AIAA JOURNAL
Vol. 43, No. 3, March 2005

High-Performance Domainwise Parallel Direct Solver for Large-Scale Structural Analysis

Jeong Ho Kim*
Korean Institute of Science and Technology Information, Taejon 305-333, Republic of Korea
and
Chang Sung Lee† and Seung Jo Kim‡
Seoul National University, Seoul 151-742, Republic of Korea

Large-scale structural analysis using a finite element method requires a high-performance and efficient parallel algorithm that is scalable in terms of both performance and storage. Most of the research for large-scale parallel structural analysis has focused on iterative solution methods because they are much easier to parallelize. In contrast, direct solution methods have generally been considered in adequate for large-scale finite element computations because of many difficulties and disadvantages for carrying out large-scale problems. However, direct solution methods are still generally preferred due to their ease of use and the numerical robustness that guarantees the solution to be obtained within an estimated time without failure. Therefore, for the general application of large-scale structural analysis to a wide range of problems, an efficient parallel direct solution method that has scalability comparable to that of iterative methods is proposed. A new parallel direct solver for large-scale finite element analysis, the domainwise multifrontal solver, is proposed by realizing domainwise parallelism for a direct solver. By the use of the proposed solver with our own structural analysis code, good scalability can be shown as a direct method and can solve the largest problem ever known to be solved by direct solvers.

Introduction

As structural analysis using a finite element method is used more and more extensively in the process of analysis and design of complex or large-scale structures such as aerospace vehicles, requirements for computational resources to perform the structural analysis are increasing rapidly. Therefore, we need high-performance structural analysis software techniques, as well as high-performance hardware systems, to satisfy these computational requirements. Thus, to meet the requirements, high-performance structural analysis software techniques should be developed that can perform high-performance scalable parallel computations by efficiently utilizing distributed computational resources such as parallel supercomputing systems or personal computer cluster systems. To develop high-performance structural analysis software, an efficient parallel algorithm is required that should be implemented in such a way that we can elicit maximum performance out of given computing systems. In this work, an efficient parallel algorithm and its efficient implementation for high-performance structural analysis software is proposed.

Structural analysis using a finite element method can be performed implicitly or explicitly. By comparison, implicit finite element analysis procedures have some significant issues and difficulties regarding parallelization because systems of linear equations should be solved in the case of implicit finite element analysis procedures. To perform implicit finite element analysis, small element stiffness matrices are assembled to build a global stiffness matrix that has a sparse nonzero pattern, and the matrix equation is solved to obtain the solution of the problem. This procedure may be repeated depending on the characteristics of the given problem. Throughout the whole finite element analysis procedure, the equation solution step is the most time consuming. In the case of large-scale implicit finite element analysis, most of the computation time is spent on the equation solution procedure, which determines the overall performance of the finite element code.

To solve the matrix equations arising from implicit finite element analysis, we typically just use a simple Gaussian elimination algorithm for dense matrices, but such an algorithm is too inefficient in terms of computation time and storage requirements because matrices assembled by the finite element procedure are generally very sparse. Therefore, various types of equation solvers that can deal with sparse matrices for finite element analysis have been developed and used. They can be classified into two categories: direct solvers and iterative solvers.

Direct solvers perform triangular factorization of a global stiffness matrix with the non-zero pattern of the matrix considered and then solve the equation by a substitution procedure. Thus, the solution always can be found after the required computational operations are completed unless the rank of the stiffness matrix is deficient. On the other hand, iterative solvers perform matrix–vector or vector–vector computations repeatedly until the solution converges, and so iterative solvers may require too many iterations, or even fail to find the solution, depending on the numerical condition of the stiffness matrix. Iterative solvers may show much better performance than direct solvers for certain classes of problems, but we do not know in advance when or whether we can find the solution with iterative solvers. In the case of direct solvers, total operation counts to solve a given problem can be estimated exactly before the beginning of the computation, and so we can anticipate when we will obtain the solution. The numerical robustness that guarantees that the solution will be obtained within an estimated time is very important. Likewise, it is advantageous and is an essential property of direct solvers used for the general application of finite element analysis techniques to a wide range of problems from the academic and industrial communities. For structural analysis or solid mechanics problems, the numerical robustness of direct solvers is much more critical because these applications have numericaly stiffer systems

Received 28 May 2004; accepted for publication 22 October 2004.

*Senior Researcher, Supercomputing Application Technologies Department, Supercomputing Center, 52 Eoeun-Dong Yusong-Gu; aresto@kisti.re.kr.

†Graduate Research Assistant, Department of Aerospace Engineering, San 56-1, Shillim-Dong, Kwanak-Gu; currently Senior Engineer, Computer and Internet Systems Division, Samsung Electronics, 416 Maetan-3Dong, Yeongtong-Gu, Suwon, Gyeonggi-Do 443-742, Republic of Korea; cs1023.lee@samsung.com.

‡Professor, Department of Aerospace Engineering, San 56-1, Shillim-Dong, Kwanak-Gu; sjkim@snu.ac.kr. Senior Member AIAA.

than these of fluid dynamics problems, which leads to difficulty in using iterative solvers. Thus, most commercial finite element packages for structural analysis or solid mechanics problems use a direct solver as their main equation solver. Direct solvers are also much more efficient than iterative solvers for problems with multiple right-hand-side vectors, such as structural analysis problems with multiple load cases or implicit time integration problems with constant stiffness matrices. The block Lanczos algorithm for the eigenanalysis of structures is a good example of a case that can benefit from direct solvers because it requires solutions for multiple right-hand-side vectors.

However, in spite of all of their strong points, direct solvers have many weaknesses for large-scale problems, especially for parallel processing of the finite element analysis procedure for very large-scale problems. Direct solvers generally have much larger storage requirements than iterative solvers, and the operation counts of a direct solver increase faster with the growing problem size than do those of an iterative solver. Furthermore, direct solvers are generally very difficult to parallelize compared with iterative solvers, and the parallelized versions of direct solvers also require much more communication between processors and have poor scalability. To make matters worse, direct solvers generally require an explicit construction of a global stiffness matrix to solve the problem, which results in another difficulty in the parallelization of the whole finite element analysis procedure. Because of the difficulties and the disadvantages of direct solvers just mentioned, iterative solvers have been preferred for large-scale parallel finite element analysis, and most of the research that has been performed so far focuses on iterative methods.[1–4] One of the most successful research codes for large-scale parallel finite element analysis with iterative solvers may be Salinas,[3] incorporating a finite element tearing and interconnection (FETI)[1,2]-style algorithm.

However, in spite of all of the difficulties and disadvantages for large-scale problems, direct methods are still more desirable, especially for structural analysis or solid mechanics problems. Thus, if one could develop a direct solver that can overcome most of the difficulties and the disadvantages encountered for large-scale problems, and could show comparable results in both the performance and the size of solvable problems, it would be considered very significant progress in large-scale parallel finite element analysis techniques.

In this research, we propose the domainwise multifrontal solver as an efficient direct solver for large-scale parallel finite element analysis, and we show the capability and performance of the proposed solver by the use of our own structural analysis code based on the proposed domainwise multifrontal solver.

To implement real and practical applications for large-scale structural analysis by the use of the proposed solver, the characteristics of smart structures, namely, active fiber composites,[5] are investigated through the direct numerical simulation (DNS) approach,[6] which requires an extremely large finite element model and tremendous computations. Vibration analysis of an aerospace launch vehicle was also successfully carried out using the block Lanczos algorithm implemented by the proposed solver.

Domainwise Multifrontal Solver

The concept of the multifrontal method was first introduced by Duff and Reid[7] as a generalization of the frontal method of Irons.[8] The frontal method was originally proposed as a solution procedure for the finite element method and was designed to minimize core storage requirements. The main idea of the frontal solution method is to eliminate the variable while assembling the equations. As soon as the coefficients of an equation are completely assembled from the contributions of all relevant elements to a small dense matrix called the frontal matrix, the corresponding variables can be eliminated. Therefore, the complete global stiffness matrix is never formed as such because after elimination the reduced equation is immediately transferred to secondary storage.

Duff and Reid analyzed the process of the frontal method and extended the concept of an "element" so that it could be applied to a general sparse matrix. Generalized frontal methods for general sparse matrices were further improved to be able to deal with multiple frontal matrices, which results in a significant reduction of total operation counts. This solution procedure is called the multifrontal method. There has been much significant progress in the multifrontal method, and it is now considered one of the most efficient direct solvers for general sparse systems of linear equations. Therefore, it can be used efficiently for implicit finite element analysis. Some of finite element analysis packages such as ABAQUS use the multifrontal method as their main equation solver and show very good performance compared with other packages that do not use the multifrontal algorithm. One of the most successful implementations of multifrontal algorithms is that of Gupta et al.[9] The algorithm is also parallelized for a distributed memory system and has very good parallel performance and scalability for a direct solver.

However, by the generalization process from the original frontal method, the multifrontal method has lost many advantages of the original frontal method for finite element analysis. The multifrontal method requires the construction of the complete global stiffness matrix as do other direct solvers besides the frontal method, which may be a significant demerit for large-scale parallel finite element analysis. In addition, most of the available multifrontal solvers,[10] including that of Gupta et al., do not use the out-of-core technique that is used extensively for the original frontal method to reduce the required core memory size. Thus, the required memory size of the multifrontal solvers may be too large to perform large-scale finite element analysis.

The idea of the domainwise multifrontal method proposed and established by Kim and Kim[11] that extends the original frontal technique to use multiple fronts without the generalization process for general sparse matrices is an excellent alternative to the multifrontal method for finite element analysis. Although both the domainwise multifrontal method and the multifrontal method can be considered to be almost equivalent algebraically, the merits of finite element analysis that the original frontal method has can be maintained for the domainwise multifrontal method by the domainwise approach. The domainwise multifrontal method does not require the global assembly of the completed stiffness matrix and can use the out-of-core technique as easily and seamlessly as the frontal method does.

As far as large-scale parallel finite element analysis is concerned, the advantage of the domainwise approach for the domainwise multifrontal method readily becomes evident. Each frontal matrix of the domainwise multifrontal method can always be mapped to a finite element domain, and so we can achieve domainwise parallelism by decomposing the whole finite element domain into multiple subdomains, whereas the original multifrontal method requires global redistribution of the global stiffness matrix after it is completely assembled from element stiffness matrices. Figure 1 shows the significance of domainwise parallelism by comparing the difference between the parallel finite element analysis procedure of the domainwise multifrontal solver with that of general sparse direct solvers, including the original multifrontal methods.

The domainwise multifrontal method can be explained in terms of structural analysis as follows. By the domainwise multifrontal method, the whole finite element domain is divided recursively into two subdomains until appropriate numbers of elements are included in each domain. Then, fully assembled degrees of freedom (DOF) in each domain are eliminated by the procedure called static condensation, and neighboring domains are merged in the reverse order to form a new domain. Fully assembled DOF in each new domain are eliminated again by static condensation, and then neighboring new domains are merged together. This process is repeated recursively until all of the finite element domains are merged. Figure 2 shows a simple example of a two-dimensional finite element domain. The domainwise multifrontal method can be regarded as the recursive substructuring technique because the elimination of internal or fully assembled DOF of the given domain is called substructuring in terms of structural analysis.

Serial Implementation

Modern cache-based systems can perform dense matrix computations very efficiently, and so we can achieve very high floating-point operations per second (FLOPS) with dense matrix computations.

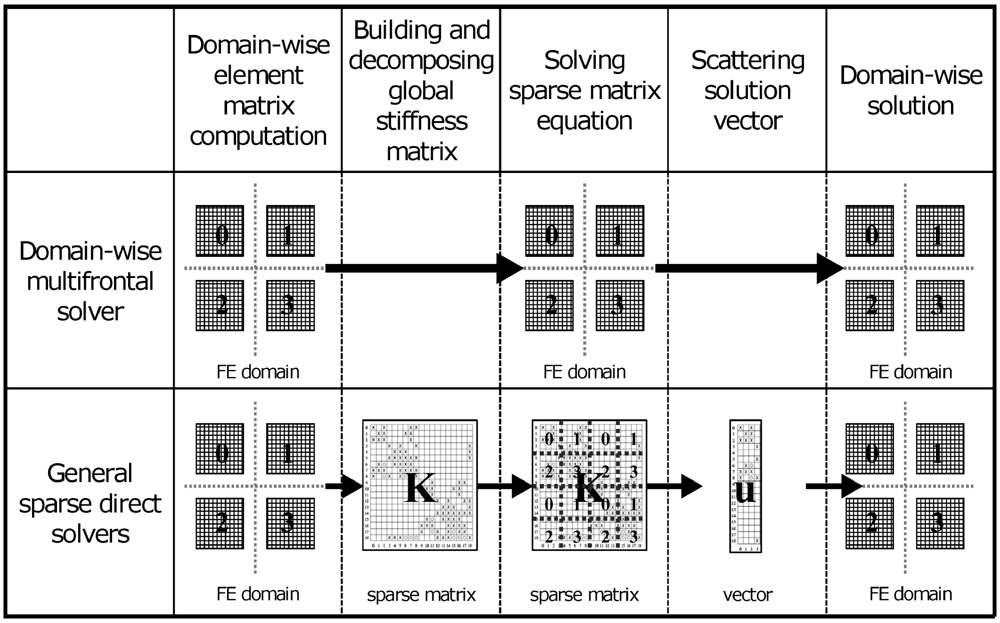

Domain-wise element matrix computation
Building and decomposing global stiffness matrix
Solving sparse matrix equation
Scattering solution vector
Domain-wise solution
Domain-wise multifrontal solver
General sparse direct solvers
FE domain
sparse matrix
vector
K
u

Fig. 1 Comparison of parallel finite element analysis procedure.

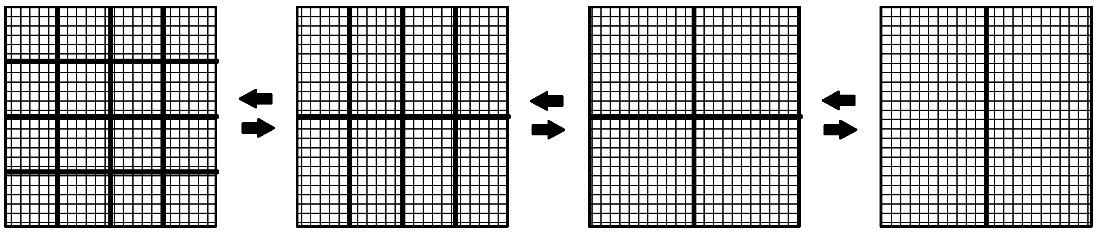


Fig. 2 Recursive substructuring procedure.

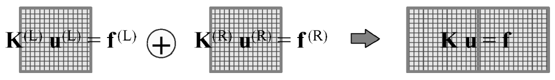


Fig. 3 Domain-merging operation for recursive substructuring.

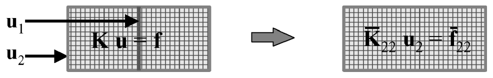


Fig. 4 Static condensation for recursive substructuring.

The domainwise multifrontal method can be organized to perform most of its computations in the form of dense matrix computations, and so we can expect very high FLOPS, as well as reduced operation counts. However, Kim and Kim,[11] who established the domainwise multirontal method, focused on the reduction of operation counts and did not focus on the method's actual performance. In this research, the performance of our domainwise multifrontal solver is greatly enhanced by implementing dense matrix computations using high-performance dense linear algebra libraries that are highly tuned to hardware architecture for optimized performance.

The recursive substructuring procedure, which is the core operation in the domainwise multifrontal method, can be divided into two stages, as shown in Figs. 3 and 4. The first stage is the domain-merging operation used to construct a new frontal matrix equation by merging old frontal matrix equations from two neighboring domains. The second stage is the static condensation or substructuring operation used to eliminate fully assembled or internal DOF from the newly formed frontal matrix equation and to produce a reduced frontal matrix equation. The reduced frontal matrix equation from the static condensation stage becomes one of the input or old frontal matrix equations for a new domain-merging operation.

Because the first stage consists of a simple rearrangement of matrix elements with addition, most of the computing time is spent on the second stage. We label the internal or fully assembled DOF as $\boldsymbol{u}_1$ and the external or surface DOF as $\boldsymbol{u}_2$, as shown in Fig. 4. Then the matrix equation of the new merged domain can be written as follows:

$$\begin{bmatrix} \boldsymbol{K}_{11} & \boldsymbol{K}_{12} \\ \boldsymbol{K}_{21} & \boldsymbol{K}_{22} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{Bmatrix} \tag{1}$$

After a static condensation process, Eq. (1) is transformed as follows:

$$\bar{\boldsymbol{K}}_{22}\boldsymbol{u}_2 = \bar{\boldsymbol{f}}_2 \tag{2}$$

$$\bar{\boldsymbol{K}}_{22} = \left(\boldsymbol{K}_{22} - \boldsymbol{K}_{21}\boldsymbol{K}_{11}^{-1}\boldsymbol{K}_{12}\right) \tag{3}$$

$$\bar{\boldsymbol{f}}_2 = \boldsymbol{f}_2 - \boldsymbol{K}_{21}\boldsymbol{K}_{11}^{-1}\boldsymbol{f}_1 \tag{4}$$

The core operation of the recursive substructuring process is to compute $\bar{\boldsymbol{K}}_{22}$ in Eq. (2), which can be computed by Eq. (3). All of the computations required to obtain $\bar{\boldsymbol{K}}_{22}$ can be implemented using level-3 BLAS[12] and LAPACK,[13] which means that we can use fully optimized library routines to perform these operations so that we can take full advantage of the given hardware performance. Furthermore, because the BLAS and LAPACK subroutine interface is being accepted as the de facto standard, the use of the BLAS and LAPACK interface guarantees high portability with high performance.

After Eq. (2) is constructed by the computation of Eqs. (3) and (4), some of the matrices such as $\boldsymbol{K}_{21}$ and $\boldsymbol{K}_{11}^{-1}$ are not needed again until a substitution procedure is performed. Therefore, they can be sent to the secondary storage hard disk if the available core memory size is not sufficient, which leads to a significant reduction in required core memory size needed to solve a given problem.

To show the efficiency of the current serial implementation of the domainwise multifrontal solver, the serial performance was compared with that of the implementation by Kim and Kim.[11] As shown in Table 1, the present implementation of the domainwise multifrontal solver (DMS) shows much better performance than the implementation by Kim and Kim[11] (labeled DMS[11]), and it can be seen that the performance ratio increases as the problem size grows.

We also evaluated the serial performance of our domainwise multifrontal solver by comparing the performance of our own structural analysis code, which uses the proposed solver, with those of the most popular finite element structural analysis packages such as MSC/NASTRAN and ABAQUS. As shown in Tables 2 and 3, our code (DMS) shows the best performance on both the EV67 and POWER4 systems. Million FLOPS/s (MFLOPS) values in Table 3 were measured using hpmcount.[14] On both systems, optimized BLAS and LAPACK routines developed by the hardware vendors were used.

For Intel-based systems, the most popular implementation of BLAS and LAPACK is ATLAS.[15] However, because the GOTO[16] library shows better performance than ATLAS for a wide range of matrix sizes on an Intel Xeon 2.4-GHz system, as shown in Fig. 5, our domainwise multifrontal solver uses the GOTO library to obtain better performance. By using the BLAS implementation with the best performance, we could achieve performance that is about twice as fast as that of ABAQUS on the Intel system, as shown in Table 4. Our structural analysis code with the proposed solver sustains 2.153 GFLOPS and shows 45% CPU efficiency.

Parallel Implementation

As shown before, to parallelize the whole finite element analysis procedure, including the parallel linear equation solution procedure, efficiently, it is best to partition the finite element domain instead of partitioning the matrix equation algebraically. We term this domainwise parallelism, and it should be implemented to achieve better parallel performance and scalability for large-scale structural analysis problems. Domainwise parallelism can be easily and naturally implemented for iterative solvers based on domain decomposition methods such as FETI, whereas it cannot be implemented for general sparse direct solution methods including the multifrontal method. However, although the domainwise multifrontal solver is a kind of sparse direct solver, by the use of a domainwise approach, the domainwise multifrontal solver can have the same level of domainwise parallelism with iterative solvers based on the domain decomposition method. In other words, the domainwise independent computations used to construct an interface problem for iterative solvers based on the domain decomposition method can be

Table 1 Elapsed time comparison with the previous serial implementation

Mesh	No. of unknowns	Alpha EV67 667 MHz, s		IBM POWER3 375 MHz, s	
		DMS[11]	DMS	DMS[11]	DMS
$20 \times 20 \times 20$	27,783	69	40	88	33
$30 \times 30 \times 30$	89,373	1103	292	1065	226
$38 \times 38 \times 38$	177,957	5215	1058	4263	804

Table 2 Performance comparison on Alpha EV67 667-MHz system[a]

Mesh	No. of unknowns	CPU time, s		
		DMS	ABAQUS	NASTRAN
$32 \times 32 \times 32$	107,811	305	331	1,345
$48 \times 48 \times 48$	352,947	2909	3150	18,885

[a] $R_{\text{peak}} = 1.3$ GFLOPS.

Table 3 Performance comparison on IBM POWER4 1.3-GHz system[a]

Mesh	No. of unknowns	CPU time, s		Performance, MFLOPS	
		DMS	ABAQUS	DMS	ABAQUS
$32 \times 32 \times 32$	107,811	112	147	1836	1331
$48 \times 48 \times 48$	352,947	1159	1213	2016	1890

[a] $R_{\text{peak}} = 5.2$ GFLOPS.

Table 4 Performance comparison[a] on Intel Xeon 2.4 GHz system[b]

Variable	DMS	ABAQUS
CPU time, s	93	203
Performance, MFLOPS	2153	986

[a]Number of unknowns 107,811 and $32 \times 32 \times 32$ mesh.
[b] $R_{\text{peak}} = 4.8$ GFLOPS.

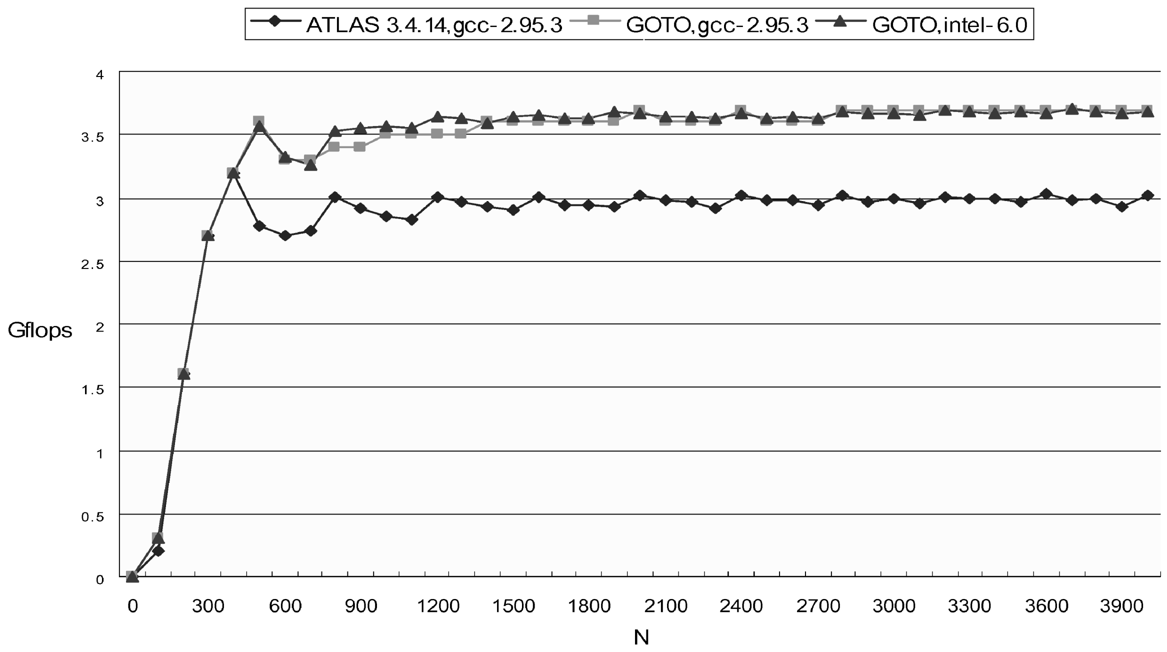


Fig. 5 Performance comparison for ATLAS and GOTO.

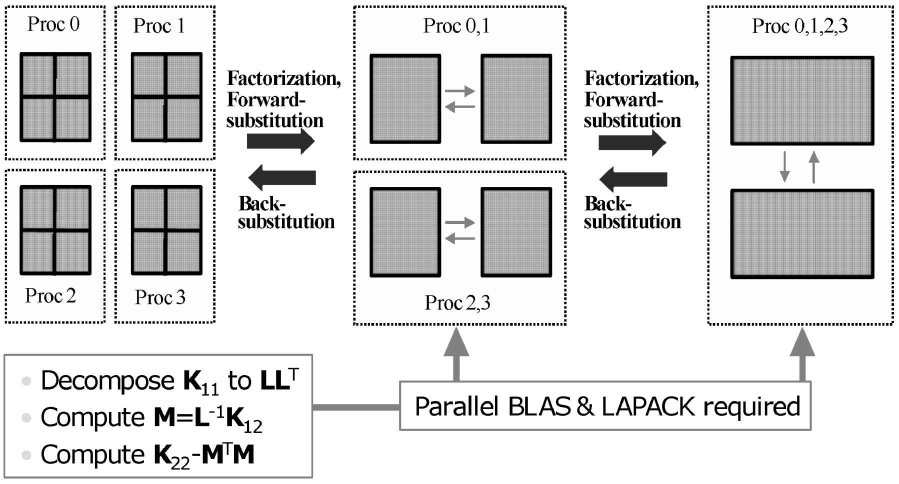

Proc 0
Proc 1
Proc 2
Proc 3
Factorization, Forward-substitution
Back-substitution
Proc 0,1
Proc 2,3
Proc 0,1,2,3
Decompose $\mathbf{K}_{11}$ to $\mathbf{LL}^T$
Compute $\mathbf{M}=\mathbf{L}^{-1}\mathbf{K}_{12}$
Compute $\mathbf{K}_{22}-\mathbf{M}^T\mathbf{M}$
Parallel BLAS & LAPACK required

Fig. 6 Parallel implementation of the domainwise multifrontal method.

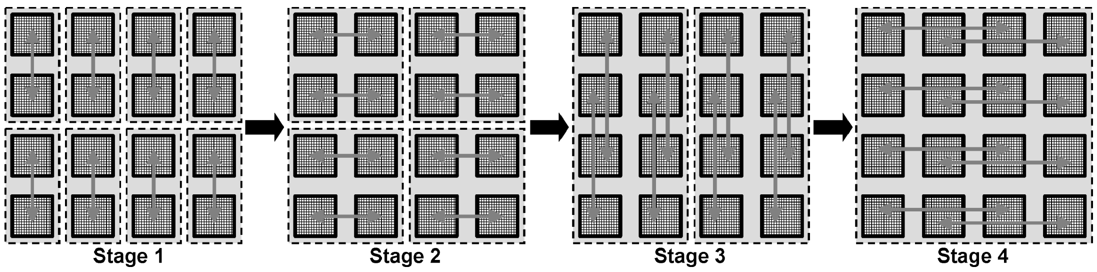


Fig. 7 Resulting communication pattern for parallel extend–add operation.

considered equivalent to the computations needed to build a frontal matrix equation for each domain assigned to each processor. After this step, iterative solvers perform parallel sparse matrix computations that require global communications for each iteration, whereas the domainwise multifrontal solver performs local parallel dense matrix computations with neighboring processors for an increasing number of domains. Figure 6 shows parallel implementation of the domainwise multifrontal solver with four domains.

As shown in Fig. 6, after the independent computations within each processor are completed, a domain-merging operation to build a new matrix equation in the form of Eq. (1) is performed across the processors. Then, dense matrix computations in Eqs. (3) and (4) needed to build the reduced matrix equation (2) by the static condensation procedure are performed in parallel over processors that have the domains included in the new merged domain. Parallel domain-merging operation to construct new matrix equations for enlarged or merged domains is very complicated and needs to be implemented carefully to achieve good scalability. The domain-merging operation for the domainwise multifrontal solver is equivalent to the "extend–add" operation for general multifrontal methods, and so an efficient parallel extend–add algorithm for general multifrontal methods can be adopted in the domain-merging operation of the domainwise multifrontal solver. In this work, we used the parallel extend–add algorithm proposed by Gupta et al.[9] to implement the domain-merging operation of our domainwise multifrontal solver because that algorithm is considered to be the most efficient and scalable. Kim et al.[17] have implemented the domainwise multifrontal solver based on the serial implementation by Kim and Kim.[11] However, because of the inefficient serial and parallel implementations of dense matrix computations, the solver's serial and parallel performance is very poor compared with the present implementation. Furthermore, when a large number of processors are used, the solver's implementation for a domain-merging operation produces inactive processors that do not participate in the parallel dense matrix computations in Eqs. (3) and (4), which leads to limited scalability for both performance and storage. In the present implementation, by adopting Gupta's parallel extend–add algorithm, we can overcome this limitation and achieve good scalability for large-scale problems.

When two neighboring domains are merged, stiffness matrices from each domain are assembled to build a new stiffness matrix in Eq. (1) by exchanging the matrix data with each other. In this procedure, a new stiffness matrix for the merged domain can be constructed only by one-to-one communication with the partner processor that is participating in the computations for the current domain. The resulting communication patterns for the present parallel extend–add algorithm are shown in Fig. 7.

In Fig. 7, boxes with dashed lines represent computational domains used to construct Eq. (1) in each stage. The ownership of each data element in the new stiffness matrix is checked, and one-to-one data exchange is performed according to that information. The ownership of each data element is determined by the bit pattern of global identification of each data element and a partitioning strategy in each stage that is changed in turn from rowwise to columnwise to guarantee scalability.

Figure 8 shows a matrix partitioning strategy in the present parallel extend–add operation. In Fig. 8, two frontal matrices once partitioned rowwise are assembled to form a new frontal matrix that is partitioned and distributed columnwise.

After new matrix equations for new merged domains are assembled by a parallel extend–add operation, they should be solved in parallel. To do this, we use existing implementations for parallel linear algebra subroutines such as parallel BLAS (PBLAS) and ScaLAPACK,[18] which are known as the most successful parallel implementations of BLAS and LAPACK. Note that to use PBLAS and ScaLAPACK, matrices should be distributed in a block-cyclic manner with a constant block size for matrix partitioning. However, in this case, the present implementation of parallel extend–add operation cannot be used because it produces a data distribution with nonconstant block size that is not supported by PBLAS and ScaLAPACK. Instead, the parallel extend–add operation of frontal matrices, which requires almost all-to-all communication among related processors, should be implemented in the use of PBLAS

Table 5 Computing time and speedup on IBM p690 system

No. of	DMS [32 × 32 × 32] (DOF = 107,811)		DMS [48 × 48 × 48] (DOF = 352,947)		ABAQUS [48 × 48 × 48] (DOF = 352,947)	
CPUs	Elapsed time, s	Speedup	Elapsed time, s	Speedup	Elapsed time, s	Speedup
1	112	1.00	1159	1.00	1345	1.00
2	57	1.96	585	1.98	710	1.89
4	32	3.50	306	3.79	485	2.77
8	17	6.59	152	7.63	468	2.87

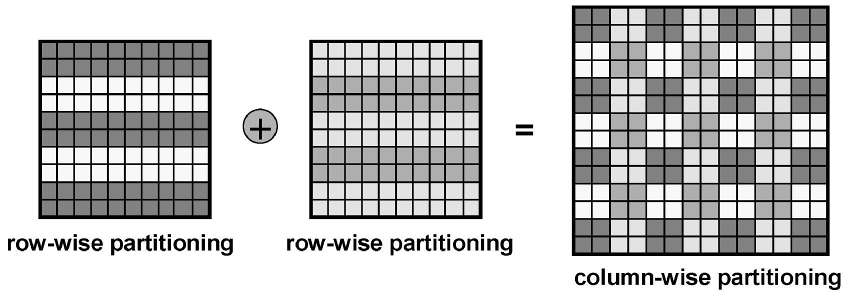


Fig. 8 Matrix partitioning for parallel extend–add operation.

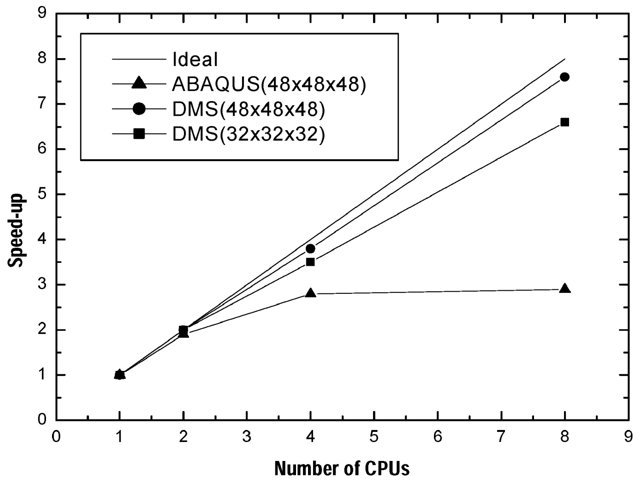


Fig. 9 Speedup comparison on IBM p690 system.

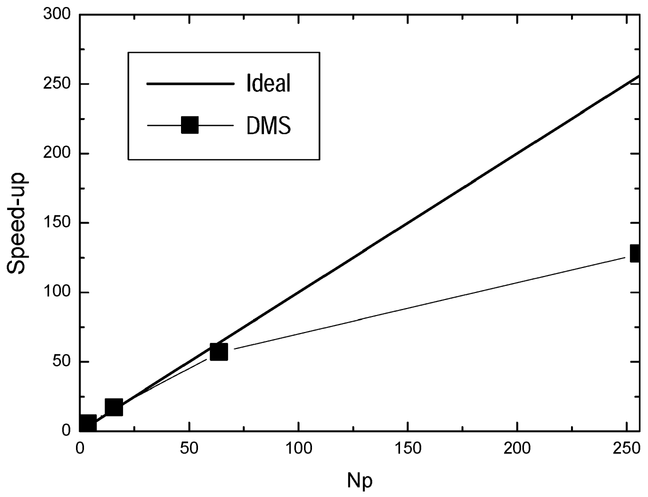


Fig. 10 Scaled speedup for two-dimensional mesh topology on Pegasus cluster.

and ScaLAPACK. Thus, it is too inefficient to use and too complicated to implement. Hence, to use the parallel extend–add algorithm proposed by Gupta et al.[9] that is much more efficient, we implemented our own parallel dense linear algebra subroutines required for our domainwise multifrontal solver instead of using PBLAS and ScaLAPACK. The primary computations of all of our parallel linear algebra subroutines are implemented using BLAS and LAPACK, and all of the communications are implemented using BLACS.[19] In this way, the performance of each developed subroutine can compete with that of the corresponding subroutine from PBLAS and ScaLAPACK. Developed library routines may also be used to balance loads for parallel matrix computations on heterogeneous computing environments by means of flexible-size blocking.

Performance Evaluation

The parallel performance of our domainwise multifrontal solver was evaluated by comparing the performance of our structural analysis code with ABAQUS, which is one of the most popular and efficient finite element structural analysis packages. Simple structural analysis problems with three-dimensional solid elements are solved on an IBM p690 system that has 32 POWER4 processors (1.3 GHz), and the resulting computing time and speedup are listed and compared in Table 5 and Fig. 9.

ABAQUS can be run in parallel only within shared memory architectures, and so we can run ABAQUS on up to 32 processors. However, parallel performance results on more than eight processors were much worse than those on only eight processors because of the hardware architecture of the IBM p690 system. Thus, speedup test results were compared and listed only up to eight processors. As shown in Table 5 and Fig. 9, the performance of our structural analysis code with the domainwise multifrontal solver scales up near the ideal speedup until the number of processors reaches eight, whereas the performance of ABAQUS stays under three times the serial performance even for four and eight processors.

To perform large-scale structural analysis, we should be able to increase the solvable problem size easily by just increasing the number of processors with limited physical memory size of a single system. Therefore, the scalability in terms of storage and performance is the most important factor for large-scale structural analysis. Thus, scalability tests were performed to show how well the domainwise multifrontal solver can scale up the solvable problem size as well as the performance.

Scalability tests for both two- and three-dimensional mesh topology were performed using 256 Intel Xeon 2.2-GHz processors of a Pegasus[20] cluster system. Problem size was increased in proportion to the number of processors used. For the two-dimensional scalability test, the size of the mesh assigned to each processor was kept constant. A 128 × 128 × 1 finite element mesh was assigned to each processor. For the three-dimensional scalability test, the total operation count in each processor was kept nearly constant. The resulting scaled speedup graphs for two- and three-dimensional scalability tests are shown in Figs. 10 and 11. Note from Figs. 10 and 11 that

Table 6 Three-dimensional scalability test results on Pegasus system

No. of CPUs	Finite element mesh	No. of unknowns	Operation count	Performance, GFLOPS	Scaled speedup
1	$40 \times 40 \times 40$	201,720	8.11×10^{11}	2.1	1.0
4	$50 \times 50 \times 50$	390,150	3.06×10^{12}	8.2	3.8
16	$64 \times 64 \times 64$	811,200	1.33×10^{13}	25.3	11.8
64	$80 \times 80 \times 80$	1,574,640	5.05×10^{13}	65.5	30.6
256	$100 \times 100 \times 100$	3,060,300	1.92×10^{14}	191.0	89.3

Table 7 Three-dimensional scalability test results on IBM p690 system

No. of CPUs	Finite element mesh	No. of unknowns	Performance, GFLOPS	Scaled speedup
1	$40 \times 40 \times 40$	201,720	3.7	1.0
4	$50 \times 50 \times 50$	390,150	15.6	4.3
16	$64 \times 64 \times 64$	811,200	48.6	13.3
64	$80 \times 80 \times 80$	1,574,640	114.0	31.1
256	$100 \times 100 \times 100$	3,060,300	350.0	95.6
512	$128 \times 128 \times 128$	6,390,144	664.0	181.4
512	$160 \times 160 \times 160$	12,442,080	770.0	210.4

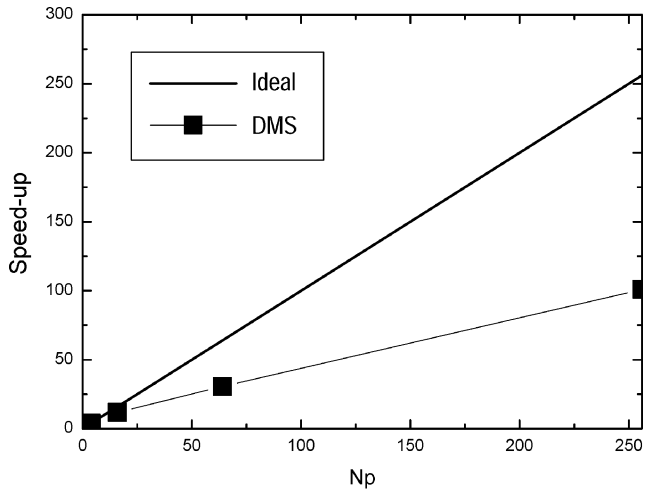


Fig. 11 Scaled speedup for three-dimensional mesh topology on Pegasus system.

the performance continuously scales up until 256 CPU, and it can be said that the performance of the proposed solver scales quite well for a direct solver.

Data specifications for the three-dimensional scalability test are listed in Table 6. As listed in Table 6, we roughly determined the largest problem size solvable on single system and increased the problem size so that the estimated operation count became proportional to the number of processors used. In contrast, the mesh topology that had the same number of elements in each dimension was not changed.

A scalability test for the three-dimensional mesh topology was also performed on an IBM p690 system. Specifications of test data and performance results are shown in Table 7. The scaled speedup on the IBM p690 system is better than that on the Pegasus cluster system for the same number of processors because the IBM p690 system has a faster network system. The performance results in GFLOPS on the IBM p690 system are also greater than those on the Pegasus cluster system.

The 512 CPU region in Table 7 represents performance results that do not follow the scaling rule for the three-dimensional scalability test. However, these results show the ability of the proposed solver in terms of performance and solvable problem size. By using the proposed solver, we could solve a three-dimensional structural analysis problem with a $160 \times 160 \times 160$ solid element mesh and more than 12 million unknowns and achieved 0.77 trillion-FLOPS performance.

Applications to Large-Scale Structural Analysis Problems

To show the availability and performance of our domainwise parallel direct solver, practical structural analysis problems were solved. One of them is the static analysis of a detailed finite element model of active fiber composites (AFC), and the other is a vibration analysis of large-scale structures.

Application 1: Static Analysis of Detailed Finite Element Model of AFC

AFC possess desirable characteristics for smart structure applications. One major advantage of AFC is the ability to create anisotropic laminate layers useful in applications requiring off-axis or twisting motions. AFC is naturally composed of two different constituents: piezoelectric fiber and a matrix. By application of the voltage to the piezoelectric fiber, mechanical characteristics of the composite panel can be controlled. Meanwhile, to predict the behavior of AFC exactly so that we can use and control them, we need a microscopic approach that enables exact prediction of local behaviors such as failure of the piezoelectric fiber of the AFC. A homogenization method, which models all of the constituents together with averaged material properties, can be used to predict the global behaviors of the composite structures in an averaged sense. However, this approach has intrinsic limitations in describing the local behaviors in the level of the constituents. Thus, we have to use the detailed finite element model of AFC shown in Fig. 12 that is modeled by a microscopic approach. A very large finite element analysis problem should be solved to analyze this finite element model of AFC. In this case, iterative methods do not converge well because these kinds of problems generally have bad numerical properties. By using the structural analysis code with the domainwise multifrontal solver that we proposed in this work as a domainwise parallel direct solver, we could solve a structural analysis problem of AFC with more than 10 million unknowns on 256 CPUs of the Pegasus cluster.

The piezoelectric fiber and matrix were modeled separately for the detailed finite element model of AFC shown in Fig. 12, and 3,369,600 solid elements were used for the model. The total number of DOF (unknowns) of the AFC model was 10,543,005. This approach, which uses detailed finite element models for composite structures is called the DNS of composite structures.[5,6]

The elapsed time was 2139 s with 256 CPUs. The von Mises effective stress distributions are shown in Fig. 13. By the use of this approach, local stress distributions that could not be observed with homogenized models can be predicted successfully. Moreover, inhomogeneous nonperiodic effects that could not be obtained with unit cell models can be evaluated using the DNS approach.

Application 2: Vibration Analysis Using an Eigensolver

In typical structural dynamic analysis procedures, natural frequency analysis is often required to analyze the dynamic properties of given structures. Natural frequencies and vibration modes of given structures can be obtained by solving a generalized eigenvalue problem with real, symmetric system matrices that can be written in the following form:

$$(\boldsymbol{K} - \lambda \boldsymbol{M})\boldsymbol{x} = \boldsymbol{0} \tag{5}$$

where $\boldsymbol{K}$ and $\boldsymbol{M}$ are, respectively, the stiffness matrix and the mass matrix of the structure. The eigenvalue λ and the eigenvector $\boldsymbol{x}$ correspond to the natural frequency and mode shape of each vibration mode of the structure.

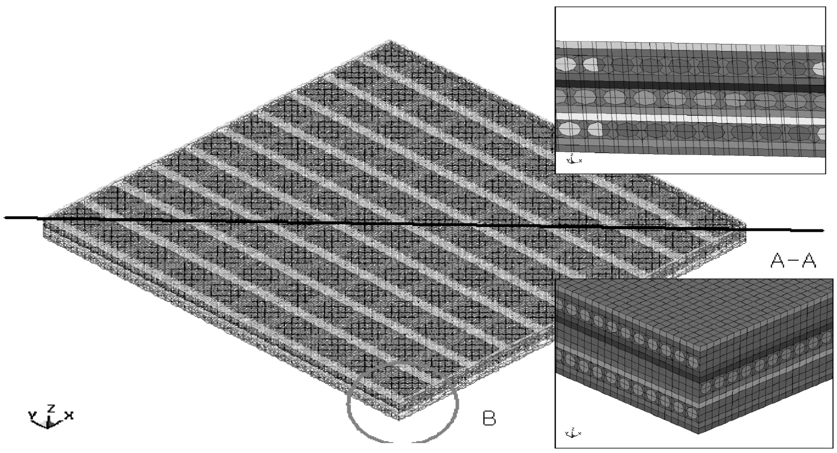


Fig. 12 Finite element model of AFC structure (10 million DOF).

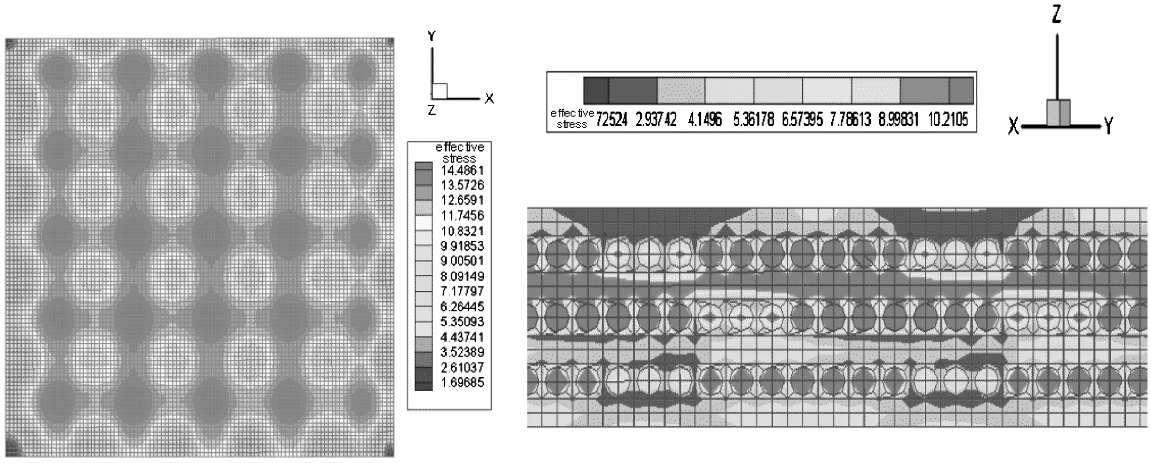


Fig. 13 Von Mises effective stress distributions of AFC.

To solve these kinds of problems, an efficient eigensolver is required. The Lanczos eigensolver is a powerful tool for extraction of the extreme eigenvalues and the corresponding eigenvectors of a sparse symmetric generalized eigenvalue problem (see Refs. 21 and 22). The block Lanczos algorithm has shown efficiency for large-scale eigenanalysis (see Ref. 23).

Direct solvers may be used much more efficiently for implementation of the Lanczos algorithm than can iterative solvers because the Lanczos algorithm, including the block Lanczos algorithm, requires repeated solutions of linear equations with the same coefficient matrix. Some iterative solvers, such as the FETI-dual primal (FETI-DP)[1,2] method used in the Salinas code, support repeated right-hand side capability that can reduce computing time for repeated solutions of linear equations with the same coefficient matrix, and the Salinas code is implemented in the Lanczos algorithm by using that capability. However, the computing time reduction of the FETI-DP method for repeated solutions with the same coefficient matrix is not that long compared with direct solvers. Therefore, our domainwise parallel direct solver can enable a more efficient implementation of the Lanczos algorithm for eigenanalysis of large-scale finite element models. Thus, we implemented the block Lanczos algorithm by using the proposed domainwise parallel direct solver and showed the effectiveness of an efficient parallel direct solver for a large-scale parallel eigensolution.

For a verification and performance check of the developed eigensolver, the vibration analysis of a plate model that has 1,638,400 eight-node solid elements and 7,389,225 DOF was performed. There were ten eigenvalues and eigenvectors extracted, and the elapsed time including I/O operations was 3107 s with 256 CPUs on the Pegasus system. The resulting mode shape of fourth mode is shown in Fig. 14.

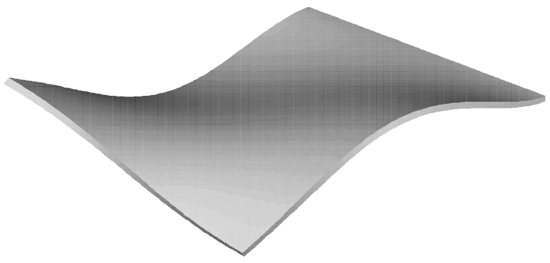
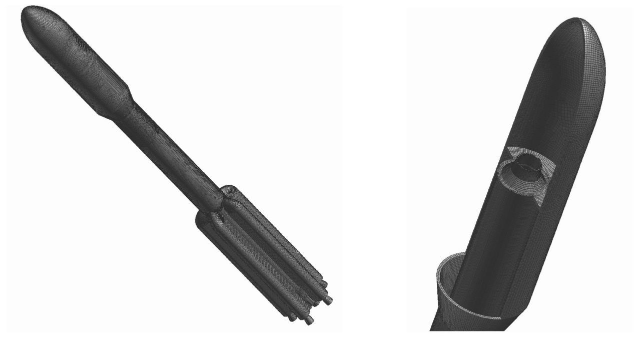


Fig. 14 Fourth-mode shape of simple plate model (7.0 million DOF).

To illustrate the usefulness and performance of the eigensolver with the domainwise parallel direct solver, a real and practical application, the full-scale vibration analysis of an aerospace launch vehicle, was carried out. The ATLAS V500 launch vehicle was modeled using 255,550 solid finite elements as shown in Fig. 15. The total number of DOF was 1,201,511.

There were 20 eigenvalues and eigenvectors extracted using 64 processors of the Pegasus system and the 7th and 10th mode shape are shown in Fig. 16. The elapsed time was 402 s including pre/postprocessing time for the mesh data input and the result data output and mesh partitioning time. This computation was also performed on an IBM p690 system using 16 processors, and the elapsed time was 208 s.

These results are comparable to the results from the Salinas code, which was said to be the only finite element software capable of computing a dozen eigenmodes for a million-DOF finite element structural model in less than 10 min (Ref. 3). By applying our domainwise parallel direct solver to the block Lanczos algorithm, we were able to cross the threshold, which means that our code can show



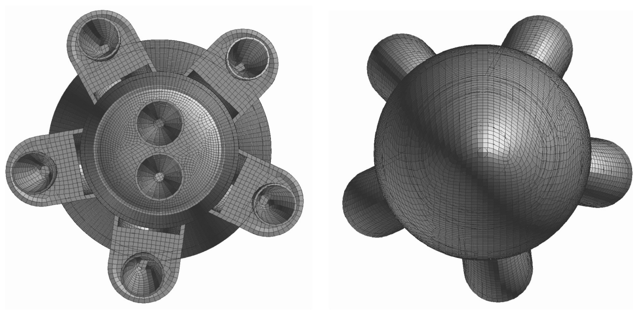


Fig. 15 Finite element mesh of ATLAS V500 launch vehicle.

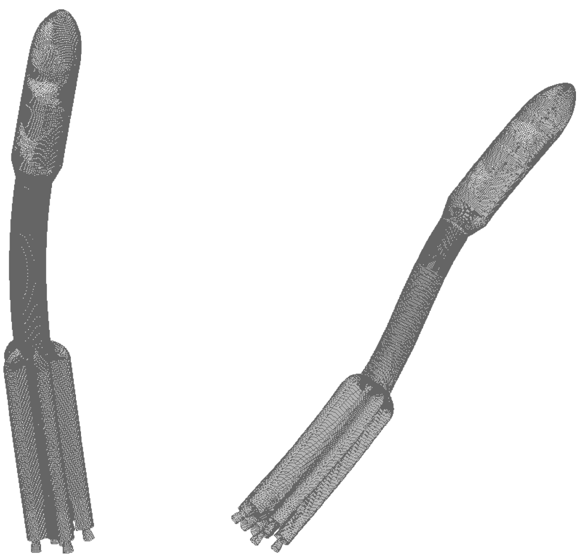


Fig. 16 Mode shapes of ATLAS V500 launch vehicle.

better or comparable performance to the Salinas code for problems with similar size to those solved here. This would not be possible without the parallel direct solver, the domainwise multifrontal solver, which enables an efficient implementation of the block Lanczos algorithm for large-scale eigenanalysis.

Conclusions

In this work, we proposed an efficient, useful, and easy-to-use parallel direct solver based on a domainwise approach, which made it easy and natural to parallelize the whole finite element analysis procedure by decomposing given problems in a domainwise manner instead of a matrixwise manner as other sparse direct solvers do. By applying the proposed solver to our own structural analysis code, we showed the scalability of the proposed solver in terms of storage as well as performance, and we were able to perform large-scale structural analyses for detailed finite element models of real-world structures successfully.

This research can be considered very significant progress in large-scale parallel finite element analysis techniques because it means that we can now benefit from the merits of applying direct solvers, including the numerical robustness that is an essential property for the general application of finite element analysis techniques, to a wide range of problems from the academic and industrial communities, even when we solve large-scale parallel finite element analysis problems.

References

[1]Farhat, C., Lesoinne, M., and Pierson, K., "A Scalable Dual-Primal Domain Decomposition Method," *Numerical Linear Algebra with Applications*, Vol. 7, No. 7, 2000, pp. 687–714.

[2]Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., and Rixen, D., "FETI-DP: A Dual-Primal Unified FETI Method—Part I: A Faster Alternative to the Two-Level FETI Method," *International Journal for Numerical Methods in Engineering*, Vol. 50, No. 7, 2001, pp. 1523–1544.

[3]Bhardwaj, M., Pierson, K., Reese, G., Walsh, T., Day, D., Alvin, K., Peery, J., Farhat, C., and Lesoinne, M., "Salinas: A Scalable Software for High-Performance Structural and Solid Mechanics Simulations," *Proceedings of the IEEE/ACM SC2002 Conference* [CD-ROM], 2002.

[4]LeTallec, P., "Domain-Decomposition Methods in Computational Mechanics," *Computational Mechanics Advances*, Vol. 1, No. 2, 1994, pp. 121–220.

[5]Kim, S. J., Hwang, J. S., and Paik, S. H., "Direct Numerical Simulation of Active Fiber Composites," *Proceedings of SPIE*, Vol. 5053, *Smart Structures and Materials 2003, Active Materials: Behavior and Mechanics*, edited by D. C. Lagoudas, Society of Photo-Optical Instrumentation Engineers, 2003, pp. 568–575.

[6]Kim, S. J., Lee, C. S., Yeo, H. J., Kim, J. H., and Cho, J. Y., "Direct Numerical Simulation of Composite Structures," *Journal of Composite Materials*, Vol. 36, No. 24, 2002, pp. 2765–2785.

[7]Duff, S., and Reid, J. K., "The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations," *ACM Transactions on Mathematical Software*, Vol. 9, No. 3, 1983, pp. 302–325.

[8]Irons, B. M., "A Frontal Solution Program for Finite Element Analysis," *International Journal for Numerical Methods in Engineering*, Vol. 2, No. 1, 1970, pp. 5–32.

[9]Gupta, A., Karypis, G., and Kumar, V., "Highly Scalable Parallel Algorithms for Sparse Matrix Factorization," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 5, 1997, pp. 502–520.

[10]Dongarra, J., "Freely Available Software for Linear Algebra on the Web," URL: http://www.netlib.org/utk/people/JackDongarra/la-sw.html [cited 28 May 2004].

[11]Kim, J. H., and Kim, S. J., "Multifrontal Solver Combined with Graph Partitioners," *AIAA Journal*, Vol. 38, No. 8, 1999, pp. 964–970.

[12]Dongarra, J., "BLAS (Basic Linear Algebra Subprograms)," URL: http://www.netlib.org/blas [cited 28 May 2004].

[13]Dongarra, J., "LAPACK (Linear Algebra PACKage)," URL: http://www.netlib.org/lapack [cited 28 May 2004].

[14]DeRose, L., "HPM Tool Kit," URL: http://www.alphaworks.ibm.com/tech/hpmtoolkit [cited 28 May 2004].

[15]Dongarra, J., "Automatically Tuned Linear Algebra Software," URL: http://math-atlas.sourceforge.net [cited 28 May 2004].

[16]Goto, K., "High-Performance BLAS by Kazushige Goto," URL: http://www.cs.utexas.edu/users/flame/goto/#overview [cited 28 May 2004].

[17]Kim, S. J., Lee, C. S., and Kim, J. H., "Large-Scale Structural Analysis by Parallel Multifrontal Solver Through Internet-Based Personal Computers," *AIAA Journal*, Vol. 40, No. 2, 2002, pp. 359–367.

[18]Dongarra, J., "The ScaLAPACK Project," URL: http://www.netlib.org/scalapack/index.html [cited 28 May 2004].

[19]Dongarra, J., "BLACS," URL: http://www.netlib.org/blacs/index.html [cited 28 May 2004].

[20]Meuer, H., Strohmaier, E., Dongarra, J., and Simon, H. D., "22nd Top500 list," URL: http://www.top500.org/lists/2003/11 [cited 28 May 2004].

[21]Parlett, B. N., and Nour-Omid, B., "Towards a Black Box Lanczos Program," *Computer Physics Communications*, Vol. 53, May 1989, pp. 169–179.

[22]Simon, H. D., "The Lanczos Algorithm with Partial Reorthogonalization," *Mathematics of Computation*, Vol. 42, No. 165, 1984, pp. 115–142.

[23]Grimes, R. G., Lewis, J. G., and Simon, H. D., "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, Vol. 15, No. 1, 1994, pp. 228–272.

A. Berman
Associate Editor