

Tetrahedral Mesh Refinement in Distributed Environments

Mehmet Balman
Department of Computer Science
Louisiana State University
balman@csc.lsu.edu

Abstract

The Adaptive Mesh Refinement is one of the main techniques used for the solution of Partial Differential Equations. Since 3-dimensional structures are more complex, there are few refinement methods especially for parallel environments. On the other hand, many algorithms have been proposed for 2-dimensional structures. We analyzed the Rivara's longest-edge bisection algorithm, studied parallelization techniques for the problem, and presented a parallel methodology for the refinement of non-uniform tetrahedral meshes. The main goal of this research is to propose a practical refinement framework for real-life applications. We describe a usable data structure for distributed environments and present a utility capable of distributing the mesh data among processors to solve large mesh structures.

1. Introduction

Adaptive Mesh Refinement (AMR) is a technique used to effectively solve numerical systems of Partial Differential Equations (PDE). Instead of processing the uniform mesh in which grid points are evenly spaced, we place more grid points to the areas where local error is large in the solution. The adaptive mesh refinement is the preferred methodology in terms of computational and storage requirements. Refinement algorithms begin with an initial mesh conforming to a particular geometry, and the conformity of the overall structure must be preserved after partitioning an element [33, 36]. Most of the research have focused on mesh components as line segments in 1-dimension, triangles in 2-dimension, and tetrahedra in 3-dimension [34, 42].

The triangle refinement process have been studied briefly in recent studies [41, 18, 39]. Since we cannot analyze the elements in a planar view, 3-dimensional structures yield to complexities and difficulties. Refining using the skeleton structure is the main idea behind the algorithms [33]; 3-dimension is reduced to 2-dimension, and then to 1-dimension. The skeleton structure of meshes in all views should preserve conformity, and partitioning of the origi-

nal mesh is refined according to the information in previous skeleton structures.

The main intention behind this research is to enhance the 8-Tetrahedra Longest-Edge (8T-LE) technique [34] and propose a parallel algorithm with a suitable data structure that is applicable in real life.

During the process of the Differential Equations, the size of the used memory will increase since the geometry of the mesh should be extracted for computation. We cannot locate all of the required elements on a single machine; therefore, we should distribute both computational power and the stored data among distributed processors.

The overall mesh structure can be partitioned in order to fit into the local memory of computational nodes, and the Rivara's longest edge bisection technique can be used to process the refinement operation locally; results in each node can be synchronized in a proper and efficient way using an appropriate data structure to handle the refinement process in a parallel manner, in which the resulting mesh data is parallely constructed.

This work analyzes the *longest-edge bisection* procedure in details and presents a novel methodology solving the refinement problem in parallel. It also proposes a data structure to store elements efficiently. The refinement framework described in this study is implemented for Message-Passing (MPI) environments.

2. Mesh refinement

Many numerical applications and simulations, solid modeling, and computer graphics require geometric objects to be partitioned into smaller pieces in order to process and solve related problems. Triangulating a set of points is the basic tool in finite element method and computational geometry [16, 6]. Therefore, mesh refinement algorithms have a critical role in adaptive finite elements of numerical computations. Especially, 3-dimensional structures have difficulties in construction of good quality and adapted to geometry solutions [33, 35, 1, 49, 5, 13, 23].

Two approaches have been mainly used to overcome the refinement problem in 2-D. The first approach is the

longest-edge bisection process which guarantees a good-quality, conforming mesh structure with linear time complexity [33, 6, 23, 38]. The second approach is based on the *Delaunay algorithm*, which can be summarized as adding non-vertex points in the circumcenter of the worst triangles of the current structure [44, 45, 37]. *Delaunay* refinement assures the construction of most equilateral triangulation at the optimum time complexity $O(N \log N)$ for a given mesh structure of N vertices [44]. However, the second method cannot be applied easily to 3-dimensions, and new approaches are needed for tetrahedral mesh refinement using a *Delaunay* triangulation based construction [44, 46]. Therefore, *Longest-Edge Bisection* method is mostly applied due to its straightforward and common implementation in the refinement process.

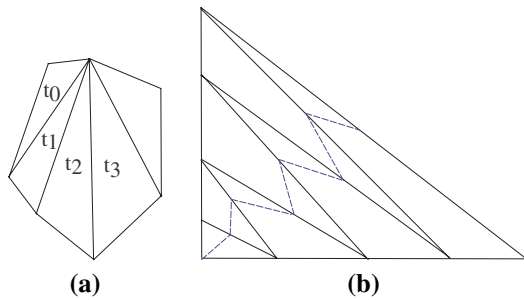


Figure 1. Longest-edge propagation

Intersection of adjacent triangles is either a common vertex or common edge. For any triangular mesh structure T , the longest-edge propagation path (*LEPP*) for a triangle t_0 is the ordered list of triangles $\{t_0, t_1, t_2, \dots\}$, such that t_i is adjacent to triangle t_{i-1} by the longest-edge of t_{i-1} [33, 42, 6, 1, 48]. Evaluating and computing the longest-edge propagation path means regarding the elements that will be bisected before partitioning any of the triangles since the longest-edge bisection method finds out effected neighbor triangles [41, 6, 40, 7, 18, 38]. $LEPP(t_0)$, the longest-edge or longest-side propagation path of triangle t_0 , is always finite and longest-edges in the list of triangles have always increasing lengths [33, 36, 34, 42, 41, 18, 39, 40]. Figure 1.a shows the $LEPP(t_0) = \{t_0, t_1, t_2, t_3\}$, propagation path for triangle t_0 . Triangulation of refinement problems can be solved via evaluating the longest-edge propagation path. We can compute the *LEPP* and bisect elements in the list to accomplish the refinement process. In the given example, terminal triangle pairs $t_3 - t_2, t_2 - t_1, t_1 - t_0$ will be bisected in order. This procedure, starting from an initial conforming geometry, will produce a good-quality nested triangulation with linear time complexity [36, 42, 37].

The Longest-edge Propagation Path algorithm can be generalized to 3-dimensional tetrahedral mesh structures. The 3-D *LEPP* for a tetrahedron τ , is the set of neigh-

boring tetrahedra that have adjacent longest-edge greater or equal to the preceding tetrahedra in the list [34, 42, 6, 1, 48, 43].

Longest-edges are bisected progressively so all angles in triangle refinement are greater or equal to half of the smallest angle in the initial mesh geometry [33, 42, 41, 18, 37, 40]. Thus, known longest-edge refinement algorithms guarantee the construction of smooth and conforming structures. Longest-edge bisection can propagate to the entire mesh in worst cases. Propagation is accomplished by traversing the longest-edge neighbor triangles of triangle t . Figure 1.b describes an eccentric situation. However, theoretical results and experiments show that successive processing of unstructured triangular mesh refinement results in mesh structures in which the average propagation path is reduced in each refinement stage and approaches to the constant of 5 [48].

For 2-dimensional mesh refinement, we can restrict the length of the propagation path. This behavior is crucial in terms of analyzing the algorithms and performance of the bisection process. However, such a limitation cannot be stated for 3-dimensional tetrahedral meshes. It should be noted that this property is the most important difference between 2-dimensional and 3-dimensional refinement algorithms; lack of such a limiting definition results in the challenge for 3-dimensional problems.

In 2-D meshes, a *LEPP-graph* forms a forest since each edge can only have two neighbor triangles. Each tree in the forest can be used to find the elements that should be refined for a conforming mesh structure [32]. A propagation path for 3-D mesh forms a directed-acyclic graph (*DAG*) such that each longest edge can be shared by many triangles and refinement operation can propagate in many directions. Therefore, the Longest-edge propagation graphs for 3-D meshes are *denser*.

The *LEPP* graph can be sequentially processed with $O(n)$ time complexity [33, 36, 18, 39, 20, 8]. Parallel implementation can be handled in $O(\log n)$ time if the structure is 2-D [32]. However, it cannot be stated for 3-D mesh, since the number of edges in the propagation graph is not linearly related to the number of elements as in a *LEPP* graph of a 2-D mesh.

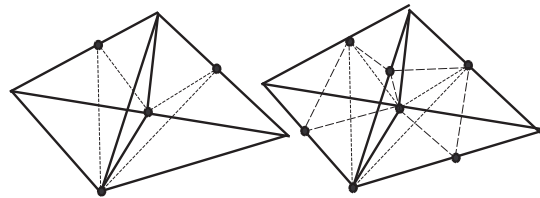


Figure 2. 4-Tetrahedra and 8-Tetrahedra

The *8T-LE* partition is defined in terms of a polyhe-

dron skeleton concept using a simple edge-midpoint bisection procedure. The 2-dimensional algorithm, which is also formulated as 4-triangle longest-edge, works over wireframe meshes containing the edges of target triangles and some neighboring triangles to prepare a conforming structure [23, 36, 7, 47].

Information in the lower dimension is used to partition appropriate triangles. The 8-tetrahedra algorithm is the generalization of skeleton algorithms in 3-dimensions. Volume refinement is based on the partitioned triangular faces of the tetrahedron in 2-dimension [33, 34, 42]. Four-triangle partitions or partial partitions of neighboring triangles are accomplished by using edge bisection; that is, by refining the wireframe mesh of the 3-dimensional edges of tetrahedra [1, 7].

The skeleton algorithm for 4-Triangle mesh refinement can be analyzed in two steps; bisecting the edges in a 1-dimensional skeleton and partitioning individual triangles according to the bisected edges [6, 7]. The 3-dimensional skeleton algorithm is a generalized version of 4-Triangles. If tetrahedral mesh τ is conforming, then 2-D skeleton, which is the triangular faces of the elements of τ , is also conforming [34]. Moreover, a 1-D skeleton of τ tetrahedral mesh is a conforming wireframe mesh of the elements of τ [34, 6, 7].

The 8-Tetrahedra longest edge partition is a 3-D algorithm that can be explained by applying 4-Triangle skeleton refinement methodology to the faces of corresponding mesh τ [34]. Partitioning any tetrahedron τ in mesh τ produces both conforming volume mesh and conforming surface mesh. Figure 2 shows the 4-Tetrahedra and 8-Tetrahedra longest-edge bisection schemes.

Volume triangulation with 8 new internal tetrahedra occurred and each face has been partitioned into 4 triangles. There is an interior edge from the midpoint of the longest-edge of τ to the midpoint of the edge opposite the longest-edge. Such a triangulation produces conformity both in volume and surface structure. Surface structure is identical to the pattern obtained by 4-Triangle partitioning.

The 3-D algorithm for refining any tetrahedron τ in a conforming mesh τ is a generalized version of a 2-D skeleton refinement algorithm. The volume structure of the refined mesh based on the 8T-LE also produces the refined 2-D skeleton surface structure. Moreover, refinement of the faces of τ as a 2-D skeleton structure in tetrahedra mesh τ produces a refined 2-D volume structure. The sequential algorithm for 3D skeleton refinement is finite with linear complexity $O(n)$ [34].

3. Parallel algorithm

Because of the excessive size of mesh structures used in current research projects, developing a parallel refinement algorithm is a crucial topic for Partial Differential Equation

(PDE) problems. There are many related projects investigating an effective procedure that is applicable to adaptive meshes [20, 30, 9, 8, 15, 31, 19, 2, 29, 12, 24, 22]. The most recent methodology for parallel refinement is based on *terminal edges*, which are defined as edges that do not propagate and do not cause other elements to be refined [43]. The *Terminal-Edge Bisection* procedure has sequential and parallel solutions, and the main idea is to bisect the *terminal-edge* which is the longest among selected edges first and to continue this process until all of required elements are refined [43]. However, there are some drawbacks in such a solution like partitioning more components than required and increasing the number of elements in the resulting 3-D structure, which already necessitates an extreme number of resources. This solution may not be practical despite the flexibility of the proposed technique.

Theoretical limits for parallel tetrahedral mesh refinement depend on the processing of the *LEPP* graph. It states the elements that are affected after an initial tetrahedron refinement. If we compute and mark those elements to be refined, other parts such as bisection steps are easily processed in a parallel environment since they will not depend on one another.

The propagation graph does not have a specific property that can be used to find reachable elements within a parallel algorithm. On the other hand, there are some approaches for similar methodologies; the *LEPP* graph is a directed-acyclic graph (*DAG*) and the *DAG* can be evaluated in many parallel ways. Most of the related techniques use random algorithms and state effective complexity times [28, 11, 25, 10, 21]. Average complexity may have proper values, but worst case complexity is not acceptable when compared to sequential algorithms. Those parallel techniques are probably not practically applicable for mesh refinement; the refinement process is another sub-component of the *PDE* problem and should be simple enough for the implementation.

The sequential algorithm has complexity $O(n)$; when data is distributed and evaluated in a parallel manner, we should deal with the relations between propagation paths. In 3-D, each element in the *LEPP* graph can have more than one propagation. Thus, the number of edges in the *LEPP* graph, which are keeping the propagation relationship, is $O(n)$ [5, 43], if n is the number of elements. Therefore, we can process *LEPP* graphs in $O(\log n)$ time with n^2 processors. For the 2-D algorithm, elements propagate to only one other element, and the number of edges in the *LEPP* graph is $O(n)$; thus, the *LEPP* graph can be processed in $O(\log n)$ time with n processors.

We start from an initial tetrahedron τ , refine according to *LE*-bisection and progress to find other elements that must be partitioned. The second step is selecting all remaining components that should be refined to form a conforming mesh structure. Propagating through the initial components

Distributed Algorithm

(P processors, Tetrahedral Mesh τ):

1. Distribute the Tetrahedral Mesh Structure τ ,
2. each processor P_i handles its local mesh structure.
3. Process the Longest-Edge Algorithm locally:
4. foreach edge e of tetra τ that needs to be refined
5. add edge e to the list of selected edges, $E_{selected}$
6. *LEPP* algorithm:
7. while all edges in $E_{selected}$ are processed;
8. if *LE* of the face f that edge e belongs is not selected
9. add edge *LE* to $E_{selected}$
10. add tetra τ that owns edge e to $T_{selected}$
11. Synchronize local Propagation Paths:
12. if local terminal point in the *LEPP* also belongs to another local mesh structure τ owned by processor P_i
13. inform processor P_i .
14. Process the *LEPP* algorithm after synchronization.
15. Bisect selected edges $E_{selected}$
16. Bisect selected tetrahedra $T_{selected}$
17. Collect local mesh structure from processing nodes.

Figure 3. Distributed algorithm

will lead us to all other elements that corrupt the conformity.

The next step is combining components to form a conforming mesh structure. Due to the nature of structural algorithms, explained in the previous section, we refine faces according to the refined one-dimensional edges; and refine tetrahedron according to 2- D faces. Components in former steps represent the edges in the one-dimensional skeleton.

The parallel algorithm can be analyzed in three steps:

- Prepare propagation graph for the refinement, (*DAG*) for 3- D mesh τ .
- Find components which must be refined.
- Partition components according to the longest-edge bisection procedure.

The 3-dimensional mesh refinement solution for adaptive structures should be effective in terms of scalability, distributed costs, and partitioned data. Required memory for tetrahedral mesh structures increases if compared to 2-dimensional structures. Therefore, distributing the computational power with partitioned data structure is crucial if large structures are concerned.

The distributed algorithm accomplishes refinement problems by utilizing the local bisection procedure and synchronizing partitioned tetrahedra. Since propagation-paths are distributed, terminal points for a local mesh may trigger another *LEPP* globally. In Figure 3, we demonstrated the overall algorithm.

The propagation path is distributed among each proces-

sor, and they compute local *LEPP*-Graphs independently. After the local refinement process, computing nodes are informed to trigger the refinement if the border element of the local mesh partition is selected to be bisected by another processor. Figure 4 presents the logically partitioned *LEPP*-Graph. In the given example, overall structure is partitioned among 3 processors. Node 13 and 14 have a common *longest-edge*; thus, refining Node 13 representing a tetrahedron in the figure results in propagation of the *LEPP* and refinement of Node 14. The other processor is informed that neighbor node in the border of the local partition should be refined. Therefore, the *LEPP* graph of the local mesh structure is synchronized and the integrity of the overall propagation paths is preserved.

The synchronization process is limited and cannot exceed a few loops due to the conforming structure of input mesh structure. In real-life problems, we usually start to refine some tetrahedra, causing an unacceptable error ratio in a *PDE* problem. Such a situation will not propagate to all other elements of the mesh; thus, handling large mesh structures and distributing them among remote processor to compute at the same time is more important. Figure 5 presents the flow-chart representation.

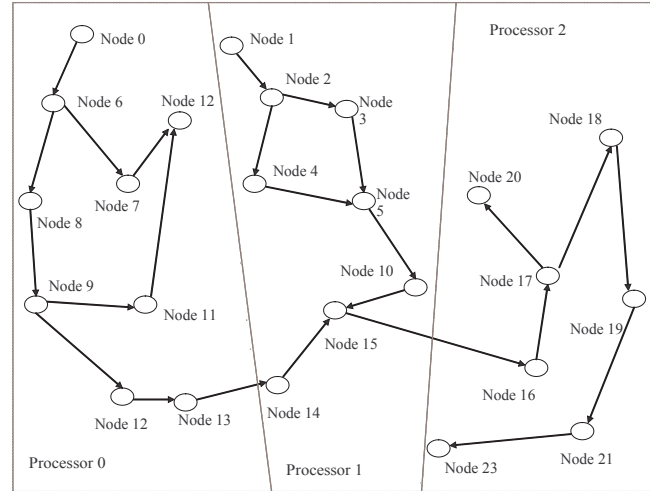


Figure 4. *LEPP*-graph synchronization

4. Implementation details

The presented implementation can be encapsulated and used by other Finite Element programs; thus, it provides a framework for the refinement process. Initially, distribution of the elements among processing nodes is accomplished. The mesh object is loaded locally and prepared for the refinement operation. After finishing the refinement process, master processor collects new vertices and tetrahedra. Another important feature is the profiler; that is, all methods are also capable of collecting elapsed time information in

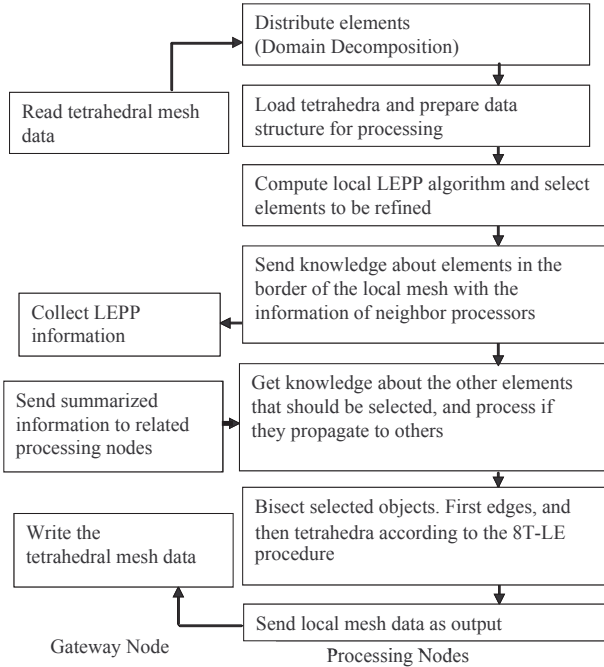


Figure 5. Parallel Algorithm

the network communication and computational code segments.

The parallel framework for tetrahedral mesh refinement requires *MPI*¹ and *PaRMetis*² libraries. Loaded mesh is partitioned in order to have minimum *edge-cuts* in the *LEPP*-Graph. Mesh partitioning is accomplished according to tetrahedra, and each processor keeps only the elements that are assigned. The mesh structure is partitioned fairly concerning the network cost between processing nodes. The overall mesh geometry fits into the memories of each processing nodes; thus, we can handle tetrahedral mesh files with excessive memory requirements. We can not deploy a very large mesh structure on a single node due to memory limitations, but we can distribute the data and operate in a parallel manner.

Tetrahedrons that should be refined initially are computed, and other elements which will be affected are selected using the *LEPP*. The selection procedure is accomplished by traversing the mesh data in the local processing node. A refined edge in one of the processors can trigger another tetrahedron which is in another processor's local memory. Therefore, related processing nodes are informed if an edge will be refined in the border of a partition.

Procedure of the parallel implementation can be stated as follows:

- Partition and distribute tetrahedral mesh elements.

- Compute initial tetrahedrons that should be refined.
- Prepare the local *LEPP*-Graph sequentially and select the edges that should be refined.
- Inform other processing nodes whether a border element in the local partition is selected.
- Refine according to the *8T-LE* procedure.
- Collect mesh data with recently produced tetrahedrons.

The gateway node is used not only to read the initial mesh data from file but also to prepare communication objects holding the information whether border elements of the local mesh partition require refinement or not. Therefore, the gateway node minimizes the number of network messages, and this situation is an important issue to enhance the performance of an *MPI* program. Each processing node sends the information about the selected border elements to be refined with the knowledge of neighbor processors that should take action. The gateway node collects the effected elements and informs other processors to start refinement process for the classified element.

4.1. Data structure

Designing a proper data structure is another challenge in the implementation. Some know techniques have been investigated [26, 4, 27, 3, 47, 17] in order to prepare a flexible architecture. A mesh structure is formed of vertices, edges, triangles and tetrahedrons. In order to process algorithms, we should be able to evaluate each element and keep relations between them. The *8T-LE* algorithm is a skeleton algorithm, and elements in lower dimensions are required for refinement. It is stated that the number of tetrahedrons in a conforming mesh is much more than the number of other elements; and any tetrahedron is adjacent to many edges and vertices [14]. Figure 6.a demonstrates the number of elements of a conforming mesh on average. Relations between elements are the related adjacencies; as an example, changing an edge will affect 5 tetrahedra on average [14].

Since the *8T-LE* is not principally parallelizable due to the sequential progress of *Longest-Edge* propagation, we developed a new data structure with a convenient parallel algorithm which is applicable to distributed environments. Each tetrahedron object keeps the list of edges it owns, and edge objects have the list of vertices that form itself. Edge objects also have the list of tetrahedrons which are adjacent, so that, while evaluating the *LEPP*, computation can be handled without searching adjacencies each time.

The data structure of the local mesh object has vertices, edges and tetrahedra. Each edge object has the list of tetrahedra it is owned by. The tetrahedron object keeps the list of edges that form this tetrahedron, and we can calculate the information of faces when required in the propagation path process.

Each vertex object has a single unique identifier which distinguishing them in the global space. Each processor

¹MPI: Message Passing Interface (www-unix.mcs.anl.gov/mpi).

²ParMetis: Parallel Graph Partitioning (www.cs.umn.edu/metis).

starts from a sequence which will not intersect with other processors. During the operation of the *LEPP* synchronization, a unique identifier which is the smallest number among all other local sequences, is selected as the identifier for the effected neighbor elements in the border of a local structure.

In *8T-LE* and *4T-LE*, we must select the longest-edge and that should be unique in the tetrahedron or in the triangle. The edge object has a simple methodology to handle the uniqueness for the length comparison. If the length of two edges are equal, identifiers of the first and then the second vertices are compared to select one of them as the longer edge.

Figure 6.b shows the used data structure skeleton. Figure 7 presents a more detailed view of the data model.

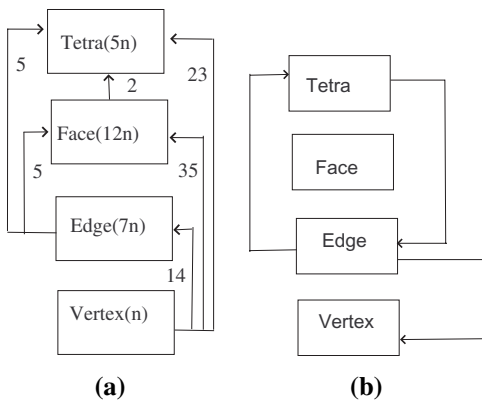


Figure 6. (a) Statistics for the number of entities of a mesh from Garimella [14] (b) Dependencies in data structure

Mesh structure is partitioned between each processing node according to cost of interaction between elements. Thus, while synchronizing the *LEPPs* between components, the parallel tetrahedron refinement process minimizes the network and computational costs. Such an implementation also covers the scalability objective. Moreover, the gateway node used in the algorithm enables us to minimize the number of network messages. The presented utility uses communication objects in the *MPI* environment in order to minimize the network traffic and handle the dynamic data transfer between processors. Each processing node uses the synchronization information to select and start the refinement from received elements if it is required. Synchronizing the borders of propagation graph, which is partitioned among processors as a result of the local mesh processing, is accomplished by summarizing the received bisection information and distributing among the concerned processors. Since the gateway node collects all messages used to synchronize the overall *LEPP*, it prevents duplicate messages in the communication environment.

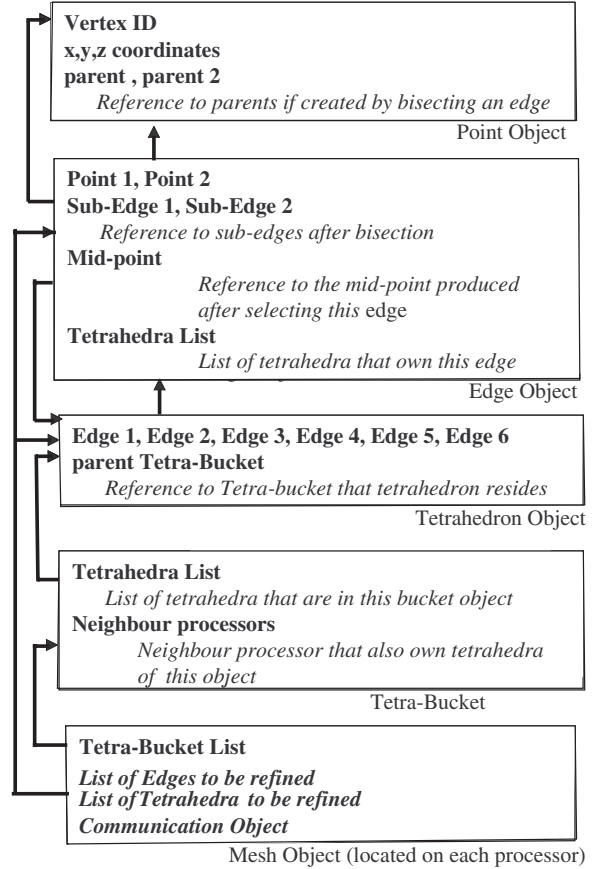


Figure 7. The Object Relationship

5. Test Results

Parallel implementation for distributed environment has been prepared and tested for homogenous platforms³. Utilized data-structure can also handle heterogeneous environments and is capable of adapting to data distribution. Since data is properly distributed, we can separate and reduce the overall computation and memory cost of the refinement process. Mesh input files from the *GAMMA* project⁴ were used as examples for testing the parallel refinement utility. We also used mesh generation tools to understand the accuracy of the methodology. Some of the mesh inputs were generated by *TetGen*⁵, which is a program for generating tetrahedral meshes for arbitrary 3-D domains based on *Delaunay* methods. The following figures present some of the generated mesh files which were produced by refining sample in-

³The cluster system with 512 nodes (SuperMike) from Center for Computation and Technology (cct.lsu.edu) at Louisiana State University has been used to test the application.

⁴GAMMA Project: The French National Institute for Research in Computer Science and Control (www-rocq1.inria.fr/gamma).

⁵Tetgen: A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator (tetgen.berlios.de).

put structures. Mesh structures were viewed by *Medit*⁶ and *Tetview*⁷ which are graphic programs for viewing tetrahedral meshes.

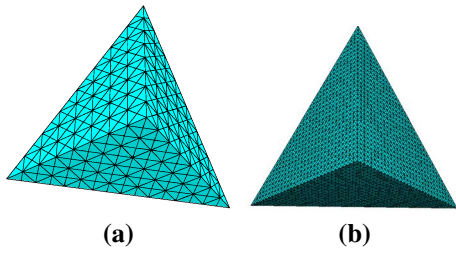


Figure 8. Examples of tetrahedral mesh

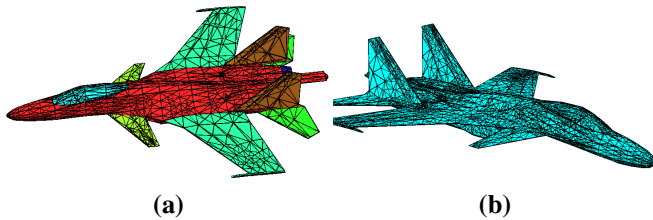


Figure 9. Examples of tetrahedral mesh

Refining the tetrahedra mesh in Figure 8.a, which has 1803 tetrahedra and 527 vertices, will produce a new structure with 13506 tetrahedra and 3387 vertices. After refining the resulting tetrahedral mesh we produced the 3-D formation shown in Figure 8.b that has 99121 tetrahedra and 23351 vertices. Refinement of mesh in Figure 9.a, 14904 tetrahedra, and 4502 vertices produced 70203 tetrahedra and 22568 vertices, as shown in Figure 9.b. Refinement of the resulting mesh second time resulted in 235941 tetrahedra and 22568 vertices. Other test results generated from the same source produced mesh structures with 450573 tetrahedra and 138842 vertices, and 770882 tetrahedra and 215017 vertices.

Figure 10 presents the maximum amount of time spent in a single processor among processing nodes during the refinement step. By defining the appropriate parameters, we can profile the software in order to optimize and test the scalability of the proposed algorithm. In future work, performance tests with larger mesh files will be done to optimize the code and to enhance the efficiency of the parallel utility.

6. Conclusions

A parallel mesh refinement algorithm for distributed environments is proposed, in which each processing node

⁶MEDIT: Mesh Visualization Tool (www.ann.jussieu.fr/free.htm).

⁷TetView: A Tetrahedral Mesh and Piecewise Linear Complex Viewer.

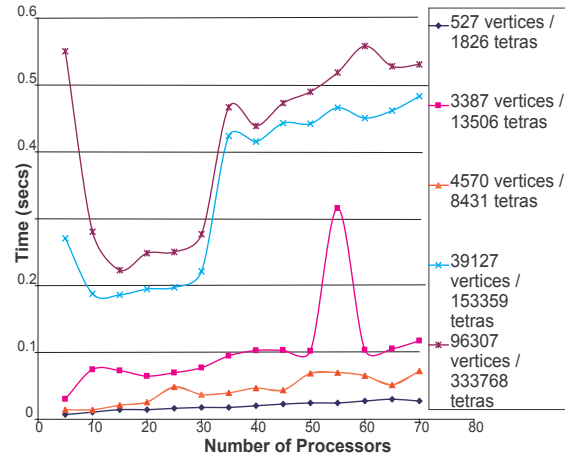


Figure 10. The refinement process

works over its local elements sequentially and synchronizes changes to update the overall mesh structure. We analyzed the *longest-edge* bisection algorithm and presented details about the parallel refinement process. 3-D structures have difficulties especially in processing the *propagation path* of selected tetrahedra. We presented a practical methodology for distributed environments which is capable of solving the refinement problem. Representation of the mesh is also crucial; the data structure must be compact not to consume so much memory, but it should be flexible and simple for computation. We explained the proposed objects to accomplish the construction of mesh topology. We also present a parallel utility that can distribute the mesh data and process in an inter-process communication environment; thus, clusters of ordinary nodes can be used to process very large mesh structures.

References

- [1] D. Arnold, A. Mukherjee, and L. Pouly. Locally Adapted Tetrahedral Meshes Using Bisection. *SISC 2000, SIAM*.
- [2] M. Bern, D. D. Eppstein, and S. Teng. Parallel Construction of Quadtrees and Quality Triangulations. In *WADS, Springer-Verlag*, 1993.
- [3] G. Berti. Generic Programming for Mesh Algorithms: Towards Universally Usable Geometric Components. In *WCCM V*, July 7-12, 2002.
- [4] G. Blandford, D.K. and Blesloch, D. Cardoze, and C. Kadow. Compact Representations of Simplicial Meshes in Two and Three Dimensions. In *12th IMR*, 2003.
- [5] C. Busch, M. Magdon-Ismael, and J. Xi. Optimal Oblivious Path Selection on the Mesh. In *IPDPS - IEEE*, 2004.
- [6] G. Carey and A. Plaza. About Local Refinement of Tetrahedral Grids Based on Bisection. In *5th International Meshing Roundtable (IMR)*, pages 123–136, 1996.
- [7] G. Carey and A. Plaza. Refinement of Simplicial Grids Based on the Skeleton. *Appl.Numer. Math.*, 32(2):195–218, 2000.

- [8] J. Castanos and J. Savage. Parallel Refinement of Unstructured Meshes. In *IASTED*, Nov. 3-6, 1999.
- [9] J. Chen and V. Taylor. Mesh Partitioning for Distributed Systems. In *HPDC '98, IEEE*.
- [10] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A Parallelization of Dijkstra's Shortest Path Algorithm. *Lecture Notes in Computer Science*, 1450, 1998.
- [11] F. K. H. A. Dehne, A. Ferreira, E. Caceres, S. W. Song, and A. Roncato. Efficient Parallel Graph Algorithms for Coarse-Grained Multicomputers and BSP. *Algorithmica*, 33, 2002.
- [12] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Shape-optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM. *Parallel Computing*, 26, 2000.
- [13] L. A. Freitag and C. Ollivier-Gooch. Tetrahedral Mesh Improvement Using Swapping and Smoothing. *NME*, Wiley, 40:3979-4002, 1997.
- [14] R. V. Garimella. Mesh Data Structure Selection for Mesh Generation and FEA Applications. *International Journal for Numerical Methods in Engineering(NME)*, Wiley, 2002.
- [15] J. Hallberg and J. S. A. Stagg. Adaptive Tetrahedral Grid Refinement and Coarsening in Message-Passing Environments. In *Grid Conf*, 2000.
- [16] O. Hammon and P. Krysl. Implementation of a General Mesh Refinement Technique. <http://hogwarts.ucsd.edu/~pkrysl>, 2003.
- [17] N. Hitschfeld. Algorithms and Data Structures for Handling a Fully Flexible Refinement Approach in Mesh Generation. In *4th International Meshing Roundtable (IMR)*, October 1995.
- [18] G. Iribarren and M. Rivara. The 4-Triangles Longest-Side Partition and Linear Refinement Algorithms. *Math.Comp*, 65:1485-1502, 1996.
- [19] M. T. Jones and P. E. Plassman. Parallel Algorithms for Adaptive Mesh Refinement. *SISC, SIAM*, 18(3), 1997.
- [20] T. Jones and P. Plassmann. Parallel Algorithms for the Adaptive Refinement and Partitioning of Unstructured Meshes. In *SHPCC, IEEE, Knoxville, TN*, pages 726-733, 1994.
- [21] P. N. Klein and S. Subramanian. A Randomized Parallel Algorithm for Single-Source Shortest Paths. *J. Algorithms*, 25(2):205-220, 1997.
- [22] C. E. Leiserson and B. M. Maggs. Communication-Efficient Parallel Algorithms for Distributed Random-Access Machines. *Algorithmica*, 3:53-77, 1988.
- [23] A. Liu and B. Joe. Quality Local Refinement of Tetrahedral Meshes based on 8-subtetrahedron Subdivision. *Math. of Computation*, 65:1183-1200, 1996.
- [24] J. Z. Lou, C. D. Norton, and T. A. Cwik. PYRAMID: Parallel Unstructured Adaptive Mesh Refinement. Jet Propulsion Laboratory, Caltech, 2005.
- [25] U. Meyer. Single-source Shortest-paths on Arbitrary Directed Graphs in Linear Average-case Time. In *Symposium on Discrete Algorithms*, 2001.
- [26] N. Molino, Z. Bao, and R. Fedkiw. A Virtual Node Algorithm for Changing Mesh Topology During Simulation. *SIGGRAPH, ACM TOG 23*, pages 385-392, 2004.
- [27] H. Nienhuys and A. Stappen. Maintaining Mesh Connectivity Using a Simplex-based Data Structure. Technical report, Utrecht University, UU-CS-2003-018, 2003.
- [28] E. Nuutila. *Efficient Transitive Closure Computation in Large Digraphs*. PhD thesis, Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering, 1995.
- [29] L. Oliker and R. Biswas. PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes. *Journal of Parallel and Distributed Computing*, 1998.
- [30] L. Oliker, R. Biswas, and H. N. Gabow. Parallel Tetrahedral Mesh Adaptation with Dynamic Load Balancing. *Parallel Comput., Elsevier*, 2000.
- [31] L. Oliker, R. Biswas, and H. N. Gabow. Parallel Tetrahedral Mesh Adaptation with Dynamic Load Balancing. *Parallel Computing, Elsevier Science*, 2000.
- [32] C. Ozturan. Worst Case Complexity of Parallel Triangular Mesh Refinement by Longest Edge Bisection. In *ICASE Report*, pages 96-56, 1996.
- [33] A. Plaza, M. Padron, and G. Carey. A 3D Refinement/Derefinement Algorithm for Solving Evolution Problems. *15th IMACS World Congress*, 32:401-418, 1997.
- [34] A. Plaza and M. Rivara. Mesh Refinement Based on 8-Tetrahedra Longest-Edge Partition. In *12th IMR*, 2003.
- [35] A. Plaza, J. Saurez, and M. Padron. Mesh Graph Structure for longest-edge Refinement Algorithms. In *7th IMR*, pages 335-344, October 1998.
- [36] M. Rivara. Mesh Refinement Based on the Generalized Bisection of Simplicies. *SIAM J. Numer. Anal.*, 1984.
- [37] M. Rivara and P. Inostrozo. A Discussion on Mixed(Longest Side Midpoint Insertion) Delaunay Techniques for the Triangulation Refinement Problem. *4th International Meshing Roundtable (IMR)*, 1995.
- [38] M. Rivara, D. Pizarro, and N. Herkovic, V. and Hitschfeld. Terminal-Edge Refinement Algorithm: A Study on a 3-Dimensional Implementation, 2003.
- [39] M. Rivara and M. Venere. Cost Analysis of the Longest-Side (Triangle Bisection) Refinement Algorithm for Triangulation. *Eng. Comp.*, pages 224-395, 1996.
- [40] M.-C. Rivara. Algorithms for Refining Triangular Grids Suitable for Adaptive and Multigrid Techniques. In *NME, Wiley*, 1984.
- [41] M.-C. Rivara. Selective Refinement/Derefinement Algorithms for Sequences of Nested Triangulations. In *NME, Wiley*, pages 2889-2906, 1989.
- [42] M.-C. Rivara. New Mathematical Tools and Techniques for the Refinement and/or Improvement of Unstructured Triangulations. In *5th IMR*, pages 77-86, 1996.
- [43] M.-C. Rivara, D. Pizarro, and N. Chrisochoides. Parallel Refinement of Tetrahedral Meshes using Terminal-Edge Bisection Algorithm. In *13th IMR, SAND 2004-3765C*, 2004.
- [44] J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-dimensional Mesh Generation. *Journal of Algorithms*, 18:548-585, 1995.
- [45] J. R. Shewchuk. Lecture Notes on Delaunay Mesh Generation. Technical report, EECS, Berkeley, 1999.
- [46] D. A. Spielman, S.-H. Teng, and A. Ungor. Parallel Delaunay Refinement: Algorithms and Analyses, <http://citeseer.ist.psu.edu/556537.html>.
- [47] J. Suarez, G. Carey, A. Plaza, and M. Padron. Graph Based Data Structures for Skeleton Based Refinement Algorithms. *University of Texas, Austin, TICAM Report 01-10*, 2001.
- [48] J. Surez, A. Plaza, and G. Carey. The Propagation Problem in Longest-Edge Refinement. *ICES Report*, 2003.
- [49] C. Walshaw and M. Cross. Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. *SIAM J. Sci. Comput.*, 22(1):63-80, 2000.