Parallel Automatic Adaptive Analysis

M.S. Shephard, J.E. Flaherty, C.L. Bottasso,
H.L. de Cougny, C. Ozturan, and M.L. Simone

Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180–3590

Abstract

Consideration is given to the techniques required to support adaptive analysis of automatically generated unstructured meshes on distributed memory MIMD parallel computers. Emphasis is placed on the structures needed to support effective parallel computations when the numerical discretization, the mesh, is defined and evolves during the computation. The key base structures are a distributed mesh based on a topological hierarchy, and a parallel distributed octree. Parallel control of the mesh and octree structures is done through a set of partition communication operations and entity migration routines. Load balance is maintained through iterative load balance, or distributed repartitioning. Building on these structures, procedures to automatically generate and adaptively refine meshes in parallel, starting from CAD geometric models, is given. Finally, the combination of these techniques to produce a parallel automated analysis procedure is demonstrated.

Keywords: Finite elements, Parallel load balancing, Mesh generation, Parallel adaptive calculations

1. Introduction

Automated and adaptive techniques provide the promise of reliably solving complex problems to the desired level of accuracy, while relieving the user of the time consuming and error prone tasks associated with mesh generation. For many problems, the computational requirements can only be met by scalable parallel computers. The development of effective parallel algorithms for adaptive techniques is challenging due to the irregular nature of adaptive discretizations and the constant modification of the discretization. This paper discusses a set of structures to support automated adaptive analysis on distributed memory MIMD parallel computers and demonstrates their application in automated three-dimensional analysis.

Three assumptions underlying the techniques presented are (i) the parallel computation algorithms assume a partitioning of the mesh onto the processors, (ii) the meshes are unstructured, and (iii) the mesh generation and enrichment processes interact directly with a geometric definition of the domain being analyzed as it exists in a CAD system. These assumptions have a defining influence on the procedures developed.

The next section overviews the components of the parallel automated adaptive finite element analysis system. The third section outlines the geometry-based problem description that represents the input to an automated adaptive analysis process. This is followed by two sections that describe the tools used to control adaptively evolving meshes on distributed memory computers. The final section overviews the three parallel application of an automatic three-dimensional mesh generator, three-dimensional mesh enrichment tools and an adaptive finite element solver for computational fluid dynamics problems.

2. Components of a Parallel Automated Adaptive Analysis System

The evolving nature of the discretization in an automated adaptive analysis procedure dictates the use of structures and constructs that can effectively account for the processor workload changes. These structures and constructs are dramatically different from those needed for fixed discretization parallel computations in that they must be able to efficiently maintain load balance, while controlling communications, as the distributed discretization evolves. Figure 1 shows the main components of an automated adaptive finite element analysis system. Each component is placed into one of four groups of ***Problem Definition, Base Structures, Parallel Mesh Control Functions,*** and ***Applications***.

The problem definition provides a geometry-based description of what is to be analyzed. It represents the input to the parallel adaptive analysis procedures. By employing a geometry-based methodology, the problem definition is independent of the type of discretization to be used in the analysis, thus allowing a variety of discretization techniques including finite element methods, finite difference methods and meshless methods.

The base structures hold the discretizations of the problem definition used by analysis applications. Currently general unstructured meshes are supported for the analysis discretizations. In addition there is an octree structure which is used by the automatic mesh generator to support parallel mesh generation. In both cases the data stored can be distributed over the processor memories in a distributed memory environment.

The parallel mesh control functions are the procedures required to support the use by the parallel applications of the distributed discretizations. Procedures included in this group support the interprocessor communications of discretization based information, the transfer of portions of the discretizations from one processor to another, procedures to determine how to carry out the redistribution of the discretizations to maintain load balance with

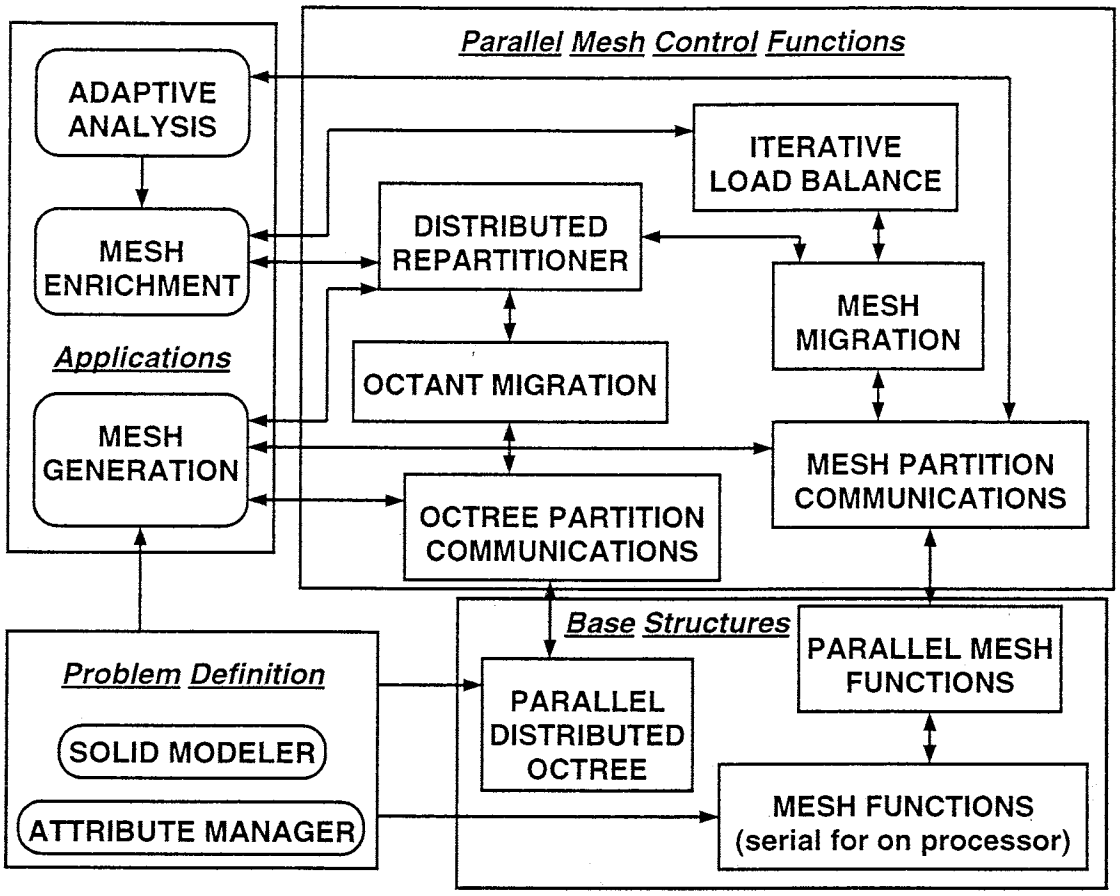

Figure 1. Components of a parallel automated adaptive analysis system.

minimum communications. All procedures in the parallel mesh control functions operate on a distributed mesh in parallel using scalable algorithms. The parallel mesh control functions are written using MPI protocols.

The final group of procedures are the application codes which carry out specific operations. The parallelization of the applications builds directly on the problem definition, base structures and parallel mesh control functions. All parallel operations are controlled through the parallel mesh control functions. The list of applications shown in Figure 1 are the minimal set to support automated adaptive analysis. The mesh generator is used to construct an initial discretization of the problem domain. The initial mesh is used in the first analysis step performed by the adaptive analysis procedures. During the analysis process, the adaptive analysis procedures invoke the mesh enrichment procedures to adapt the discretization as prescribed by the error indicators and correction indicators.

The next four sections provide an overview of the four groups of components.

3. Problem Definition

The goal of an analysis is to solve a set of partial differential equations over the geometric domain of interest. The specification of the analysis problem requires a complete and unique definition of the domain as well as the physical attributes required to indicate the partial differential equations, and the specific values of the equation parameters, for the analysis at hand. The solid modeler is responsible for maintaining the description of the domain and the attribute manager is responsible for housing the definition of the needed physical attributes and their association with the geometric domain.

Although a number of schemes are possible for defining a geometric domain [27], the most advantageous are boundary-based schemes in which the closure of the geometric domain to be analyzed is defined as

$$\overline{\Omega}_G(T_G, S_G)$$

where S_G represents the shape information of the entities which define the domain and T_G represents the topological types and adjacencies[1] of the entities which define the domain. In addition to being unique, the use of topological entities and their associativities provides a convenient abstraction for defining the relationship of different models of the same domain. Boundary representations allow the convenient specification, with respect to the geometric domain, of the physical attributes for that analysis [36, 37]. An additional advantage of boundary representations is the fact that current computer aided design systems support boundary representations of the domains defined within them. This allows the effective combination of these packages with automatic analysis procedures. A final advantage of recent boundary representations is their ability to properly represent the non-manifold geometric domains commonly used for analysis [47, 15].

Solid modeling systems are major software packages developed independently of the engineering analysis procedures. The solid modeling software used varies from organization to organization. However, by definition of the functions a solid modeler must support, they all have a level of functional equivalence. This functional equivalence is take advantage in that the various components of the parallel automated adaptive analysis procedures interact with the problem definition through a well defined set of operators [39] (Fig. 2). These operators are all keyed based on the basic topological entities which exist in any valid boundary representation. The arrows going from the problem definition in only one direction implies the capability to interrogate geometric model for any geometry based

[1] In the context of a domain representation, adjacencies are the relationships among topological entities which bound each other. For example, the edges that bound a face is a commonly used topological adjacency.

information, such as the intersection of a given line with a surface, but do not support the capability to modify the model. For many classes of analysis this is sufficient. However, in the case of evolving domain problems, analysis information may be used to update the geometric representation. The inclusion of operators to support such operations is a straight forward extension.

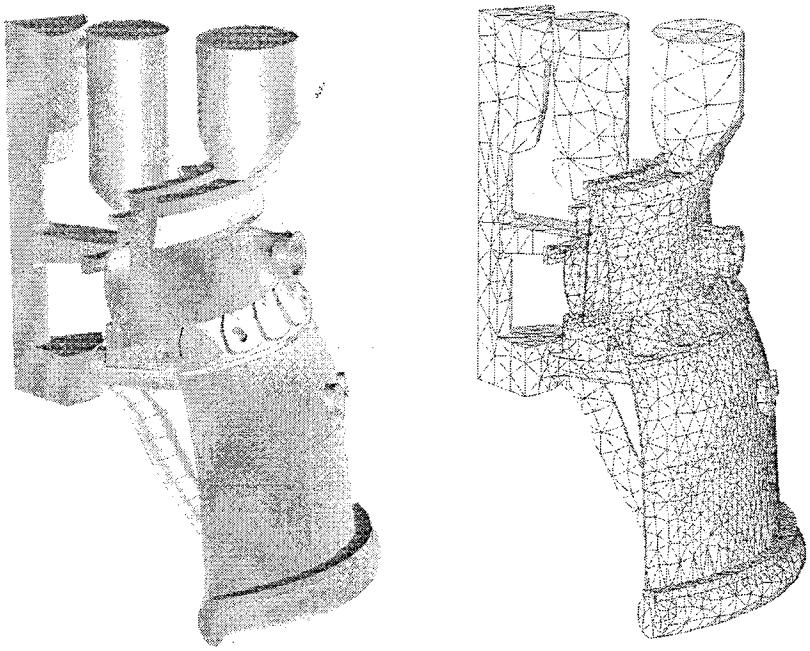

Figure 2. Solid model and automatically generated mesh for the casting of engine diffuser with gating.

Since the solid modeler is central to the problem definition is an external piece of software that runs in serial, it is not practical to develop a parallel version of it. However, in the currently depicted case where an existing model is only interrogated, it is straight forward to run an independent version of the solid modeler on each processor. Since the size of the solid model data structure is small compared to the other data structures, this is an acceptable solution which in itself is not scalable, but which does not influence the scalability of the other pieces. If one wanted to make this piece scalable, the parallel distributed octree discussed below could be used to distribute the model to the processors.

4. Base Structures

The key structure used by all procedures is the distributed mesh structure. The distribution

of the mesh among the processors is used to define and control the computational workload for the adaptive analysis and mesh enrichment procedures.

Since the goal of the automatic mesh generator is to create a mesh for a given geometric domain, it is not possible to control the parallel computations on the information that is to be generated. To support the parallelization of this process the parallel distributed octree was introduced. In addition to its use in controlling parallelization of the mesh generation process, the spatially based octree can be effectively used for parallel control of other aspects of the adaptive analysis process.

4.1 Distributed Mesh Structures

The classic mesh structure of node point coordinates and element connectivities can not support the needs of automated adaptive analysis. Richer structures are required to support adaptive mesh enrichment and to provide the links to the original domain definition needed by critical functions, including ensuring the automatic mesh generator has produced a valid discretization of the domain. A number of alternative mesh structures have been proposed for various forms of mesh adaptation. All of them can be effectively captured by a general structure based on a hierarchy of topological entities.

Since individual volume mesh entities will be limited to simple regions, bounded by simply connected faces, consideration of the topological entities can focus on the basic 0 to d dimensional topological entities, which for the three-dimensional case (d=3) are the vertices, edges, faces and regions defining the domain.

Mesh entities are always classified with respect to the lowest order object entity possible. Classification of the mesh against the geometric domain is central to (i) ensuring that the automatic mesh generator has created a valid mesh [30, 31], (ii) transferring analysis attribute information to the mesh [37], (iii) supporting h-type mesh enrichments, and (iv) integrating to the exact geometry as needed by higher order elements.

The adjacencies of various order mesh topological entities and their classification are used to support a great number of the operations required by a parallel automated adaptive analysis. Therefore, it is important that they can be quickly determined. Clearly, if the adjacencies of each order entity against all other entities were stored, all possible adjacency information would be readily available. This approach would be highly wasteful with respect to the amount of data storage required. On the other hand, storing only a minimal number of adjacencies could require extensive searches and sorts to determine other specific adjacencies. An examination of the specific adjacencies used by the various algorithmic operations provides guidance as to the minimum number of adjacencies needed. For example references [2, 6, 14, 20] define adjacencies used in specific finite volume and finite element procedures. Since the procedures used here support any form of

adaptive analysis on conforming unstructured meshes, all adjacencies are either stored, or can be quickly determined through a set of local traversals and sorts which are not a function of the mesh size. One set of relationships that can effectively meet these requirements is to maintain adjacencies between entities one order apart. Fig. 3 graphically depicts this set of relationships, as well as the classification with respect to the geometric domain representation. For more information on the use and implementation of alternative topologically-based mesh structures see [1].

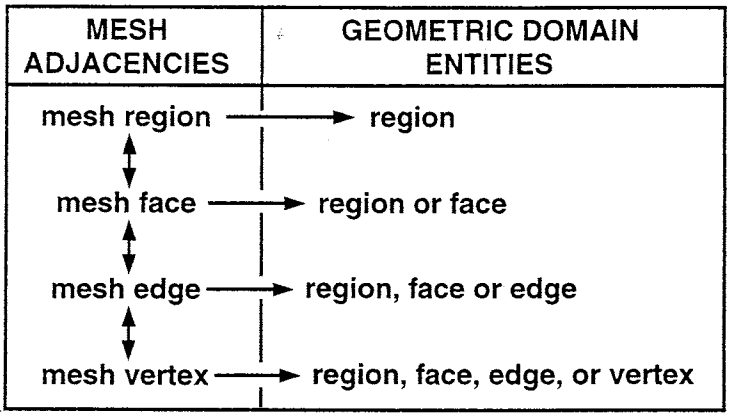

Figure 3. Mesh topological adjacencies and classification information.

The distributed mesh structure builds directly on the topologically based mesh database in the following manner. The mesh assigned to each processor is housed in a local version of the mesh structure and the links to neighboring mesh entities on other processors is viewed analogous to the modeling of material interfaces in geometric objects.

The uses of shared entities are mapped onto the owner entity by a many-to-one relation, $\Phi : (p_k, a_k) \rightarrowtail (p_o, a_o)$ where p_k indicates a processor ID and a_k in the local address of the entity on that processor. The distributed mesh structures were designed to provide the full set of adjacencies. A partition boundary entity can get all the links of an entity on other processors. Each partition boundary entity stores all the uses on other processors as a linked list (Fig. 4). The bold edges and vertices indicate the owners of the shared entities. This ownership information can be used in the implementation of the owner computes rule, for example, during link updates in mesh migration or scalar product computation in an iterative linear solver.

Since each processor stores the uses on all the processors that hold a shared entity, the ownership can be computed as a function of these uses. Since the processor ID and local address pair *is the global key*, there is no need to generate and store a separate key. On a processor, at the level of entities, the sets of entities that are on the partition boundary or

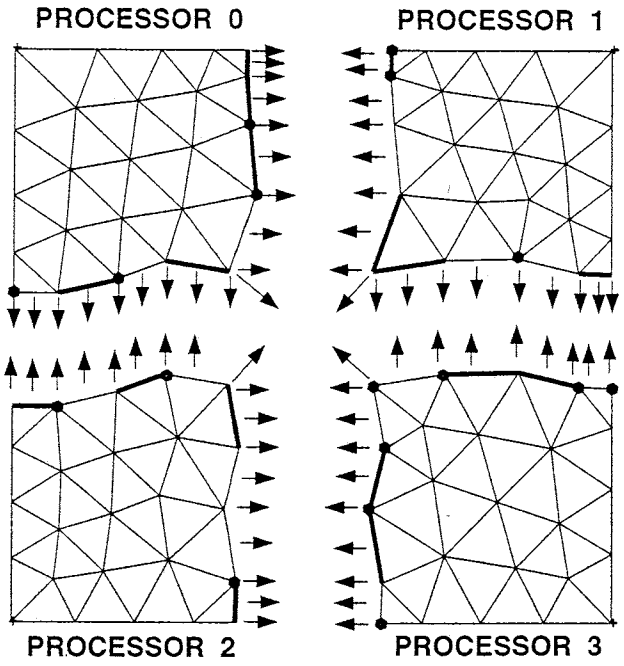

Figure 4. Two-dimensional example of the interprocessor links and entity ownership.

adjacent to a specific processor are organized in doubly linked lists which provide constant insertion and deletion.

4.2 Parallel Distributed Octree

An octree [28] is a spatially-based structure commonly used for the representation of three-dimensional objects. The basic idea of an octree is to place the domain of interest into an enclosing cube and to recursively subdivide selected octants into their eight suboctants until each suboctant meets a selected criteria. This subdivision process naturally defines a tree where each node represents a cube of a specific size, and the children of a parent node represent the eight octants defined by bisecting the cube in the three orthogonal directions. The typical use of octrees in finite element methods [35] is to have the finite element edge lengths equal to that of the terminal octant, and to vary the size of the terminal octants through the domain based on some spatially-based mesh control function. Two key issues faced in the use of octree structures in parallel applications on distributed memory computers are the distribution of the octree among the processors and the control of interprocessor communications when the tress is used.

One approach to the distribution of the octree is to have a copy on each processor. Since octree structures require much less memory than the mesh or solution structures, this approach is viable for a reasonable number of processors [9, 10]. However, it is clearly

not scalable when the problem size grows and the number of processors is increased. To obtain scalability, the octree must be distributed. One approach to a distributed octree [41, 42] is to employ the concept of local roots with interprocessor links from local roots to parent octants, and from parent octants to their children on other processors. For purposes of parallel efficiency, each processor contains a list of local roots, and specific consideration is given during tree construction to minimize the number of local roots on a processor.

A common use of an octree in finite element applications is to employ tree traversal to locate neighboring octants. In general such traversals introduce a $\log_8 N$, where N is the number of terminal octants, term into the growth rate, which in itself can be undesirable. More importantly to a distributed octree is the amount of interprocessor communications such traversals can introduce. In general each traversal can introduce a number of interprocessor hops into the process thus greatly increasing the communication costs required to get the neighbor information. An alternative to avoid the traversals is to store octant neighbor pointers in the tree. However, the indiscriminate inclusion of neighbor pointers can greatly increase the tree storage requirements. A useful compromise is to store only face neighbor pointers to neighboring octants of equal or greater size. Edge and vertex neighbors can be determined from the face neighbors [41, 42]. The advantage of this approach is that it limits the amount of storage. In the present case it increases the storage of the combined octree and mesh structures by less than three percent.

The disadvantage of this approach is in the case when the neighboring octant of interest is at a finer level, tree traversal from the octant of equal size down the terminal octant is required. In worst case we are back to $O(\log_8 N)$ hops. However, it is typical in finite element applications to limit the size difference of face neighbors to one level in final form of the octree. Since the octree is in its final form of one level difference when the neighbor determinations are required, the maximum number of hops is limited to two.

5. Parallel Mesh Control Functions

The evolving nature of an adaptive discretization introduces load imbalance into the solution process. Therefore, it is critical that the load be dynamically rebalanced as the adaptive calculation proceeds. The tools needed to support dynamic rebalancing must be able to move mesh entities from one processor to another and to determine which mesh entities should be migrated. The partition communication and entity migration routines control the process of moving mesh entities between processors, while either a distributed repartitioner or iterative load balancer can be used to determine which mesh entities should be moved to different processors.

5.1 Partition Communication and Entity Migration

Analogous to the owner computes rule, the mesh migration procedure uses an *owner updates rule* to collect and update any changes to the links on partition boundaries after moving entities among processors. The migration of a set of mesh, or octant, entities from a given processor to destination processors proceeds in three stages (Fig. 5). Firstly, sender processors migrate the mesh entities to receiver processors. Secondly, the senders and receiver processors report the deletions, or new addresses, of migrated mesh entities to the owner processors. In the last stage, the owner processors inform the affected processors about the updates in links. The processing which is done in the first stage is proportional to the number of mesh entities being migrated, whereas in the second and third stages, it is proportional to the union of the boundary of the migrated mesh entities. An explanation of the procedure and demonstration of it's scalability is given in references [25, 38].

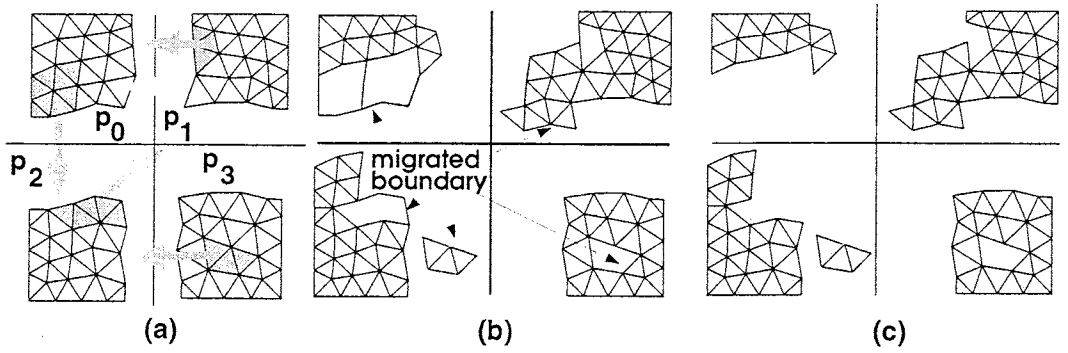

Figure 5. Example showing steps of mesh migration.

5.2 Dynamic Balancing Based on Repartitioning

The dynamic balancing based on repartitioning employs the Inertial Recursive Bisection (IRB) method [24] which bisects a set of entities by considering the median of the set of corresponding centroids with respect to the inertial coordinate system for the set of entities to be bisected. The main assumption for performing repartitioning in parallel is that the entities are distributed and the number of entities across processors before repartitioning is not uniform. The key algorithm in IRB is the determination of the median for a given set of doubles (referred to as "keys") [32]. With respect to this algorithm, the "keys" are the first coordinates, in the inertial frame, of the entities to be bisected. The method used here is to sort the "keys" and then pick the entry at the middle of the sorted list. In this case, efficiently performing IRB in parallel can be reduced to efficiently sorting in parallel [17]. A parallel sample sort algorithm [3] is well suited to large data sets, and is used here to efficiently support IRB on distributed meshes [10, 38].

Fig. 6 shows a randomly distributed mesh (approximately 35,000 elements) and the resulting dynamically repartitioned mesh for eight processors. Fig. 7 shows timings for that particular mesh on 2, 4, 8, and 16 processors. It should be noted that a randomized mesh as the initial state is a worst-case scenario for the migration part of the repartitioning procedure. Past four processors, the time spent decreases as the number of processors increases, which is a good indication of scalability.

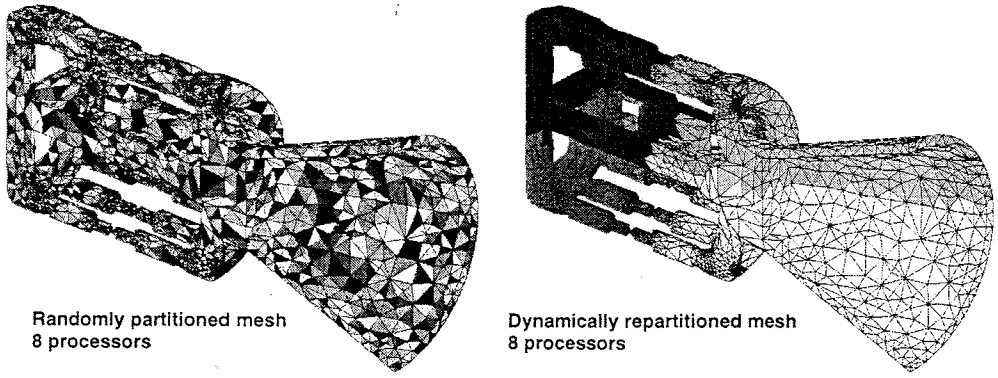

Figure 6. Dynamic repartitioning on a randomly distributed mesh.

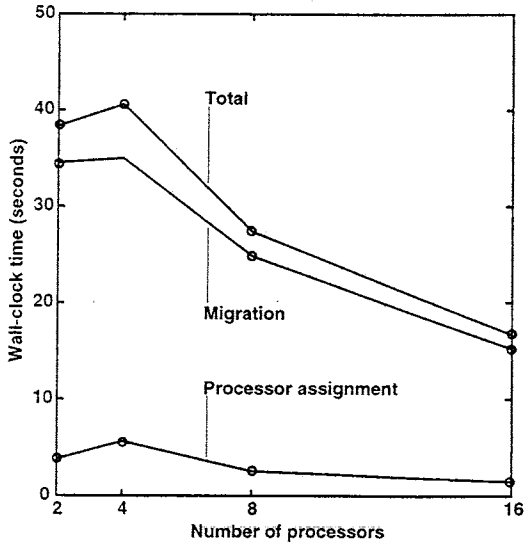

Figure 7. Timings for dynamic repartitioning.

5.3 Dynamic Balancing based on Iterative Load Migration

Iterative load migration techniques can be used to dynamically balance adaptive refined meshes. These techniques exchange load between neighboring processors to improve the load balance and/or decrease the communication volume. One approach to iterative load migration is cyclic pairwise exchange [16] where pairs represent processors connected by a hardware link which can exchange nodes of the mesh. Leiss/Reddy [22] use the hardware link as the neighborhood to transfer work from heavily loaded to less loaded processors. The Tiling system [11, 12] extends the Leiss/Reddy algorithm to the case where the neighborhood is defined by the connectivity of the split domains. The algorithm of Lohner and Ramamurti [24] exchanges elements between subdomains according to a deficit difference function which reflects the imbalance between an element and its neighbors. Vidwans et al. [46] uses a divide and conquer approach to pair processors and uses connectivity as well as coordinate information to decide which elements to migrate.

The current iterative load balancing procedure is based the Tiling heuristic of requesting load from the most heavily loaded neighbor. To incorporate more global information and to direct load transfers, the current procedure views the processor requests for load from heavily loaded processors as forming a forest of trees. Given this hierarchical arrangement of processors as the nodes of trees, the individual trees are easily balanced. Since the average load on trees can vary, and the migration process can change which neighbors are connected, the forest of trees can change. Therefore, the basic tree balancing process is iteratively repeated until the load distribution converges to optimal load balance within a user supplied tolerance. The basic algorithmic steps are given in Fig. 8.

In step 4, load differences are computed by each processor sending its load value to its neighbors and receiving load values from its neighbors. Step 5 invokes the Tiling load request process. Since each processor can receive requests from multiple processors, but can only request from a single processor, a forest of trees is formed.

In step 6, the trees are linearized for efficient scan operations using depth-first-links [21, 44]. Step 7 computes the amounts of load migrations on the tree using logarithmic scan operations on the linearized tree.

6. Applications

6.1 Parallel Automatic Mesh Generation from Solid Models

The development of automatic mesh generation techniques for complex three-dimensional configurations has been an active area of research for over a decade [13, 40]. Parallel mesh generation is difficult to effectively control since the only structure known at the start of the process is that of the geometric model which has no discernible relationship to the work

```
procedure tree_load_balance(tol_load, max_iter)
in tol_load imbalance load tolerance
in max_iter : maximum number of iterations
begin
1   iter = 0
2   while (max. load difference > tol_load ) and
    (iter < max_iter) do
3     iter = iter + 1
4     Compute neighboring load differences
5     Request load from neighbor processor having
      largest load difference (creates processor trees)
6     Linearize processor trees
7     Compute amounts of load migration
8     Select and migrate load
9   endwhile
end
```

Figure 8. Iterative load migration balancing procedure.

load needed to generate the mesh. The lack of initial structure and ability to accurately predict work load during the meshing process underlies the selection of an octree structure to support the distribution and redistribution of computational effort to processors.

The parallel mesh generator presented here meshes three-dimensional non-manifold objects [9, 10, 41]. In the current implementation the model boundary is meshed first, and the model regions are meshed based on the resulting boundary triangulation. The parallel distributed octree is the key structure used to control parallelization of the volume meshing process.

The steps involved with meshing a model face include: [8]:

1. Determination of the face's boundary.
2. Initial coarse meshing based on boundary discretization invoking the Delaunay criteria in the parametric space of the face.
3. Boundary recovery and deletion of exterior elements.
4. Insertion of interior vertices using a quasi-Delaunay criteria in three-space to yield mesh faces with sizes as dictated by the mesh control information applied to that model face.
5. Mesh surface approximation check.
6. Mesh self intersection check.

The basic approach to parallelization of surface triangulation is to distribute the model faces to individual processors to be meshed in serial on that processor. Issues that arise in the parallelization of this process include:

1. Ensuring a consistent discretization of the faces boundary.
2. Maintaining load balance and ensuring scalability.
3. Performing the surface intersection checks against other faces.
4. Constructing the octree structure from the surface triangulation.

Consistent face boundary discretization can be accomplished either by their distribution to the processors, or based on the distribution of the faces using appropriate coloring and synchronization. Since the number of faces relative to the number of processors may not be large, and/or the number of mesh faces to be generated per model face can vary dramatically, a procedure is included to split faces which are "large" in parallel so that reasonable load balance can be maintained during face meshing.

The parallel distributed octree is constructed as the faces are being meshed. During the process of setting-up the face meshing process, an octree is constructed from the mesh size parameters. The distribution of faces to processors for meshing is based on this octree. As the faces are meshed, the distributed octree is refined if necessary such that the octants containing the mesh faces being generated are on the order of the size of those mesh faces. This octree structure can then be used as the localization device to check for self intersection of the surface meshes.

After the model faces are triangulated, the octree is refined to reflect any additional volume based mesh control information and to ensure a one-level difference between face and edge neighboring octants.

The volume meshing process employs the octree structure and the surface mesh. Fig. 9 graphically depicts the basics of the volume mesh generator for an inplane two-dimensional case. Given the octree reflecting the surface mesh size and volume mesh control information, and with only level neighbor difference enforced, the octants are classified as interior, outside, or boundary. Those classified as outside receive no further consideration. Some interior octants are reclassified boundary if they are too close to mesh entities classified on the boundary of the model region (boundary-interior). The purpose of this reclassification is to avoid the complexities caused when interior octant mesh entities are too close to the boundary which can lead to the creation of poorly shaped elements in that neighborhood. Interior octants are meshed using templates. The remainder of the region is meshed using face removals to connect the given surface triangulation to the interior octants. The octree is used in the parallelization of both meshing steps. This

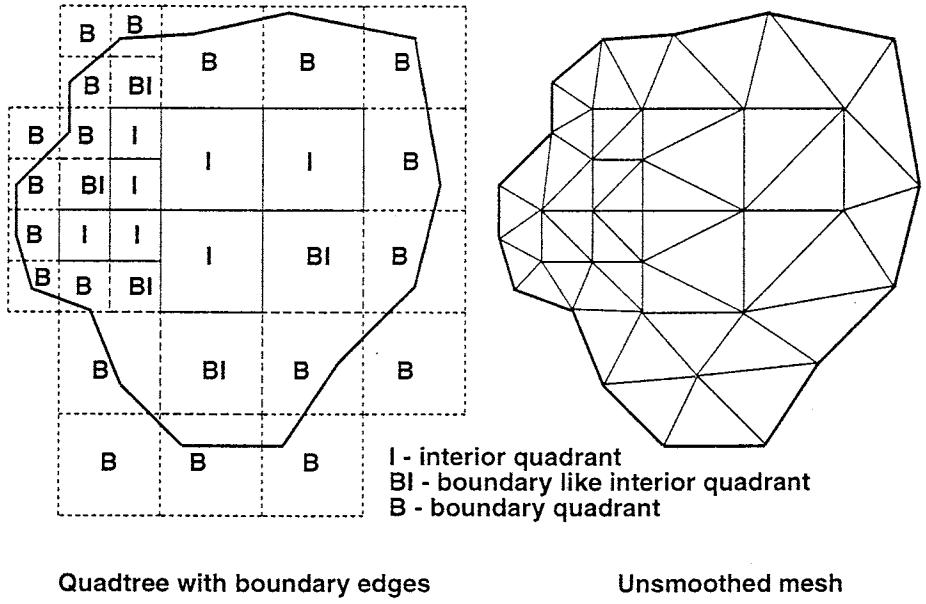

Figure 9. Graphical depiction of the basics of the volume meshing process in a two-dimensional setting.

Figure 10. Volume meshing example done on 8 processors.

process is explained in more detail in references [9, 10]. Figure 10 shows an example mesh generated with the volume mesher using 8 processors.

6.2 Mesh Enrichment

Since mesh enrichment is an integral part of an adaptive solver, it must be run in parallel as well in order to not become a bottleneck. A variety of approaches are possible to perform the required mesh enrichments. Currently the local mesh modification procedures of edge-based refinement, edge-based derefinement and local mesh optimization, as described in [7], are employed. These procedures have been developed to interact directly with the geometric representations and deal with all the complexities introduced by curved geometries. The basic steps in this process are:

1. Coarsening using edge collapsing.
2. Grid optimization to improve element shapes.
3. Refinement using a full set of subdivision patterns.
4. Refinement vertex snapping (to the model boundary).
5. Grid optimization to improve element shapes.

If a mesh edge is marked for coarsening, it is attempted to be collapsed. If the affected polyhedron (made up of all connected mesh regions) is on processor p_i, the edge collapsing is performed on p_i. If the affected polygon is not fully on p_i, the missing mesh regions are requested from the appropriate processors. When all processors are done traversing their lists of mesh edges, the processors that have received requests send (migrate) the requested mesh regions. In figure 11, processor p_0 requests mesh regions from processors (p_1, p_2, p_3) and the requested mesh regions are migrated. If there is conflict, the processor with the lowest p_i has priority. On the next iteration, it is the processor with the highest p_i that will have priority. This switching is done to prevent too much load imbalance at completion. The process of traversing the list of mesh edges and sending/receiving requests continues until all marked mesh edges have been attempted to be collapsed. Because mesh regions are migrated, it is possible that the processors are not well balanced after the derefinement step. The triangulation is therefore submitted to a load balancing step before going further. For a triangulation of 85,000 elements where 50% of the mesh edges are coarsened (the resulting triangulation has approximately 46,000 elements), the speed-ups are approximately 1.8, 3.3, and 5.4 for 2, 4, and 8 processors, respectively.

The grid optimization process is parallelized using the same basic approach as the derefinement process. For a triangulation of 85,000 elements, the speed-ups are approximately 1.7, 3.1, and 4.8 for 2, 4, and 8 processors.

Refinement relies on the marking of mesh edges. Any mesh region is given a weight proportional to the number of marked bounding edges. Prior to applying the parallel refinement scheme, the triangulation is load-balanced based on that weight. Any mesh face on some partition boundary with at least one marked mesh edge is triangulated using

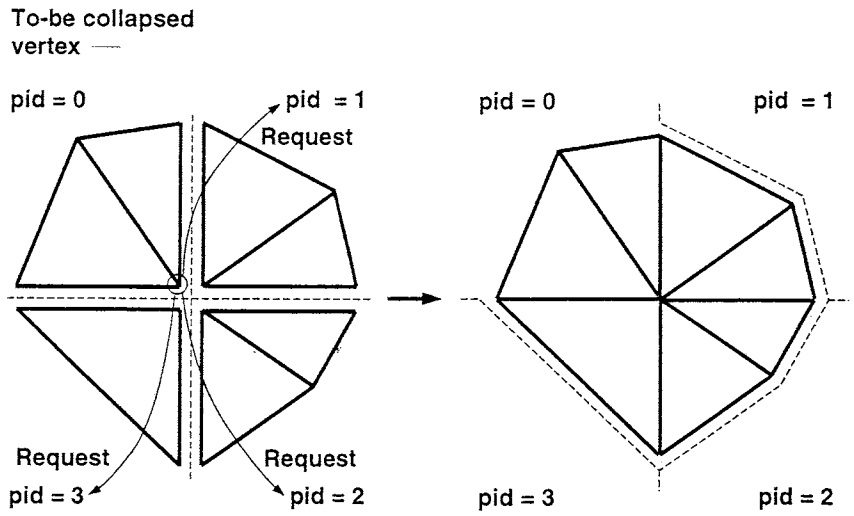

Figure 11. Mesh migration to support parallel distributed derefinement.

two-dimensional subdivision patterns. Once all mesh faces on the partition boundary are subdivided, links for all new mesh entities are updated. Then, each processor can apply three-dimensional templates on any mesh region with at least one marked edge without any communication [7]. Once all appropriate mesh regions have been subdivided, the refinement vertices which are classified on the model boundary need to be snapped to the corresponding model entity. Since snapping makes use of the local retriangulation tools, the technique to parallelize that part of the process is similar to the one used to parallelize the derefinement and optimization steps. For a triangulation of 36,000 elements when 20% of the mesh edges are refined (resulting triangulation has 88,000 elements), speed-ups are approximately 1.9, 3.4, and 5.8 for 2, 4, and 8 processors, respectively.

6.3 Parallel Adaptive Solution Procedures

Details of the finite element formulation of our parallel adaptive code for compressible flow problems are discussed in [5]. Here its basic capabilities are reviewed and representative results provided.

The program can solve steady and unsteady compressible Euler and Navier Stokes flows, in fixed and rotating frames, and with specific extensions for the simulation of helicopter rotors in hover and forward flight. The numerical discretization is based on the Time–Discontinuous Galerkin Least–Squares finite element method [33, 34] with the flow equations written in terms of the entropy variables. A least–squares operator and a discontinuity capturing term are added to the formulation for improving stability without sacrificing accuracy [33, 34]. This approach is characterized by a very limited amount

of numerical diffusion, allowing a nice resolution of vortical structures and sharp shocks in transonic regions.

Two different three dimensional space–time finite elements have been implemented. The first is based on a constant in time interpolation, and, having low order of time accuracy but good stability properties, it is well suited for solving steady problems using a local time stepping strategy. The second makes use of linear–in–time basis functions and, exhibiting a higher order temporal accuracy, is well suited for addressing unsteady problems. In these cases, moving boundaries are handled by means of the space–time deformed element technique [4, 45].

Discretization of the weak form implied by the TDG/LS method leads to a nonlinear system, which is solved iteratively using a quasi–Newton approach. At each Newton iteration, a non–symmetric linear system of equations is solved using the GMRES algorithm. We have developed scalable parallel implementations of the preconditioned GMRES algorithm and of its matrix–free version [19, 18]. Preconditioning is achieved by means of a nodal block–diagonal scaling transformation.

Error indication is currently limited to the evaluation of the norm of the gradient of the flow variables, or on estimates of the second derivatives of the solution due to [23]. The edge values of the error indicator are computed by averaging the corresponding two nodal values. These edgewise error indicator values are then used for driving the mesh adaptation procedure. Appropriate thresholds are supplied for the error values, so that the edge is refined if the error is higher than the maximum threshold, while the edge is collapsed if the error is less than the minimum threshold. The mesh adaptations are performed in parallel using the parallel mesh enrichment procedures described above.

The first example problem presented is the standard CFD three-dimensional test case of an Onera M6 wing in transonic flight. This wing has been studied experimentally by Schmitt and Charpin [29]. The wing is characterized by an aspect ratio of 3.8, a leading edge sweep angle of 30^o, and a taper ratio of 0.56. The airfoil section is an Onera D symmetric section with 10% maximum thickness–to–cord ratio.

The simulation performed is for steady flow at an angle of attack $\alpha = 3.06^o$ and a value of $M = 0.8395$ for the freestream Mach number. In such conditions, the flow pattern around the wing is characterized by a double–lambda shock on the upper surface of the wing with two triple points.

An initial coarse mesh of 85,567 tetrahedra was partitioned on 32 processing nodes of an IBM SP-2 and the analysis was carried out to convergence as previously explained. The results obtained were then used for computing an error indicator based on density and Mach number, which was employed for performing a first level of refinement, bringing the

mesh to 131,000 tetrahedra. The solution was projected on the new vertices using a simple edge interpolation technique, and the analysis was then performed on the refined mesh for 80 time steps at a CFL number of 10. Similarly, two levels of refinement followed by subsequent analysis were performed, obtaining an intermediate 223,499 tetrahedron mesh and a final 388,837 tetrahedron mesh.

As expected, considerable improvement in the resolution of the shocks is obtained when mesh adaptation is employed. Figure 12 shows the initial and final meshes. Elements assigned to the same subdomains are denoted by the same grey level. For the final mesh, the partitions shown are those obtained with the iterative load balancing algorithm.

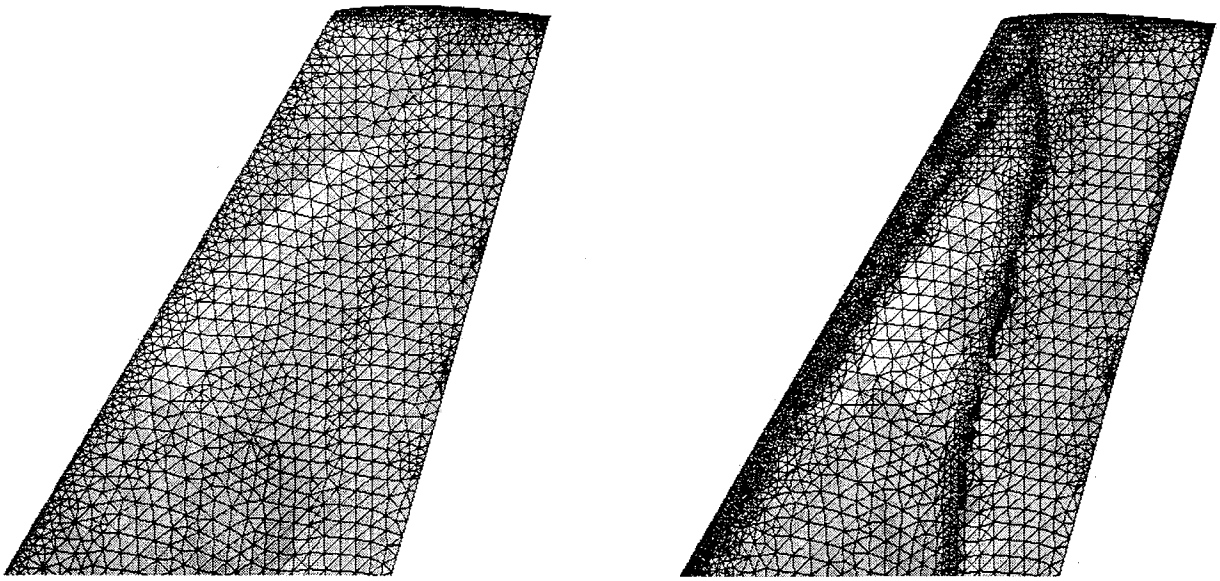

Figure 12 Onera M6 wing in transonic flight, $\alpha = 3.06^o$, $M = 0.8395$.
Initial and final meshes. Grey levels indicate processor assignment.

The second test problem demonstrates the use of the parallel adaptive procedures to perform the CFD calculations needed in high speed impulsive noise computations. The CFD results are interfaced with a Kirchhoff integral solver for the noise calculation. Such a coupled CFD/Kirchhoff procedure would allow a solution adaptive unstructured CFD simulation of the acoustic field close to the rotor, while the Kirchhoff formulation would be responsible for propagating the acoustic signal to the far field.

For this purpose, a test case experimentally investigated by Purcell [26] was selected. The same problem has been numerically simulated by a number of researchers, including Strawn

et al. [43]. The problem is that of a rectangular blade hovering rotor with NACA0012 airfoil sections and aspect ratio of 13.71, denoted by a tip Mach number of 0.95. The test case is characterized by a marked tip delocalization. The simulation is conducted for the non-lifting case, therefore the problem is symmetric about the plane of the rotor and the computational domain extends only on one side of the plane itself. The far field boundaries are located at 1.5 radii above the plane and at 3 radii from the hub. Periodic boundary conditions are applied at the symmetric faces, while slip conditions are applied at the rotor disk plane to account for symmetry.

The parallel adaptive analysis was conducted with four refinement levels, each followed by subsequent analysis to convergence. The error indicator selected in this case is based on the norm of the gradient of pressure, with limiters on the minimum allowable edge lengths. The principal scope of the limiters is to prevent the excessive refinement of the leading edge of the blade tip, that otherwise would lead to excessively large meshes. Figures 13 and 14 show the final mesh, characterized by 575,026 tetrahedra. Note the refinement along the tip blade shock and along the acoustic wave.

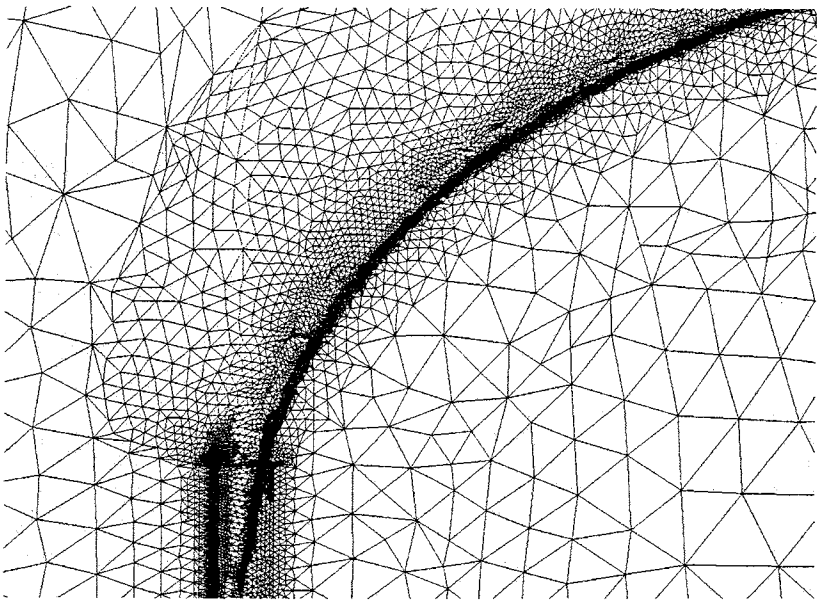

Figure 13 Final refined mesh (4 levels) for the acoustic wave problem.

The pressure distribution on the blade surface and on the plane of the rotor is presented in Figure 15. Note that the shock wave is very nicely captured, resulting in a very sharp jump. The encouraging results obtained in this preliminary simulation is the ability to target for refinement different features of the flow field, such as the strong blade shock

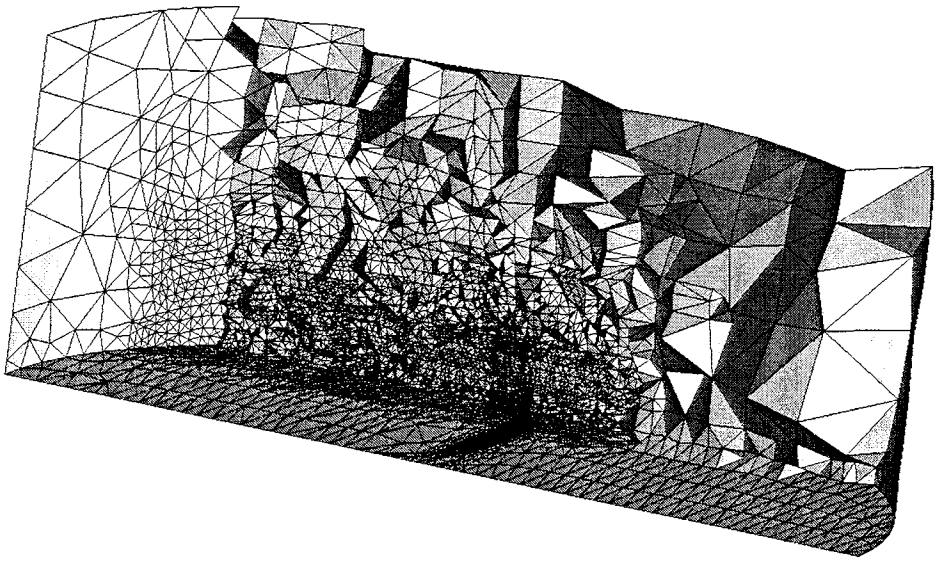

Figure 14 Cut-away of final mesh.

and the acoustic signal, indicate that a combined CFD/Kirchhoff procedure can achieve a high level of reliability and efficiency.

7. Closing Remarks

This paper has presented progress made to date on the development of parallel automated adaptive analysis procedures for unstructured meshes which operate on distributed memory MIMD computers. The procedures presented allow for the reliable analysis, through the use of automated adaptive analysis, of large problems which can only be supported by the computational power of parallel computers. Specific emphasis was placed on the techniques needed to effectively support evolving meshes such that computational load balance was maintained throughout the simulation process.

8. Acknowledgments

The authors would like to acknowledge the support of NASA Ames Research Center under grants NAG 2–832 and NCC 2–9000, the Army Research Office under grants DAAH04–93–G-0003 and DAAH04–95–G-0091, the Office of Naval Research through grant No. N00014-94-1-0962 and the National Science Foundation under grant DMS-93–18184.

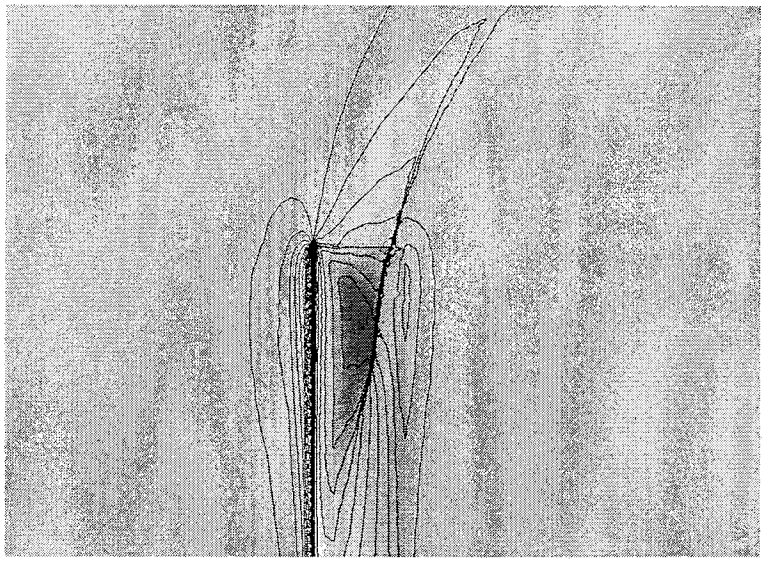

Figure 15 Pressure distribution on the blade and on the plane of the rotor for the acoustic wave problem.

9. References

[1] M. W. Beall and M. S. Shephard. Mesh data structures for advanced finite element computations. Technical Report 19-1995, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1995. submitted to Int. J. Num. Meth. Engng.

[2] R. Biswas and R. Strawn. A new procedure for dynamic adaptation of three-dimensional unstructured grids. In *31st Aero. Sci. Meet.*, 1993.

[3] G. Blelloch, C. Leiserson, B. Maggs, C. Plaxton, S. Smith, and M. Zagha. A comparison of sorting algorithms for the connection machine cm-2. *ACM*, pages 3–16, 1991.

[4] C. L. Bottasso. On the computation of the boundary integral of space–time deforming finite elements. *Comm. Num. Meth. Eng.*, 1995. under review.

[5] C. L. Bottasso and M. S. Shephard. A parallel adaptive finite element flow solver for rotary wing aerodynamics. In *12th AIAA CFD Conference*, San Diego, CA, 1995.

[6] S. D. Connell and D. G. Holmes. 3-dimensional unstructured adaptive multigrid scheme for the euler equations. *AIAA J.*, 32:1626–1632, 1994.

[7] H. L. de Cougny and M. S. Shephard. Parallel mesh adaptation by local mesh modification. Technical Report 21-1995, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1995. submitted.

[8] H. L. de Cougny and M. S. Shephard. Surface meshing using vertex insertion. In *5th Int. Meshing Roundtable '96*, number SAND96-2301, pages 243–256, Durbuy, Belgium, 1996. Sandia, Corp.

[9] H. L. de Cougny, M. S. Shephard, and C. Ozturan. Parallel three-dimensional mesh generation. *Computing Systems in Engineering*, 5(4-6):311–323, 1994.

[10] H. L. de Cougny, M. S. Shephard, and C. Ozturan. Parallel three-dimensional mesh generation on distributed memory MIMD computers. *Engineering with Computers*, 12(2):94–106, 1996.

[11] K. Devine, J. Flaherty, A. Maccabe, and S. Wheat. A massively parallel adaptive finite element method with dynamic load balancing. In *Proc. Supercomputing '93*, pages 2–11, 1964.

[12] K. M. Devine. *An adaptive HP-finite element method with dynamic load balancing for the solution of hyperbolic conservation laws on massively parallel computers*. PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institute, Troy, New York, 1994.

[13] P. L. George. *Automatic Mesh Generation*. John Wiley and Sons, Ltd, Chichester, 1991.

[14] N. Golias and T. Tsiboukis. An approach to refining three-dimensional tetrahedral meshes based on delaunay transformations. *Int. J. Numer. Meth. Engng.*, 37:793–812, 1994.

[15] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner, and K. Priess, editors, *Geometric Modeling Product Engineering*, pages 107–130. North Holland, 1990.

[16] S. W. Hammond. *Mapping Unstructured Grid Computations to Massively Parallel Computers*. PhD thesis, Computer Science Dept., Rensselaer Polytechnic Institue, Troy,, 1991.

[17] J. JaJa. *An introduction to Parallel Algorithms*. Addison Wesley, Reading Mass., 1992.

[18] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*. PhD thesis, Stanford University, July 1992.

[19] Z. Johan, T. J. R. Hughes, K. K. Mathur, and S. L. Johnsson. A data parallel finite element method for computational fluid dynamics on the connection machine system. *Comp. Meth. Appl. Mech. Engng.*, 99:113–134, 1992.

[20] Y. Kallinderis and P. Vijayan. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. *AIAA J.*, 31(8):1440–1447, August 1993.

[21] C. P. Kruskal, L. Rudolph, and M. Snir. Efficient parallel algorithms for graph problems. *Algorithmica*, 5:43–64, 1990.

[22] E. Leiss and H. Reddy. Distributed load balancing: Design and performance analysis. Technical Report Vol. 5, W. M. Keck Research Computation Laboratory, 1989.

[23] R. Löhner. An adaptive finite element scheme for transient problems in cfd. *Comp. Meth. Appl. Mech. Engng.*, 61:323–338, 1987.

[24] R. Löhner and R. Ramamurti. A parallelizable load balancing algorithm. In *Proc. of the AIAA 31st Aerospace Sciences Meeting and Exhibit*, 1993.

[25] C. Ozturan. *Distributed Environment and Load Balancing for Adaptive Unstructured Meshes*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, August 1995.

[26] T. W. Purcell. Cfd and transonic helicopter sound. Milan, Italy, 1988. 14th European Helicopter Forum.

[27] A. A. G. Requicha and H. B. Voelcker. Solid modeling: Current status and research directions. *IEEE Computer Graphics and Applications*, 3(7):25–37, 1983.

[28] H. Samet. *Application of Spatial Data Strucutres*. Addison-Wesley, 1990, Reading, Mass, 1990.

[29] V. Schmitt and F. Charpin. Pressure distributions on the onera m6 wing at transonic mach numbers. Technical Report R-702, AGARD, 1982.

[30] W. J. Schroeder and M. S. Shephard. A combined octree/Delaunay method for fully automatic 3-D mesh generation. 29:37–55, 1990.

[31] W. J. Schroeder and M. S. Shephard. On rigorous conditions for automatically generated finite element meshes. In J. Turner, J. Pegna, and M. Wozny, editors, *Product Modeling for Computer-Aided Design and Manufacturing*, pages 267–281. North Holland, 1991.

[32] R. Sedgewick. *Algorithms in C*. Addison-Wesley Publishing Company, 1990.

[33] F. Shakib. *Finite Element Analysis of the Compressible Euler and Navier-Stokes Equations*. PhD thesis, Stanford University, 1988.

[34] F. Shakib, T. J. R. Hughes, and Z. Johan. A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier Stokes equations. *Comp. Meth. Appl. Mech. Engng.*, 89:141–219, 1991.

[35] M. S. Shephard. Approaches to the automatic generation and control of finite element meshes. *Applied Mechanics Review*, 41(4):169–185, 1988.

[36] M. S. Shephard. The specification of physical attribute information for engineering analysis. *Engineering with Computers*, 4:145–155, 1988.

[37] M. S. Shephard and P. M. Finnigan. Toward automatic model generation. In A. K. Noor and J. T. Oden, editors, *State-of-the-Art Surveys on Computational Mechanics*, pages 335–366. ASME, 1989.

[38] M. S. Shephard, J. E. Flaherty, H. L. de Cougny, C. Ozturan, C. L. Bottasso, and

M. W. Beall. Parallel automated adaptive procedures for unstructured meshes. In *Parallel Computing in CFD*, volume R-807, pages 6.1–6.49. AGARD, Neuilly-Sur-Seine, France, 1995.

[39] M. S. Shephard and M. K. Georges. Reliability of automatic 3-D mesh generation. *Comp. Meth. Appl. Mech. Engng.*, 101:443–462, 1992.

[40] M. S. Shephard and N. P. Weatherill, editors. volume 32. Wiley-Interscience, Chichester, England, 1991.

[41] M. L. Simone, H. L. de Cougny, and M. S. Shephard. Tools and techniques for parallel grid generation. In *Numerical Grid Generation in Computational Field Simulations, Vol. II*, pages 1165–1174, Mississippi State, Miss., 1996. NSF Research Center for Computational Field Simulation.

[42] M. L. Simone and M. S. Shephard. A constant-time neighbor-finding algorithm for spatial octrees: Serial and parallel implementation. Technical report, Scientific Computation Research Center, RPI, Troy, NY, 1996.

[43] R. Strawn, M. Garceau, and R. Biswas. Unstructured adaptive mesh computations of rotorcraft high-speed impulsive noise. Long Beach, CA, 1993. 15th AIAA Aeroacoustics Conference.

[44] B. K. Szymanski and A. Minczuk. A representation of a distribution power network graph. *Archiwum Elektrotechniki*, 27(2):367–380, 1978.

[45] T. E. Tezduyar, M. Behr, S. Mittal, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces - the deforming-spatial-domain/space time procedure: I. the concept and preliminary tests. *Comp. Meth. Appl. Mech. Engng.*, 94:339–351, 1992.

[46] A. Vidwans, Y. Kallinderis, and Venkatakrishnan. Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids. *AIAA Journal*, 32(3):497–505, March 1994.

[47] K. J. Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 3–36. North Holland, 1988.