ELSEVIER

Comput. Methods Appl. Mech. Engrg. 121 (1995) 137–162

Computer methods in applied mechanics and engineering

Recursive spectral algorithms for automatic domain partitioning in parallel finite element analysis

Shang-Hsien Hsieh[a,*,1], Glaucio H. Paulino[b,2], John F. Abel[b,3]

[a]*School of Civil Engineering, 1284 Civil Engineering Building, Purdue University, West Lafayette, IN 47907-1284, USA*

[b]*School of Civil and Environmental Engineering, Hollister Hall, Cornell University, Ithaca, NY 14853-3501, USA*

Received 19 July 1993; revised manuscript received 24 January 1994

Abstract

Recently, several domain partitioning algorithms have been proposed to effect load-balancing among processors in parallel finite element analysis. The recursive spectral bisection (RSB) algorithm [1] has been shown to be effective. However, the bisection nature of the RSB results in partitions of an integer power of two, which is too restrictive for computing environments consisting of an arbitrary number of processors. This paper presents two recursive spectral partitioning algorithms, both of which generalize the RSB algorithm for an arbitrary number of partitions. These algorithms are based on a graph partitioning approach which includes spectral techniques and graph representation of finite element meshes. The 'algebraic connectivity vector' is introduced as a parameter to assess the quality of the partitioning results. Both node-based and element-based partitioning strategies are discussed. The spectral algorithms are also evaluated and compared for coarse-grained partitioning using different types of structures modelled by 1-D, 2-D and 3-D finite elements.

1. Introduction

To take advantage of distributed-memory parallel-processing in finite element analysis, the domain is usually partitioned (or decomposed) into a number of subdomains which are distributed among the processors. The computation for each subdomain is carried out by a separate processor. The key problems of this approach are how to partition the domain to achieve well-balanced workload distribution among processors and how to minimize the amount of interprocess communication so that significant speed-up can be obtained in the parallel analysis.

The problem of domain partitioning of finite element meshes is equivalent to the problem of partitioning the graph associated with the mesh. Graph partitioning is an important NP-complete problem [2] and so is the general nature of mesh partitioning. Therefore, obtaining optimal solutions is practically intractable, but is also unnecessary because satisfactory near-optimal solutions can always be sought at a relatively low cost by the use of fast and efficient heuristic algorithms.

Since the finite element meshes considered may be large and generic (irregular shapes, including multiply connected and/or branched domains), manual partitioning may be difficult even with the help of interactive computer graphics. Simon [1] has shown that visual perception alone may be inadequate for the task of partitioning large three-dimensional structures. Therefore, automatic algorithms are generally required.

* Corresponding author.

[1] Postdoctoral Research Associate; formerly Graduate Research Assistant at the School of Civil and Environmental Engineering and the Program of Computer Graphics, Cornell University.

[2] Graduate Research Assistant.

[3] Professor of Structural Engineering.

Elsevier Science S.A.
SSDI 0045-7825(94)00704-7

Several automatic partitioning algorithms have been proposed. For example, Flower et al. [3] presented a simulated annealing method for mapping irregular finite element domains to parallel processors. Although these types of methods can give almost optimal results, they are usually expensive and time-consuming, especially for problems of large size. Malone [4] proposed an automated mesh decomposition method for transient dynamic analysis on hypercube multiprocessor computers using an explicit time integrator. The method is called reduced bandwidth decomposition (RBD) and is based on a scheme which reduces the bandwidth of the matrix representation of the nodal connectivities in the mesh. Farhat [5] also proposed an automatic finite element domain decomposer which is based on a greedy algorithm and seeks to decompose a finite element mesh into a set of balanced domains sharing a minimum number of common nodal points. In an attempt to avoid domain splitting problems, Al-Nasra and Nguyen [6] incorporated geometrical information of the finite element meshes into an automatic decomposition algorithm similar to the one proposed by Farhat [5]. Padovan and Kwang [7] developed a direct element connect filling (DECF) scheme which employs multiple starting nodes and proceeds with an algorithm similar to the one by Farhat [5], simultaneously for each starting node, to avoid possible domain splitting. The symmetry of the finite element domain is considered in the selection of starting nodes and during the partitioning process. Recently, Miler et al. [8] developed a partitioning method which makes use of the geometric structure of a given finite element or finite difference mesh to find the desired partitions.

Simon [1] proposed the recursive spectral bisection (RSB) algorithm for hypercube architectures, which consist of 2^d processors (d is the dimension of the hypercube). Simon [1] also compared the RSB algorithm with two other algorithms: the recursive coordinate bisection (RCB) and the recursive graph bisection (RGB) algorithms, and showed the superiority of the RSB algorithm over both the RCB and the RGB algorithms. Recently, Hendrickson and Leland [9, 10] extended the spectral bisection through the use of multiple eigenvectors to allow for partitioning of a domain into 4 or 8 subdomains at each stage of a recursive decomposition. They proposed the recursive spectral quadrisection (RSQ) algorithm for 4^k partitions and the recursive spectral octasection (RSO) for 8^k partitions; in each, k is a positive integer number. They have also concluded that the RSQ and RSO algorithms are less expensive than multiple levels of the RSB algorithm. However, all these algorithms (RSB, RSQ and RSO) have been specifically developed for hypercube or mesh machine architectures.

Williams [11] evaluated the performance of three partitioning algorithms for dynamic load-balancing using a 16-processor NCUBE machine. He compared the simulated annealing, the orthogonal recursive bisection and the eigenvector recursive bisection (analogous to the RSB) methods. He concluded that the eigenvector recursive bisection seems to be a good compromise between the other two methods.

In numerical comparative studies using structures of different types, Hsieh [12] has demonstrated that the RSB algorithm is more effective than the algorithms proposed by Al-Nasra and Nguyen [6], Farhat [5] and Malone [4] for coarse-grained partitioning. A main reason the spectral algorithms are so effective is that they use *global* properties of the graph associated with the mesh to perform the partitioning, i.e. a spectral analysis is conducted to compute separators based on eigenvector components of the graph. The algorithms by Al-Nasra and Nguyen [6], Farhat [5] and Malone [4] mainly use *local* information in the graph, such as the neighboring information of a vertex.

The bisection nature of the RSB algorithm is suitable for hypercube computers, but it is too restrictive for applications on coarse-grained parallel computing environments consisting of an arbitrary number of processors. To generalize the RSB algorithm for an arbitrary number of partitions, two new recursive spectral algorithms called the recursive spectral sequential-cut (RSS) algorithm and the recursive spectral two-way (RST) algorithm, respectively, are investigated in this paper.

Different solution methods used in the analysis may require different strategies for domain partitioning. Two types of partitioning strategies may be classified: *element*-based partitioning and *node*-based partitioning. The element-based partitioning focuses on partitioning elements in finite element meshes, while the node-based partitioning focuses on partitioning nodes. For parallel solutions of structural dynamics, implicit solution methods often require element-based partitioning (see, e.g., [13]), while explicit methods may require either type depending on the parallel implementation (see, e.g. [4, 14]). Both types of partitioning are considered in this paper.

The partitioning algorithms addressed in this paper are not specific to a single hardware configura-

tion, but instead are suitable for a variety of machine environments which share a few common features. The parallel computing environment used in this work consists of up to six DECsystem 5000s. These UNIX workstations are connected by Ethernet and communicate via the TCP/IP (Transmission Control Protocol/Internet Protocol). The interprocess communication and synchronization are achieved in a message-passing environment provided by ISIS [15]. This is a coarse-grained, distributed-memory environment where the number of processors used is small and no global memory is shared among processors.

The remainder of this paper is organized as follows. Section 2 presents relevant concepts of graph theory. This includes a discussion about spectral techniques applied to graphs and various graph representations of finite element meshes. Moreover, element-based and node-based partitioning strategies for implicit and explicit solution methods, respectively, in dynamic analysis are discussed. In Section 3, the spectral algorithms RSB, RSS and RST are presented. These algorithms are evaluated and compared in Section 4 using practical finite element examples such as a space station, an L-shaped building, a 12-bladed turbine disk, and a turbine blade. Finally, in Section 5, conclusions are drawn and directions for future work are discussed.

2. Preliminaries

For the purpose of discussion of domain partitioning algorithms, it is necessary to introduce some essential concepts and definitions of graph theory and to establish their relationship to matrix analysis and finite element analysis. These concepts and definitions are presented in this section. Next, various graph representations of finite element meshes and different partitioning strategies for implicit versus explicit dynamic analysis are also discussed.

2.1. Applied graph theory

This section presents some basic concepts and definitions of graph theory which are central to this paper. For further details the reader is referred to the books by Harary [16] and Deo [17].

A graph $G = (V, E)$ consists of a non-null finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ together with a set of edges $E = \{e_1, e_2, \ldots, e_m\}$ which are unordered pairs of distinct values form V; $E = \{(v_i, v_j) \,|\, v_i \in V, v_j \in V, i \neq j\}$. Edges are elements of the set E and $|E| = m$ is the cardinality of the set E. Vertices are elements of the set V and $|V| = n$ is the cardinality of the set V. A graph obeying the above definition is an undirected graph because the set E is comprised of unordered pairs of vertices.

The correspondence between a graph and a matrix can be established by considering a symmetric matrix K of order N with non-null diagonal components k_{ii}. The ordered and undirected graph of K is denoted by $G^K = (V^K, E^K)$. This graph has N vertices numbered from 1 to N, and $\{v_i, v_j\} \in E^K$ if and only if $k_{ij} = k_{ji} \neq 0$, $i \neq j$, where v_i denotes the vertex of E^K with label i.

The vertices u and v in G are *adjacent* vertices if $\{u, v\} \in E$. Two graphs G and H are said to be isomorphic ($G \cong H$) if there exists a one-to-one correspondence between their vertices which preserves adjacency, i.e. the graphs have the same topology.

The adjacent set of the subset W of the vertices of G, $\mathrm{Adj}(W)$, is defined as

$$\mathrm{Adj}(W) = \{u \in (V - W) \,|\, (u, v) \in E, v \in W, W \subset V\}$$

If $W = \{v\}$, $\mathrm{Adj}(W) = \mathrm{Adj}(v)$. The degree of the set W is defined as

$$\mathrm{Deg}(W) = |\mathrm{Adj}(W)|$$

Similarly, the degree of a vertex v is defined as

$$\mathrm{Deg}(v) = |\mathrm{Adj}(v)|$$

A path in a connected graph $G = (V, E)$ is an ordered set of vertices $(v_1, v_2, \ldots, v_{n+1})$ such that v_k and v_{k+1} are adjacent for $k = 1, \ldots, n$. This path has length n and it may also be interpreted as an

ordered set of n edges $(e_1, e_2, \ldots, e_n)$. The distance $d(u, v)$ between two vertices u and v of G is the length of the shortest path joining these vertices. The eccentricity of a vertex u is defined as

$$e(u) = \max\{d(u, v) \mid v \in V\}$$

The diameter of G is given by

$$\delta(G) = \max\{e(u) \mid u \in V\} = \max\{d(u, v) \mid u, v \in V\}$$

Consider, for example, the finite element mesh of Fig. 1(a) with nine nodes, four T3 elements, two Q4 elements, and no boundary conditions. For the sake of simplicity, assume one degree of freedom (dof) per node. The stiffness matrix representation associated with this mesh is given in Fig. 1(b) by the symmetric matrix K (of order nine) with non-null elements k_{ij} represented by X's. With respect to the corresponding (node) graph in Fig. 1(c), $V^K = \{1, \ldots, 9\}$ and $E^K = \{\{1, 2\}, \{1, 4\}, \{1, 5\}, \ldots, \{8, 9\}\}$, where each pair $\{i, j\}$ contains two adjacent vertices. If $W = \{1, 2, 3\}$, then $\mathrm{Adj}(W) = \{4, 5, 6\}$ and $\mathrm{Deg}(W) = 3$. For the sixth vertex, $\mathrm{Adj}(6) = \{2, 3, 5, 8, 9\}$ and $\mathrm{Deg}(6) = 5$. By inspection, one can easily verify that for this graph $\delta(G) = 3$.

Suppose that the vertex No. 5 in the graph G^K of Fig. 1(c) is relocated between the vertices No. 7 and No. 8, as shown in Fig. 1(d) by the graph H^L. The graphs G^K and H^L are isomorphic under the correspondence $i \leftrightarrow i - 10$ $(i = 1, \ldots, 9)$ for the vertex labels.

Next, consider the finite element mesh of Fig. 2(a) with fifteen nodes, eight T3 elements, four Q4 elements, and no boundary conditions. The connectivity of the nodes can be represented topologically by the node graph (NG) of Fig. 2(b). To establish the connectivity of the finite elements in a topological sense, a dual graph (DG) representation is used as illustrated by Fig. 2(c).

The geometric aspect of the dual graph is used here for the purpose of representing the connectivity of the finite elements in a generic mesh. The vertices in the dual graph represent finite elements in the original mesh. The edges in the dual graph represent adjacent finite elements that share a common boundary in the original mesh. According to this definition, a finite element of dimension n $(n = 1, 2, 3)$

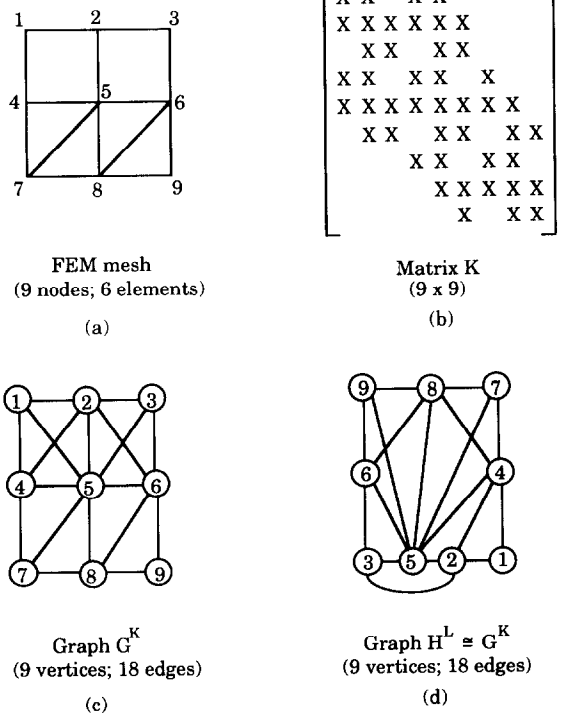
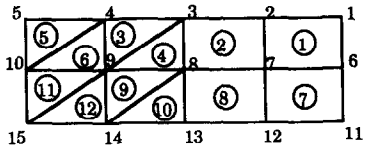
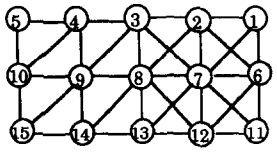


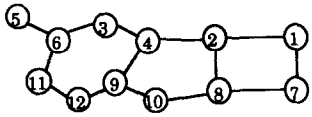
Fig. 1. Correspondence among mesh, matrix and graph.



(a) Finite Element Mesh (arbitrary nodal and element numbering)



(b) Node graph NG^A



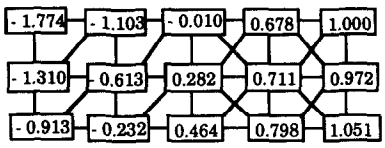
(c) Dual graph DG^A

$$\begin{bmatrix}
3&-1&0&0&0&-1&-1&0&0&0&0&0&0&0&0\\
-1&5&-1&0&0&-1&-1&-1&0&0&0&0&0&0&0\\
0&-1&5&-1&0&0&-1&-1&-1&0&0&0&0&0&0\\
0&0&-1&4&-1&0&0&0&-1&-1&0&0&0&0&0\\
0&0&0&-1&2&0&0&0&0&-1&0&0&0&0&0\\
-1&-1&0&0&0&5&-1&0&0&0&-1&-1&0&0&0\\
-1&-1&-1&0&0&-1&8&-1&0&0&-1&-1&-1&0&0\\
0&-1&-1&0&0&0&-1&7&-1&0&0&-1&-1&-1&0\\
0&0&-1&-1&0&0&0&-1&6&-1&0&0&0&-1&-1\\
0&0&0&-1&-1&0&0&0&-1&4&0&0&0&0&-1\\
0&0&0&0&0&-1&-1&0&0&0&3&-1&0&0&0\\
0&0&0&0&0&0&-1&-1&-1&0&0&-1&5&-1&0&0\\
0&0&0&0&0&0&0&-1&-1&0&0&0&-1&4&-1&0\\
0&0&0&0&0&0&0&0&-1&-1&0&0&0&-1&4&-1\\
0&0&0&0&0&0&0&0&0&-1&-1&0&0&0&-1&3
\end{bmatrix}$$

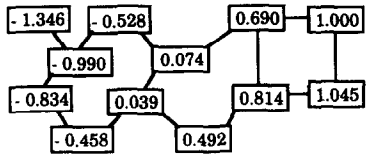
(d) $L(NG^A)$

$$\begin{bmatrix}
2&-1&0&0&0&0&-1&0&0&0&0&0\\
-1&3&0&-1&0&0&0&-1&0&0&0&0\\
0&0&2&-1&0&-1&0&0&0&0&0&0\\
0&-1&-1&3&0&0&0&0&-1&0&0&0\\
0&0&0&0&1&-1&0&0&0&0&0&0\\
0&0&0&-1&0&-1&3&0&0&0&0&-1&0\\
-1&0&0&0&0&0&0&2&-1&0&0&0&0\\
0&-1&0&0&0&0&0&-1&3&0&-1&0&0\\
0&0&0&-1&0&0&0&0&3&-1&0&-1\\
0&0&0&0&0&0&0&0&-1&-1&2&0&0\\
0&0&0&0&0&0&0&0&0&-1&0&0&0&2&-1\\
0&0&0&0&0&0&0&0&0&0&-1&0&-1&2
\end{bmatrix}$$

(e) $L(DG^A)$



(f) Fiedler vector for $L(NG^A)$ ($\lambda_2 = 0.639$)



(g) Fiedler vector for $L(DG^A)$ ($\lambda_2 = 0.264$)

Fig. 2. Spectral analysis.

has boundaries of dimension $(n-1)$. As an example, by applying this definition to the finite element mesh of Fig. 2(a), the dual graph of Fig. 2(c) is obtained.

Spectral techniques for graph partitioning

A spectral method, based on algebraic properties of the graph associated with the finite element mesh, can be used to solve the partitioning problem [18]. The spectral method associates an adequate graph representation (G) to the finite element mesh and forms the Laplacian matrix $L(G)$. Here, G is assumed to be a connected graph. The spectral properties of the Laplacian matrix are of special interest to the present work. For example, a particular eigenvector of this matrix can be used to partition the vertices of a graph into two sets.

The Laplacian matrix $L(G)$ is a symmetric matrix of order N, where $N = |V|$. The components l_{ij} of $L(G)$ are defined as

$$l_{ij} = \begin{cases} -1 & \text{if } \{v_i, v_j\} \in E \\ \text{Deg}(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

From this definition, it follows that the Laplacian matrix $L(G)$ can be obtained from the degree matrix $D(G)$ and the adjacency matrix $A(G)$ of a given labeled graph G

$$L(G) = D(G) - A(G)$$

where $D(G)$ is a diagonal matrix of order N with diagonal components $d_{ii} = \mathrm{Deg}(v_i)$, and $A(G)$ is an adjacency matrix of order N with components $a_{ij} = 1$ if $\{v_i, v_j\} \in E$ and zero otherwise. Fig. 2(d) shows the Laplacian matrix $L(NG^A)$ associated with the node graph NG^A, and Fig. 2(e) shows the Laplacian matrix $L(DG^A)$ associated with the dual graph DG^A.

The Laplacian matrix $L(G)$ is positive semidefinite and it determines G up to isomorphism. Let the eigenvalues of $L(G)$ be ordered as

$$0 = \lambda_1 \leqslant \lambda_2 \leqslant \cdots \lambda_N \leqslant |V| = N$$

The smallest eigenvalue is always $\lambda_1 = 0$ and the associated eigenvector y_1 has all of its normalized elements equal to 1. If G is connected, the second eigenvalue λ_2 is always non-zero and positive. The eigenvalue λ_2 is designated 'algebraic connectivity' and is related to the vertex and edge connectivities of the graph. The components of the second eigenvector y_2 are associated with the corresponding vertices of the graph. Differences in the values of the components of y_2 give topological distance information about the vertices of the graph. The components of y_2 also provide a weighting for these vertices which can be used for partitioning the graph. For example, all vertices with weights below the median weight may be assigned to one partition, and the rest to the other partition.

The special properties of the second eigenvalue λ_2 and its corresponding eigenvector y_2 have been studied by Fiedler [19] and Anderson and Morley [20], among other researchers. The second eigenvector y_2 is called the Fiedler vector by Simon [1]. (It should be noted that Simon uses the opposite sign for the Laplacian matrix defined above, probably because it may be more convenient from a numerical point of view.)

Fig. 2(f) and (g) show both the Fiedler vector y_2 and the algebraic connectivity λ_2 for the node graph NG^A and the dual graph DG^A. In Fig. 2(f), a bisection of the vertices of the graph can be executed by assigning the vertices with values smaller than the median weight (= 0.282) to one subdomain and the remaining to the other. The same procedure can be used to partition the vertices in Fig. 2(g) into two sets.

Algebraic connectivity of the partitioning

The algebraic connectivity (λ_2) of a graph (or subgraph) serves as a measure of connectivity of the graph (or subgraph). Here, we introduce the algebraic connectivity vector (ACV) for the purpose of investigating the algebraic connectivity of the partitioning

$$\mathbf{AC}_{\lambda_2}^{N_p}[\mathrm{Alg}(G)] = (\lambda_2^1, \lambda_2^2, \ldots, \lambda_2^{N_p})$$

where N_p is the number of partitions, Alg denotes the partitioning algorithm being used and G denotes a graph associated with the finite element mesh. Each element of the ACV represents a measure of the algebraic connectivity of the ith partition, λ_2^i, $i \in [1, N_p]$. Some comments about relationship of λ_2 with the size and connectivity of a finite element mesh are given in the following. First, consider a square grid with $n \times n$ (n is a positive integer number) Q4 elements and its associated NG. As n increases, λ_2 decreases because the diameter of the associated NG increases. Next, for the same $n \times n$ grid, if Q8 elements are used instead of Q4 elements, λ_2 increases. This is because $|E|$ is larger for the associated NG when Q8 elements are used.

The ACV gives useful information for assessing load balancing among processors. If a graph (corresponding to a finite element domain) is partitioned into well-balanced subgraphs, one would expect that all the elements of the ACV are approximately equal, i.e. all the subdomains have nearly the same value for λ_2^i, $i \in [1, N_p]$. This means that the normalized ACV would be close to a unity vector.

The ACV gives conclusive information about the occurrence of the domain splitting problem. This problem occurs in the ith partition (or subgraph) when $\lambda_2^i = 0$, where $i \in [1, N_p]$. Moreover, the multiplicity of the null eigenvalue is equal to the number of distinct connected components in the

specific partition which has domain splitting problems. If the subgraph is connected, its algebraic connectivity is always positive.

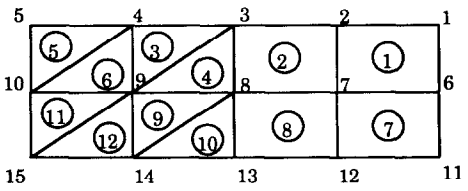
The amount of information contained in the ACV should not be overstated because isomorphic graphs are isospectral (or cospectral) graphs, i.e. graphs with the same spectrum. (However, isospectral graphs are not necessarily isomorphic.) In this sense, the ACV can be viewed as a useful but not conclusive parameter for evaluating the quality of domain partitioning.

2.2. Graph representation of finite element meshes

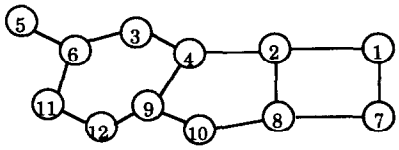
One approach to partition a generic finite element mesh is to associate a proper graph representation with the mesh and to partition the graph. There are several ways to associate graphs with meshes. In particular, three types of graphs associated with a mesh are considered here: the node graph (NG), the dual graph (DG) and the communication graph (CG).

An advantage of using the DG for partitioning of finite element meshes is that the number of edges in the connectivity graph is reduced because the DG defines the connectivity of the finite elements by means of their boundaries instead of their nodes. This leads to smaller data storage requirement and a lower-order algebraic eigenvalue problem (compare Figs. 2(b) and (c)). However, for finite element meshes consisting of only 1-D finite elements such as framed structures, the elements are connected to their adjacent elements through 0-D boundary nodes (pointwise). Therefore, there is no clear advantage in using the DG approach for domain partitioning of meshes of 1-D elements.

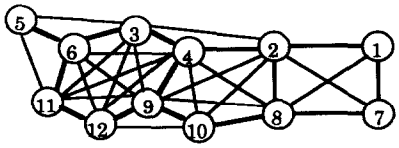
A disadvantage of the DG approach is that it does not represent the true interprocess communication in parallel finite element analysis. The actual interprocess communication occurs across shared nodes in the original finite element mesh, but not shared boundaries (e.g. edges between 2-D elements or faces between 3-D elements). To better describe the communication pattern, Venkatakrishnan et al. [21] proposed the use of a communication graph (CG) which is defined as follows. The vertices in the communication graph represent finite elements in the original mesh. The edges in the CG represent adjacent finite elements that share a common node in the original mesh. Fig. 3(a) and (b) show the same finite element mesh and its associated dual graph (DG^A) as in Figs. 2(a) and (c), respectively. Fig.



(a) Finite element mesh



(b) Dual graph DG^A



(c) Communication graph CG^A

Fig. 3. Correspondence among mesh, dual graph and communication graph.

3(c) shows the associated communication graph (CG^A). The heavy lines in Fig. 3(c) show that DG^A is contained in CG^A.

Both the DG and the CG do not differentiate among different types of finite elements. For example, if one or more Q4 elements in Fig. 3(a) are replaced by Q8 elements, both the DG and the CG remain the same as in Figs. 3(b) and (c), respectively. Therefore, the partitioning algorithms employing these graphs may not produce partitions with well-balanced computational loads for generic meshes with mixed finite element types. Nevertheless, in this case, the CG is expected to provide better results than the DG because of its more realistic representation of interprocess communication.

For node-based partitioning, since the focus is on partitioning nodes in finite element meshes, the NG seems to be a natural choice. The use of the NG is advantageous for meshes consisting of only 1-D finite elements (e.g. framed structures) because meshes of this type usually have a fewer number of nodes than of elements, so the NG requires less data storage. However, the opposite is true for meshes consisting of 2-D or 3-D elements, especially of higher interpolation order.

The best choice among different possible graph representations may depend on several factors such as the type of partitioning desired (node-based or element-based), the parallel solution algorithms (explicit or implicit), the size and type of the structure, and the effectiveness and computational cost of the partitioning algorithm. This topic is further investigated in the remainder of this paper.

2.3. Element-based versus node-based partitioning

As discussed previously, two different types of partitioning strategies, *element*-based partitioning and *node*-based partitioning, are required for implicit and explicit solution methods, respectively. For the transient structural dynamics analyses adopted as prototype problems in the present paper, the parallel central difference method proposed by Hajjar and Abel [14] is considered an example of explicit solution methods which require the node-based partitioning, while a slightly modified version of the parallel Newmark constant average acceleration method [12] described by Hajjar and Abel [13] is considered an example of implicit solution methods which require the element-based partitioning. Descriptions of these two parallel solution methods and their partitioning requirements are briefly discussed next.

Parallel explicit analysis

The parallel central difference method developed by Hajjar and Abel [14] has been implemented for explicit solutions of structural dynamics. The formulation for the central difference method is well known [22, 23] and is not repeated here. The central difference algorithm is inherently amenable to parallel processing. With the use of lumped masses to yield a diagonal mass matrix, the solution scheme may proceed on a degree-of-freedom level without assembly of the global matrices and solution of simultaneous equations.

To take advantage of parallel processing, the finite element domain is first partitioned into a number of subdomains which are then distributed among the processors, and the computation involved in each subdomain is carried out by a separate processor. The structural partitioning needed by the present central difference algorithm is shown in Fig. 4 for a simple two-dimensional frame. The elements in a particular substructure are either interior elements if both their nodes lie completely within the boundary of the substructure, or border elements if one of their nodes is resident in a neighboring substructure. The interior nodes in a particular substructure are nodes lying within the boundary of the substructure. The boundary nodes are a subset of the interior nodes and connected to at least one border element. The adjacent nodes (see Fig. 4) are also connected to at least one border element, but lie outside the boundary of the substructure.

With this partition of structural data, the interprocess communication can be minimized by including the border elements on both of the processors associated with neighboring substructures. In this case, only one nearest-neighbor communication for exchanging the displacements of the boundary nodes is required per time step. This communication efficiency is achieved at the expense of a duplication of effort to perform the element calculations for the border elements on the processors sharing these elements. This approach is suitable for parallel analyses in which the number of border elements is

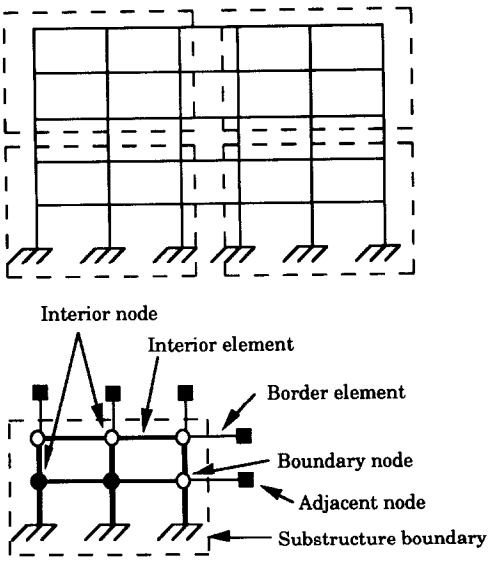


Fig. 4. Node-based partitioning of framed structural data for the parallel explicit algorithm [14].

significantly smaller than the number of interior elements and the cost of interprocess communication is relatively high. This is usually the case for the type of problem and environment characterizing the current work.

Parallel implicit analysis

The present research implements a slightly modified version of the parallel Newmark constant average acceleration method [12] described by Hajjar and Abel [13] for implicit transient solution of structural dynamics. The formulation of the Newmark constant average acceleration method is also well known in the technical literature [22, 23] and is not repeated here. This parallel implicit method uses a domain decomposition approach within the Newmark time stepping outer loop. The domain decomposition approach starts with partitioning the structure into a number of subdomains and assigns each subdomain to a separate processor. Then, substructure condensation is carried out on each processor independently and concurrently without any interprocess communication. Finally, a condensed set of system equations associated with unknowns along subdomain interfaces is solved by a parallel diagonally-preconditioned conjugate gradient algorithm (for discussion of the preconditioned conjugate gradient algorithm, see [24]).

As discussed previously, the domain decomposition approach requires that the finite element domain be partitioned into a number of subdomains for substructuring analysis. The structural partitioning needed is shown in Fig. 5 for the same two-dimensional frame of Fig. 4. The nodes in a particular subdomain are either interior nodes, which lie within the boundary of the subdomain, or boundary nodes, which lie along the interfaces between subdomains. The boundary nodes are further categorized into primary and secondary boundary nodes. Each boundary node is a primary node in only one subdomain but, at the same time, a secondary boundary node in all other subdomains which share it. The interior elements in a particular subdomain are connected to at least one interior node. Finally, for 1-D finite elements only, a second category exists, boundary elements, which are those elements connected only to boundary nodes.

3. The spectral algorithms

Three recursive spectral partitioning algorithms are discussed in this section. The first one is the recursive spectral bisection (RSB) algorithm presented by Simon [1], while the other two are the recursive spectral sequential-cut (RSS) algorithm and the recursive spectral two-way (RST) algorithm

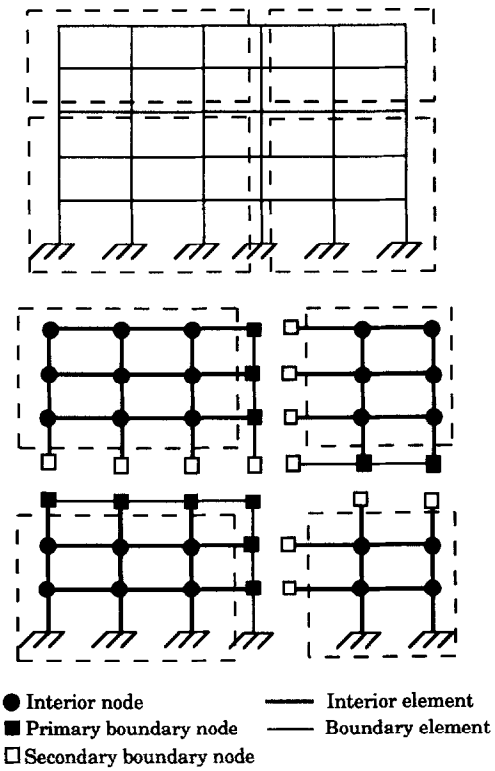


Fig. 5. Element-based partitioning of framed structural data for the parallel implicit algorithm [13].

both proposed by the writers. The RSS and RST algorithms employ the spectral partitioning method used by the RSB algorithm. However, unlike the RSB algorithm in which the number of partitions is restricted to an integer power of two, both the present algorithms can efficiently yield an arbitrary number of partitions.

Fig. 6 illustrates the overall approach of the spectral partitioning algorithms investigated in this work. This figure shows that the spectral partitioning method is used in a preprocessing stage of the parallel finite element analysis. Note that the element-based partitioning is naturally obtained from a DG or CG approach. The node-based partitioning can be obtained from a NG, DG or CG approach; in this case, if the DG or CG is used, common boundary nodes of adjacent subdomains must be assigned uniquely to a subdomain. The recursive part of the partitioning process consists of building a Laplacian matrix of the associated graph, computing the second eigenpair of the Laplacian matrix, and partitioning the graph according to the bisection, sequential-cut or two-way strategy. Note that these three strategies only differ in the way a simple domain is partitioned into two subdomains. This process is repeated until reaching a specified number of partitions (N_p). Finally, the results are collected into a database which can be used later for visualization of the partitions and by the parallel finite element analysis.

3.1. *Recursive spectral bisection (RSB) partitioning algorithm*

Based on the spectral partitioning method proposed by Pothen et al. [18], Simon [1] presented a recursive bisection algorithm for automatic partitioning of unstructured grids. To establish the partitioning of the grid, a DG representation has been used. The partitioning algorithm is called recursive spectral bisection (RSB) algorithm by Simon (1991) and is summarized in Table 1.

There are particularly two notable features of this algorithm. First, the DG is used to construct the partitioning problem. As discussed previously, this approach decreases significantly the size of the eigenproblem associated with the Laplacian matrix. As a result, the computational storage and execution time required to solve the eigensystem are reduced. Second, this algorithm employs the

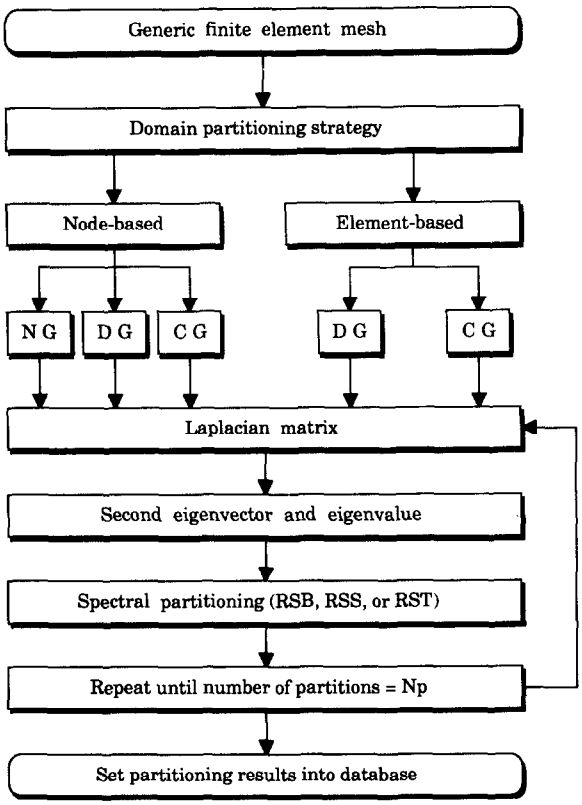


Fig. 6. Spectral domain partitioning method.

global information of the graph provided by the Fiedler vector to obtain an edge separator which partitions the graph into two subgraphs of nearly equal size (one vertex difference at most).

Excellent partitioning results have been obtained using this algorithm, as reported by Simon [1] and more recently by Hsieh [12]. It has also been shown by Pothen et al. [18] that the separators computed by this algorithm compare quite favorably with separators computed by other previously proposed algorithms.

However, there are some drawbacks associated with the RSB algorithm. First, the DG associated with a finite element mesh may be a disconnected graph. This situation may happen when n-dimensional finite elements are not connected through all their $(n-1)$-dimensional boundaries and when the adjacent finite elements do not have the same geometric dimensions [25]. Although theoretically this algorithm is capable of handling non-connected graphs [18], this situation may not only create difficulties and complexities in the partitioning process but also deteriorate the partitioning results.

Second, as discussed earlier, the DG does not represent the true interprocess communication in parallel finite element analysis. To better describe the communication pattern, the CG may be used. In the present work, an extended version of the RSB algorithm has been implemented, as illustrated by

Table 1
The recursive spectral bisection (RSB) algorithm [1]

(1) Construct the dual (DG), communication (CG), or node graph (NG) associated with the finite element mesh.[a]
(2) Compute the second eigenvector of the Laplacian matrix (Fiedler vector) of the graph using the Lanczos algorithm.
(3) Sort vertices of the graph according to the value of their associated components in the Fiedler vector.
(4) Assign half of the vertices to each subdomain.
(5) Repeat recursively for each subdomain.

[a] Only the dual graph is considered by Simon [1]. Both the dual and communication graphs are considered by Venkatakrishnan et al. [21].

Fig. 6. For the node-based partitioning, the NG, DG or CG can be used, while the DG or CG can be used for the element-based partitioning.

Third, the bisection nature of the RSB algorithm is suitable for hypercube computers in which the number of processors is an integer power of two, but it is too restrictive for computing environments consisting of an arbitrary number of processors, such as the networked workstation environment used by the writers.

The last drawback associated with the RSB method is related to its computational cost. In general, the solution of the second eigenvector (y_2) for large meshes can be quite expensive, even when the dual graph approach is used. Venkatakrishnan et al. [20] have compared the execution time of the RSB, RCB and RGB algorithms for a graph with 15 606 vertices on a Silicon Graphics workstation (Iris 4D/70). They found that the RSB algorithm is 'quite expensive' (1750 s for the RSB algorithm, while 4 and 3 seconds for the RCB and RGB algorithms, respectively) although the performance of the algorithm improves considerably on a vector computer such as the Cray Y-MP because matrix vector products in the Lanczos algorithm can be vectorized. Barnard and Simon [26] have recently developed a fast multilevel implementation of the RSB algorithm which is reported to attain 'about an order-of-magnitude improvement in run time on typical examples.' This multilevel approach has not been implemented in the present work.

3.2. *Recursive spectral sequential-cut (RSS) partitioning algorithm*

The recursive spectral sequential-cut (RSS) algorithm partitions the graph in such a way that subgraphs are cut out of the original graph one by one (or sequentially) in a recursive fashion. The algorithm is given in Table 2. The present implementation allows the use of the DG, CG or NG (see Fig. 6). Fig. 7 illustrates the process of partitioning a graph with seventeen vertices into five subdomains using this algorithm.

3.3. *Recursive spectral two-way (RST) partitioning algorithm*

Instead of using a bisection approach, the recursive spectral two-way (RST) algorithm uses a two-way partitioning approach which partitions the graph into two parts not necessarily equal in size. The algorithm is given in Table 3. The number of vertices in each subdomain D_i when the partitioning task is completed is denoted by m_i. This is computed in advance by sequentially employing the following equation for $i = 1, \ldots, N_p$

$$m_i = \left[\frac{N - \sum_{j=1}^{i-1} m_j}{N_p - (i-1)} \right] \quad (+\,1 \text{ if remainder} \neq 0)$$

in which N_p is the number of available processors and N is the total number of vertices of the whole domain. The number of partitions desired in the intermediate subdomain in each two-way partitioning step is denoted by n_p (initially $np = N_p$ for the whole domain). Each intermediate subdomain maintains a list l of subdomain numbers associated with D_i which has been assigned to it (initially the list is $\{1, \ldots, N_p\}$ for the whole domain). The kth component of this list is denoted by $l(k)$. Fig. 8

Table 2
The recursive spectral sequential-cut (RSS) algorithm

(1) Construct the dual (DG), communication (CG) or node graph (NG) associated with the finite element mesh.
(2) Compute the second eigenvector of the Laplacian matrix (Fiedler vector) of the graph using the Lanczos algorithm.
(3) Sort vertices of the graph according to the value of their associated components in the Fiedler vector.
(4) Assign $1/np$ of the vertices (discard remainder) to one subdomain and the remaining to the other, in which np is initially equal to the number of partitions (or processors) desired.
(5) Repeat recursively for the larger subdomain in the previous step with $np = np - 1$ until all subdomains are defined (i.e. $np = 1$).

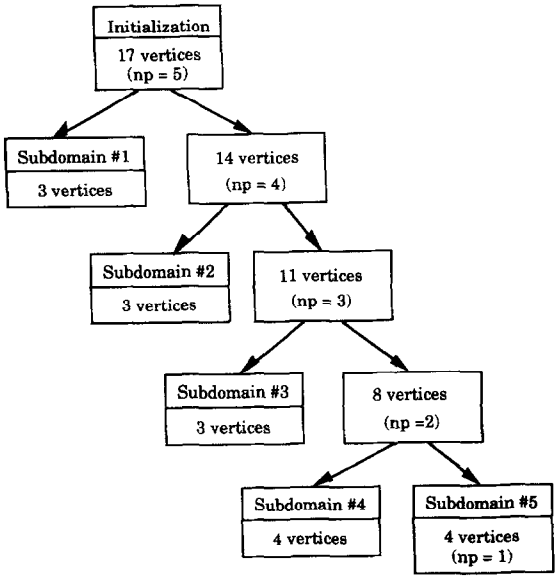


Fig. 7. Example of the RSS partitioning process.

demonstrates the process of partitioning a graph with seventeen vertices into five subdomains using this algorithm.

Comparing Tables 1 and 3, and making use of Fig. 8 as an example, one can easily verify that the RSB algorithm is obtained as a particular case of the more general RST algorithm when the number of processors (N_p) is equal to an integer power of two.

4. Numerical examples

The three partitioning algorithms discussed in the previous section have been evaluated and compared using different types of structures modelled by 1-D, 2-D and 3-D finite elements. Some typical results are reported here. The following nomenclature is adopted in the present study:

N_p = the number of partitions (or processors),
N_b = the number of boundary nodes in a subdomain,
N_n = the number of nodes in a subdomain,
N_e = the number of elements in a subdomain,
N_{e1} = the number of 1-D elements in a subdomain,
N_{e2} = the number of 2-D elements in a subdomain,

Table 3
The recursive spectral two-way (RST) algorithm

(1) Construct the dual (DG), communication (CG) or node graph (NG) associated with the finite element mesh.
(2) Compute the second eigenvector of the Laplacian matrix (Fiedler vector) of the graph using the Lanczos algorithm.
(3) Sort vertices of the graph according to the value of their associated components in the Fiedler vector.
(4) Compute the following integers:
$p1 = np/2$ (discard remainder)
$p2 = np - p1$
$n = \sum_{k=1}^{p1} m_{l(k)}$
Assign n vertices and the list of the first p1 components in $l(k)$ to one subdomain, and set $np = p1$ for this subdomain. Assign the remaining vertices and the list of the remaining components in $l(k)$ to the other subdomain, and set $np = p2$ for this subdomain.
(5) Repeat recursively for each subdomain with $np > 1$.

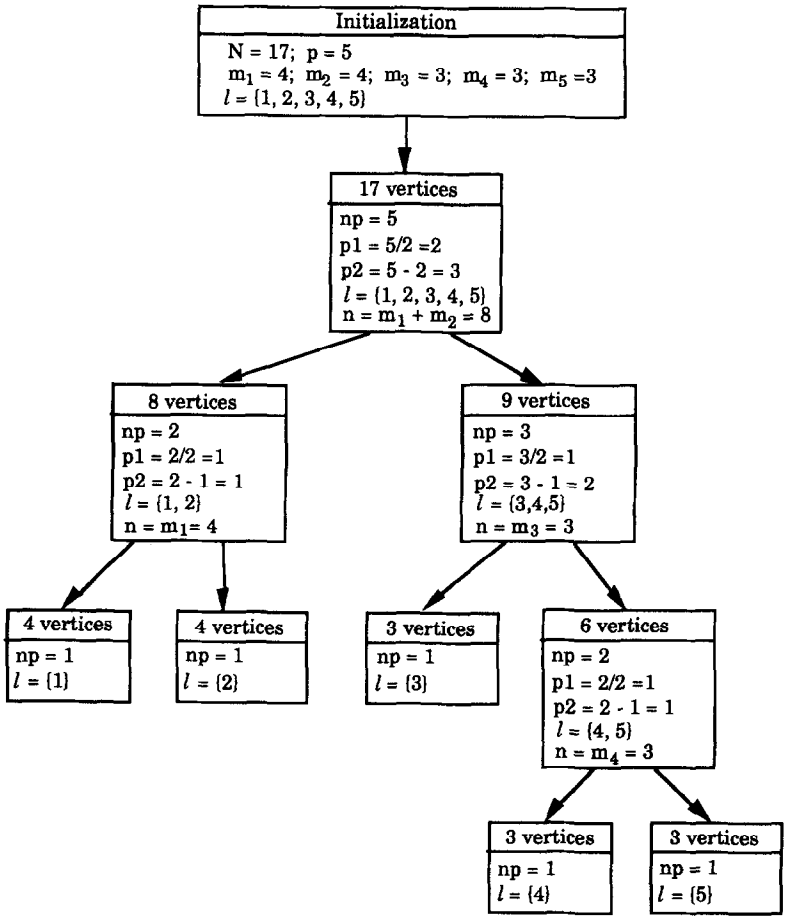


Fig. 8. Example of the RST partitioning process.

N_{e3} = the number of 3-D elements in a subdomain,
N_d = the number of subdomains that have disconnected regions,
Tot(x) = total values of x in the whole domain,
Sum(x) = summation of values of x over all subdomains,
Max(x) = maximum value of x among subdomains,
Min(x) = minimum value of x among subdomains,
NG = node graph,
DG = dual graph,
CG = communication graph,
T_{cpu} = the CPU time (s) required for partitioning on a single DECsystem 5000, which includes time spent in data preparation for the algorithm, execution of the algorithm, and setting results into the database.

$$S = \text{speed-up} = \frac{\text{Elapsed wall clock time for solution on one processor}}{\text{Elapsed wall clock time for solution on } N_p \text{ processors}}$$

$$E\ (\%) = \text{efficiency} = \frac{\text{Speed-up}}{N_p} \times 100$$

4.1. *Element-based partitioning for implicit analysis*

Four structures of different types have been used to evaluate and compare the partitioning algorithms discussed in the previous section. They include a framed structure modelled by 1-D elements, a building structure modelled by 1-D and 2-D elements, and two solid structures modelled by 3-D elements.

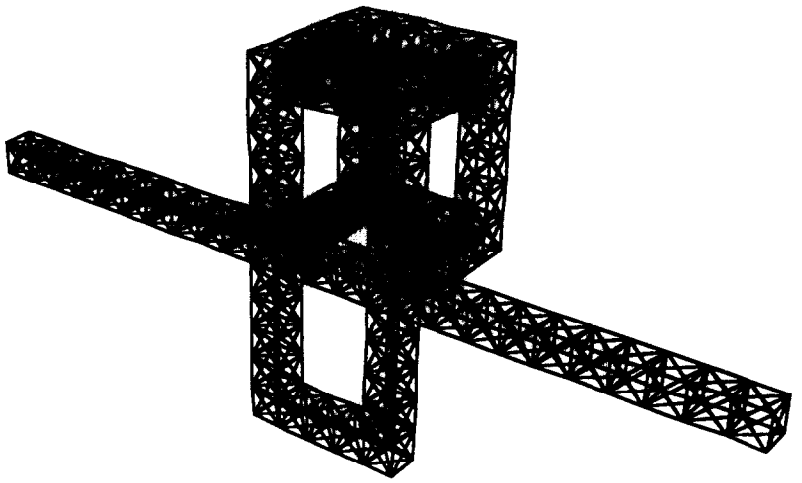


Fig. 9. The finite element model of a space station with 1428 1-D elements and 304 nodes.

In the comparative studies below, $\mathrm{Tot}(N_b)$ is used to indicate the amount of interprocess communication required in the parallel implicit analysis. It is also directly related to the size of the condensed set of system equations that is solved by a parallel preconditioned conjugate gradient algorithm. Therefore, the larger $\mathrm{Tot}(N_b)$ is, the more interprocess communication and synchronization are required. $\mathrm{Max}(N_b)/\mathrm{Min}(N_b)$ shows the balancing of communication loads among processors. $\mathrm{Max}(N_e)/\mathrm{Min}(N_e)$ indicates the balancing of the computational loads of element calculations and substructure condensation among processors. In addition, it is undesirable to have fragmented subdomains (which is indicated by N_d) because they usually result in longer subdomain boundaries which implies more communication overhead.

A space structure modelled by 1-D elements

The space station of Fig. 9 has been used to evaluate the performance of the partitioning algorithms. The results are given in Table 4 and the following observations are obtained:

(1) The CPU time spent by all the partitioning algorithms is only a few seconds, which is practically negligible when compared to the execution time required by a dynamic analysis.

(2) As expected, the RST algorithm produces the same results as the RSB algorithm when N_p is an integer power of two.

Table 4
Comparison of different algorithms for element-based partitioning of the space station of Fig. 9

N_p	Parameter	RSB	RSS	RST
3	T_{cpu}	–	15	14
	$\mathrm{Tot}(N_b)$	–	38	38
	$\mathrm{Max}(N_b)$	–	28	28
	$\mathrm{Min}(N_b)$	–	20	20
	N_d	–	0	0
	$\mathrm{Max}(N_{e1})$	–	476	476
	$\mathrm{Min}(N_{e1})$	–	476	476
4	T_{cpu}	15	18	15
	$\mathrm{Tot}(N_b)$	44	50	44
	$\mathrm{Max}(N_b)$	32	37	32
	$\mathrm{Min}(N_b)$	16	17	16
	N_d	1	1	1
	$\mathrm{Max}(N_{e1})$	357	357	257
	$\mathrm{Min}(N_{e1})$	357	357	357

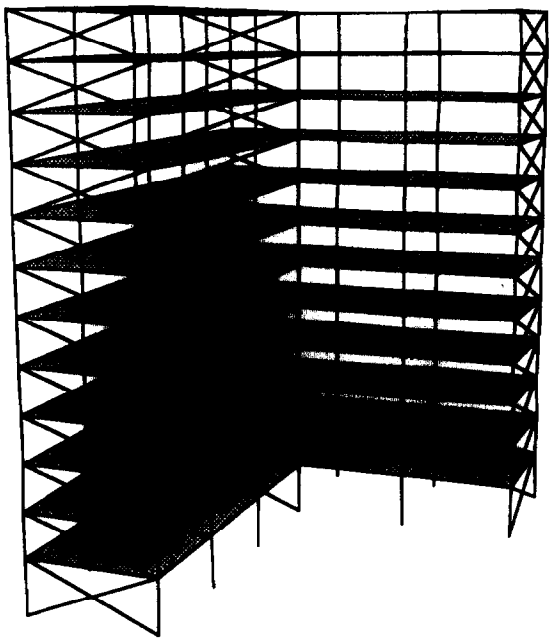


Fig. 10. A finite element model of a 12-story L-shaped building with 468 1-D beam-column elements, 72 2-D shell elements and 482 nodes.

(3) For all cases studied, the RST algorithm gives partitions with smaller or at least not larger $\mathrm{Tot}(N_b)$ than the RSS algorithm.
(4) Domain splitting (i.e. $N_d \neq 0$) occurs for all three algorithms for the case of $N_p = 4$.

A building structure modelled by 1-D and 2-D elements

The 12-story L-shaped building of Fig. 10 has been partitioned using different algorithms. The results are reported in Table 5. Although the present implementation assumes that all elements in the mesh have the same weight in the partitioning process, the use of the CG approach with all the three

Table 5
Comparison of different algorithms for element-based partitioning of the 12-story L-shaped building of Fig. 10

N_p	Parameter	RSB (DG)	RSB (CG)	RSS (DG)	RSS (CG)	RST (DG)	RST (CG)
3	T_{cpu}	–	–	3	3	3	3
	$\mathrm{Tot}(N_b)$	–	–	139	28	139	28
	$\mathrm{Max}(N_b)$	–	–	139	28	139	28
	$\mathrm{Min}(N_b)$	–	–	75	14	75	14
	N_d	–	–	1	0	1	0
	$\mathrm{Max}(N_{e1})$	–	–	180	156	180	156
	$\mathrm{Min}(N_{e1})$	–	–	108	156	108	156
	$\mathrm{Max}(N_{e2})$	–	–	72	24	72	24
	$\mathrm{Min}(N_{e2})$	–	–	0	24	0	24
4	T_{cpu}	3	4	4	4	3	4
	$\mathrm{Tot}(N_b)$	155	42	155	42	155	42
	$\mathrm{Max}(N_b)$	156	28	156	28	156	28
	$\mathrm{Min}(N_b)$	56	14	56	14	56	14
	N_d	1	0	1	0	1	0
	$\mathrm{Max}(N_{e1})$	135	117	135	117	135	117
	$\mathrm{Min}(N_{e1})$	63	117	63	117	63	117
	$\mathrm{Max}(N_{e2})$	72	18	72	18	72	18
	$\mathrm{Min}(N_{e2})$	0	18	0	18	0	18

algorithms considered still produces a satisfactorily balanced distribution of elements among subdomains in this example which represents a mixture of 1-D and 2-D elements (see, e.g. Fig. 11(a)). However, the use of the DG approach with all the three algorithms considered yields unacceptable distribution of elements among subdomains (see, e.g. Fig. 11(b)), mainly because the DG of the mesh in this case is a non-connected graph. In addition, both the RSS(CG) and RST(CG) algorithms give the same results and domain splitting does not occur. The RSS(DG) and RST(DG) algorithms also give the same results; however, domain splitting occurs.

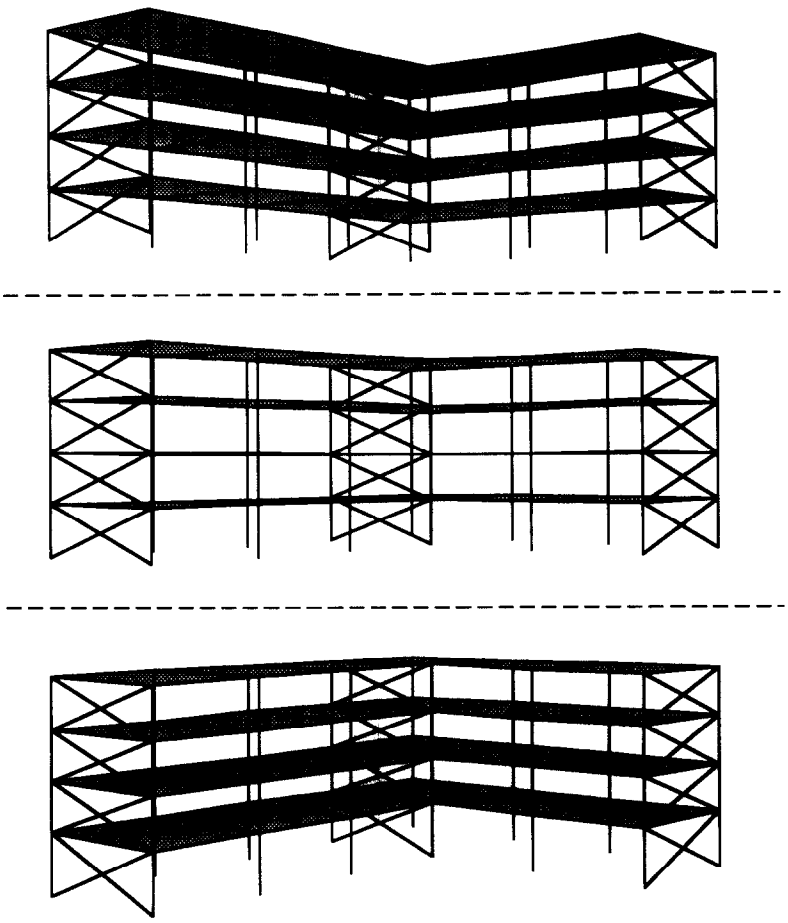
To further evaluate the quality of domain partitioning, the ACV is given for the RST(CG) algorithm and $N_p = 4$:

$$\mathbf{AC}^4_{\lambda_2}[\text{RST(CG)}] = (15.30, 15.20, 15.08, 15.19)$$

Note that all the elements of this vector are about the same value. This result is consistent with the statistics of Table 5 and the partitioning result shown in Fig. 11(a) where the RST(CG) algorithm is shown to produce well-balanced subdomains for this example.

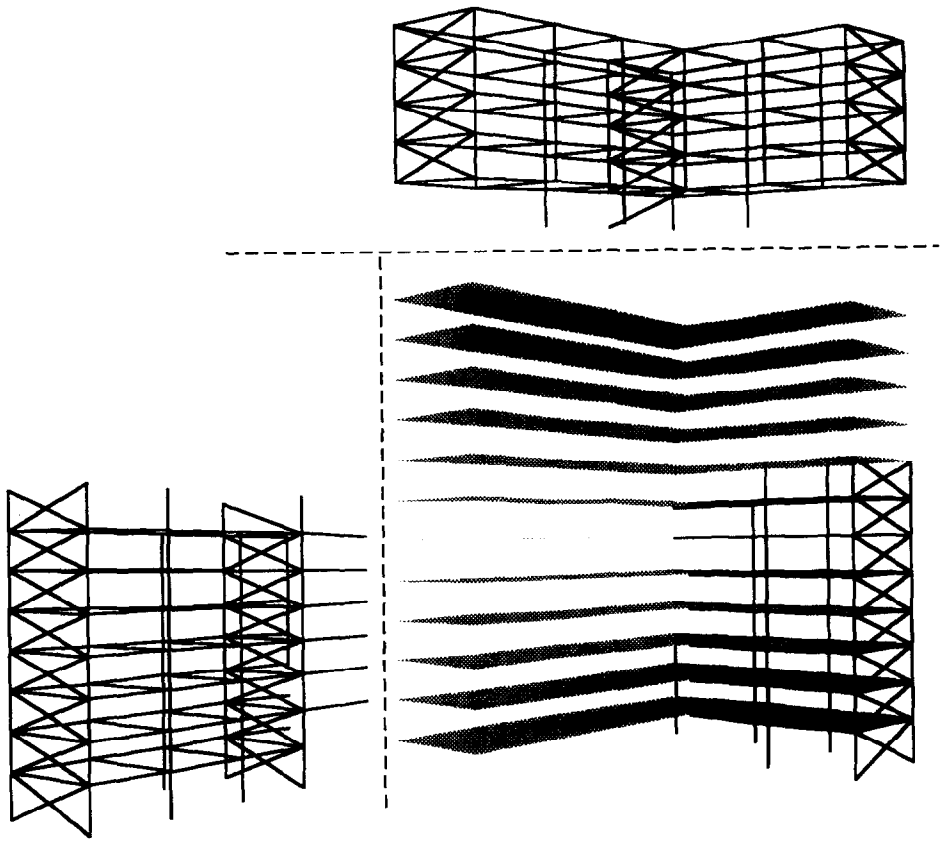
Solid structures modelled by 3-D elements

Two solid structures have also been used in the present comparative studies. The first one is the turbine blade of Fig. 12 created by Wawrzynek [27], while the second one is the 12-bladed turbine disk of Fig. 13. The partitioning results are given in Tables 6 and 7.



(a)

Fig. 11. Partitionings of the 12-story building of Fig. 10 by the (a) RSS(CG) and (b) RST(DG) algorithms.



(b)

Fig. 11 (Contd.)

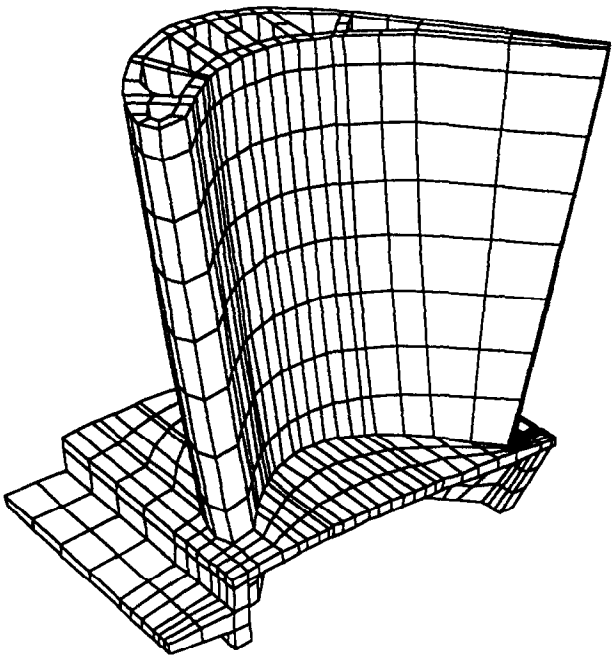


Fig. 12. A finite element model of a turbine blade with 944 20-noded 3-D elements and 6427 nodes [27].

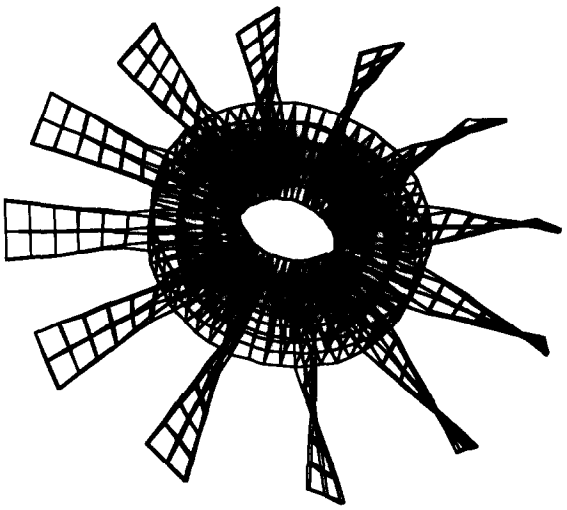


Fig. 13. A finite element model of a 12-bladed turbine disk with 504 3-D elements and 3828 nodes.

From Tables 6 and 7, the following observations are obtained:

(1) In most cases studied, the DG approach produces smaller Tot(N_b) than the CG approach for the RSS algorithm, while the CG approach produces smaller Tot(N_b) than the DG approach for the RST algorithm.

(2) For the bladed disk example, the optimal partitioning for $N_p = 3$, 4 or 6 seems to be a trivial task for the human. However, it is not the case for the partitioning algorithms. The RSS(DG), RST(DG) and RST(CG) algorithms deliver the optimal solutions, while the RSS(CG) algorithm does not.

(3) Except for the case of $N_p = 8$ in Table 6, the RST(CG) algorithm gives partitions with smallest Tot(N_b).

In addition, the evolution of the ACVs is given for the RST(CG) algorithm during the recursive process of partitioning the 12-bladed disk of Fig. 13 into 8 subdomains:

$$\mathbf{AC}^1_{\lambda_2}[\text{RST(CG)}] = (2.31)$$

$$\mathbf{AC}^2_{\lambda_2}[\text{RST(CG)}] = (2.39, 2.52)$$

$$\mathbf{AC}^4_{\lambda_2}[\text{RST(CG)}] = (3.17, 3.46, 3.25, 3.45)$$

Table 6
Comparison of different algorithms for element-based partitioning of the turbine blade of Fig. 12

N_p	Parameter	RSB (DG)	RSB (CG)	RSS (DG)	RSS (CG)	RST (DG)	RST (CG)
6	T_{cpu}	–	–	117	126	124	123
	Tot(N_b)	–	–	881	868	839	821
	Max(N_b)	–	–	435	381	382	377
	Min(N_b)	–	–	169	180	166	160
	N_d	–	–	0	0	0	0
	Max(N_{e3})	–	–	158	158	158	158
	Min(N_{e3})	–	–	157	157	157	157
8	T_{cpu}	113	118	119	125	113	118
	Tot(N_b)	1138	1179	989	1043	1138	1179
	Max(N_b)	444	460	374	365	444	460
	Min(N_b)	195	155	144	143	195	155
	N_d	0	0	0	0	0	0
	Max(N_{e3})	118	118	118	118	118	118
	Min(N_{e3})	118	118	118	118	118	118

Table 7
Comparison of different algorithms for element-based partitioning of the 12-bladed turbine disk of Fig. 13

N_p	Parameter	RSB (DG)	RSB (CG)	RSS (DG)	RSS (CG)	RST (DG)	RST (CG)
3	T_{cpu}	–	–	40	40	39	38
	Tot(N_b)	–	–	87	87	87	87
	Max(N_b)	–	–	58	58	58	58
	Min(N_b)	–	–	58	58	58	58
	N_d	–	–	0	0	0	0
	Max(N_{e3})	–	–	168	168	168	168
	Min(N_{e3})	–	–	168	168	168	168
4	T_{cpu}	41	40	41	40	40	40
	Tot(N_b)	116	116	116	153	116	116
	Max(N_b)	58	58	58	79	58	58
	Min(N_b)	58	58	58	74	58	58
	N_d	0	0	0	1	0	0
	Max(N_{e3})	126	126	126	126	126	126
	Min(N_{e3})	126	126	126	126	126	126
6	T_{cpu}	–	–	40	41	39	38
	Tot(N_b)	–	–	174	258	174	174
	Max(N_b)	–	–	58	92	58	58
	Min(N_b)	–	–	58	74	58	58
	N_d	–	–	0	3	0	0
	Max(N_{e3})	–	–	84	84	84	84
	Min(N_{e3})	–	–	84	84	84	84
8	T_{cpu}	39	41	42	42	41	39
	Tot(N_b)	304	276	308	387	304	276
	Max(N_b)	76	69	79	121	76	69
	Min(N_b)	76	69	76	81	76	69
	N_d	4	0	3	4	4	0
	Max(N_{e3})	63	63	63	63	63	63
	Min(N_{e3})	63	63	63	63	63	63

$$\mathbf{AC}^8_{\lambda_2}[\text{RST(CG)}] = (4.72, 5.84, 4.76, 6.35, 4.86, 6.00, 4.74, 5.99)$$

For the cases of $N_p = 2$ and $N_p = 4$, all the elements of the ACV are approximately equal. This is consistent with the fact that the RST(CG) algorithm delivers well-balanced subdomains for both cases. For the case of $N_p = 8$, the elements of the ACV have wider range of values than the former two cases. This is also consistent with the fact that the subdomains delivered by the RST(CG) algorithm in this case are not as well-balanced as those in the former two cases. Note that the magnitude of the eigenvalues show a tendency to increase as N_p increases. This is expected because, in general, the diameter of the subgraphs (associated with the subdomains) decreases. The statistics of the partitioning results for $N_p = 4$ and $N_p = 8$ are given in Table 7. Moreover, the ACV for the RST(DG) algorithm and $N_p = 4$ is

$$\mathbf{AC}^4_{\lambda_2}[\text{RST(DG)}] = (1.69, 1.65, 1.44, 1.39)$$

Comparing the numerical results for $\mathbf{AC}^4_{\lambda_2}[\text{RST(CG)}]$ and $\mathbf{AC}^4_{\lambda_2}[\text{RST(DG)}]$, one can see that the magnitude of the eigenvalues is larger in the former case than in the later one. In this example, it has also been observed that the four subdomains obtained using the RST(CG) algorithm are similar to those obtained using the RST(DG) algorithm. Therefore, $|E|$ for each subdomain is larger when the CG approach is used than when the DG approach is used. This fact is reflected in the magnitude of the eigenvalues.

Table 8
Comparison of different algorithms for node-based partitioning of the space station of Fig. 9 ($N_p = 3$)

Parameter	RSS		RST	
	NG	DG & CG	NG	DG & CG
T_{cpu}	2	16	2	16
Sum(N_b)	72	83	75	83
Max(N_b)	29	39	29	39
Min(N_b)	18	18	20	18
Max(N_n)	132	156	132	156
Min(N_n)	120	115	120	115
Max(N_{e1})	521	605	523	605
Min(N_{e1})	497	456	494	456
N_d	0	0	0	0

4.2. Node-based partitioning for explicit analysis

In addition to the use of the node graph (NG) approach for the node-based partitioning, the present research performs a simple extra step at the end of the algorithms with either DG or CG approach to assign uniquely common boundary nodes of two or more subdomains to a subdomain so that they can be also used for node-based partitioning. In this extra step, boundary nodes common to any two subdomains are assigned to the subdomain which has fewer nodes. The boundary node assignment is made such that a smooth boundary is generated between subdomains, i.e. a 'zig-zag' type of boundary is avoided.

Numerical comparative studies among the partitioning algorithms discussed have been conducted using three structures of different types: (a) the space station of Fig. 9 consisting of 1-D elements, (b) the 12-story L-shaped building of Fig. 10 consisting of both 1-D and 2-D elements, and (c) the 12-bladed turbine disk of Fig. 13 consisting of 3-D elements. As has been shown, the RST algorithm produces the same results as the RSB algorithm when N_p is an integer power of two. In the studies here, the focus is on the comparison between the RSS and RST algorithms. The partitioning results are given in Tables 8–10. Sum(N_b) is related to the amount of interprocess communication, while Max(N_b)/Min(N_b) shows the balancing of communication loads among processors. Max(N_i)/Min(N_i) and Max(N_e)/Min(N_e) indicate the balancing of the computational loads of equation solving and element calculations, respectively, among processors.

From Tables 8–10, the following observations can be obtained:

Table 9
Comparison of different algorithms for node-based partitioning of the 12-storey L-shaped building of Fig. 10 ($N_p = 3$)

Parameter	RSS			RST		
	NG	DG	CG	NG	DG	CG
T_{cpu}	3	3	3	3	3	3
Sum(N_b)	90	439	81	87	439	81
Max(N_b)	39	290	39	34	290	39
Min(N_b)	18	64	14	19	64	14
Max(N_n)	218	474	223	214	474	223
Min(N_n)	175	232	170	176	232	170
Max(N_{e1})	207	211	215	204	211	215
Min(N_{e1})	153	95	137	155	95	137
Max(N_{e2})	30	72	30	29	72	30
Min(N_{e2})	24	34	24	24	34	24
N_d	0	1	0	0	1	0

Table 10
Comparison of different algorithms for node-based partitioning of the 12-bladed disk problem of Fig. 13

N_p	Parameter	RSS			RST		
		NG	DG	CG	NG	DG	CG
4	T_{cpu}	139	28	27	123	27	26
	Sum(N_b)	280	280	426	356	280	280
	Max(N_b)	70	70	131	89	70	70
	Min(N_b)	70	70	79	89	70	70
	Max(N_n)	1027	1027	1101	1046	1027	1027
	Min(N_n)	1027	1027	1023	1046	1027	1027
	Min(N_{e3})	132	132	144	134	132	132
	Min(N_{e3})	132	132	126	134	132	132
	N_d	0	0	1	0	0	0
6	T_{cpu}	190	28	29	145	29	28
	Sum(N_b)	675	420	624	566	420	420
	Max(N_b)	118	70	131	130	70	70
	Min(N_b)	105	70	79	80	70	70
	Max(N_n)	768	708	779	768	708	708
	Min(N_n)	740	708	725	718	708	708
	Max(N_{e3})	98	90	100	98	90	90
	Min(N_{e3})	88	90	84	88	90	90
	N_d	2	0	3	1	0	0
8	T_{cpu}	234	31	33	148	29	27
	Sum(N_b)	795	759	947	672	777	740
	Max(N_b)	126	116	149	94	130	116
	Min(N_b)	82	76	81	66	76	69
	Max(N_n)	594	610	635	572	711	613
	Min(N_n)	553	537	542	544	531	529
	Max(N_{e3})	74	78	84	74	87	80
	Min(N_{e3})	66	63	63	66	63	63
	N_d	3	3	4	0	4	0

(1) In all examples studied, the CPU time spent by all the algorithms is very small compared to the execution time of a dynamic analysis.
(2) In the example of the space station, the NG approach produces better results than the DG and CG approaches for both the RSS and RST algorithms. In addition, the RSS(NG) algorithm gives the best results.
(3) In the example of the 12-storey L-shaped building, the RSS(CG) and RST(CG) algorithms give partitions with smallest Sum(N_b), while the RST(NG) algorithm produces partitions with most balanced N_b, N_n, N_{e1}, and N_{e2} among subdomains. In addition, the use of the DG approach gives very bad results, and domain splitting occurs.
(4) In the example of the 12-bladed disk, although the optimal partitioning into four or six subdomains seems to be a trivial task for the human, it is not the case for the partitioning algorithms. Only the RSS(DG), RSS(NG), RST(DG) and RST(CG) algorithms give the optimal solution for the case of $N_p = 4$. The RSS(DG), RST(DG) and RST(CG) algorithms also give the optimal solution for the case of $N_p = 6$. For the case of $N_p = 8$ which does not have a trivial optimal solution, the RST(NG) algorithm gives the best results among all the considered algorithms. In addition, the CG approach produces better or at least not worse partitioning results than the DG approach for the RST algorithms, while it is opposite for the RSS algorithm.
(5) As the number of partitions (N_p) increases, the RSS(CG), RSS(DG), RSS(NG) and RST(DG) algorithms seem to have a tendency to generate fragmented subdomains.
(6) In all the cases studied, the RST(CG) algorithm, which consistently delivers good partitioning results, has the best overall performance among all the considered algorithms.

To show the effect of different partitions on the performance of the parallel central difference method, actual parallel analyses have been carried out on the rotating bladed-disk (shown in Fig. 13)

partitioned into six subdomains by different algorithms. Both geometric and rotational non-linearities are considered in the dynamic analyses [12]. Fig. 14 shows the partitionings of the bladed-disk model by the RSS(NG), RSS(CG), RST(NG) and RST(CG) algorithms. It should be noted that border elements (elements in gray) shared by adjacent subdomains are included in each of these subdomains, as illustrated by Fig. 14. The partitioning results have already been given in Table 10. The performance of the parallel analyses using six DECsystem 5000s are given in Table 11. From Table 10 (for the case of $N_p = 6$) and Table 11, the following observations can be obtained:

(1) The RST(CG) algorithm delivers partitions with balanced subdomains and the smallest subdomain boundaries, resulting in the best performance of the parallel analysis.

(2) Although the RST(NG) algorithm gives partitions with Sum(N_b) smaller than that of the RSS(NG) algorithm, the former produces less balanced communication loads among processors.

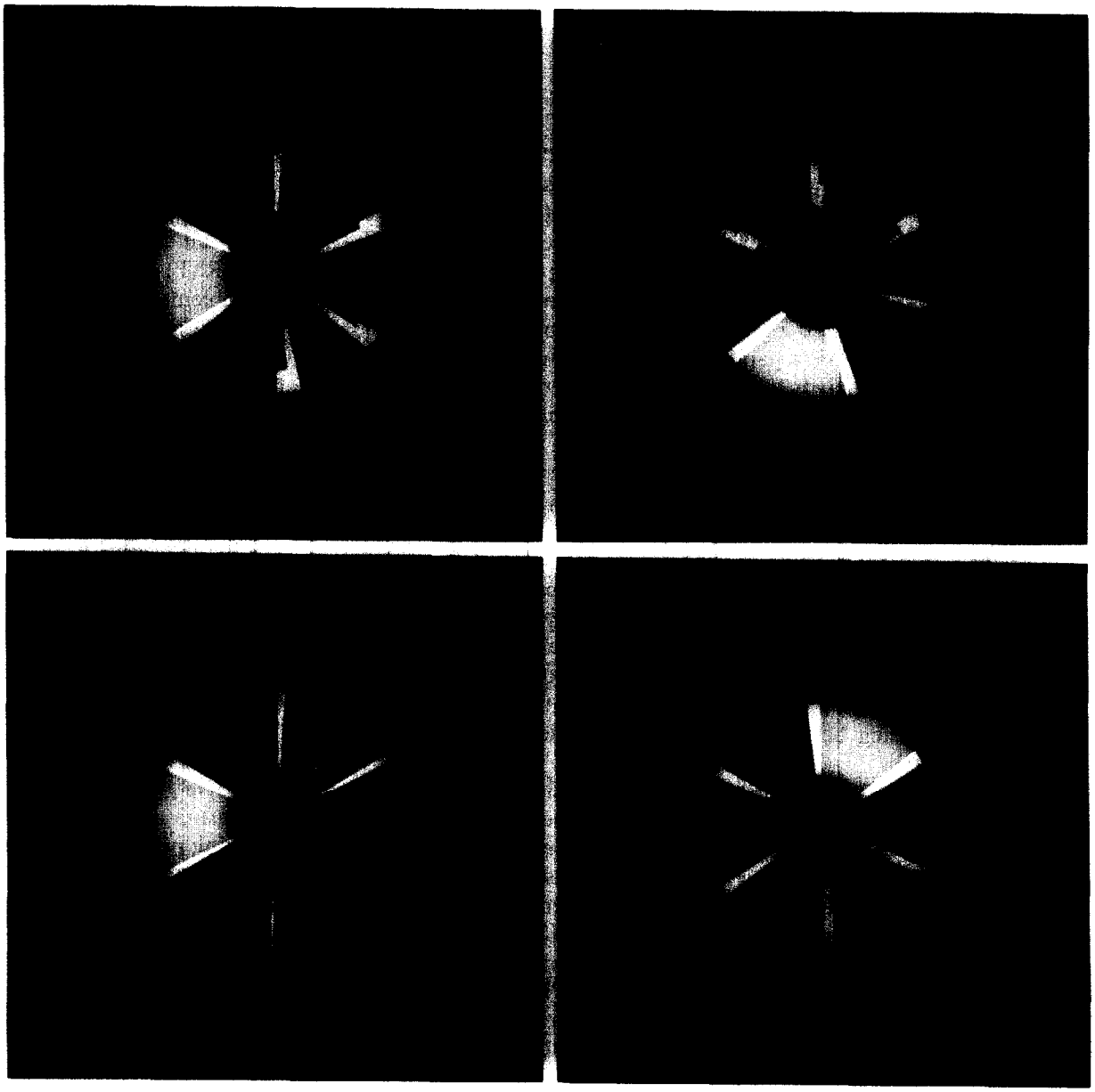


Fig. 14. Partitionings of the bladed-disk model of Fig. 13 by the (a) RSS(NG), (b) RSS(CG), (c) RST(NG) and (d) RST(CG) algorithms.

Table 11
Performance of the parallel finite element analysis of the 12-bladed disk problem of Fig. 13 partitioned by different algorithms ($N_p = 6$)

Partitioning algorithm	S (speed-up)	E (%) (efficiency)
RSS(NG)	4.9	82
RSS(CG)	4.7	78
RST(NG)	4.8	80
RST(CG)	5.4	90

In this case, as indicated in Table 11, the synchronzation overhead associated with the unbalanced communication loads among processors is slightly more significant than the overall communication overhead required.

5. Conclusions

This paper has shown that both the RSS and RST algorithms can efficiently produce an arbitrary number of partitions for generic finite element meshes. The use of different graph representations of the finite element meshes in the partitioning algorithms has also been investigated. It is found that in most cases the RST(CG) algorithm gives the best partitioning results among all the considered algorithms. However, the best combination of the partitioning algorithm (RSS or RST) and the graph (NG, DG or CG) associated with the finite element mesh is, in general, problem-dependent. In addition, all the considered algorithms deliver partitions in a very small amount of time compared to the computational time of a parallel non-linear dynamic analysis.

The algebraic connectivity vector (ACV) has been introduced as a parameter to assess the quality of the domain partitioning for load-balancing among processors. It also gives direct information about the occurrence of domain splitting problem (i.e. the ith partition is split if $\lambda_2^i = 0$). In addition, the multiplicity of the null eigenvalue is equal to the number of distinct connected components in the specific split partition. Of course, split partitions are also apparent if displayed graphically. It is important to avoid domain splitting because it usually increases communication overhead, and performance of parallel computations (especially implicit computations) can be seriously degraded. Future research is required to examine the important question of how much extra information concerning the quality of the partitioning can be extracted from the ACV.

It is apparent, from the results of the numerical examples studied, that all of the spectral partitioning algorithms considered do not guarantee the avoidance of domain splitting problems. However, compared to many non-spectral partitioning algorithms, the application of spectral algorithms to this problem is promising because of existing theoretical support [19, 20]. Further research is therefore needed in this area.

The RSS and RST algorithms have been considered in the context of a coarse-grained parallel computing environment. However, the use of these algorithms for fine-grained partitioning should be investigated in future research. In this case, the computational cost of the partitioning process, which is mainly determined by the efficiency of the eigensolver used, needs to be carefully addressed. The eigensolver has to cope efficiently with eigenproblems of a broad range of sizes. To this effect, the fast multilevel implementation of Barnard and Simon [26] might be considered.

Very recently, a node weighting scheme has been implemented by Simon in the RSB algorithm [28] for meshes with different types of finite elements. In this scheme, each vertex in the graph associated with the mesh has a positive weight to account for different computational loads in different types of elements. The bisection is performed according to weights instead of the number of vertices. The Fiedler vector (y_2) components provide an ordering of the vertices. Then, the weights are added up from the first vertex, the second one, and so on, until half of the total weight is reached. This scheme can also be adopted by both the RSS and RST algorithms presented in this paper. Future research is

needed to investigate the effectiveness of this approach for parallel solution of finite element meshes with different types of elements, such as the example in Fig. 10.

In the present work, attention has been on static load-balancing techniques which distribute computational loads among processors only once and before the actual computation starts. This approach is suitable for problems in which the distribution of computational loads among processors is constant throughout the course of the analysis. However, for dynamic analysis involving localized non-linear responses, the presence of localized non-linearity may create unbalanced workloads among processors, resulting in reduction in parallel efficiency. Therefore, there is a need to study and develop dynamic load-balancing techniques that can redistribute computational loads among processors during parallel analysis in a cost-effective fashion. In the present networked workstation environment where the cost of interprocess communication is high, the development of suitable dynamic load-balancing techniques is a particularly challenging research task.

Acknowledgment

The writers wish to thank Dr. Horst D. Simon of NASA Ames Research Center for kindly providing the source code of the RSB algorithm. The support of NASA Lewis Research Center under Grant No. NAG 3-1063 has been essential to this work and is appreciated. The partial support from the U.S. Air Force Office of Scientific Research under Contract No. AFOSR-90-0211 is also appreciated. Equipment grants from both the Digital Equipment Corporation and Hewlett-Packard to the Cornell Program of Computer Graphics are gratefully acknowledged. In addition, Glaucio H. Paulino would like to acknowledge the financial support provided by the Brazilian Agency CNPq (National Council for Research and Development). Any opinions expressed herein are those of the writers and do not necessarily reflect the views of the sponsors or equipment donors.

References

[1] H.D. Simon, Partitioning of unstructured problems for parallel processing, Comput. Syst. Engrg. 2 (1991) 135–148.
[2] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (W.H. Freeman and Company, New York, 1979).
[3] J. Flower, S. Otto and M. Salama, Optimal mapping of irregular finite element domains to parallel processors, in: A.K. Noor, ed., Parallel Computations and Their Impact on Mechanics (ASME, New York, 1987) 239–250.
[4] J.G. Malone, Automatic mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computers, Comput. Methods Appl. Mech. Engrg. 70 (1988) 27–58.
[5] C. Farhat, A simple and efficient automatic FEM domain decomposer, Comput. & Structures 28 (1988) 579–602.
[6] M. Al-Nasra and D.T. Nguyen, An algorithm for domain decomposition in finite element analysis, Comput. & Structures 39 (1991) 277–289.
[7] J. Padovan and A. Kwang, Hierarchically parallelized constrained nonlinear solvers with automated substructuring, Comput. & Structures 41 (1991) 7–33.
[8] G.L. Miler, S.-H. Teng, W. Thurston and S.A. Vavasis, Automatic mesh partitioning, Technical report CTC92TR112, Cornell Theory Center, Cornell University, Ithaca, NY 14853, 1992.
[9] B. Hendrickson and R. Leland, An improved spectral partitioning algorithm for mapping parallel computations, Sandia Report SAND92-1460, Category UC-405, Sandia National Laboratories, Albuquerque, NM 87185, 1992.
[10] B. Hendrickson and R. Leland, Multidimensional spectral load balancing, Sandia Report SAND93-0074, Category UC-405, Sandia National Laboratories, Albuquerque, NM 87185, 1993.
[11] R.D. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency: Practice and Experience 3 (1991) 457–481.
[12] S.H. Hsieh, Parallel processing for nonlinear dynamics simulations of structures including rotating bladed-disk assemblies, Ph.D. dissertation, Cornell University, Ithaca, New York, 1993.
[13] J.F. Hajjar and J.F. Abel, Parallel processing for transient nonlinear structural dynamics of three-dimensional framed structures using domain decomposition, Comput. & Structures 30 (1988) 1237–1254.
[14] J.F. Hajjar and J.F. Abel, Parallel processing of central difference transient analysis for three-dimensional nonlinear framed structures, Commun. Appl. Numer. Methods 5 (1989) 39–46.
[15] K.P. Birman, R. Cooper, T.A. Joseph, K.P. Kane and F. Schmuck, ISIS – A distributed programming environment, version 3.0 – user's guide and reference manual, Department of Computer Science, Cornell University, Ithaca, New York, 1991.

[16] F. Harary, Graph Theory (Addison Wesley, Reading, Massachusetts, 1969).
[17] N. Deo, Graph Theory with Applications to Engineering and Computer Science (Prentice-Hall, Englewood Cliffs, New Jersey, 1974).
[18] A. Pothen, H. Simon and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM J. Matrix Anal. Appl. 11 (1990) 430–452.
[19] M. Fiedler, A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory, Czechoslovak Math. J. 25 (1975) 607–618.
[20] W.N. Anderson Jr. and T.D. Morley, Eigenvalues of the Laplacian of a graph, Linear and Multilinear Algebra 18 (1985) 141–145 (originally published as University of Maryland Technical Report TR-71-45, October 1971).
[21] V. Venkatakrishnan, H.D. Simon and T.J. Barth, A MIMD implementation of a parallel Euler solver for unstructured grids, J. Supercomputing 6 (1992) 117–137.
[22] K.J. Bathe, Finite Element Procedures in Engineering Analysis (Prentice Hall, Englewood Cliffs, New Jersey, 1982).
[23] R.D. Cook, D.S. Malkus and M.E. Plesha, Concepts and Applications of Finite Element Analysis, 3rd edition (Wiley, New York, 1989).
[24] G.H. Golub and C.F. Van Loan, Matrix Computations, 2nd edition (The John Hopkins University Press, Baltimore and London, 1989).
[25] S.J. Fenves and K.H. Law, A two-step approach to finite element ordering, Internat. J. Numer. Methods Engrg. 19 (1983) 891–911.
[26] S.T. Barnard and H.D. Simon, A fast implementation of recursive spectral bisection for partitioning unstructured problems, in: R.F. Sincovec et al., ed., Parallel Processing for Scientific Computing (SIAM, 1993) 711–718.
[27] P.A. Wawrzynek, Discrete modeling of crack propagation: theoretical aspects and implementation issues in two and three dimensions, Ph.D. dissertation, Cornell University, Ithaca, New York, 1991.
[28] H.D. Simon, Personal communication (1994).