Status Report for NASA Grant NAG 2-832

Covering Period from 4/1/93 to 9/30/93

PARALLEL UNSTRUCTURED GRID GENERATION FOR COMPUTATIONAL AEROSCIENCES

Mark S. Shephard
Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

Abstract

The objective of this research project is to develop efficient parallel automatic grid generation procedures for use in computational aerosciences. This effort is focused on a parallel version of the Finite Octree grid generator [17]. This report represents discusses the progress made during the first six months of support.

1. Introduction

The development of automatic grid generation techniques for complex three-dimensional configurations has been an active area of research for over a decade. The most complex aspects of these techniques are the computational geometry issues associated with ensuring the grids generated represent a valid discretization [14, 13, 16] of the original geometric domain as it is defined in a geometric modeling system. All of the complex issues of automatic grid generation are associated with domains bounded by general curved surfaces. From a computational geometry viewpoint, the automatic generation of 3-D grids for domains bounded by planar surfaces (including those already discretized into a planar surface grid) is straightforward. This effort is focused on the development of parallel grid generation techniques for general domains bounded by curved surfaces with no requirement to pre-triangulate the surface.

As discussed in the original proposal, the previous efforts on creating parallel automatic mesh generators have focused on performing triangulation of a given discrete domain [3, 8], or have simulated the parallelization of the computations without full consideration of the communications [12] overhead. In this effort consideration is being given to the parallel implementation of the entire mesh generation process. The non-uniform nature of the computations associated with automatic mesh generation dictate the use of MIMD computing methodologies. To provide the highest degree of transportability between MIMD computers, the procedures being developed employ message-passing.

The efforts to date on the development of a parallel Finite Octree have considered the changes required for those portions of the current serial procedure that should be implemented using different methods, and the parallel implementation of those parts that will stay much the same in the final version. The portion of the Finite Octree procedure requiring a detailed evaluation of alternative implementations is the construction of the first level discretization, the finite octree. The portions requiring the least consideration of major change are associated with the creation of the finite element mesh within the finite octree. As indicated below, a parallel implementation of most of the element creation

steps is in place, and, after evaluation of a number of alternatives, an approach for the parallel implementation of the finite octree construction process has been selected.

2. Construction of the Finite Octree

As discussed in the original proposal, much of the effort in the Finite Octree automatic mesh generation is spent during tree building determining the interactions of the boundary of the object with the boundary of the octree cells. Since the amount of computational effort required to determine these interactions varies dramatically from octant to octant, and it is not known until it has been determined, the effective parallelization of this portion of the process is quite complex. The two most obvious approaches to parallelize the tree-building process are (i) to control the process through the octree structure distributing effort to processors on an octant by octant basis, and (ii) the parallel insertion of geometric entities into the Finite Octree by distributing the edges and faces bounding the object among the processors. Substantial effort has been spent on devising alternative strategies within those two approaches.

Careful evaluation of the controlling parallelization by distributing geometric entities to the processors indicated the following disadvantages:

1. Interprocessor communication would most likely be quite high since information on individual octants would reside on each processor housing a geometric entity which interacts with it. The performance of mesh validity checks involving the interaction of different geometric entities would require an extremely high amount of interprocessor communication with little computation per communication.
2. Since parallel implementation of later steps in the process will be best coordinated through the octree structure, a redistribution of the mesh information based on the tree, after the tree is built would be required.

Therefore, effort has focused on methods to parallelize through the distribution of the tree. In this approach the major question is the most effective method to construct the tree in parallel. An examination of procedures developed in image synthesis [5] and ray tracing [6] were useful here. One approach to do this starting from the current serial tree building procedure is to consider the distribution of octants to processors as arise during the edge, face and region insertion process. Although having the potential advantage of best load balance, this approach has the two disadvantages of (i) a high level of interprocessor communications and (ii) difficulty in trying to distribute neighboring

octants to the same processor, leading to additional interprocessor communications in later steps of the meshing process.

An alternative approach to the construction of the finite octree, which will reduce the amount of interprocessor communication, is to employ recursive subdivision. In recursive subdivision the process of determining the interactions of octant and model entities begins with the root octant. This octant is then subdivided into eight octants only determining the pointwise interactions of the model entities with the new octant entities. The process of subdividing octants is recursively performed until the correct sized terminal octants are obtained.

The introduction of a recursive subdivision approach, which has been used in octree/Delaunay techniques [4, 15], does introduce a number of alterations to the ordering and specific aspects of algorithmic steps within the Finite Octree procedure. The main reason for the required changes is that in the current Finite Octree procedure the various finite element vertices, edges and faces are constructed as the geometric entity is processed. In recursive subdivision, it is only reasonable to determine the mesh vertices due to the intersection of model edges with octant faces and the intersection of octant edges with model faces. The process of constructing the appropriate mesh edges and faces must then be done after all the mesh vertices, to ensure the required interactions are complete. Therefore, the steps in the mesh generation become:

1. Tree building by recursive subdivision
2. Enforcement of 2:1 level of difference between octant edge neighboring terminal octants
3. Creation of mesh edges classified on model edge
4. Creation of mesh edges classified on the closure of the octant face and model face
5. Creation of octant level loops classified on model face
6. Classification of terminal octant corners and creation of mesh edges classified on the closure of the octant face and model region
7. Creation of octant level loops classified on model region
8. Octant level loop triangulation
9. Terminal octant tetrahedronization
10. Node point repositioning

The basic steps involved with the parallel recursive subdivision process are:

1. Create 8 child octants
2. Transfer mesh vertices from parent to child octants

3. Intersect all model edges in parent octant with 3 auxiliary parent octant faces (Fig. 1)
4. Intersect all model faces in parent octant with up to 15 auxiliary parent octant edges (Fig. 1)
5. For each intersection, create a mesh vertex and make it known to the proper terminal octants
6. Update global tree (known by each processor)
7. Repartition tree
8. Migrate octants

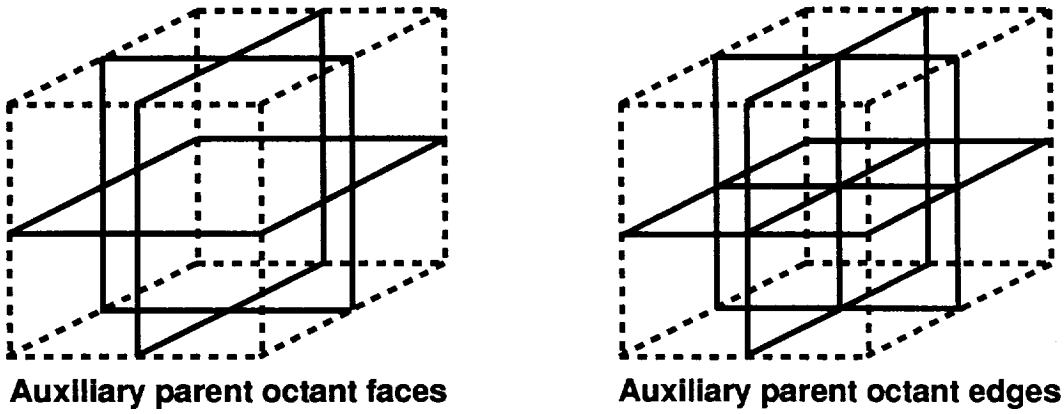

Figure 1. Auxiliary parent octant faces and edges.

A critical aspect of the parallel version of the procedures is the distribution of the octants to processors and performing repartitioning, when needed, to maintain load balance [7, 9, 11]. As the octree workload becomes too unbalanced, it will be repartitioned by first estimating the workload to determine the work per processor. A linear traversal of the tree will then be used to reassign the terminal nodes to the available processors. This process is depicted in Figure 2. Other partition algorithms to be considered are orthogonal recursive bisection [10], and the repartitioning procedure given by Berger and Bokhari in reference [2]. In this example four processors are assumed and the amount of effort per terminal octant is assumed equal. Figure 3 shows the partitions of the FLANGE model example created using the (re)partitioning procedures.

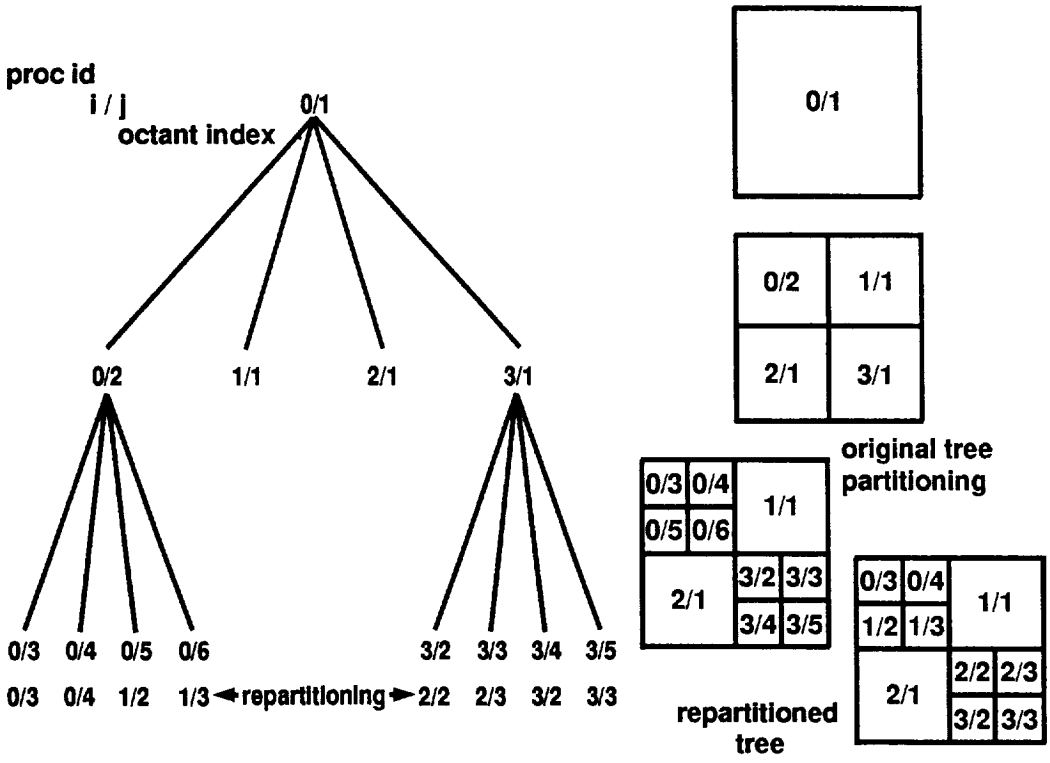

Figure 2. Redistribution of terminal octants to processors.

3. Construction of Mesh Edges on Model Edges

The mesh edges classified on the model edges are created based on the mesh vertices defined during the recursive subdivision process by the intersection of model edges with octant faces. The overall procedure for the parallel construction of this information is:

1. For each terminal octant known by given processor and for each model edge known by the octant:
   a. Create a mesh edge each time the model edge enters and exits the octant (Fig. 4)
   b. If mesh edge is classified on an octant edge or face, communicate its creation to neighboring processors (if any)
2. Receive pending messages and create mesh edges accordingly

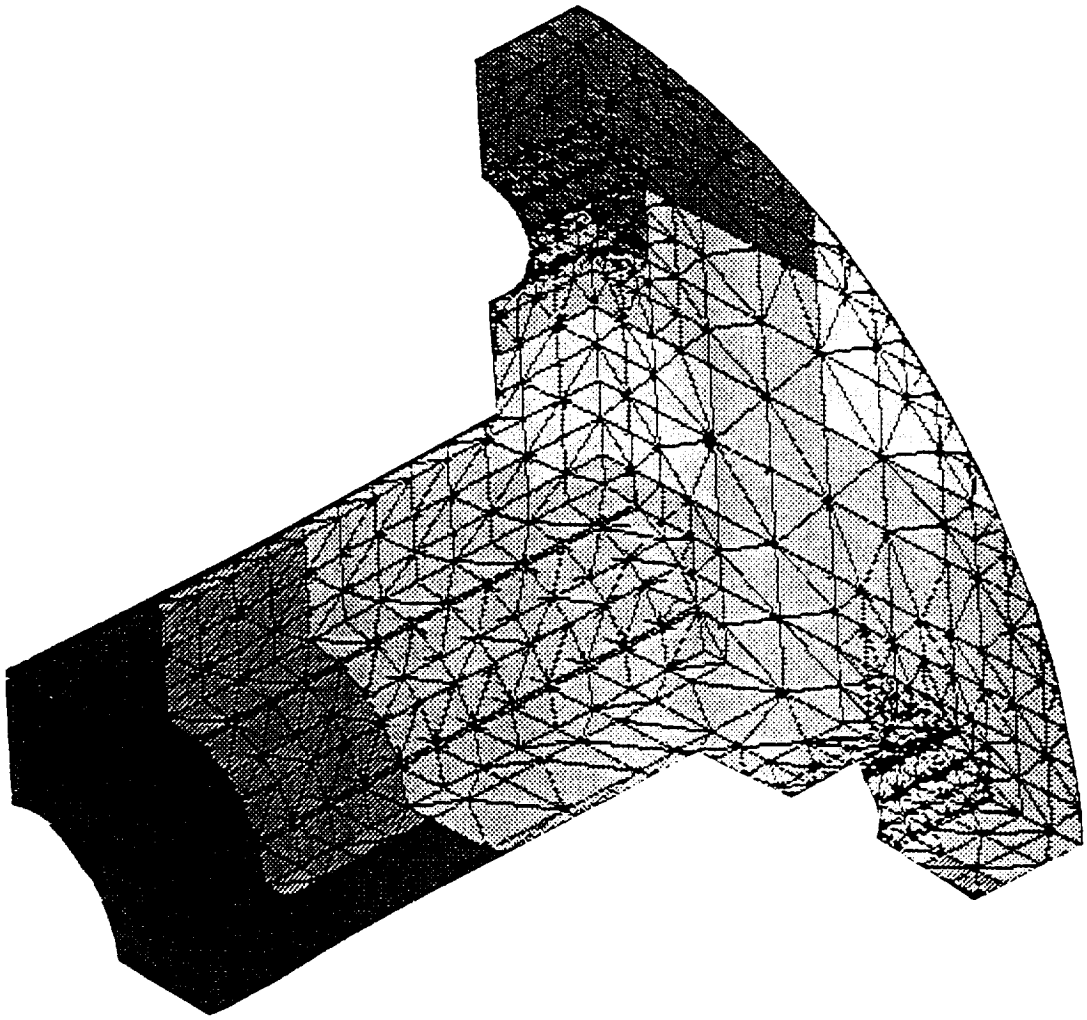

Figure 3. Partitioned mesh for FLANGE model.

4. Mesh Edges on Octant Faces and Model Faces

The mesh edges classified on octant faces and model faces are created based on the mesh vertices defined during the recursive subdivision process by the intersection of model edges with octant faces, and octant edges with model faces. The overall procedure for the parallel construction of this information is:

1. Given a terminal octant face and a model face known by the octant face:

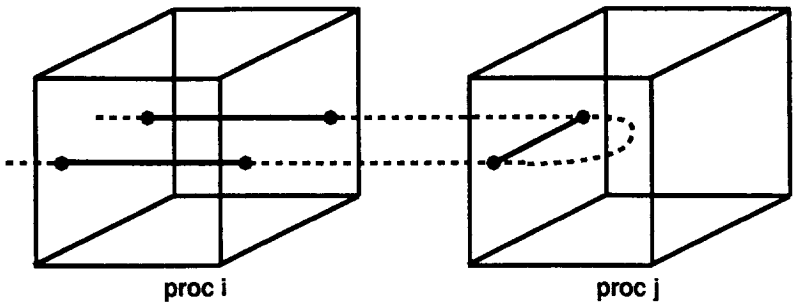

Figure 4. Creation of mesh edges on model edges.

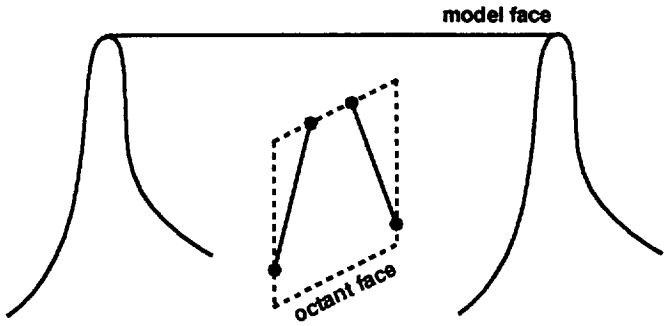

Figure 5. Mesh edges on octant faces and model faces.

   a. Gather all mesh vertices classified both on octant face's closure and model face's closure
   b. Create a mesh edge between 2 vertices if it discretizes properly the trace of the model face on the octant face (Fig. 5)

2. For each terminal octant known by given processor:

   a. Process "left" octant face with respect to any model face known by the octant face
   b. Send data to neighboring processors (if any)

3. Receive pending messages and create mesh edges accordingly
4. For each terminal octant known by given processor, process "right" octant face with respect to any model face known by the octant face (if not already done)
5. Do the same for bottom-top and back-front octant faces

5. Creation of Octant Level Loops Classified on Model Faces

Once all the mesh edges have been created, the mesh edge loops needed by the octant level triangulation procedures must be created. The overall procedure for the parallel construction of this information is:

1. For each terminal octant and each model face known by the octant (Fig. 6):
    a. Gather all mesh vertices and mesh edges classified both on octant's closure and model face's closure
    b. Assign 1 directed edge_use to any mesh edge classified on a peripheral model edge and 2 on any other
    c. Create loops so that all edges are used the proper number of times
    d. Reject any loop that goes in opposite direction of the model face
    e. Possibly connect loops together to form simply connected loops

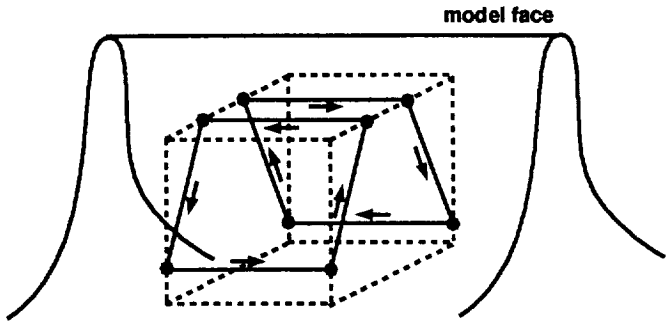

Figure 6. Octant level model face loop building.

2. If an octant face is shared by 2 processors and a loop is defined fully within that face, the 2 loop versions are implicitly guaranteed to be identical

6. Octant Level Loop Triangulation

Once all the octant level loops have been constructed they can be triangulated. The current implementation of Finite Octree employs an element removal procedure based on the triangle removal operators shown in Figure 7. The major complexity in the parallelization of this procedure is the need to ensure that only a single triangulation is obtained for each loop. The basic approach taken to the parallelization of this process is to order the triangulating of octant face loops based on position on the octant boundary, and

then, for those octant faces on interprocessor boundaries, to communicate triangulations once they have been created. The procedures developed use the CM5 message passing procedures [1].

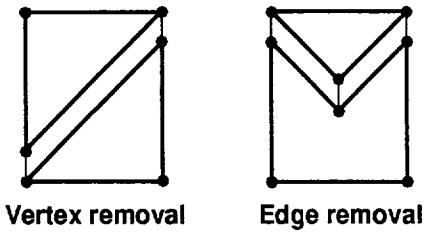

Figure 7. Triangle removal operators.

The steps in the octant level loop triangulation process are:

1. For each terminal octant known by given processor P triangulate "left" octant faces and prepare communication information needed for partition neighbors
   a. Triangulate all loops (faces) classified on the "left" octant face
   b. For each created mesh face on the partition boundary:
      - Get neighboring processor P'
      - Pack information about how to create that mesh face and the neighboring processor into array A
      - Place a 1 into array C to indicate that P will send at least 1 message to P'
2. For each P' appearing in A, reshuffle A and send to P' asynchronously (Fig. 8)

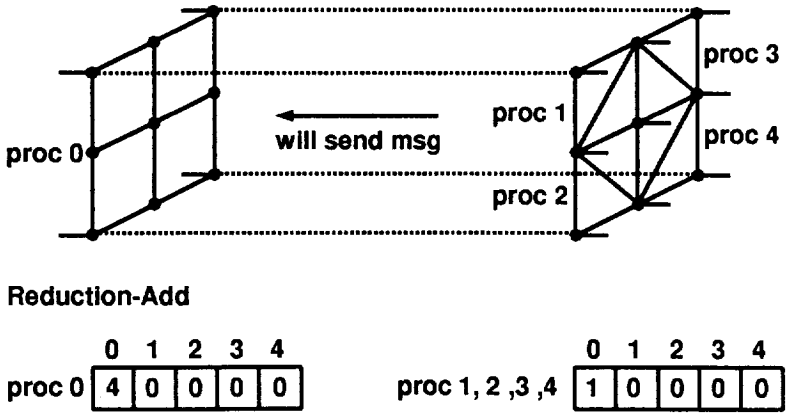

Figure 8. Message marking for octant face triangulation.

3. Perform reduction operation (add) on array C to indicate how many messages should be received (Fig. 8)
4. For each pending message:
   a. Get size of message
   b. Receive message (block)
   c. For each set of data, create up to 3 mesh edges and a mesh face (if not already existing)
   d. If mesh entities already exist, they may have to be reclassified if the message indicates that they are classified on a lower order model entity (Fig. 9)
5. For each terminal octant known by the processor, triangulate the "right" face loops on the right hand boundary of the model that have not yet been triangulated.
6. Repeat steps 1–5 for "bottom-top" and "back-front" octant faces
7. For each terminal boundary octant known to a given processor, triangulate the interior face loops:
   a. Triangulate all loops classified in the octant interior
   b. If an already existing mesh face needs to be reclassified (and possibly bounding mesh edges), reclassify mesh face (and possibly bounding mesh edges) (Fig. 10)

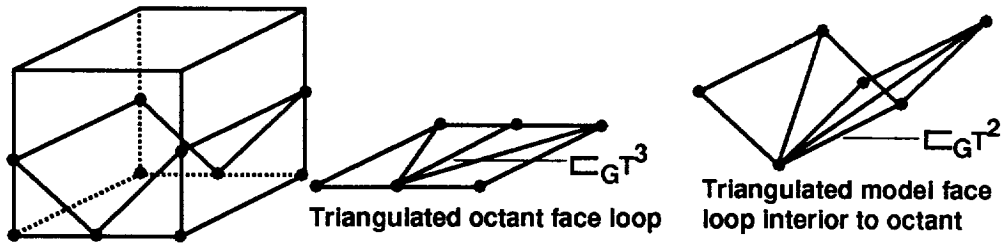

Figure 9. Model face loop interior to octant: Case of mesh edge reclassification.

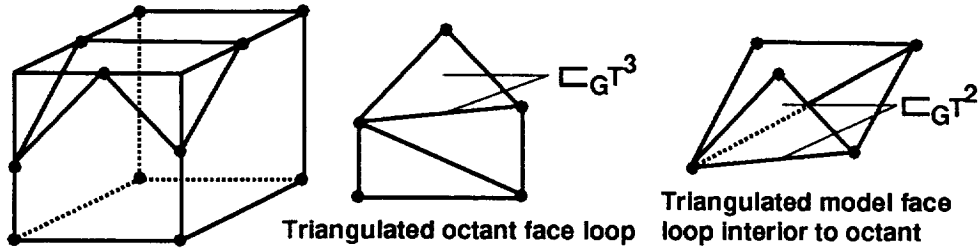

Figure 10. Model face loop interior to octant: Case of mesh face and edge reclassification.

The octant level loop triangulation procedures have been implemented using the above procedure. Figure 11 shows the CPU times required for triangulation for the FLANGE model example shown in Figure 3 as the number of processors is increased from 1 to 64.

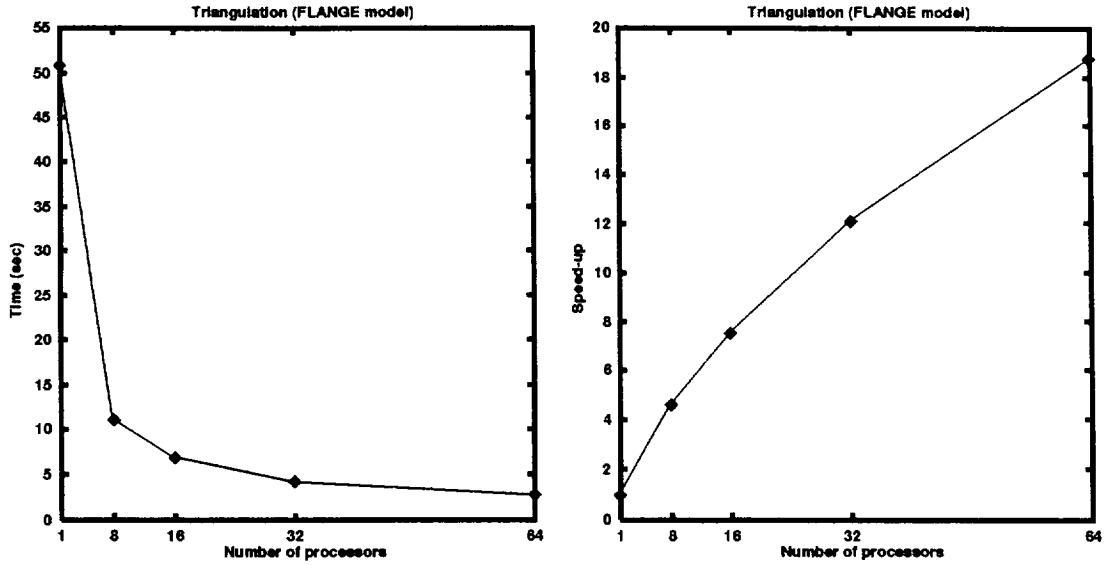

Figure 11. Triangulation timings (FLANGE model).

7. Octant Tetrahedronization

Once the octant level mesh loops have been triangulated the tetrahedra are generated in parallel on an octant-by-octant basis using element removal operators (Fig. 12). This is a straight forward process since no interprocessor communication is required. This process is carried out using the partitioning of the finite octree already constructed. Figure 13 shows the CPU time required for the FLANGE model example shown in Figure 3 as the number of processors is increased from 1 to 64.

8. References

[1] *CMMD reference manual version 3.0*. Thinking Machines Corporation, Cambridge, MA, 1993.

[2] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

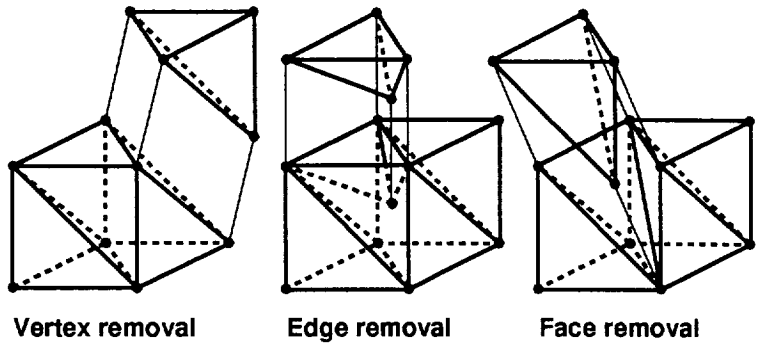

Figure 12. 3-D element removal operators.

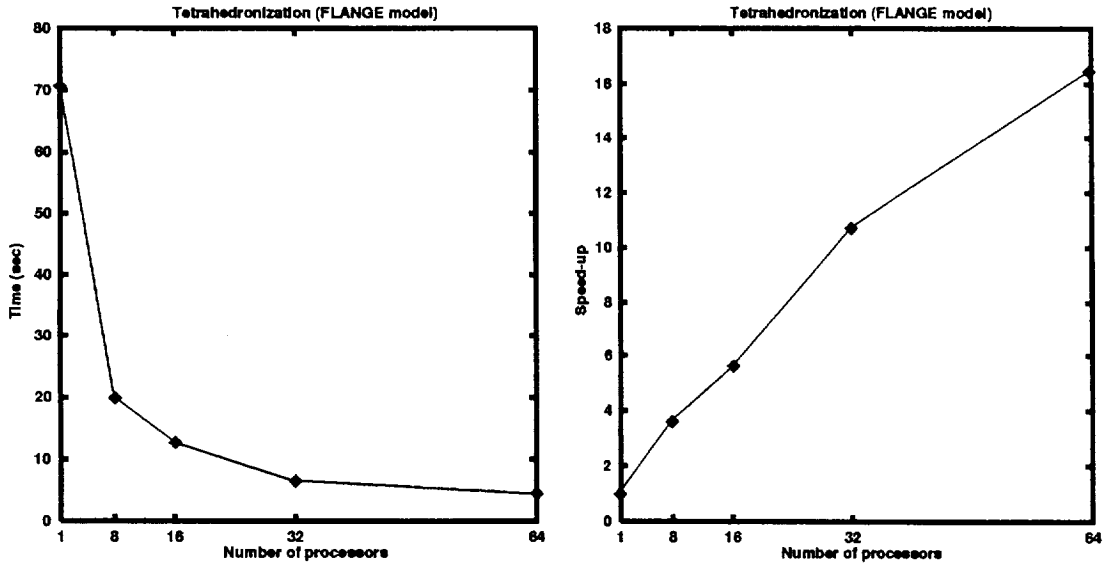

Figure 13. Tetrahedronization timings (FLANGE model).

[3] F. Cheng and M. Mehra. Parallel mesh generation for objects composed of 3D free-formed surfaces. 1991. submitted for publication.

[4] H. L. de Cougny. Automatic generation of geometric triangulations based on octree/Delaunay techniques. Master's thesis, Civil and Environmental Engineering, Scientific Computation Research Center, Rensselaer Polytechnic Institute,Troy, NY 12180-3590, May 1992. SCOREC Report # 6-1992.

[5] M. Dippe and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *Computer Graphics*, 18(3):149-158, 1984.

[6] S. Green and D. Paddon. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications*, 9(6):12–26, 1989.

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

[8] R. Löhner, J. Camberos, and M. Merriam. Parallel unstructured grid generation. In *Proc. 1991 AIAA CFD Conf.*, pages 627–637. AIAA, 1991. AIAA-91-1582-CP.

[9] D. Nicol and R. Fujimoto. Parallel simulation today. Technical Report ICASE Report No. 92-62, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia, 1992.

[10] C. Pommerell, M. Annaratone, and W. Fichtner. A set of new mapping and coloring heuristics for distributed memory parallel processors. *SIAM J. on Scientific Computing*, 13(1):194–226, January 1992.

[11] H. H. Reif, editor. *Synthesis of Parallel Algorithms*. M. Kaufmann, San Matro, CA, 1993.

[12] M. Saxena and R. Perucchio. Parallel fem algorithms based on recursive spatial decompositions - I. automatic mesh generation. *Computers and Structures*, 45:817–831, 1992.

[13] W. J. Schroeder. *Geometric Triangulations: with Application to Fully Automatic 3D Mesh Generation*. PhD thesis, Rensselaer Polytechnic Institute, Scientific Computation Research Center, RPI, Troy, NY 12180-3590, May 1991.

[14] W. J. Schroeder and M. S. Shephard. An O(N) algorithm to automatically generate geometric triangulations satisfying the Delaunay circumsphere criteria. *Engng. with Computers*, 5(3/4):177–194, 1989.

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

[16] 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.

[17] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *Int. J. Numer. Meth. Engng.*, 32(4):709–749, 1991.