AIAA JOURNAL
Vol. 44, No. 7, July 2006

Parallel Implementation of Structural Dynamic Analysis for Parachute Simulation

Wenqing Zhang,* Michael L. Accorsi,† and John W. Leonard‡
University of Connecticut, Storrs, Connecticut 06268

An iterative parallel finite element procedure for the simulation of nonlinear structural behavior of parachute systems is presented. The Hilber–Hughes–Taylor method is employed in this procedure for nonlinear implicit time integration. Parallel techniques derived from the geometrical parallelism algorithm are developed in a finite element code. This involves the parallel implementation of generating the tangent stiffness matrix and the effective force vector, solving the resulting linearized simultaneous system equations and other required operations. Two sample problems are presented to demonstrate the inherent features of parachute simulations and to analyze the efficiency of the parallel algorithms.

Nomenclature

$\tilde{C}$	=	damping matrix
d	=	proportionality coefficient between mass matrix and damping matrix
$\boldsymbol{F}_{\text{eff}}$	=	effective force vector
K^*	=	tangent stiffness matrix at the current Newton–Ralphson iteration step
$\tilde{K}_{\text{eff}}$	=	effective stiffness matrix
$\tilde{M}$	=	mass matrix
$\boldsymbol{P}_n$	=	external force vector at time step n
$\boldsymbol{P}^*$	=	external force vector at the current Newton–Ralphson iteration step
$\boldsymbol{R}_n$	=	internal force vector at time step n
$\boldsymbol{R}^*$	=	internal force vector at the current Newton–Ralphson iteration step
$\boldsymbol{U}_n$	=	displacement vector at time step n
$\dot{\boldsymbol{U}}_n$	=	velocity vector at time step n
$\ddot{\boldsymbol{U}}_n$	=	acceleration vector at time step n
α	=	Hilber–Hughes–Taylor parameter
β, γ	=	Newmark parameter
Δt	=	length of each time step
$\boldsymbol{\Delta U}$	=	displacement increment

I. Introduction

COMPUTER finite element (FE) simulation of parachute behavior has recently been used in parachute design. A good simulation model can tremendously reduce a parachute system design cost. However, the simulation of parachute dynamic behavior is extremely complex and computationally intensive because it involves transient nonlinear FE analysis of huge models. The computation time needed for a simulation may be unreasonably long even with very fast modern computer processors. Therefore, it is necessary to develop a technique to speed up the computations for transient nonlinear FE analysis of parachute dynamics.

Recently, much research has been conducted in the area of parallel nonlinear structural dynamic (SD) analysis. For example, Chiang and Fulton introduced the basic concepts of parallel FE analysis in 1990.[1] Useful parallel-vector computation techniques for nonlinear FE analysis and SD analysis are discussed by Baddourah and Nguyen[2] and Jaques et al.[3] A parallel implementation of nonlinear FE structural dynamics analysis by using FORTRAN 90 and message-passing interface (MPI) is presented by Danielson and Namburu.[4] More systematic explanations for the usage of parallel techniques in FE analysis may be found in Ref. 5. All of these research works show that the parallel computation technique has great potential in FE nonlinear SD analysis. It can significantly reduce computation time by taking advantage of the combined power of multiple computer processors.[6]

Unfortunately, usage of parallel computation techniques in the area of parachute simulation has been limited. Most of the available parallel developments and implementations in parachute simulations focus on the computational fluid dynamic (CFD) area.[7] Coupled fluid–structure simulation, the most accurate simulation approach for parachute behavior, requires both CFD simulation and SD simulation. Thus, it is necessary to develop an efficient parallel SD analysis algorithm, as well as a parallel CFD analysis algorithm.

The objective of this study is to incorporate parallel computation techniques into the nonlinear FE analysis of SD for parachute simulations. A total Lagrangian method is employed for the analysis of structural mechanical behavior. Iterative procedures are involved that require force vector and tangent stiffness matrix updates during the iterations. Therefore, as mentioned by Chiang and Fulton[8] and Gummadi and Palazotto,[6] the most time-consuming parts in transient nonlinear FE analysis of structures are the generation of element force and stiffness matrix, as well as the solution of the linearized simultaneous system equations. This study is primarily focused on developing the techniques to make these two parts parallel. Parallel strategies used in other required miscellaneous operations, such as input/output, displacement updating, are also discussed. The available parallel computers in this research have the single-instruction multiple-data (SIMD) control mechanism[9] and the distributed memory arrangement. The MPI library[10] is used as the communication method in the implementation of parallel techniques into a C program for general FE analysis of membrane, cable, and payload structures. With the explicit calls of the message-passing commands, the task can be divided independently and the data may be transferred among multiple processors; thus, the desired combined power of multiple processors is obtained.[4]

In the first part of this paper, the dynamic algorithm and governing equations used for the nonlinear structural analysis in parachute simulations are presented. Then, the techniques employed for the development of the parallel computation algorithms, including the parallel storage strategies and the incorporation methods of parallel techniques into the FE dynamic procedures, are described. Two

Received 3 December 2004; revision received 4 December 2005; accepted for publication 1 January 2006.

*Ph.D. Student, Civil and Environmental Engineering Department; currently Postdoctoral Research Assistant, Civil and Environmental Engineering Department; wenqingz@ engr.uconn.edu.

†Professor, Civil and Environmental Engineering Department; accorsi@ engr.uconn.edu. Senior Member AIAA.

‡Professor, Civil and Environmental Engineering Department.

parallel implementations of transient nonlinear SD problems are also presented for validation, and the efficiencies of the parallel strategies are discussed.

II. FE Formulations

A typical parachute system consists of fabric canopy, suspension lines, and attachments. A general cable–membrane structural model has been developed where membrane elements, cable elements, and payload elements, which are dimensionless points with finite mass, are used to simulate the transient nonlinear structural behavior of fabric canopies, suspension lines, and attachments, respectively.[11,12] The surrounding air pressure and fluid drags are treated as external forces when standalone structural behavior of the parachute system is considered. The governing motion equation derived by Accorsi et al.[11] is adopted here. The nonlinear dynamic responses are integrated in time using the Hilber–Hughes–Taylor (HHT) method (see Ref. 13).

A. Governing Motion Equations

The governing FE equation controlling the nonlinear structural response is[11]

$$\tilde{M}\ddot{\boldsymbol{U}} + \boldsymbol{R} = \boldsymbol{F} \tag{1}$$

where R and F are the internal restoring force and the external applied force, respectively; $\boldsymbol{U}$ is the displacement vector, and $\dot{\boldsymbol{U}}$ and $\ddot{\boldsymbol{U}}$ are the first and second derivative of displacement with respect to time. Real damping effects are considered in this research; therefore, the governing equation has changed to

$$\tilde{M}\ddot{\boldsymbol{U}} + \tilde{C}\dot{\boldsymbol{U}} + \boldsymbol{R} = \boldsymbol{F} \tag{2}$$

where $\tilde{C}$ is the damping matrix. A time incremental scheme and implicit iterative approach are used to find the solution to this equation. The whole analysis time duration is divided into multiple time steps. When nonlinear structural response at time step n is known, the response at time step $n+1$ can be obtained by using an iterative approach. Detailed expression of the terms in Eqs. (1) and (2) may be found in Ref. 11.

B. HHT Method

The implicit HHT method (see Ref. 13) is used in this research to simulate the transient nonlinear SD behavior because of its second-order accuracy when using numerical damping. Because typical parachute simulations involve long time durations, implicit methods are preferable over explicit methods due to their unconditional stability. When considering numerical damping, the preceding FE equations governing the nonlinear SD response can be expressed as[11,13]

$$\begin{aligned}\tilde{M}\ddot{\boldsymbol{U}}_{n+1} + (1+\alpha)\tilde{C}\dot{\boldsymbol{U}}_{n+1} - \alpha\tilde{C}\dot{\boldsymbol{U}}_n + (1+\alpha)R_{n+1} - \alpha\boldsymbol{R}_n \\ = (1+\alpha)\boldsymbol{P}_{n+1} - \alpha\boldsymbol{P}_n\end{aligned} \tag{3}$$

The subscript represents the time-step number. That is, the displacements at time step $n+1$ are calculated from the results of the previous time step, that is, time step n, where the displacements, velocities, and accelerations for each node are already known. A dot indicates derivatives with respect to time. The HHT parameter α controls the strength of numerical damping. If α is chosen to be 0, the governing equations of motion are identical to those in the Newmark method (see Ref. 14).

The displacement and velocity expansions from time step n to time step $n+1$ are given by

$$\boldsymbol{U}_{n+1} = \boldsymbol{U}_n + \Delta t\boldsymbol{U}_n + \left[\left(\tfrac{1}{2} - \beta\right)\ddot{\boldsymbol{U}}_n + \beta\ddot{\boldsymbol{U}}_{n+1}\right]\Delta t^2 \tag{4a}$$

$$\dot{\boldsymbol{U}}_{n+1} = \dot{\boldsymbol{U}}_n + \Delta t[(1-\gamma)\ddot{\boldsymbol{U}}_n + \gamma\ddot{\boldsymbol{U}}_{n+1}] \tag{4b}$$

When α is chosen in the range of $[-\frac{1}{3}, 0]$, β and γ are determined to obtain the optimum accuracy by using[13]

$$\beta = \tfrac{1}{4}(1-\alpha)^2 \tag{5a}$$

$$\gamma = \tfrac{1}{2}(1-2\alpha) \tag{5b}$$

Once the expressions for $\boldsymbol{U}_{n+1}$, $\dot{\boldsymbol{U}}_{n+1}$, and $\ddot{\boldsymbol{U}}_{n+1}$ in terms of $\boldsymbol{U}_n$, $\dot{\boldsymbol{U}}_n$, and $\ddot{\boldsymbol{U}}_n$ are derived from Eqs. (4a) and (4b), they are substituted into Eq. (3). Because of its quadratic convergence rate, the Newton–Ralphson iteration (N–R) method is used to seek an incremental solution $\boldsymbol{\Delta U}$ up to time step $n+1$, given that the SD response at time step n is known. An identical linearized transient problem can be stated as

$$\tilde{\boldsymbol{K}}_{\text{eff}}\{\boldsymbol{\Delta U}\} = \boldsymbol{F}_{\text{eff}} \tag{6}$$

where $\tilde{\boldsymbol{K}}_{\text{eff}}$ is the effective stiffness matrix and $\boldsymbol{F}_{\text{eff}}$ is the effective force vector. When mass proportional damping is assumed to be used, the damping matrix $\tilde{C}$ can be written as

$$\tilde{C} = d\tilde{M} \tag{7}$$

where d is the proportionality coefficient. Therefore, the terms in Eq. (6) have the following forms:

$$\tilde{K}_{\text{eff}} = (1+\alpha)K^* + \tilde{M}/\beta\Delta t^2 + (1+\alpha)(d\gamma\tilde{M}/\beta\Delta t^2) \tag{8}$$

$$\boldsymbol{F}_{\text{eff}} = (1+\alpha)(P^* - R^*) - \alpha(P_n - R_n) + \tilde{M} \times \text{temperature} \tag{9}$$

$$\begin{aligned}\text{temp} = {} & \dot{U}_n/\beta\Delta t + (1/2\beta - 1)\ddot{U}_n - (U^* - U_n)/\beta\Delta t^2 \\ & + d\Big(\{(1+\alpha)(\gamma/\beta - 1) + \alpha\}\dot{U}_n + (\Delta t/2)(\gamma/\beta - 2) \\ & \times (1+\alpha)\ddot{U}_n - (\gamma/\beta\Delta t)(1+\alpha)(U^* - U_n)\Big)\end{aligned} \tag{10}$$

where the asterisks represent the current N–R iteration step.

III. Parallel Computation Techniques

The concept of parallel computation is to divide a large problem into multiple, independent, smaller subproblems and use a group of computer processors to solve them simultaneously. The power of multiple processors is efficiently combined together to reduce the needed problem-solving time.[6] In this research, the geometrical parallelism algorithm[5] is used for the transient nonlinear SD analysis. The entire structural model is divided into many independent or semi-independent substructures according to the geometrical discretization and the mesh connectivity, and the FE analysis for each substructure is assigned to different computer processors. Tremendous amounts of data communications are needed among the processors due to the enforcement of boundary conditions and the globalization of the model. A number of researchers have addressed the parallel techniques used in the CFD analysis of parachute simulation.[7,15,16] The parallel communication techniques employed by them are adopted in this research for the convenience of combining the SD analysis and CFD analysis to produce coupled fluid and structural simulation.

A. Domain Decomposition Technique

A divide-and-conquer[10] approach is used in this research to divide a large FE structural model into multiple smaller substructures. When n computer processors are assumed to be used, the whole large FE model is divided into n substructures according to the geometrical connectivity.[5,15] All elements in one specific substructure are geometrically connected. Each processor is assigned with a substructure, and every processor performs the FE analysis on its own substructure simultaneously. A combination step is taken to produce the FE response of the whole model after every processor finishes its own analysis. Efforts have been made to distribute elements and nodes on each processor equally to avoid synchronization delay.

Table 1 Element and node distributions among processors

Quantity	Processor 0	Processor 1	Processor 2
Nodes stored in system mode	1, 2, 3	4, 5, 6	7, 8
Subelement group contained	(1), (2)	(3), (4)	(5), (6)
Nodes needed to form submesh	1, 2, 7, 8	2, 3, 6, 7	3, 4, 5, 6
Nodes stored in element mode[a]	1-1, 2-2, 3-7, 4-8	1-2, 2-3, 3-6, 4-7	1-3, 2-4, 3-5, 4-6

[a]Number after hyphen is the corresponding global node.

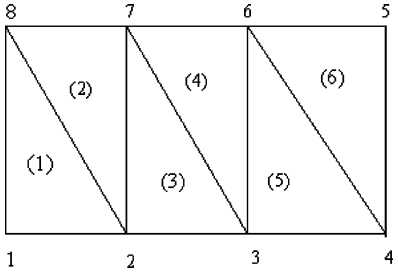

Fig. 1 Membrane structure.

B. Parallel Computation Storage

Operations involved in FE structural dynamic analysis are grouped as system nodal-level operations and element-level operations. For example, generating the stiffness matrix for each element is an element-level operation, whereas updating the displacements of each node at every time step is a system nodal-level operation. To make both kinds of operations parallel, variables are stored on each processor as system mode and element mode,[7] respectively. Variables stored in system mode on a processor represent the nodal information of the global nodes stored on this processor that are part of the total model, and variables stored in element mode have information for the nodes imbedded in the submesh contained in this processor.

A simple example is shown in Fig. 1, which represents a membrane system consisting of eight nodes and six three-node membrane elements. Assuming that three processors are used, Table 1 demonstrates the distribution of the node variables stored in system mode and the node variables stored in element mode on each processor. As can be seen in Table 1, when storing element and node information on multiple processors, efforts are made to assign the number of elements and nodes among processors as equally as possible according to the number of processors used. In this example, each processor has information for its own set of global nodes and two elements. Specifically, the system mode variables represent the global information for nodes 1–3 on processor 0, for nodes 4–6 on processor 1, and for nodes 7 and 8 on processor 2. The element subgroup on processor 0 includes elements 1 and 2 that consist of global nodes 1, 2, 7, and 8 in the mesh. Therefore, the element mode variables on processor 0 represent the elemental information of the nodes imbedded in this mesh, which correspond to global node 1, 2, 7, and 8. The element mode variables on processor 1 show the elemental information of the nodes building up the element subgroup on processor 1, which correspond to global nodes 2, 3, 6, and 7. Similar explanations apply to the mesh on processor 2. Table 1 also shows that redundant storage of certain variables occurs as a tradeoff for the increasing computational speed. However, little concern has been put on this issue because the requirements for space are not as stringent as for speed. When globalizing or localizing takes place, communications between the corresponding system mode variables and element mode variables are needed and realized by using MPI. More detailed explanations on communication techniques are given in next section.

Consequently, variables stored in system mode are processed when a system nodal-level operation takes place, and variables stored in element mode are used when processing an element-level operation. Most of the operations involved in the FE procedures are, therefore, suitable for parallel computation by taking advantage of these two storage modes.

C. Data Communication Among Processors

Data communications required during the stage of globalizing element mode variable values to system mode variables and the stage of localizing system mode variable values to element mode variables are realized by using MPI. The correspondence between global system mode nodal variables and local element mode nodal variables are built up during a preprocessing step; thus, communication trace only needs to be computed once and is repeatedly used during analysis.[7,15] With explicit calls of different MPI commands, data will transfer along both directions of communication trace to achieve globalization and localization effects.[7,10,15] When situations such as assembling the global force vector for the whole structure arise, every processor used will check the local substructure and send out element mode variables embedded to their corresponding destination processors by calling

```
MPI_Isend(&Local, LocalSize, MPI_DOUBLE,
 DESTINATION, TAG, MPI_COMM_WORLD, &Request)
```

At the same time, every processor will also check the system mode variables stored on it and receive data from their corresponding source processor by

```
MPI_Irecv(&Global, GlobalSize, MPI_DOUBLE,
 SOURCE, TAG, MPI_COMM_WORLD, &Request)
```

where Local is the element mode data for the substructures on each processor, Global is the system mode data, and LocalSize and GlobalSize are sizes of local data and global data, respectively. DESTINATION is the identity of the destination processor where local data should be sent to, and SOURCE is the identity of the source processor where global data should be received from. Thus, values in Local are transferred to Global at their corresponding locations along the communication trace.

When system mode nodal data need to be transferred to the element mode nodal data on each processor, a similar procedure is employed except that the system mode variables are sent out by each processor and received as element mode variables by their corresponding processors. Data are transferred in opposite direction, from Global to Local, along the communication trace.

Each communication step takes some time. For a specified problem with certain size, the amount of communication increases with the number of processors used; as a result, the ratio of the time spent on data communication vs real FE analysis time also increases. An appropriate number of processors should be chosen for a finite-sized problem to achieve best efficiency. If too many processors are used, data are sparsely stored on each processor, and tremendous communication is needed. Speed up will drop down because much time is wasted on data communication. This is explained in sample problems in later sections.

D. Parallel Computation Algorithm

The procedure of setting up the model is ignored, and a summary of steps involved in FE analysis of the transient nonlinear SD problem is given. The information in parenthesis at each step indicates what kind of operations is processed and which mode of variables is used in this step:

1) Begin time incremental loop.

2) Calculate initial estimation of displacement $\boldsymbol{U}_0$ using Taylor's expansion (system).

3) Begin N–R iteration loop.

4) Generate stiffness matrix $\tilde{K}_{\text{eff}}$ and force vector $\boldsymbol{F}_{\text{eff}}$ for each element subgroup and assemble them into global level (element and system).

5) Solve system algebraic equations (system).

6) Check for convergence, and go back to step 3 if not converged (system).

7) Update velocities and accelerations (system).

8) Go back to step 1 if more time steps are to be calculated.

Obviously, every step in this summary may take advantage of parallel computation techniques due to the two variable storage modes. However, we note that steps 4 and 5 occur in both the time incremental and N–R iterative loops and, therefore, are the most computationally intensive parts. For this reason, the concentration of this research lies on the parallel implementation of these two steps, which is demonstrated next.

E. Parallel Generation and Assemblage of $\tilde{K}_{\text{eff}}$ and F_{eff}

A geometrical parallelism algorithm and a domain decomposition approach are used in this step to generate and assemble $\tilde{K}_{\text{eff}}$ and $\boldsymbol{F}_{\text{eff}}$. The total number of elements and nodes in the model are divided into many nearly equal-sized subgroups through geometrical discretization according to the number of processors used. An element subgroup is allocated to a processor, and the elemental information of the nodes embedded in the element subgroup is stored on this processor as element-level mode variables. A global node subgroup is allocated to a processor and is stored as system-level mode variables. Every processor independently generates the matrices and vectors for its own subgroup of elements in parallel. After all processors finish the generation, the matrices and vectors generated on all processors are assembled. Then the global values are simultaneously assigned to corresponding system mode node variables to produce algebraic equation (6), indicating the system response.

Figure 2 shows the structure of parallel generation and assembly of matrices and vectors. The top part explains the stage of generating element matrices and vectors for each element subgroup concurrently. The middle communication interface is responsible for linking the element mode variables with their corresponding system mode variables and transferring data among processors if necessary. The bottom part represents the procedure of assembling the global values on the global node subgroup.

As can be seen, variables stored in element mode are used for generating matrices and vectors for the element subgroup on each processor because operations involved here are element level. On each processor, a sequential approach is used to generate matrices and vectors for the element subgroup. That is, elements in the element subgroup on a certain processor are considered one by one to form the matrices. According to Eqs. (8) and (9), the tangent stiffness matrix $\tilde{K}^*$ at current N–R iteration step is calculated for every element in the element subgroup and assembled to form the stiffness matrix for this subgroup. The term $(P^* - R^*)$ at current N–R iteration step is computed for every element to produce the force vector of each node consisting of the mesh in the element subgroup before storing into the corresponding element mode variable.

Figure 2 also shows that variables stored in system mode and element-level mode are both used in assembling matrices and vectors. The purpose of this step is to generate global system effective stiffness matrix and force vector. The operations here are system nodal level. As shown in Eqs. (8–10), the $(\tilde{M} \times \text{temp})$ term is calculated on every processor for each node consisting of the element subgroup mesh on the processor, and the results are stored in element-level mode variables at first. Next, the values of the variables stored as element mode are globalized to the corresponding variables stored as system mode. The results are then combined with the corresponding globalized $(\boldsymbol{P}^* - \boldsymbol{R}^*)$ term and $(\boldsymbol{P}_n - \boldsymbol{R}_n)$ term to produce the effective force vector for the system, as explained earlier in Eq. (9). Apparently, the components of this effective force vector are still distributed among processors according to the distribution of system mode nodal variables. On the other hand, $\tilde{K}_{\text{eff}}$ for each element subgroup is combined with its corresponding $\tilde{K}^*$, $\tilde{M}$, and $\tilde{C}$ term. The resultant $\tilde{K}_{\text{eff}}$ only represents the effective stiffness matrix for this subgroup of elements and is not assembled to the global level. The reason for this is that the element-level matrix is used in the equation solver. This is explained in the next section.

Synchronization delay may happen at the assembling step because of the unbalanced load on each processor. Some processors may finish their job of generating the elemental matrices for their own element subgroups ahead of other processors and have to be in an idle state waiting because the assembly of the system equations requires all parts of the model to participate. Many facts such as inappropriate geometrical discretization and unequal number of elements assigned on different processors can cause the load unbalance. Note that even when the number of elements on each processor is equal, the unbalance may still occur because of different type of elements allocated on different processors. The analysis of membrane elements is more complicated than cable and payload elements and, thus, requires more computational time. Efforts should be made to allocate computational load evenly among processors to reduce the synchronization delay.

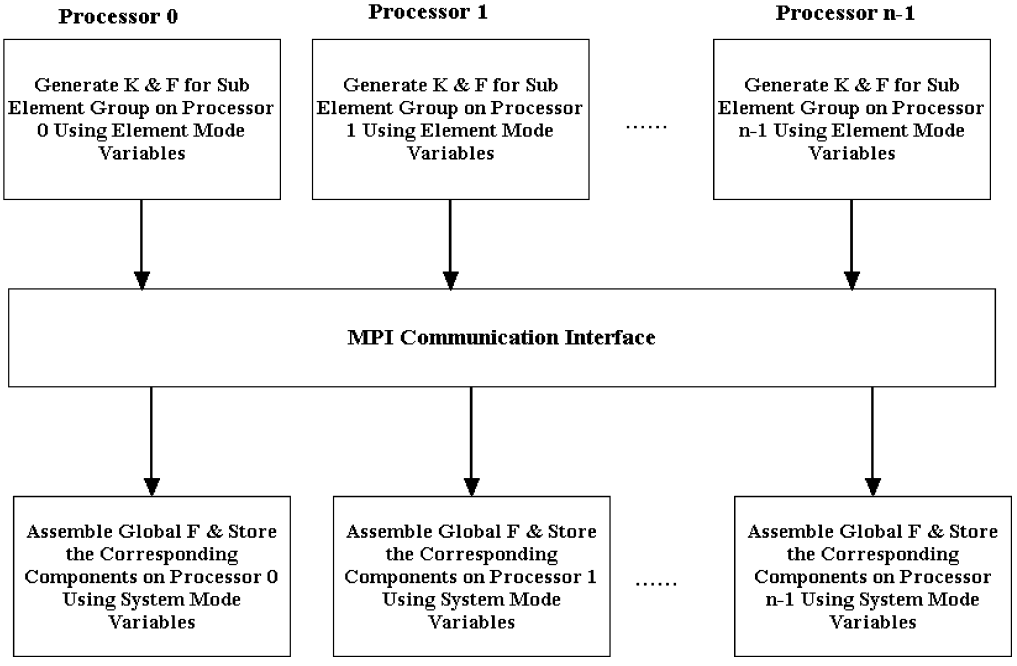

Fig. 2 Parallel generation and assembly of stiffness matrices and force vectors.

F. Parallel Equation Solver

As one of the most time-consuming parts of the FE analysis, the method for solving the system equations has been extensively investigated. Many researchers have devoted attention to developing parallel direct and iterative algorithms for solving large-scale linear equations. For example, Chiang and Fulton[8] presented a parallel skyline Cholesky decomposition method and lower–upper decomposition method by Gaussian elimination; Topping and Khan[5] demonstrated a parallel Jacobi conditioned conjugate gradient algorithm. Recently, an iterative method, generalized minimal residual method (GMRES),[17] has been widely used. Much research has been conducted to introduce parallel techniques to the implementations of GMRES method.[7,18,19]

In this research, because of its small memory requirements and good convergence features, the diagonal preconditioned GMRES method[20] is adopted to solve the algebraic equations. Because the equations needed to be solved are system-level algebraic equations and the operations involved in the equation solver are system level, operations are performed and parallelized mostly using variables stored in system mode. The algorithm for this method to solve an equation $\tilde{A}X = b$ can be expressed as follows[7,17]:

1) Guess initial solution x_0 and calculate residual $R_0 = b - \tilde{A}x_0$,
   $v_1 = r_0/||r_0||$
2) Inner iteration:
   for $j = 1, 2, \ldots, k$, do:
$v_j = v_j/\tilde{A}(j, j)$ (diagonal preconditioning)
   for $i = 1, 2, \ldots, j$, do:
      $h_{i,j} = (\tilde{A}v_j, v_i)$
      $\hat{v}_{j+1} = \tilde{A}v_j - h_{i,j}v_i$
   end for
   $h_{j+1,j} = \|\hat{v}_{j+1}\|$
$v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
end for
3) Form the solution:
   $\tilde{H} = [h_{i,j}]$
   $\tilde{V} = [v_{i,j}]$
   $y_k = \tilde{H}^{-1}||r_0||e_1$
   $x_k = x_0 + \tilde{V}y_k$
   if convergent, exit
   else $x_0 = x_k$ and repeat the whole procedure

Here lowercase letters with only one subscript letter represent vectors, lowercase letters with two subscript letters separated by comma denote scalars, and uppercase letters with a tilde represent matrices. As can be seen, e_1 is a unit vector of $(1, 0, 0, \ldots, 0)^T$, $\tilde{V}$ is the Krylov subspace whose columns are orthonormal base vectors $[v_1, v_2, \ldots, v_n]$, and $\tilde{H}$ is the Hessenberg matrix with entries of scalar $h_{i,j}$ (Refs. 7 and 20). Detailed algorithm derivation may be found in Ref. 20.

Note that every step in the algorithm involves either a vector dot product operation, a matrix–vector multiplication operation, or a vector update operation. Parallelization of the equation solver is achieved by making all of these basic operations parallel.

Vector dot product operations are naturally parallizable due to the existence of the system mode variables. A pictorial demonstration is shown in Fig. 3, which represents parallel procedure for the dot product of a vector $\boldsymbol{V}$ with a vector $\boldsymbol{T}$ to produce the scalar result S. V_1, V_2, etc., are the components of $\boldsymbol{V}$, and T_1, T_2, etc., are the components of $\boldsymbol{T}$. They are distributed among the processors evenly and correspondingly. The ellipses mark the subset of the vectors' components on each processor in Fig. 3. As can be seen, each processor at first simultaneously performs dot product operations using the partial vector components stored on it. Then the results obtained by every processor are added together to produce the dot production result for the whole vector.

A vector update operation is also made parallel by using a system mode variable, and the procedure is very similar to the parallel vector dot operations. Node variables stored in the system mode on each processor are modified sequentially. Every processor works on its own subgroup of global nodes simultaneously, and therefore, the updating of the whole vector is naturally concurrent.

Most of the matrix–vector multiplications in the algorithm involve multiplying the stiffness matrix $\tilde{K}_{\text{eff}}$ with a vector. As already mentioned, $\tilde{K}_{\text{eff}}$ is the effective stiffness matrix for each element subgroup on the corresponding processor and is stored in element mode. The vector may be stored in either element mode or system mode. If the vector is stored in element mode, a matrix and vector can be multiplied directly on each processor. Otherwise, the vector stored in the system mode should be localized to be a vector stored in the element mode by using parallel communication skills. Then the matrix and the vector stored in the element mode are multiplied on each processor. This procedure is parallel because every processor

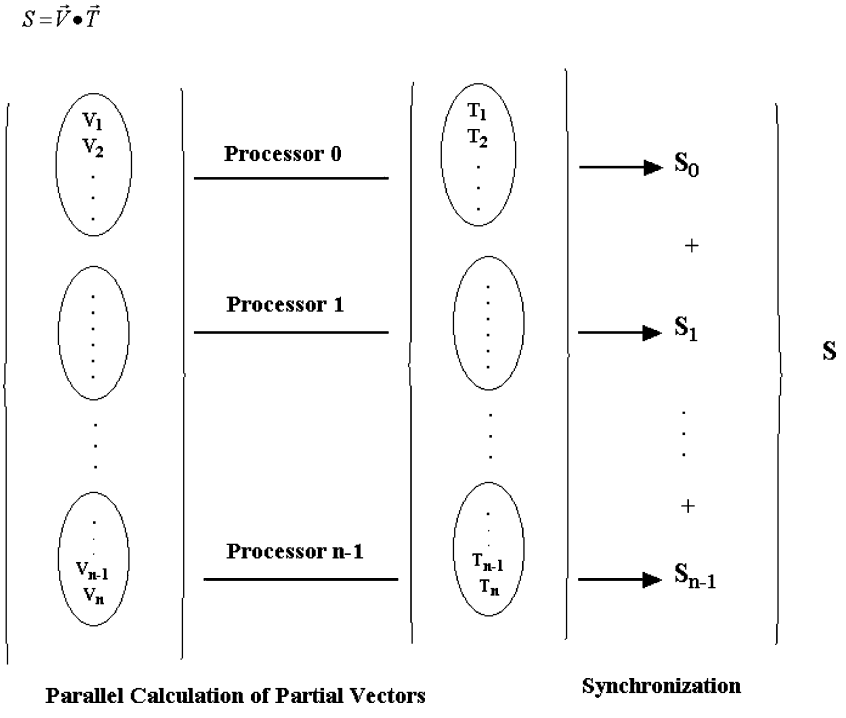

Fig. 3 Parallel vector dot production.

can process its own subgroup of elements at the same time. After each processor finishes its own multiplication, it obtains a vector corresponding to the nodes imbedded in the element mesh on this processor. A parallel communication step is used to add all of these element mode vectors into a system mode vector.

G. Parallel Miscellaneous Operations

The complete FE analysis of a SD model involves many other operations, such as input/output and displacements/velocities updating operations. Although these operations are not as time consuming as the steps of element matrix generation and system equations solving, which already have been demonstrated, they still take much time during real simulations because the models are normally large and very frequent input/output are required. Thus, the amount of time for input/output increases, and the time needed for the updating of displacements and velocities accumulates and becomes too large to be neglected. In this research, parallel strategies are also applied to these miscellaneous operations whenever possible to achieve better efficiency for the whole analysis. The techniques used here are very similar to the methods explained earlier. For instance, techniques used to update vectors in the step of solving system equations are also employed here to execute the displacements/velocities updating operations in parallel. Many input files are formatted so that every processor can read in the information for its own subgroup of global nodes concurrently.

IV. Example Problems

The parallel techniques described for the SD analysis have been incorporated into a nonlinear FE computer code[15] written in C language using the MPI library. This FE code has been designed for the general nonlinear analysis of membrane, cable, and payload structures.[11] The procedures of geometrically subdividing the whole model and the strategies used for data transfer among processors have been imbedded into the FE program. CPU time used for each FE step is recorded to evaluate efficiency. The features needed for coupled fluid–structural dynamic analysis are also included.

Two example problems have been implemented in this section to investigate the validity and the efficiency of the parallel model developed. The general information of the parallel computers used in this research is introduced first. Then the mechanical results and the discussion of parallel efficiency for the inflation analysis of a square airbag are presented, followed by a parallel simulation of the inflation of a G12 parachute. Although the parallel program developed is perfectly suitable for coupled fluid–structural analysis, only standalone three-dimensional SD analysis is discussed in this paper.

A. Computational Environment

The parallel computer used in this research is a silicon graphic interface (SGI) Origin 2000, which has 128 parallel processors with the speed of 400 MHz for each. This computer is specially designed for parallel computations and has a distributed memory arrangement. That is, each processor has its own memory and can store information for its own substructures. The SIMD control mechanism[9] is also used. During the analysis, every processor performs the same part of program instructions, whereas the input/output data are corresponding to their own substructure and, therefore, different. The data communications among processors, which are required in analysis, are implemented by utilizing MPI library.[10] With the explicit calls of the message-passing commands, data may be exchanged among multiple processors, and the control for each processor's status can be realized.

B. Inflation of a Square Airbag

In this example, inflation of a square airbag from the initial flat unstressed state to the final steady inflated state is simulated using the parallel FE code. The airbag is made up of two square membranes joined along their perimeters. The width and the length of the airbag are both 1 ft, and the thickness of the membranes is 0.0001 ft (3.048×10^{-5} m). The membrane material used is isotropic with a Young's modulus of 4.32×10^6 lb/ft^2 (2.068×10^8 N/m^2) and Poisson ratio of 0.3. When the internal pressure of magnitude 0.5 lb/ft^2 (23.94 Nm^2) is applied, the airbag inflates. Figures 4 and 5 show the two-dimensional and three-dimensional views of the initial mesh configuration for the FE model, respectively. This model consists of 5000 membrane elements and 5002 nodes, which leads to 15,006 equations. A three-dimensional analysis of the airbag inflation is performed using 500 time steps with each step of 0.0001 s. The fully inflated configuration of the airbag at the time of 0.05 s is shown in Figs. 6 and 7. Note that in this model the membrane wrinkling algorithm[12] is utilized and also implemented in parallel.

This problem is first executed using eight processors. Table 2 shows the records of the CPU time spent on different operations. As

Table 2 CPU time distributions for different operations

Operations	Time, CPU s
FE model setup	0.026966
Communication setup	11.219217
FSI setup	0.826638
Newmark operation	0.413030
Matrices generation and assemblage	2945.135546
GMRES solver	311.803631
Convergence check	8.200840
Input/output	1.111705
Initial acceleration calculation	0.064446
Total	3278.802019

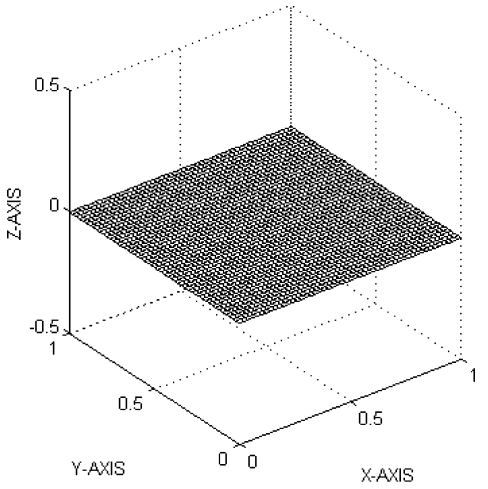

Fig. 4 Initial FE mesh of a square airbag (three-dimensional view).

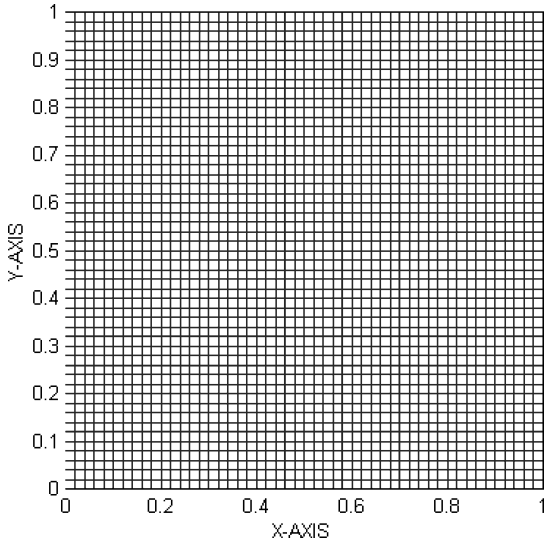

Fig. 5 Initial FE mesh of a square airbag (two-dimensional view).

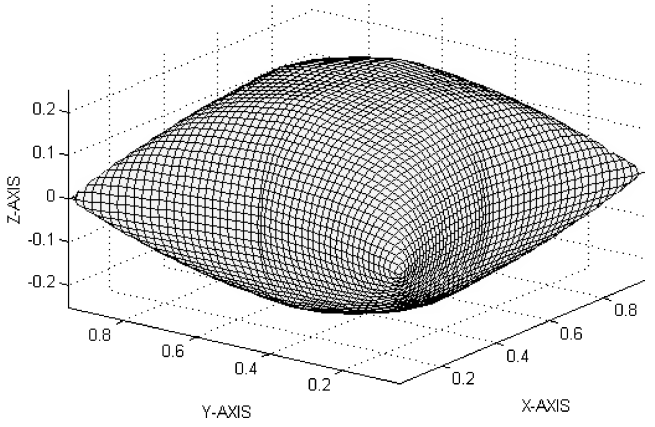

Fig. 6 Inflated shape of a square airbag (three-dimensional view).

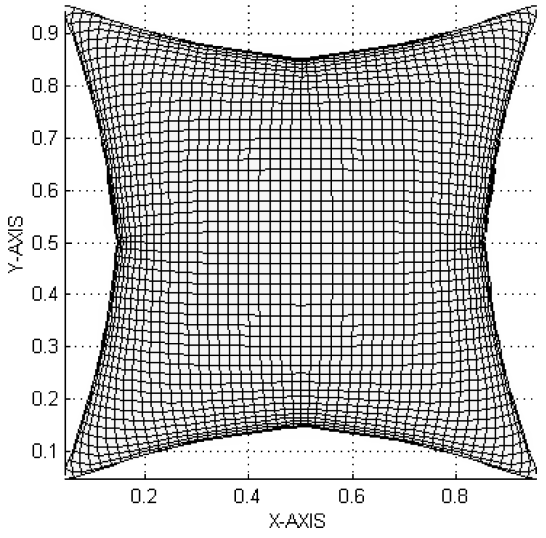

Fig. 7 Inflated shape of a square airbag (two-dimensional view).

can be seen, the step of the matrices generation and assemblage takes about 89.8% of the total time, and the GMRES solver takes about 9.5% of the total time. This means that the decision of making these two parts highly parallel is correct and very useful for the parallel FE structural analysis.

The speedup for using different number of processors is also compared. The serial runtime is defined as the time used for one sequential computer processor to complete the model analysis. The parallel runtime is defined as the time used for several computer processors to finish the same analysis running in parallel. The speedup is then defined as the ratio of the serial run time vs parallel time.[6] Under an ideal condition, the speedup should equal the number of processors used. However, this never happens in practice because data communication among processors takes time. The problem of airbag inflation was executed in parallel using one to eight processors, respectively. The ideal speedup and the speedups for the step of matrices generation and assemblage, the step of GMRES equation solver, and the total run are shown in Fig. 8. Figure 8 demonstrates that using parallel computation techniques can significantly reduce the running time by combining several processors' computation power. However, with the increase of the number of processors, the practical speedup curves become flatter and even drop down. This is because more data communications are needed when more processors are used. Also note that the speedup for the parallel model is dependent on the parallel machine used. In this study, all of the results are based on the usage of SGI Origin 2000. If a different parallel computer, such as Cray, is used, different speedup curves may be obtained.

C. Opening of a G12 Parachute

The parallel FE code developed earlier is also used to simulate parachute behavior. In this example, the opening of a round G12 parachute from the initially unstressed state to the fully inflated state is simulated. The G12 parachute has a round canopy [radius of 32 ft (9.75 m)] consisting of 64 gores, 64 suspension lines [length of 51.2 ft (15.61 m) each], and a 2200-lb (9785.6-N) payload. The material used for the canopy fabrics is isotropic with Young's modulus of 4.32×10^6 lb/ft^2 (2.068×10^8 N/m^2), and Poisson ratio of 0.3. Young's modulus for the cables is 4.32×10^6 lb/ft^2 (2.068×10^8 N/m^2), and the density for all of the materials is 0.6 lb/ft^3 (94.25 N/m^2). The membrane thickness is 0.001 ft (3.048×10^{-4} m), and the cross-sectional area of the cables is 0.0005 ft^2 (4.65×10^{-5} m^2).

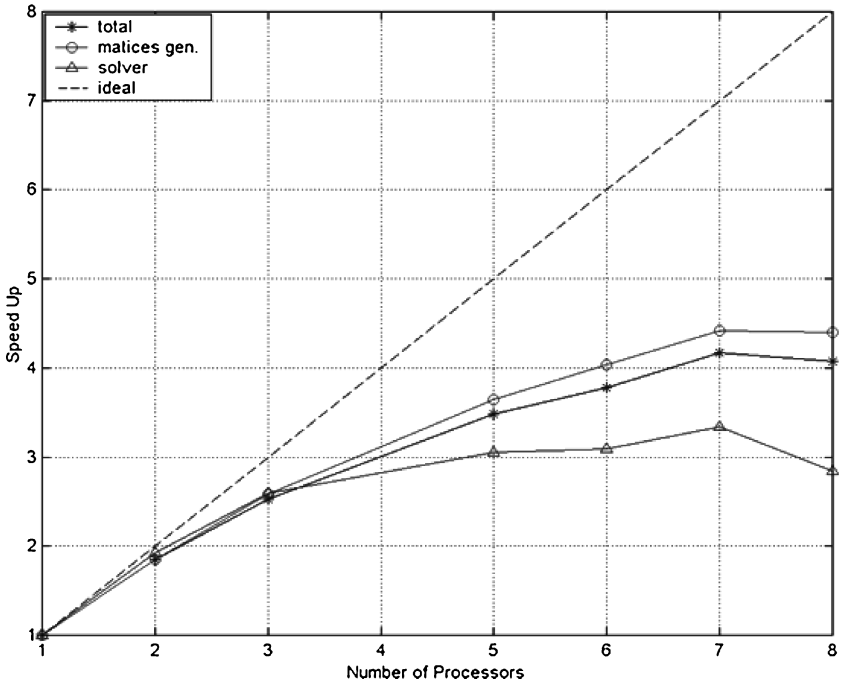

Fig. 8 Speedup analysis for airbag inflation problem.

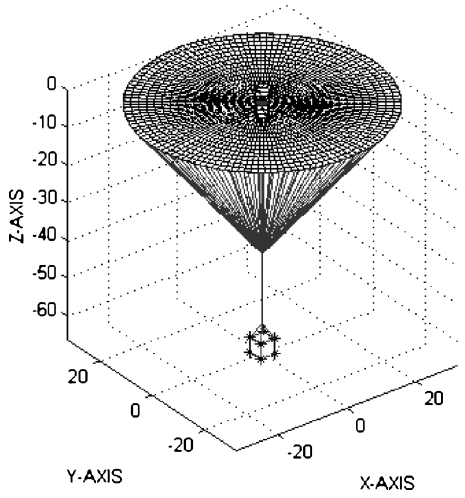

Fig. 9 Initial FE mesh of a G12 parachute.

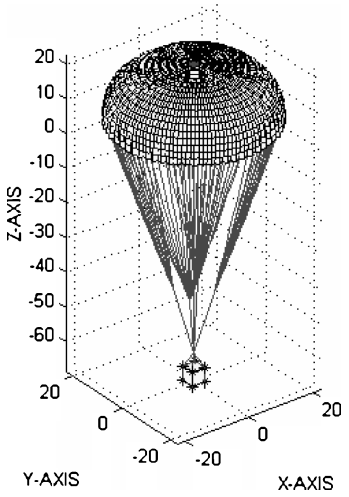

Fig. 10 Fully inflated shape of a G12 parachute.

A standalone three-dimensional FE SD model is used during the parallel simulation. The initial mesh configuration is shown in Fig. 9. This model involves 9633 nodes, 3037 two-node cable elements, 2182 nine-node membrane elements, and 9 payload elements, which lead to 28,899 system equations. During the opening, the model is subject to a constant pressure [0.5 lb/ft^2 (23.94 N/m^2)] on the canopy, a velocity proportional fluid drag force on the suspension lines, and gravity. There are 6000 time steps of analysis, with 1.0×10^{-3} s every step is performed. Figure 10 shows the fully inflated configuration of the G12 parachute at 6.0 s. These results demonstrate the capability of the parallel FE code presented earlier to simulate the complex parachute behavior. Note that in this example the membrane wrinkling algorithm[12] is also implemented in parallel to produce realistic simulations for the canopy membranes.

To compare the speedup of the parallel code, this example is executed in parallel by using 1, 2, 4, 8, and 16 computer processors, respectively. The CPU time needed to finish 10 time steps with 4 iterations every step is recorded and compared. Figure 11 shows the speedup for the step of matrices generation and assemblage, the step of GMRES solver, and the total time. This result also demonstrates that the usage of parallel techniques into FE methods can tremendously reduce the computation time for parachute simulations.

As can be seen, the G12 problem does not gain much speedup increase when more than 8 processors are used. This is because of the relatively small size of the G12 model. For a certain sized problem, tremendous communication among processors is needed when too many processors are used. The ratio of communication time vs FE analysis time becomes unreasonably big and speedup drops. This proves that the choice of processors number for a certain problem to achieve maximum efficiency is dependent on the problem size.

Comparing Figs. 8 and 11, we see that the G12 problem achieves better speedup than the airbag inflation problem in total analysis and the iterative solver part. The reason is that the speedup of the parallel code is dependent on the size of the problem. The problem with bigger size has a smaller ratio of the communication time vs total time, assuming the same number of processors is used.

We also see that the speedup for the G12 problem is worse than the airbag inflation problem in generating and assembling the matrix, when same number of processors is used. This is caused by different element types involved. The airbag model comprises only uniform membrane elements. The computation load on each processor is equal because the number of elements on each processor is

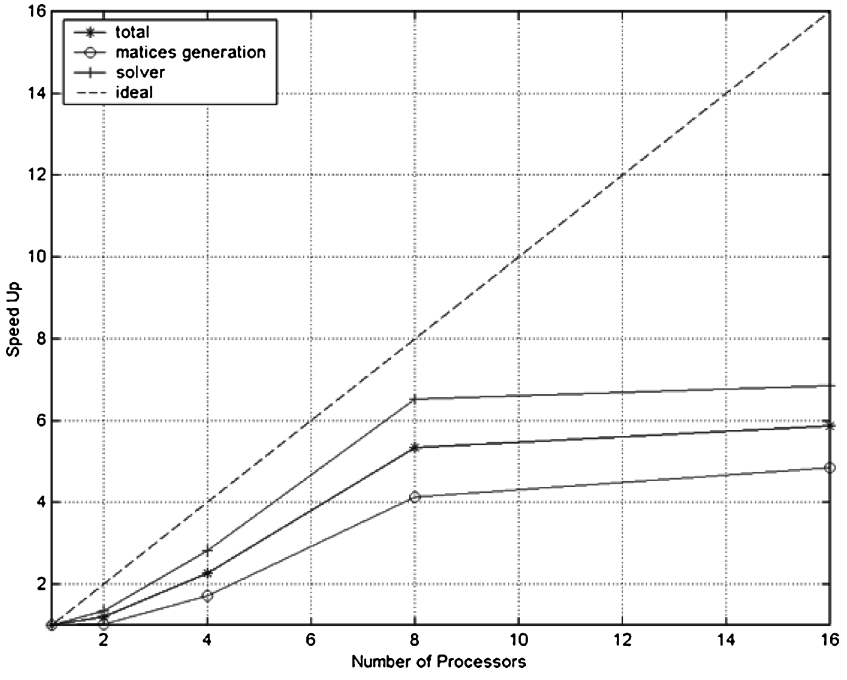

Fig. 11 Speedup analysis for the simulation of a G12 parachute.

equal. No synchronization delay is introduced. On the other hand, the G12 parachute model has different types of membrane, cable, and payload elements. The types of elements on each processor are very different, although the number of elements is the same. Generating and assembling matrix for membrane elements is much more complex and time consuming than for cable and payload elements. Some processors with mostly cable elements and payload elements may finish their job much earlier than those with membrane elements and have to wait. Synchronization delay is caused and speedup drops. For example, when 2 processors are used, the types of the elements on these two processors are very different, although the numbers of elements allocated are both 2614. The elements on processor 0 include all of the 2182 membrane elements and 432 cable elements, whereas the elements on processor 1 include 2605 cable elements and 9 payload elements. Processor 1 can finish its job much earlier than processor 0 and has to be waiting idle. A significant amount of processor 1's computation power is wasted, and as a result, the speedup is very low. In future research, more efforts are needed to balance the load among processors by balancing both the number and the type of elements among processors.

V. Conclusions

Parallel computation techniques used in FE simulations of nonlinear SD behavior for flexible structures were introduced. An iterative FE procedure using the HHT method was presented. Parallel techniques derived from the geometrical parallelism algorithm were developed and realized in an FE code. The parallel storage strategies were explained. The parallel implementations of generating the tangent stiffness matrix and the effective force vector, solving the linear simultaneous system equations, and other miscellaneous operations were also performed.

Two sample problems were presented to show the mechanical features of the FE model and the speedup of the parallel algorithm. The results obtained demonstrate the ability of the parallel FE code developed in this research to simulate the nonlinear SD behavior for flexible structures, such as parachute systems. The computation time for the SD analysis is reduced tremendously by incorporating parallel techniques into FE code. This is essential to the ultimate goal of reducing the parachute design cost. In the future, efforts should be made to improve the current preliminary parallel FE model to make it more robust and efficient. This includes balancing computation load among processors according to both the number and the type of elements on each processor, making many miscellaneous operations parallel, and implementing in parallel new mechanical features into the current preliminary parallel FE model.

With parallel implementations of CFD analysis and SD analysis available, parallel fluid structural interface (FSI) simulations, which are the most accurate ways to simulate transient nonlinear parachute behaviors,[11] become possible. Coupling of fluid mesh in which parachute systems are merged with structural mesh of parachute systems themselves are achieved by transferring FSI information between them using an interface surface.[16] Nodal displacements from SD analysis are transferred from the structural model to the interface and then used to update the fluid mesh, whereas pressure and fluid drag computed from the fluid model are employed as the external loads on the structural model. An iterative procedure is used to fulfill the coupling. Information change between CFD mesh and SD mesh takes place at every FSI step. Multiple CFD time steps and SD time steps may occur at each FSI step.[16] When parallel techniques are taken advantage of, the CFD and SD analysis between two consecutive coupling phases can be performed on large number of high-speed computer processors simultaneously. Tremendous parachute design time and cost can be saved as a result. More studies will be carried out on the implementation of parallel FSI simulation to predict efficiently more realistic parachute behavior in the future.

Acknowledgment

This research work was sponsored by the U.S. Air Force of Scientific Research under Contract F49620-98-1-02/4 to the University of Connecticut.

References

[1]Chiang, K. N., and Fulton, R. E., "Concepts and Implementation of Parallel Finite Element Analysis," *Computers and Structures*, Vol. 36, No. 6, 1990, pp. 1039–1046.

[2]Baddourah, M. A., and Nguyen, D. T., "Parallel-Vector Computations for Geometrically Nonlinear Finite Element Analysis," *Computers and Structures*, Vol. 51, No. 6, 1994, pp. 785–789.

[3]Jaques, M. W. S., Ross, C. T. F., and Strickland, P., "Exploiting Inherent Parallelism in Nonlinear Finite Element Analysis," *Computers and Structures*, Vol. 59, No. 4, 1996, pp. 801–807.

[4]Danielson, K. T., and Namburu, R. R., "Nolinear Dynamic Finite Element Analysis on Parallel Computers Using FORTRAN 90 and MPI," *Advances in Engineering Software*, Vol. 29, No. 3–6, 1998, pp. 179–186.

[5]Topping, B. H. V., and Khan, A. I., *Parallel Finite Element Computations*, Saxe–Coburg Publications, Stirling, Scotland, U.K., 1996.

[6]Gummadi, L. N. B., and Palazotto, A. N., "Nonlinear Finite Element Analysis of Beams and Arches Using Parallel Processors," *Computers and Structures*, Vol. 63, No. 3, 1997, pp. 413–428.

[7]Behr, M., and Tezduyar, T. E., "Finite Element Solution Strategies for Large-Scale Flow Simulations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 112, No. 1–4, 1994, pp. 3–24.

[8]Chiang, K. N., and Fulton R. E., "Structural Dynamics Methods for Concurrent Processing Computers," *Computer and Structures*, Vol. 36, No. 6, 1990, pp. 1031–1037.

[9]Kumar, V., Grama, A., Gupta, A., and Karypis, G., *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, New York, 1994.

[10]Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, 1997.

[11]Accorsi, M. L., Leonard, J. W., Benney, R., and Stein, K., "Structural Modeling of Parachute Dynamics," *AIAA Journal*, Vol. 38, No. 5, 2000, pp. 139–146.

[12]Lu, K., "Enhanced Membrane Elements for Simulation of Parachute Dynamics," Ph.D. Dissertation, Civil and Environmental Engineering Dept., Univ. of Connecticut, Storrs, CT, Aug. 2000.

[13]Hughes, T. J., *The Finite Element Method—Linear Static and Dynamic Finite Element Analysis*, Prentice–Hall, Englewood Cliffs, NJ, 1987.

[14]Bathe, K. J., *Finite Element Procedures*, Prentice–Hall, Upper Saddle River, NJ, 1996.

[15]Kalro, V. J., "3D Flow Analysis and Applications with Parallel Finite Element Methods," Ph.D. Dissertation, Univ. of Minnesota, Minneapolis, MN, Nov. 1996.

[16]Stein, K. R., "Simulation and Modeling Techniques for Parachute Fluid-Structure Interactions," Ph.D. Dissertation, Univ. of Minnesota, Minneapolis, MN, Dec. 1999.

[17]Saad, Y., and Schultzm, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *Society for Industrial and Applied Mathematics*, Vol. 7, No. 3, 1986, pp. 856–869.

[18]Spyropoulos, A. N., Boudouvis, A. G., and Markatos, N. C., "An Implementation of Parallel Preconditioned GMRES(m) on Distributed Memory Computers," *Computational Fluid Dynamics '98, European Computational Fluid Dynamics Conference*, 4th ed., Wiley, New York, 1998, pp. 478–483.

[19]Vuik, C., Van Nooyen, R. R. P., and Wesseling, P., "Parallelism in ILU-Preconditioned GMRES," *Parallel Computing*, Vol. 24, No. 14, 1998, pp. 1927–1946.

[20]Tezduyar, T. E., Behr, M., Aliabadi, S. K., Mittal, S., and Ray. S. E., "A New Mixed Preconditioning Method for Finite Element Computations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 99, No. 1, 1992, pp. 27–42.

S. Saigal
Associate Editor