# Parallel-Vector Solution of Large-Scale Structural Analysis Problems on Supercomputers

Olaf O. Storaasli*
*NASA Langley Research Center, Hampton, Virginia 23665*
and
Duc T. Nguyen† and Tarun K. Agarwal‡
*Old Dominion University, Norfolk, Virginia 23529*

**A new linear equation solution method, which exploits both the parallel and vector capabilities of modern supercomputers, is presented. This direct method is based on the Choleski factorization procedure and uses a skyline storage scheme. Unique features of the method include its parallel computation at the outermost DO-loop and vector computation at the innermost DO-loop. A novel way to use the "loop unrolling" technique is introduced to improve the performance of this dot-product-based method. This novel approach is in contrast to the traditional loop unrolling approach that is suitable for those Choleski methods that use saxpy rather than dot product operations. Several small- to large-scale structural analysis problems are solved on various parallel-vector high-performance computers to show the effectiveness of the new method.**

## I. Introduction

THE solution of linear systems of equations on advanced parallel and/or vector computers is an important area of ongoing research.[1-5] The development of efficient equation solvers is particularly important for static and dynamic structural analyses, eigenvalue and buckling analyses, sensitivity analysis, and structural optimization procedures.[6-12] Current research has been directed towards developing either effective vector methods or parallel methods to solve linear systems of equations.[10-14] However, modern supercomputers now have both parallel and vector capabilities. Since the total speed-up of an algorithm is a product of the vector speed-up times the parallel speed-up, algorithms that exploit both capabilities are the most desirable.

This paper presents a direct Choleski-based equation solver that exploits both parallel and vector features of supercomputers. The objective of this paper is to describe this new equation solver and evaluate its performance by solving structural analysis problems on three high-performance computers.

Section II discusses two Choleski-based linear equation solution methods: a row-oriented method and a new column-oriented parallel-vector method. Section III discusses the parallel FORTRAN language, Force,[15] used to implement this method. Section IV presents results obtained for small- to large-scale structural analysis problems to evaluate the performance of the method.

## II. Choleski-Based Solution Strategies

The key to reducing the computation time for structural analysis is to reduce the time to solve the resulting linear system of equations. On sequential computers, direct methods based on Choleski factorization[16] are both accurate and fast in solving a wide range of structural analysis problems. These methods are used in most commercial finite element codes. Choleski-based methods have also been found to be accurate and fast in solving structural analysis problems on parallel computers.[9]

### Basic Choleski Decomposition of $[K]$

In Choleski-based methods, a symmetric positive definite stiffness matrix $[K]$ can be decomposed as

$$[K] = [U]^T[U] \tag{1}$$

where $U = L^T$ is an upper triangular matrix. The $U_{ij}$ terms can be computed by expressing the product in Eq. (1) as

$$U_{ij} = 0 \text{ for } i > j \tag{2}$$

$$U_{11} = \sqrt{K_{11}}; \qquad U_{1j} = K_{1j}/U_{11} \tag{3}$$

$$U_{ii} = (K_{ii} - \sum_{k=1}^{i-1} U_{ki}^2)^{1/2} \text{ for } i > 1 \tag{4}$$

$$U_{ij} = (K_{ij} - \sum_{k=1}^{i-1} U_{ki}U_{kj})/U_{ii} \text{ for } i,j > 1 \tag{5}$$

Having obtained the upper triangular matrix $[U]$ from the Choleski decomposition phase, the solution for the system of simultaneous equations is found by the forward and backward substitution phases.[1,5] The row- and column-oriented Choleski methods to be described in subsequent sections refer to the storage scheme used during the computation of $[U]$.

### Row-Oriented Choleski Method

A basic Choleski factorization, which can be used to compute $[U]$ by rows, is summarized in Refs. 1 and 5. The major computations for the row-oriented Choleski method are vector saxpy operations[5] in which additions and multiplications can often be performed simultaneously on some high-performance computers. The performance of this basic Choleski factorization strategy can be improved by using the so-called "loop-un-

Received March 20, 1989; presented as Paper 89-1259 at the AIAA/ASME/ASCE/AHS 30th Structures, Structural Dynamics and Materials Conference, Mobile, AL, April 3-5, 1989; revision received Sept. 26, 1989.

*Aerospace Engineer, Computational Mechanics Branch. Associate Fellow AIAA.

†Associate Professor, Department of Civil Engineering.

‡Graduate Student.

rolling'' technique.[5] Using this technique, the total number of load/store instructions and operations between the main memory and vector registers can be reduced significantly for nested DO-loops. The modified outer loop has an increment equal to the level of unrolling; the innermost loop contains more arithmetic computations in a single arithmetic statement than the basic methods. For vector supercomputers, such as Cray 2, the use of a variable bandwidth storage scheme can further enhance the performance of the resulting vectorized code.[14] Such enhanced versions of the basic Choleski factorization can be quite effective on vector computers, since they exploit the powerful loop-unrolling technique.

In a row-oriented variable bandwidth Choleski approach, the (variable) bandwidth of each row can be defined as the distance between the diagonal term and the last nonzero term of the corresponding row in $[U]$. This method requires more zero terms to be stored than the well-known skyline storage scheme.[16] Furthermore, storing more unnecessary zero terms of the stiffness matrix increases the number of operations required to solve any given structural problem. These ''extra operations'' can be inconsequential or significant, depending on the nature of the problem and node reordering scheme used. Examples of these two storage schemes (variable bandwidth and skyline) for the upper half of the stiffness matrix of a typical structural analysis panel problem are illustrated in Figs. 1 and 2. The row-oriented variable bandwidth storage, shown in Fig. 1, takes considerably more storage than the skyline storage scheme shown in Fig. 2. The number of the vertical gaps in Fig. 2 indicates the storage saved by using the skyline method.

## New Column-Oriented Skyline Choleski Method

Column-oriented methods for the decomposition of $[K]$ into $[U]^T[U]$, compute $[U]$ one column at a time. The major computations in column-oriented Choleski algorithms occur in the dot product operations of Eq. (5). Each element of $[U]$ is computed by using dot products of each column of $[U]$. These dot product operations involve two columns of the decomposed upper triangular matrix $[K]$.

Vector computers, such as the Cray, provide maximum vector speed to access data that are stored contiguously. ''Stride'' in vector computer terminology is the distance between two adjacent elements of a vector involved in the computations. The elements of $[U]$ are stored in column form, in order that

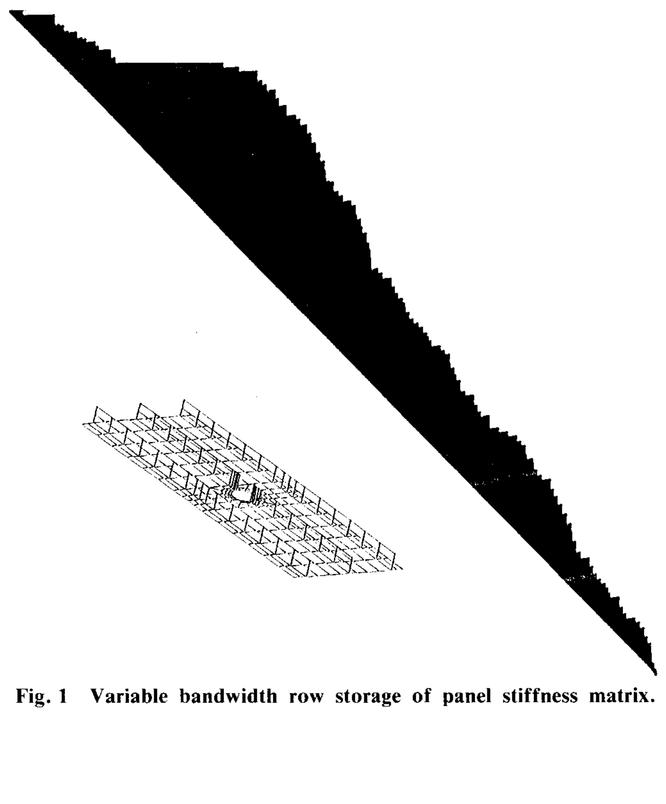
**Fig. 1 Variable bandwidth row storage of panel stiffness matrix.**

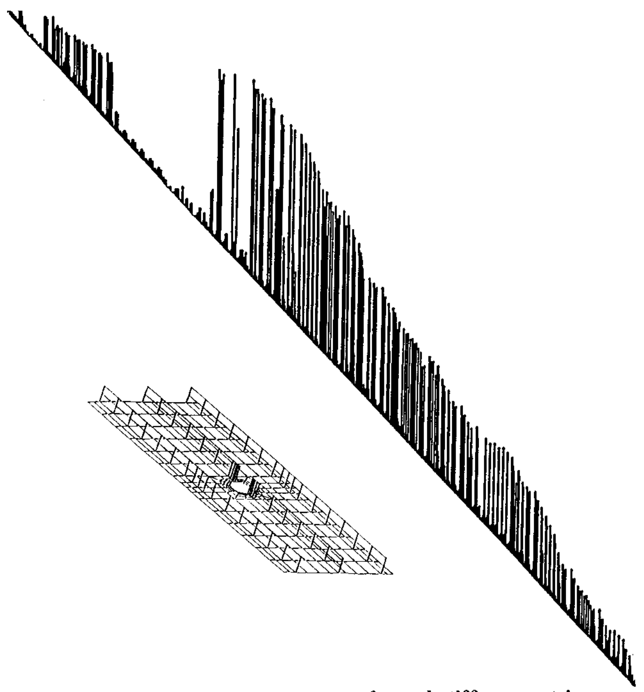
**Fig. 2 Skyline column storage of panel stiffness matrix.**

**Table 1 Basic Choleski decomposition algorithm**

```
      DO 10 I_COL = 1, NEQ
      M_INIT = row = no. of 1st element of I_COL
      M_COL = I_COL-1
      DO 20 J_COL = M_INIT,M_COL
      MAX_ROW = max(row_no. of I element
        of I_COL and J_COL
      DO 30 K = MAX_ROW, J_COL
        SUM-DOT(I_COL,J_COL)
30    Continue
      Modify J_COL th element of I_COL
20    Continue
      Modify diagonal element of I_COL
10    Continue
```

they will be in contiguous storage locations. The column-oriented skyline storage scheme offers stride 1 storage and, hence, the optimum memory retrieval speed. The fast local memory of the Cray 2, which can improve the computation rate by approximately 50%,[14] is not available for parallel computation.

The column-oriented skyline Choleski method was implemented in a computer code to exploit both parallel and vector capability of supercomputers. In this code, the elements of $[U]$ overwrite the elements of $[K]$ as the decomposition takes place to reduce memory use. The columns of $[K]$ are stored one after the other in a single vector array, and elements of each column are arranged from the diagonal up. This storage arrangement is referred to as skyline storage[16] and is illustrated in Fig. 3. Column 5 in Fig. 3, for example, represents the terms $K_{55}$, $K_{54}$, $K_{53}$, and $K_{52}$ of the stiffness matrix $[K]$. In a vector skyline storage scheme, the corresponding terms of $K_{55}$, $K_{54}$, $K_{53}$, and $K_{52}$ are stored in a one-dimensional array $V$ at the locations $V(10)$, $V(11)$, $V(12)$, and $V(13)$, respectively.

The column-oriented skyline Choleski algorithm results in three nested DO-loops, as shown in Table 1 for the factorization phase. On a vector computer, such as the Cray 2, the innermost loop is vectorized by the compiler automatically. This traditional algorithm has a single dot product in the innermost loop, as shown in Table 1. However, Table 2 shows four dot products in the innermost loop (loop 30). This novel use of loop unrolling technique is hereafter referred to as ''vector unrolling.'' In Table 2, the DO 30 loop computes four elements of $[U]$ contained in two columns of $[U]$. This allows

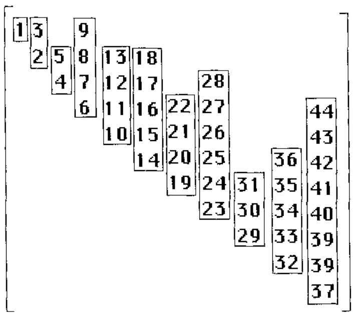

**Fig. 3 Skyline storage scheme for stiffness matrix before and after decomposition.**

four elements to be computed with only one-half the memory references made by the standard algorithm. This is accomplished by unrolling, to the second level, the DO 10 and DO 20 loops in pairs. This vector unrolling strategy allows the elements of a vector brought to vector registers to be used in more than one arithmetic statement in the innermost loop (loop 30). This new vector unrolling technique is different from the traditional loop unrolling technique used in row-oriented Choleski schemes to enhance the performance of the Cray 2.

Coarse-grained parallelism, often referred to as macro-level parallelism, can be accomplished at the outermost loop for the column-oriented Choleski algorithm. Table 2 shows parallelization at DO loop 10, which assigns different columns of $[K]$ to different processors. This strategy allows the algorithm to achieve a high level of computational speed-up. This strategy requires a synchronization (check) statement to assure that a processor accesses the columns of $[K]$ only after the processor assigned to that column has completed its computations. Statement 100 executes the checking of work by other processors and holds the processor until the required work is completed by the other processors.

For some parallel computers, where the time for synchronization is small relative to computation time, the synchronization statement is executed just prior to each dot product in the DO 20 loop. However, on parallel-vector supercomputers, such as the Cray 2, that have high computational speeds, the synchronization time relative to computation time is significant.[17] This synchronization overhead time is reduced by segmenting the loop into two nearly equal sections (see the integer variable M-ROW before statement 100 in Table 2) and performing the synchronization only twice for each column.

**Table 2 New parallel decomposition algorithm**

```
Modify Columns
     1 if odd number of equations
     (1 & 2 if even number of equations)
Parallel DO 10 I_COL = 2(or 3), NEQ, 2
      M_INIT = row_no. of 1st element of I_COL
      M_ROW = row_no. of mid element of I_COL
100   Check/Proceed if column M_ROW complete
      DO 20 J_COL = M_INIT, M_COL,2
      MAX_ROW = max (row no. of 1st element of
        I_COL and J_COL)
      DO 30 (compute 4 dot products of length
        J_COL-MAX_ROW)
          SUM1 = DOT(I_COL,J_COL)
          SUM2 = DOT((I + 1)_COL,J_COL)
          SUM3 = DOT(I_COL,(J + 1)_COL)
          SUM4 = DOT(I + 1)_COL,(J + 1)_COL)
30    CONTINUE
      Modify J_COL elements of I_COL,
        (I + 1)_COL
      Modify (J + 1)_COL elements of
        I_COL, I + 1_COL
20    Continue
      If (M_COL.ne.(I_COL-1))
        M_INIT = M_COL + 1
        Set M_COL = row(I_COL-1)th element and
        GO TO 100 to modify lower half of elements
      Endif
      Modify Diagonal elements of I_COL,
        (I + 1)_COL
      Declare I_COL and (I + 1)_COL complete
10    End Parallel DO
```

**Table 3 Code for stiffness and load generation**

```
      MAXA(1) = 1
      A(1) = 2.
      DO 51 I = 2, NEQ
      COLHT = MIN(I-1, HALFBW)
      MAXA(I) = MAXA(I-1) + COLHT
      A(MAXA(I)) = 2.
51    B(I) = 2.
      IEND = MIN(I + HALFBW,NEQ)
      DO 52 I = 1,NEQ
      DO 52 J = I + 1, IEND
      LOCATE = MAXA(J) + J-1
      A(LOCATE) = 1.0/(I + J)
      B(I) = B(I) + A(LOCATE)
      B(J) = B(J) + A(LOCATE)
52    CONTINUE
```

### Solution of Triangular Systems

The forward elimination and backward substitution phases can be made parallel in the first loop, which requires synchronization statements. This parallel implementation was tested, and resulted in excellent computation speed-up for an increasing number of processors, but it suffered from the added time for synchronization on Cray computers. Because of this synchronization overhead time, the forward-backward solution phases were found to be faster on one Cray processor without parallel constructs than on multiple processors. Further time reduction for one processor was also obtained by using second-level vector unrolling in the forward elimination and second-level loop-unrolling in the backward substitution.

## III. Force: A Portable Parallel FORTRAN

Force is a machine-independent tool for parallel programming and, with certain exceptions noted in the "Force User's Manual,"[15] it includes all FORTRAN constructs and compiler options. It is a preprocessor that produces executable parallel code from FORTRAN augmented with several simple parallel extensions. These parallel extensions include such constructs as Pre- and Self-scheduled DO-loops (for parallel computations), Barriers, Produce and Consume (for synchronization). Force permits users to write efficient, yet portable, parallel code without referring to the many details of multitasking or parallel programming found in vendor manuals. Thus, engineers and numerical analysts can concentrate on developing effective parallel algorithms or solution strategies for different engineering and/or scientific applications. Programs written in Force are easily ported to and run on other parallel computers on which Force is installed. In this paper, Force is used to develop and implement the parallel FORTRAN code on high-performance computers, such as the Convex, Cray 2, and Cray Y-MP.

## IV. Evaluation of Method on Example Problems

To test the effectiveness of the parallel-vector skyline Choleski solver, several structural analysis problems were solved on various high-performance computers. In the solution of the following example problems, code was inserted to measure the time spent by each processor during the equation solution. The Cray timing function, tsecnd, was used to measure the time for equation solution taken by each Force processor. In addition, for each problem, the number of million floating point operations, MFLOP, was calculated and then divided by the solution time, in seconds, to determine the overall performance rate of the solver in MFLOPS.

### 10,000 Degree-of-Freedom, 800 Bandwidth Test Problem

The newest, fastest parallel-vector supercomputer available for testing the column-oriented skyline Choleski solver is currently the Cray Y-MP. Although it is fast and has eight processors, the Cray Y-MP used has 32 million words of main memory, of which only eight million are available for a single job without special permission. This eight million word restriction limited the size of the largest problem that could be solved since the current implementation of the equation solver described uses only main memory to store $[K]$. Thus, the largest problem possible to solve on all three high-performance computers was designed to have a stiffness coefficient matrix with 10,000 degrees of freedom (DOF) and 800 bandwidth. In this test problem, the coefficient matrix $A$ (stored in a vector skyline form) and the load vector $B$ are generated according to Table 3. In this table, NEQ represents the number of equations (DOF); an integer array MAXA is used to store the location of the diagonal terms of the stiffness matrix. This test problem is useful for debugging and can also serve to evaluate the performance of this and other methods on any shared memory parallel-vector computer.

The total processing time (for both the decomposition and the forward-backward solution phases) for the test problem on the Cray 2 using 1, 2, 3, and 4 processors is shown in Fig. 4. The figure indicates a computation speedup of 3.9 (74 s divided by 19 s), at a rate of 328 MFLOPS on four processors. The major computation time was spent in the decomposition phase, whereas very little time was spent in the forward-backward solution phase, as indicated in Fig. 4.

The total computation time for this test problem on the Convex-220, Cray 2, and Cray Y-MP computers is shown in Fig. 5. From this figure, one can see the relative performance of the skyline Choleski solution method on a single processor for the three computers. Here again the forward-backward solution phase represents a very small fraction of the total solution time. Figure 5 also indicates that the Cray Y-MP processors run almost twice as fast as the Cray 2 processors, which in turn run about thre times as fast as the Convex-220 processors. By using Force (see Sec. III), the same code was implemented on the three computers. The code was then modified on the Cray Y-MP and was found to achieve performance rates of up to 2.025 GigaFLOPS for matrix decomposition when using eight processors.[18]

To measure the overhead of Force, the same code, with all Force statements removed, was implemented on the Cray 2 and Cray Y-MP. The results obtained using Force on the Cray 2 and Cray Y-MP (denoted Force in Fig. 5) showed no significant overhead in computation time.

### Three-Dimensional Cube Problem

Solution algorithms often display different characteristics for different classes of problems. To investigate the behavior

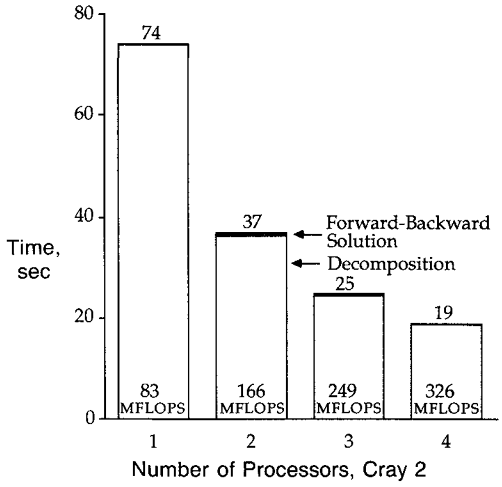

**Fig. 4 Computation time reduction for test problem with 10,000 equations and 800 bandwidth.**

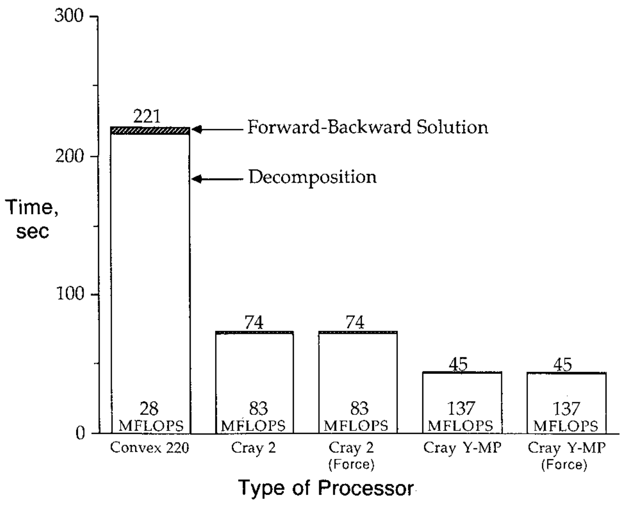

**Fig. 5 Time comparison for one processor with 10,000 equations and 800 bandwidth (*F* denotes Force used)**

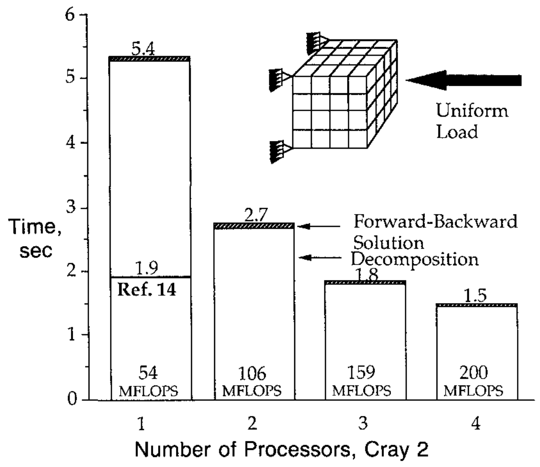

**Fig. 6 Computation time reduction for 3000 equation cube.**

of the solver on a three-dimensional problem, a cube-shaped isotropic solid undergoing compression was solved. This 3000 degree-of-freedom problem has a maximum bandwidth of 336 and an average bandwidth of 313. The 1000 node cube (10 nodes along each axis) was constrained at the corner nodes and contained 729 eight-node solid elements (nine along each axis). The computation times for multiple processors are shown in Fig. 6. The computation time reduces in direct proportion to the number of processors, with a speed-up of 3.6 for four processors. A similar proportional computation time reduction was also found, using a revised solution strategy on an eight-processor Cray Y-MP.[18] The revised solution time is similar to that for a vectorized code on one processor, also shown in Fig. 6.[14]

### Space Shuttle Solid Rocket Booster (SRB) Problem

To evaluate the performance of the skyline Choleski solver on a large-scale static structural analysis problem, a two-dimensional shell model of the Space Shuttle solid rocket booster, shown in the upper right of Fig. 7, was solved. This SRB model was used to investigate the overall deflection distribution for the SRB when subjected to mechanical loads corresponding to selected times during the launch sequence.[19] The model contains 9205 nodes, 9156 four-node quadrilateral shell elements, 1273 two-node beam elements, and 90 three-node triangular elements, with a total of 54,870 DOF. This problem has a maximum bandwidth of 894 and an average bandwidth of 381. A detailed description and analysis of this problem is given in Refs. 19 and 20.

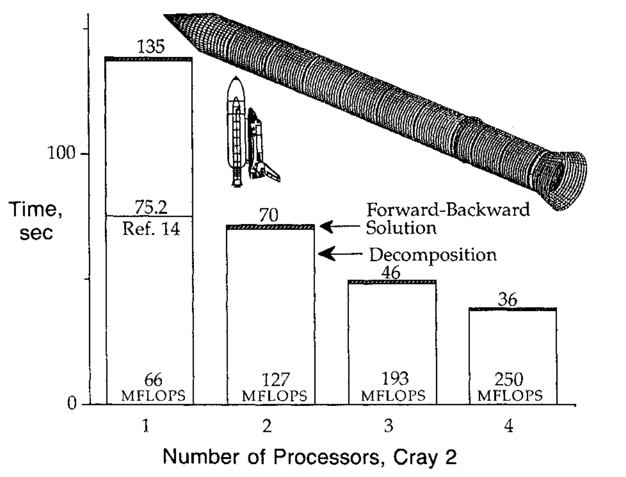

**Fig. 7 Computation time reduction for Space Shuttle solid rocket booster with 54,870 equations.**

The times taken for a typical finite element code to generate the mesh, form the stiffness matrix, and factor the matrix are 344.9, 215.2, and 821.3 s, respectively, on a Cray 2, and 2209.2, 2097.3, and 51,480.7 s, respectively, on a VAX 11/785.[19] At the time the new parallel-vector skyline Choleski method was developed, the fastest time reported in the literature to factor this SRB matrix was 75.2 s on one Cray 2 processor with a vectorized variable band solver.[14] For this same problem, the new skyline Choleski solver took 135 and 36 s on one and four processors, respectively, on the Cray 2. This is a reduction in matrix decomposition time by a factor of 3.75 on four processors. The time to factor and solve the SRB matrix using eight processors on a Cray Y-MP is 6 s elapsed time using a revised solution strategy.[18]

The computation rate (i.e., MFLOPS), shown in Figs. 4–7, is best for problems with a large average bandwidth (i.e., Fig. 4). The skyline Choleski solver operates on vectors ranging in length from one to the column height of each column of the upper triangular matrix $[U]$. For problems with a small average column height, the major portion of the computation is performed on short vectors, which results in a low MFLOPS rate. In large structural analysis problems with large average column height, the majority of the vector operations are performed on long vectors, which results in higher MFLOPS rates.

## V. Concluding Remarks

A portable and efficient skyline Choleski method for the solution of large-scale structural analysis problems has been developed and tested on three high-performance computers—Convex-220, Cray 2, and Cray Y-MP. Although existing equation solvers have been tailored to either vector or parallel computers, the newly developed equation solver exploits both the parallel and vector capabilities of modern high-performance computers. A unique feature of this method is the strategy used to minimize computation time by performing parallel computation at the outermost DO-loop of the decomposition phase, the most time-consuming part of the total equation solution time. Most, if not all, existing solution methods introduce parallelism only at the second DO-loop of the decomposition phase. In addition, the most intensive computation of the decomposition phase was vectorized at the innermost DO-loop. The dot-product-based factorization scheme prohibits the traditional use of the well-known loop-unrolling technique used for saxpy operations. To overcome this difficulty, a novel use of loop unrolling, termed "vector unrolling," has been introduced in the column-oriented Choleski algorithm to reduce computation time. For the forward and backward solution phases, it was found to be more effective to perform vector unrolling and loop unrolling, respectively, using vector, rather than parallel, code.

The new method was coded in a modular and portable fashion using a generic parallel FORTRAN, called Force. The generality and portability of the method should make the use of it attractive for other engineering and scientific applications.

The newly-developed parallel-vectorized Choleski method has been applied to the solution of several small- to large-scale structural analysis problems. For all problems, the total equation solution time was reduced significantly, in direct proportion to the number of processors used. Factoring the stiffness matrix for the Space Shuttle solid rocket booster, which typically takes hours on most computers and minutes on Cray computers, was reduced to seconds using the parallel-vector Skyline Choleski method. This new method is not only attractive for current supercomputers, but also for future supercomputers, where the number of processors is expected to increase far beyond that of today's supercomputers.

## Acknowledgments

The second and third authors acknowledge the support provided by NASA Grant NAG-1-858.

## References

[1]Dongarra, J. J., Gustafson, F. G., and Karp, A., "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine," *SIAM Review*, Vol. 26, No. 1, Jan. 1984, pp.91–112.

[2]Chen, S., Dongarra, J., and Hsiung, C., "Multiprocessing Linear Algebra Algorithms on the Cray XMP-2: Experiences With Small Granularity," *Journal of Parallel Distributed Computing*, Vol. 1, No. 1, 1984, pp. 22–31.

[3]Farhat, C., Wilson, E., and Powell, G., "Solution of Finite Element Systems on Concurrent Processing Computers," *Engineering with Computers*, Vol. 3, 1987, pp. 157–165.

[4]Utku, S., Salama, M., and Melosh, R., "Concurrent Factorization of Positve Definite Banded Hermitian Matrices," *International Journal of Numerical Methods in Engineering*, Vol. 23, No. 11, 1986, pp. 2137–2152.

[5]Ortega, J. M., *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, NJ, 1988.

[6]Nguyen, D. T., and Niu, K. T., "A Parallel Algorithm for Structural Sensitivity Analysis on the FLEX/32 Multicomputer," *Proceedings of the 6th ASCE Structures Congress*, Aug. 1987.

[7]Nguyen, D. T., Shim, J. S., and Zhang, Y., "The Component Mode Method In a Parallel Computer Environment," *Proceedings of the 29th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference*, April 1988, AIAA Paper 88-2438.

[8]Storaasli, O. O., Bostic, S. W., Patrick, M., Mahajan, U., and Ma, S., "Three Parallel Computation Methods for Structural Vibration Analysis," *Proceedings of the 29th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference*, April 1988, pp. 1401–1411, AIAA Paper 88-2391.

[9]Storaasli, O. O., Poole, E. L., Ortega, J. M., Cleary, A., and Vaughan, C., "Solution of Structural Analysis Problems on a Parallel Computer," *Proceedings of the AIAA/ASME/ASCE/AHS 29th Structures, Structural Dynamics and Materials Conference*, April 1988, pp. 596–605, AIAA Paper 88-2287.

[10]Farhat, C., and Wilson, E. L., "A Parallel Active Column Equation Solver," *Computers and Structures*, Vol. 28, No. 2, 1988, pp. 289–304.

[11]Law, K., "A Parallel Finite Element Solution Method," *Computers and Structures*, Vol. 23, No. 6, 1986, pp. 845–858.

[12]Storaasli, O. O., and Bergan, P. G., "Nonlinear Substructuring Method for Concurrent Processing Computers," *AIAA Journal*, Vol. 25, June 1987, pp. 871–876.

[13]Ashcraft, C. C., Grimes, R. G., Lewis, J. G., Peyton, B. W., and Simon, H. D., "Progress in Sparse Matrix Methods for Large Linear Systems on Vector Supercomputers," *The International Journal of Supercomputer Applications*, Vol. 1, No. 4, Winter 1987, pp. 10–30.

[14]Poole, E. L., and Overman, A. L., "The Solution of Linear Systems of Equations with a Structural Analysis Code on the NAS Cray 2," NASA CR-4159, Dec. 1988.

[15]Jordan, H. F., Benten, M. S., Arenstorf, N. S., and Ramann, A. V., "Force User's Manual," Dept. of Electrical and Computer Engineering, Pub. 88-2-4R, Univ. of Colorado, Boulder, CO, July 1988.

[16]Bathe, K. J., *Finite Element Procedures in Engineering Analysis,* Prentice Hall, NY, 1982.

[17]Benten, M., Farhat, C., and Jordan, H., "The Force for Efficient Multitasking on the Cray Series of Supermultiprocessors," *Proceedings of the Fourth International Symposium: Science and Engineering on Cray Supercomputers,* Oct. 1988, pp. 389–406.

[18]Storaasli, O., Nguyen, D., and Agarwal, T., "Force on the Cray Y-MP," Numerical Aerodynamic Simulation Program Newsletter of NASA Ames Research Center, Vol. 4, No. 7, July 1989, pp. 1–4.

[19]Knight, N. F., McCleary, S. L., Macy, S. C., and Aminpour, M. A., "Large Scale Structural Analysis: The Structural Analyst, The CSM Testbed, and The NAS System," NASA TM-100643, March 1989.

[20]Knight, N. F., Gillian, R. E., and Nemeth, M. P., "Preliminary 2-D Shell Analysis of the Space Shuttle Solid Rocket Boosters," NASA TM-100515, 1987.