during execution, dynamic load balancing is required. Parallel codes are more difficult to test and debug than sequential or vectorized codes.

Granularity

The level of parallel processing depends on the granularity of the computation. A high level of granularity refers to executing large sections of code, such as complete subroutines, concurrently. The initial parallel processing software mechanism on the CRAY, referred to as macrotasking, had a high overhead and required a high level of granularity to be efficient. If the computation can be divided into large independent tasks that are equal and can be performed simultaneously, macrotasking can be invoked with a minimum amount of overhead.

The computations necessary for eigenvalue analysis typically do not have tasks that can be carried out concurrently at the subroutine level. For the small granularity inherent in these algorithms, microtasking is used to process tasks within a subroutine. For instance, in a loop that will be generated many times, the number of times through the loop can be divided up among the available processors.

Autotasking is a feature now available on CRAY systems and other computers with sophisticated compilers that detects parallel regions in a pre-processing phase. The autotasking capability detects regions that can be microtasked and automatically generates code to assign tasks to all processors that are available. The programming effort in this case is minimal, as is the computational overhead.

Dedicated versus Batch Mode

The decrease in wall clock time depends on many factors, one of which is the mode in which the program is executed. The high-performance computers used in research labs and in production environments are typically heavily utilized. Parallel programs running in a batch mode are competing with all the other programs in the system for hardware resources. In a batch mode programs use those processors available at the time and there is no guarantee that more than one processor will be assigned to a given execution. For particular high priority, long-running jobs, where it is important to get the answers as fast as possible, such as weather prediction, jobs may run in a dedicated mode where all processors are assigned to the one job. In a dedicated mode all processors will be available to the program and the work will be divided as directed. In both cases a decrease in wall-clock time should result for the execution of a program, but when run in batch mode, the decrease could be significantly less. In the examples following, statistics will be presented showing the difference between running in the two modes.

Implementation of the Lanczos Method

The first step in the algorithm development process is to identify the time-consuming calculations. Software tools are available on today's high-performance computers to analyze the computations in a program. The Lanczos algorithm was analyzed using the flowtrace capability on the CRAY-2. This utility computes the percentage of time spent in each section of the code. For the structural analysis problems presented in this study, the three dominant computational steps are: factorization of the matrix $\bar{K}$, as in equation 10, the forward/backward solution steps in equations 14 and 15, and the matrix-vector multiplications in equations 28 and 29. For typical structural analysis problems, the factorization and forward/backward solution steps combined in the Lanczos method take over 50 percent of the total computation time and the matrix-vector multiplications take another 20 to 25 per cent of the computation time [6].

The following sections will address some of the issues involved in exploiting the architectural capabilities of high-performance computers in order to decrease the computation time of the Lanczos eigensolver. The first section will describe the direct Choleski solver for variable-band matrices that was used in the implementation of the Lanczos method presented earlier in this chapter.

Variable Band Choleski Solver

An important area of research on parallel-vector computers has been the solution of linear systems of equations. In many algorithms, the equation solution, including the factorization of the matrix and the forward/backward solves, is the dominant factor in terms of the amount of computation. This is particularly true in the Lanczos algorithm, as the matrix to be factored can be extremely large and the forward/backward solution is repeated many times. Comparisons have been made between direct and indirect solvers and various implementations have been tested. Memory requirements, the number of floating point arithmetic operations required and the speed at which the operations can be performed influence the choice of which solver to use.

The decomposition of a symmetric, positive-definite matrix into lower and upper triangular matrices which can then be used in the forward and backward solution steps, is attributed to Choleski [2]. The solver used in the example problems that follow is a variation of the Choleski solver, described as a variable band Choleski solver. The decomposition, or factorization, used in the Lanczos eigensolver is an LDL^T factorization, which results in a lower triangular matrix, L, a diagonal matrix, D, and an upper triangular matrix L^T which is the transpose of the lower triangular matrix. This solver has been shown to be efficient and accurate, and outperformed iterative solvers, as well as sparse solvers, on a vector computer for representative structural analysis problems [26,27]. The high computation rate for this solver more than made up for the increase in memory and the greater number of arithmetic floating point operations.

As previously stated, the matrices that result from the finite element method in structural analysis are often large, sparse and banded. The amount of computation involved in the factorization of the matrix $\bar{K}$ and the equation solution steps depend on the size of the problem and the bandwidth of the matrices. In the variable band storage scheme used in the described implementation, the number of degrees of freedom of the finite element model determine the number of rows in the stiffness and mass matrices. The length of each row, (or column, as the matrices are symmetric), or bandwidth, is determined by the connectivity of the elements. The number of rows in the matrices or the number of degrees-of-freedom for a complex aircraft or space station model can be several hundred thousand. For these large problems the issue of data storage and access is most important in determining the efficiency of the implementation. The Choleski factorization and equation solver to be described uses column-oriented variable-band storage.

Storage Schemes

The most efficient type of data storage is determined by the computation algorithm to be implemented. For sparse, banded matrices a choice must be made between storing the banded matrix which contains zero elements but results in long, efficient vector operations and storing only the non-zero elements, referred to as sparse storage, which conserves storage and reduces the amount of computation but often seriously decreases the computation rate. Poole compares banded equation solvers with sparse equation solvers in reference 26. Results vary, depending on both the problem to be solved and the hardware on which the program is executing. For the typical structural analysis problems on a CRAY-2 the variable band Choleski equation solver was the fastest in terms of the megaflop rate and computation time. One other advantage of the variable band solver is the type of computation dictated by the algorithm.

The two vector computations encountered most in the factorization and forward/backward solve are the inner product ($x^t x$) and the saxpy, or $\Sigma(ax_i+y_i)$, where x and y are vectors and a is a scalar. On vector machines the saxpy is by far the more efficient operation. With proper use of loop unrolling, the saxpy operation allows overlapping of memory accesses with simultaneous use of both the add and multiply functional units. The variable band storage scheme's efficiency is in part due to the fact that it enables the use of the saxpy operations.

Reordering of Nodes

When using a banded solver, the amount of computation involved in the factorization and forward/backward solves is directly proportional to the semi-bandwidth. It is, therefore, important to decrease the semi-bandwidth, or the distance from the diagonal to the last non-zero, as much as possible. The non-zero quantities in the stiffness matrix represent the connectivity of the elements in the finite element model. Often, the numbering of the nodes is done by a computer program, or an analyst to whom the structure of the resulting matrix is not of concern. In some cases, rows or columns of the matrix may be exceptionally long and have very large semi-bandwidths, but contain mostly zeros. For maximum efficiency the nodes of the finite element model often need to be renumbered to reduce the semi-bandwidth of the matrices. The particular method used to renumber the nodes for the applications discussed in this chapter was a reverse Cuthill-McKee profile minimizer [28]. Such algorithms can significantly reduce the semi-bandwidth of the matrices and for the example problems, a significant amount of storage and computation is saved by using this reordering scheme.

Column Storage versus Skyline Storage

For the variable band Choleski algorithm, the lower triangular part of the symmetric matrix is stored by columns, beginning with the main diagonal down to the last non-zero entry in each column, including zeros. This data storage scheme is in contrast to the skyline or profile schemes which store the upper triangular part of the matrix by columns beginning with the main diagonal and storing all coefficients up to the first non-zero in each column. The advantage of the skyline storage scheme is that it requires less storage. One advantage of using the variable band storage scheme is the type of floating point operations associated with the method, particularly the saxpy operation. The vector lengths are also longer which helps to offset the fact that more total computations are required. A schematic of the storage scheme is shown in figure 2. The numbers in figure 2 indicate the order in which the elements of the matrix are stored.

```
1
2 5
3 6 8
4 7 9  13
     10 14
     11 15
     12 16
        .
          .
```

Figure 2. Variable band storage of lower triangle by columns

Reference 26 describes the variable band Choleski solver method in detail. This method is able to exploit key architectural features of vector computers and runs well in excess of 100 Mflops on the CRAY-2 and CRAY Y-MP computers. The storage scheme allows the factorization routine to be carried out with stride one vectors, or those with elements stored in contiguous locations. To increase the speed of the factorization, an immediate update strategy is used where each column is used to update the other columns as soon as it is computed. The forward solution uses a column sweep approach, thus accessing the data in the most efficient way. The variable band storage format results in using the efficient saxpy operation in the factorization, allowing addition and multiplication to be performed simultaneously.

The lower triangular matrix, L and the diagonal matrix, D are stored in the location previously occupied by $\bar{K}$, as this original matrix is not needed again. A by-product of the factorization of $\bar{K}$ is that the number of eigenvalues less than the given shift (σ) can be found. The number of negative entries in the diagonal matrix D produces this information.

Matrix-Vector Multiplication

Another time-consuming operation in the solution process was the matrix-vector operation which is carried out three times for each iteration. For this calculation, it proved more efficient to eliminate the zeros in the variable band matrix and to store only the non-zero coefficients of the lower triangular part of the matrix by columns in a single dimensioned array.

Two integer pointer arrays are used to store the beginning location of each column and the length of each column. The matrix-vector multiplication takes advantage of the fast saxpy operation, explained previously. This storage scheme can effectively shorten the vector lengths, so a trade-off exists between storing only the non-zeros and the variable band storage. Statistics comparing the sparse matrix-vector multiplier versus a banded matrix-vector multiplier will be shown in the applications section.

Applications

Predicting the structural response of the next generation of aerospace structures will place great demands on available computational power. The complexity of these structures dictates finite element models of small granularity which result in very large problems to be solved. The applications to be addressed here are representative, although on a smaller scale than what can be realistically expected.

The first example, a blade-stiffened panel with cutout, is a representative component of an aerospace vehicle. The second example is a preliminary model of a high-speed civil transport. The examples are used to best determine the most efficient exploitation of parallel-vector computers.

Vectorized Lanczos Implementations

Since the benefits of vectorization, in terms of reducing computation time, are much greater than the benefits of multitasking on the parallel-vector computers used in this study, the Lanczos algorithm was first vectorized using the techniques described in the previous section. The effects on the computation time of these optimization techniques, including automatic vectorization, compiler directives and loop unrolling are shown with the buckling of the laminated blade-stiffened panel problem as an example.

Panel Problem

The finite element model of a graphite-epoxy blade-stiffened compression panel with a discontinuous stiffener is depicted in figure 3.

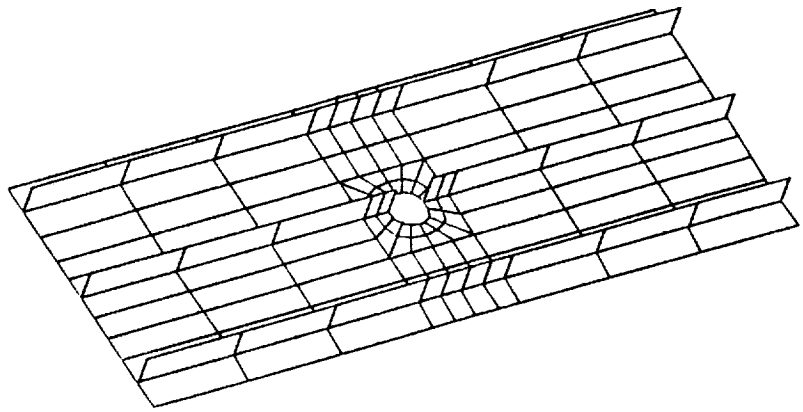

Figure 3. Finite element model of blade-stiffened panel with cutout.

This graphite-epoxy panel represents a generic class of laminated composite structures whose properties must be understood before being incorporated into future aerospace vehicles. This problem was selected as an example because experimental results are available and the characteristics, such as a discontinuity, large displacements, and a brittle material system, are representative of practical composite structures [29]. The panel skin is a 25-ply laminate and each blade stiffener is a 24-ply laminate. The panel was loaded in axial compression. The loaded ends of the panel are clamped and the sides are free. The Lanczos method is used to find the buckling load of this stiffened panel with cutout. The first buckling mode is shown in figure 4.

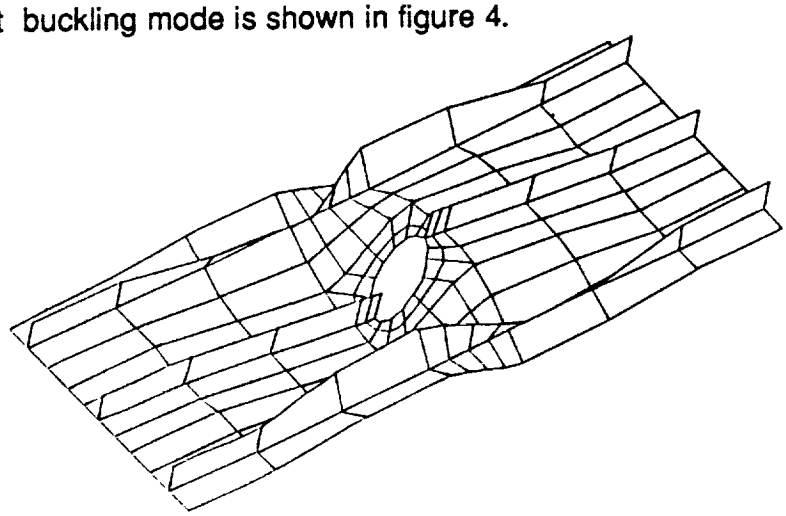

Figure 4. First buckling mode of blade-stiffened panel.

The Lanczos algorithm found eigenvalues that agreed with those found by a subspace iteration method and the Lanczos method was an order of magnitude faster.

A discretization of the panel which resulted in 6426 degrees of freedom was used as a model for this study and the first five buckling modes were computed. The Lanczos eigensolver was first coded and run in a scalar mode, with no optimization, on the Convex 220. The total computation time for this problem was 181.6 seconds, with the factorization of the matrix taking 55% of the time and the forward/backward solutions taking 30% of the time. The automatic vectorization option of the compiler was then exercised and the total execution time was decreased to 64 seconds. The loops that the compiler could not vectorize without intervention of the analyst were studied for data dependencies. Compiler directives were inserted where applicable, reducing the computation time to 21 seconds. The main loop in the factorization routine was unrolled to an optimal level 6 decreasing the computation time to 14.1 seconds. The total savings in execution time for the panel buckling problem on the Convex obtained by automatic vectorization, compiler directives and loop unrolling are shown in figure 5. An overall speedup of almost 13 is achieved for the optimized vector code over the sequential implementation for this representative problem.

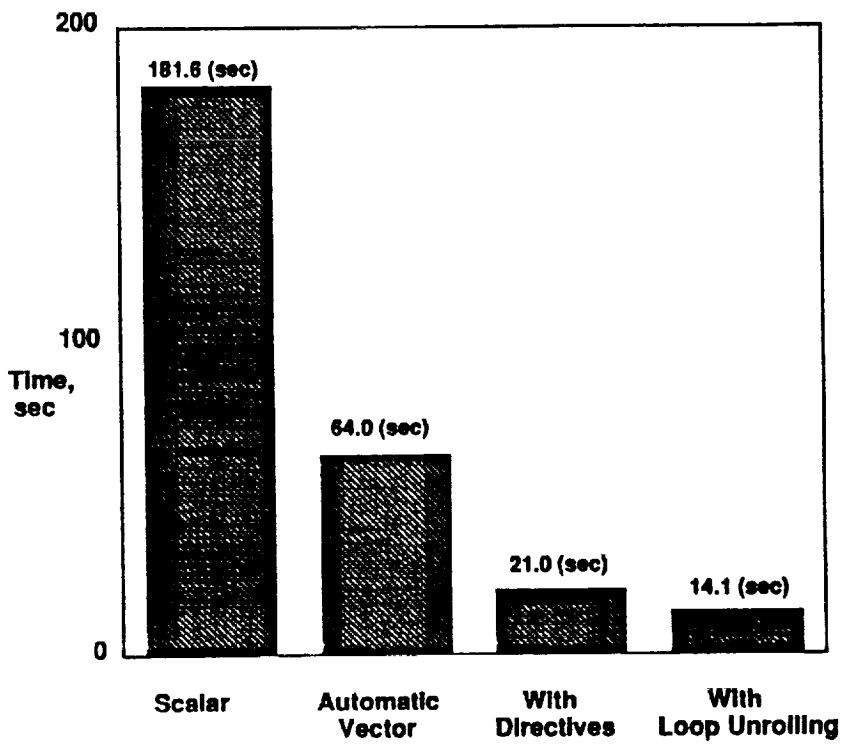

Figure 5. Improvement in computation times for panel buckling problem on Convex 220.

Another major time-consuming calculation for the solution to the panel problem was determined to be the matrix-vector multiplies. A comparison was made between using the variable band storage scheme for this operation and converting the matrix to a sparse storage, or eliminating the zeros within the columns, thereby reducing the number of floating point operations but at the same time decreasing the vector length. It was found that the sparse storage scheme resulted in a significant decrease in computation time even though the megaflop rate was decreased.

The comparison of the number of operations, the computation times and the megaflop rate (Mflops) between a variable band matrix multiply and a sparse matrix multiply for the panel problem on the CRAY-2 are shown in Table 1.

Table 1 Variable band matrix-vector multiply vs. sparse matrix-vector multiply for panel problem with 6423 degrees of freedom

Type of Matrix	Number of Operations	Time, secs	Mflops
Variable Band	4,129,044	.042	97.8
Sparse(Non-zeros, only)	300,512	.013	22.7

As shown, the time to multiply the stiffness matrix in the sparse format by a vector was .013 seconds while the time to multiply the same matrix stored in variable band format was .042 seconds. The megaflop rate is over four times faster using the variable band algorithm, but the fewer floating point operations of the sparse storage scheme results in reducing the overall execution time by 69%. seconds. This matrix-vector multiply was performed over three times for each Lanczos step, thus even a small reduction in time results in a significant saving in overall computation time.

The use of local memory on the CRAY-2 can speed up calculations by decreasing the number of memory references. In the factorization step, those vectors that will be accessed consecutively many times are stored in the local memory. The comparison of the computation times for factoring the matrix for the panel problem on the CRAY-2 using local memory and not using local memory are shown in Table 2.

Table 2 Effects of using local memory in factorization step for panel problem with 6423 degrees o f freedom on a CRAY-2.

	Time, secs	Mflops
No local memory	1.57	127.0
Local memory	1.49	133.7

High Speed Civil Transport on CRAY-2

Considerable research in the aerospace field is being directed toward the development of supersonic civil transport aircraft. A finite element model for the preliminary design studies of a high speed civil transport is shown in figure 6. The symmetric half of the structure is composed of 2851 nodes, 5189 two-noded rod elements, 4329 four-noded quadrilateral elements and 114 three-noded triangular elements. This structure has 17,106 degrees of freedom. Eliminating the constrained degrees of freedom results in 16,146 active degrees of freedom, resulting in stiffness and mass matrices of that size. After resequencing the node numbering for minimal bandwidth, the maximum semi-bandwidth of the problem was 594 and the average semi-bandwidth was 319.

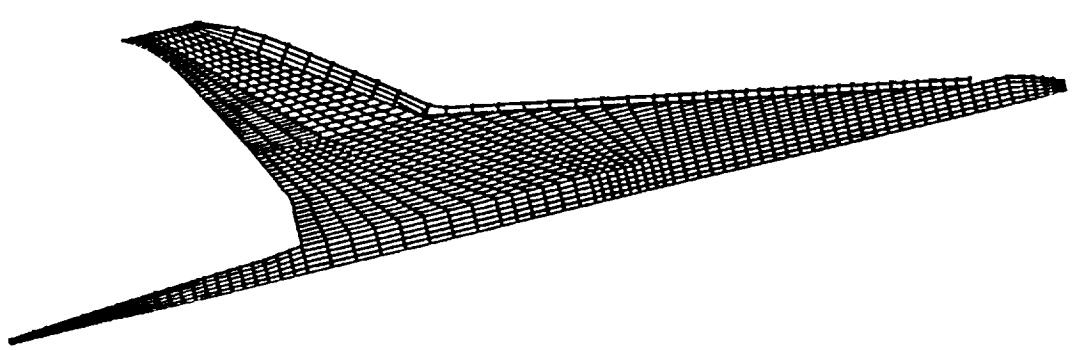

Figure 6. Finite element model of symmetric half of high-speed civil transport.

Timing results for the high speed civil transport problem when run on the CRAY-2 with the optimized variable band factor and solve routines and the sparse matrix-vector routine as described previously and without any parallelization are shown in Table 3. The value of m, or the number of approximate eigenvalues to be calculated, was 60. This results in 30 "acceptable" eigenvalues and associated eigenvectors. This input value was held constant for all of the examples that follow. The size of the matrices resulted in long vector lengths, making the vector operations efficient and the megaflop rate large. As shown in the table, the megaflop rate for the factorization step was 158.

Table 3 Solution time for Vectorized HSCT problem on CRAY-2.

Computation Step	Time (seconds)	Number of Floating Point Operations	Mflops
Factorizaton	8.5	1,345,852,840	158
Forward/Backward Solve	11.7	1,260,167,646	107
Reorthogonalization	1.4	114,313,680	80
Matrix-Vector Multiply	15.3	379,409,549	25
Total CPU time	40.3		

Parallel Lanczos Implementation

The vectorized code was next parallelized using the CRAY autotasking capability with compiler directives inserted where data dependencies could not be resolved by the compiler. The HSCT problem was run on the CRAY-2 and the CRAY Y-MP and performance measurements were made.

Multitask Performance Measurement

There are many different types of measurement tools available on the CRAY systems. One of these, the job accounting report, lists multitasking time usage. Several timing routines are available that report CPU time, wall clock time and the number of clock ticks used for each job, or section of a job. Even with these measurement tools, the performance of a multitasked program is sometimes difficult to measure and the timing results vary from run to run particularly when run in a batch mode. An example of timing statistics obtained when running the high speed transport problem on the CRAY Y-MP is shown in Table 4. in a batch mode. In this case an average of 3.63 processors of the eight processors were used. The total CPU time was 23 seconds, the time on only one processor was 2.65 seconds and the time spent using more than one processor was 3.71 seconds. The CPU time is obtained by multiplying the number of processors used (column 1) by the amount of time spent using those processors concurrently (column 2).

Table 4 Solution time for High Speed Civil Transport Problem.on the CRAY Y-MP

Concurrent CPUS	* Connect seconds	= CPU seconds
1	2.650	2.650
2	0.037	0.074
3	0.135	0.404
4	0.792	3.167
5	0.554	2.770
6	1.649	9.894
7	0.212	1.482
8	0.332	2.652
3.63	6.3587	23.0925

The purpose of multitasking is to decrease the wall clock time for a particular computer run. The CPU time will increase due to overhead associated with the parallelization. In the Lanczos algorithm, the matrix factorization step was the calculation that benefited most from the parallelization. Computer runs were made on the CRAY-2 in a dedicated mode on two, three and four processors, respectively. Table 5 shows the wall clock time taken for the factorization for these cases. The actual speedup of 3.2 on four processors represents a considerable decrease in wall clock time for this computational

step. Although not shown in the table, a megaflop rate of 826 was obtained using four processors concurrently in the factorization step.

Table 5. Wall Clock time for Matrix Factorization CRAY-2

Number of Processors Operating Concurrently	Time, secs	Actual Speedup	Theoretical Speedup
1	7.9		
2	4.4	1.8	2
3	3.2	2.5	3
4	2.5	3.2	4

Effects and Timing of Reorthogonalization

Without any reorthogonalization of the Lanczos vectors, repeated and spurious eigenvalues will appear. The HSCT problem is used to demonstrate the loss of orthogonality that occurs when implementing the Lanczos method. The first twelve natural frequencies of the HSCT are shown in the left-hand column of Table 6 with the vectors reorthogonalized at every step. The values in the right-hand column represent the eigenvalues found with no reorthogonalization. Using a value of m equal to 60, 30 eigenvalues were accepted as approximations to the eigenvalues of the system. When no reorthogonalization was performed, the recursion converged to those eigenvalues on the right. The first eigenvalue was repeated 8 times before the second eigenvalue was found. The computation times for the total solution and for total reorthogonalization on the CRAY-2 are shown in the table. The reorthogonalization computation is highly vectorizable and high megaflop rates, up to 227 on one processor on the CRAY Y-MP, were achieved.

Table 6 Effect of reorthogonalization

Eigenvalues found with total Reorthogonalization	Eigenvalues found with no Reorthogonalization
Radians/second	
.02331	.02331
.36548	.02331
.60044	.02331
.70849	.02331
.74632	.02331
.90365	.02331
.91478	.02331
1.0005	.02331
1.1048	.36548
1.1271	.36548
1.3130	.36548
1.3760	.60044

	CONVEX		CRAY-2		CRAY Y-MP	
	Time (sec)	Mflop	Time (sec)	Mflop	Time (sec)	Mflop
Reorthogonalization	8.0	50	1.4	80	0.5	227
Total Solution	728.0		40.0		24.0	

Summary

The Lanczos method is an efficient method for solving the eigenvalue problem and is adaptable to vectorization and parallelization. This method is being incorporated into large finite element codes to solve the vibration and buckling problems where only a few of the natural frequencies and mode shapes are needed. Many adaptations and enhancements to the method as originally proposed are being developed to increase the efficiency and reliability of the eigenvalue and eigenvector approximations. Block Lanczos methods have been developed to overcome the difficulties in determining multiple roots. Many uses for the Lanczos vectors are being discovered and implemented. The Lanczos vectors can be used as basis vectors in reduced basis methods for structural dynamics, flexible body vibration control

and transient thermal problems. Their many uses make it important to have the most efficient and accurate algorithm possible. Ongoing research is aimed at improving the algorithm and applying the vectors in diverse types of problems. The total reorthogonalization used in the example problems executed at a high megaflop rate, but the substitution of more sophisticated reorthogonalization schemes, such as selective or partial reorthogonalization may reduce the overall computation time.

The many vector operations inherent in the Lanczos method exploited the vector capabilities of the Convex and Cray computers. The automatic vectorization of the Convex compiler resulted in a 65 % decrease in computation time for the example panel buckling problem. Further reductions in computation time were achieved using compiler directives and loop unrolling. The computationally-intensive step of factoring the large matrix benefited most from the parallelization on the Cray computers. The speedup in computation time for the factorization of the matrix in the transport example problem was 1.8 on two processors and 3.2 on four processors. Efficient equation solvers are now available on parallel-vector computers, significantly decreasing the computation time. To analyze the large, complex aerospace structures now being designed will require powerful computers and efficient algorithms that can use the computational power to the best advantage. The next generation of parallel computers will most likely incorporate massively parallel processors to perform computationally intensive tasks. This concept will again influence algorithm development.

References

1. J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

2. A. Jennings, *Matrix Computation for Engineers and Scientists*, John Wiley & Sons, Ltd., 1977.

3. B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, 1980.

4. K. J. Bathe and E. L. Wilson, 'Large Eigenvalue Problems in Dynamic Analysis', *ASCE J. Eng. Mech. Div.*, vol. 98, pp. 1471-1485, 1972.

5. P. S. Jensen, 'The Solution of Large Symmetric Eigenproblems by Sectioning', *SIAM, J. Num. Anal.*, vol. 9, pp. 534-545, 1972.

6. S. W. Bostic, 'A Vectorized Lanczos Eigensolver for High-Performance Computers', Proceedings of the AIAA/ASME/ASCE/AHS 31st Structures, Structural Dynamics and Materials Conference, AIAA Paper No. 90-1148, Long Beach, California, April 2-4, 1990, pp.652-662.

7. S.W. Bostic and R. E. Fulton, 'A Lanczos Eigenvalue Method on a Parallel Computer', AIAA Paper 87-0725-CP, Proceedings of the AIAA/ASME/ASCE/AHS 28th Structures, Structural Dynamics and Materials Conference, Monterey, California, April 6-8, 1987, pp.123-135.

8. S. W. Bostic and R. E. Fulton, 'A Concurrent Processing Implementation for Structural Vibration Analysis', AIAA Paper 85-0783-CP, Proceedings of the AIAA/ASME/ASCE/AHS 26th Structures, Structural Dynamics and Materials Conference, Orlando, Florida, April 15-17, 1985, pp.566-572.

9. S. W. Bostic and R. E. Fulton, 'Implementation of the Lanczos Method for Structural Vibration Analysis', AIAA Paper 86-0930-CP, Proceedings of the AIAA/ASME/ASCE/AHS 27th Structures, Structural Dynamics and Materials Conference, San Antonio, Texas, May 19-21, 1986, pp.400-410.

10. M. T. Jones and M. L. Patrick, 'The use of Lanczos's Method to Solve the Large Generalized Symmetric Definite Eigenvalue Problem', NASA CR-181914, *ICASE Report 86-69*, September, 1989.

11. O. Storaasli, S. Bostic, M. Patrick, U. Mahajan, S. Ma, 'Three Parallel Computation Methods for Structural Vibration Analysis', *Journal of Guidance, Control, and Dynamics*, Volume 13, Number 3, May-June 1990, pages 555-561.

12. V. K. Gupta, V. K. and J. F. Newell, ' Band Lanczos Vibration Analysis of Aerospace Structures', *Proceedings of the Symposium on Parallel Methods on Large-Scale Structural Analysis and Physics Applications*, Pergamon Press, New York, N.Y., July, 1991.

13. J. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations Vol. 1 Theory*, Birkhauser Boston, Inc., 1985.

14. K. H. Huebner and E. A. Thornton, *The Finite Element Method for Engineers*, John Wiley & Sons, Inc. , 1982.

15. R. R. Craig Jr., Structural Dynamics, *An Introduction to Computer Methods*, John Wiley & Sons, New York. 1981.

16. R. C. Hibbeler, *Engineering Mechanics, Dynamics,* Macmillan Publishing Co., Inc., New York,1983.

17. L. Komzsik, Editor, *Handbook for Numerical Methods. MSC/NASTRAN version 66,* The Macneal-Schwendler Corporation, Los Angeles, California, April, 1990,PP. 4.4-1,4.4-12.

18. C. Lanczos, 'An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators', *J. Res. Natl. Bureau of Standard,* Vol. 45, pp. 255-282, 1950.

19. I. U. Ojalvo and M. Newman, 'Vibration Modes of Large Structures by an Automatic Matrix Reduction Method', *AIAA Journal* , vol. 8, pp. 1236-1239, 1970.

20. C. C. Paige, 'Accuracy and Effectiveness of the Lanczos Algorithm for the Symmetric Ax $=\lambda$ Bx Problem', Rep. STAN-CS-72-270, Stanford University Press, Palo Alto, CA, 1972.

21. B. Parlett, 'The State-of-the-Art in Extracting Eigenvalues and Eigenvectors in Structural Mechanics Problems', Department of Mathematics, University of California, Berkeley, 1986.

22. D. S. Scott, 'The Advantages of Inverted Operators in Rayleigh-Ritz Approximations', *SIAM J.Sci, Stat. Comput.*, vol. 3, No. 1, March, 1982, pp. 68-75.

23. H. D. Simon, 'The Lanczos Algorithm for Solving Symmetric Linear Systems', Center for Pure and Applied Mathematics, University of California at Berkeley, distributed by Defense Technical Information Center, Alexandria, Virginia, February, 1984.

24. B. Nour-Omid and R. W. Clough, 'Dynamic Analysis of Structures using Lanczos Co-ordinates', *Earthquake Engineering and Structural Dynamics*, Vol. 12, 1984. pp. 565-577.

25 R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger Ltd, Bristol, Great Britain, 1981, pp. 27-29.

26. E. L. Poole and A. L. Overman, 'Parallel Variable-Band Choleski Solvers for Computational Structural Analysis Applications on Vector Multiprocessor Supercomputers', *Proceedings of the Symposium on Parallel Methods on Large-Scale Structural Analysis and Physics Applications*, Pergamon Press, New York, N.Y., July, 1991.

27. E. L. Poole, 'Comparing Direct and Iterative Equation Solvers in a Large Structural Analysis Software System',*Computing Systems in Engineering,* Pergamon Press, Oxford, England, to be published in 1991.

28. A. George and J. W-H Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

29. N. F. Knight and J. W. Stroud, 'Computational Structural Mechanics: A New Activity at the NASA Langley Research Center', NASA TM 87612, Sept., 1985.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA TM-104108	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle: Lanczos Eigensolution Method for High-Performance Computers		5. Report Date: September 1991
		6. Performing Organization Code
7. Author(s): Susan W. Bostic		8. Performing Organization Report No.
		10. Work Unit No. 505-63-53
9. Performing Organization Name and Address: NASA Langley Research Center, Hampton, VA 23665-5225		11. Contract or Grant No.
		13. Type of Report and Period Covered: Technical Memorandum
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, DC 20546-0001		14. Sponsoring Agency Code

15. Supplementary Notes

This paper is to appear as a chapter in the book: Solving Large Scale Problems in Mechanics, edited by Prof. M. Papadrakakis of the National Technical University of Athens, and published by J. Wiley & Sons, Chichester, England.

16. Abstract

The paper presents the theory, computational analysis and applications of a Lanczos algorithm on high-performance computers. The computationally-intensive steps of the algorithm are identified as: the matrix factorization, the forward/backward equation solution and the matrix-vector multiples. These computational steps are optimized to exploit the vector and parallel capabilities of high-performance computers. The savings in computational time from applying optimization techniques such as: variable-band and sparse data storage anc access, loop-unrolling, use of local memory and compiler directives are presented. Two large-scale structural analysis applications are described: the buckling of a composite, blade-stiffened panel with a cutout, and the vibration analysis of a high speed civil transport. The sequential computational time for the panel problem executed on a CONVEX computer of 181.6 seconds was decreased to 14.1 seconds with the optimized vector algorithm. The best computational time of 23 seconds for the transport problem with 17,000 degrees of freedom was on the CRAY-YMP using an average of 3.63 processors.

17. Key Words (Suggested by Author(s)): Eigenvalues, Eigenvectors, Lanczos Method, High-Performance, Buckling, Vibration Analysis		18. Distribution Statement: Unclassified-Unlimited; Subject Category 39	
19. Security Classif. (of this report): Unclassified	20. Security Classif. (of this page): Unclassified	21. No. of pages: 30	22. Price: A03

NASA FORM 1626 OCT 86