[3] E. Bänsch, *An adaptive finite-element strategy for the three-dimensional time-dependent navier-stokes equations*, J. of Computational and Applied Mathematics, 36 (1991), pp. 3–28.

[4] J. G. Castanos, *The dynamic adaptation of parallel mesh-based computation*, Master's thesis, Department of Computer Science Department, Brown University, May 1996.

[5] I. Fried, *Condition of finite element matrices generated from non-uniform meshes*, AIAA J., 10 (1972), pp. 219–221.

[6] J. Jaja, *An introduction to Parallel Algorithms*, Addison Wesley, 1992.

[7] M. T. Jones and P. E. Plassmann, *Parallel algorithms for adaptive mesh refinement*, Tech. Rep. MCS-P421-0494, Mathematics and Computer Science Division, Argonne National Laboratory, Illinois, 1994. (to appear in *SIAM J. on Scientific Computing*).

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

[9] A. Liu and B. Joe, *Quality local refinement of tetrahedral meshes based on bisection*, SIAM J. on Scientific Computing, 16 (1995), pp. 1269–1291.

[10] M.-C. Rivara, *Algorithms for refining triangular grids suitable for adaptive and multigrid techniques*, Int. J. Numer. Methods in Engineering, 20 (1984), pp. 745–756.

[11] ——, *Mesh refinement processes based on the generalized bisection of simplices*, SIAM J. Numer. Analysis, 21 (1984), pp. 604–613.

[12] M.-C. Rivara and C. Levin, *3d refinement algorithm suitable for adaptive multigrid techniques*, Comm. in Applied Numer. Methods, 8 (1992), pp. 281–290.

[13] W. J. Schroeder and M. S. Shephard, *A combined octree/Delaunay method for fully automatic 3d mesh generation*, Int. J. of Numer. Methods in Engineering, 29 (1990), pp. 37–55.

[14] M. S. Shephard, J. E. Flaherty, H. L. DeCougny, C. Özturan, C. L. Bottasso, and M. Beall, *Parallel automated adaptive procedures for unstructured meshes*, in Parallel Computing in CFD, AGARD, Neuilly-Sur-Seine, 1995.

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

[16] R. D. Williams, *A dynamic solution-adaptive unstructured parallel solver*, Tech. Rep. CCSF-21-92, Supercomputing Facility, California Institute of Technology, California, 1992.

u has been marked for refinement and $w(u_0) = 0$ if u has been not marked for refinement. In addition, we assign $w(u_1) = 0$. Let $pw(u_i)$, $i = 0, 1$, denote the prefix-type computation defined on this linked list, i.e. the summation of weights of nodes from the beginning of the list until and including the node u_i.

Table 1

node u_i	6_0	7_0	7_1	5_0	4_0	3_0	3_1	4_1	5_1	8_0	8_1	6_1
$w(u_i)$	0	0	0	0	0	1	0	0	0	0	0	0
$pw(u_i)$	1	1	1	1	1	1	0	0	0	0	0	0
$pw(u_0) - pw(u_1)$	1	0		1	1	1				0		

Table 1 shows the initial weight assignments and the values of $pw(u_i)$. If $pw(u_0) - pw(u_1) > 0$, then we know that the node u has been either initially marked for refinement or it should be marked because of propagation of refinement.

5 Discussion and Conclusion

In this paper, we have presented a logarithmic algorithm for adaptive mesh refinement by the longest edge bisection. The main advantage of the proposed algorithm is not its significantly lower worst case complexity, but rather its simplicity and its data parallel nature. The algorithm uses parallel prefix-type computation on linked list of mesh entities. Hence, it is a good candidate for implementation by a data parallel language like High Performance Fortran. The previous methods which have been proposed will be difficult to implement with a data parallel language.

We have assumed our input to be in the form of faces pointing to edges and edges pointing to vertices. Whereas mesh generators such as the one described in [13] generate this type of input by default, there are also mesh generators that output meshes by giving a set of vertices and a set of faces with each face pointing to three vertices. One possible conversion procedure for this type input is this: We can identify edges with the indices of its two vertices which make it up. We can then have each face generate these two-tuples (vertex-vertex pair in ascending order) for each of its edges. These two tuples can then be sorted to get the face-to-edge relationship.

In this paper, we have developed our algorithm for two-dimensional triangular meshes. One immediate question that pops up is this: Can we employ the same strategy for adaptive refinement of three-dimensional tetrahedral meshes ? Various methods have been proposed for tetrahedron refinement [3][9] including one based on longest edge bisection [12]. In two dimensions, an edge can be shared by at most two faces. In three dimensions, there can be arbitrary number of faces which can share an edge. This introduces some complications. These problems will be the target of future investigations.

References

[1] I. Babuska and K. Aziz, *On the angle condition in the finite element method*, SIAM J. Numer. Analysis, 13 (1976), pp. 214–226.

[2] R. Bank, A. Sherman, and H. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, in Scientific Computing, R. S. et al., ed., Amsterdam, 1983, IMACS/North Holland Publishing Company, pp. 3–17.

part of any face which have been marked for refinement. Therefore, they all have weight 0 and are drawn with a white circle to indicate this weight. Additionally, Figure 4(a) shows the forest of directed trees that represent the dependency of refinement propagation for the mesh. As is illustrated in Figure 4(b), the refinement of edge 3 will propagate to edges 4, 5 and 6. Once these four edges are marked for refinement, application of the refinement templates given in Figure 2 will lead to the final refined and conforming mesh given in Figure 4(c).

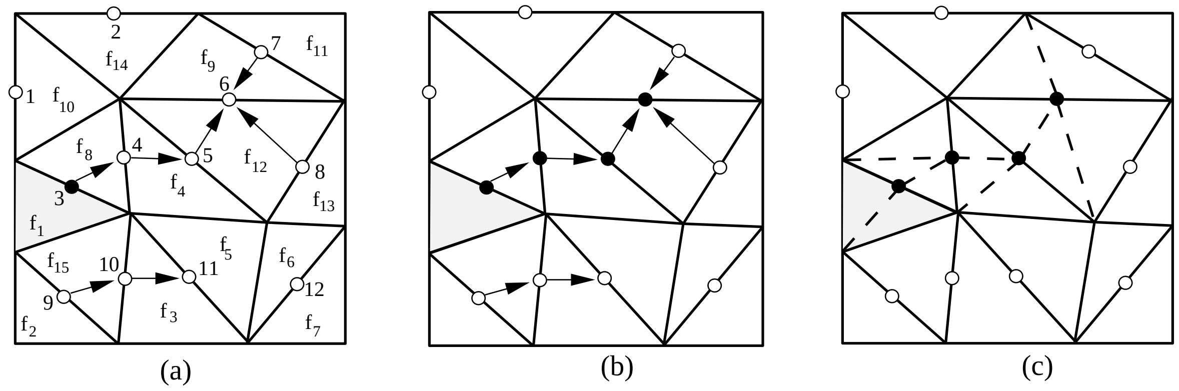

FIG. 5. *An example showing the construction of directed trees (a), the propagation of refinement (b) and the final refined mesh (c)*

Given the example mesh and the corresponding forest of directed trees, we illustrate the Euler-Tour representation of one of the trees (the one containing the nodes 3, 4, 5 and 6) in Figure 6(a). Euler Tour representation replaces each edge of the tree with two directed edges. In distributed memory implementations, this may lead to some complications due to variable number of links pointing to or from a tree node. A more practical preorder tree traversal representation involve replacement of each tree node, $u \in V_r$, with two nodes, u_0 and u_1. This representation has been used by [15] and recently by [8]. Figure 6(b) illustrates the example tree in this representation.

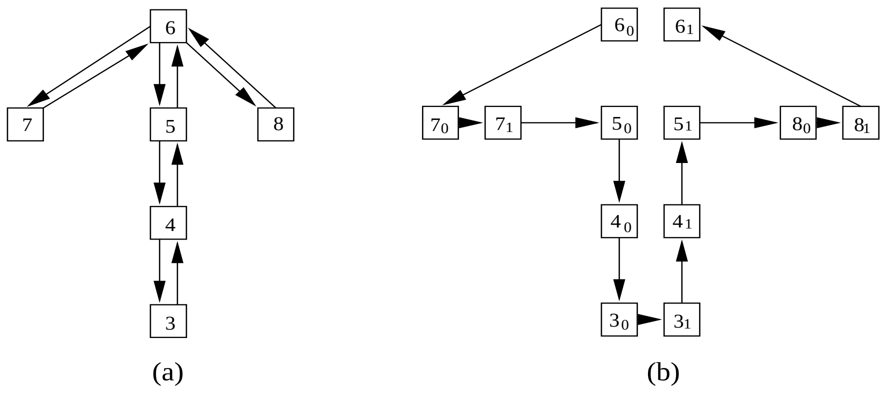

FIG. 6. *Linearized representation for one of the trees in the example: Euler Tour representation (a) and another more practical representation (b)*

Given the representation in Figure 6(b), we assign the following weights; $w(u_0) = 1$ if

```
Algorithm ParallelRefine
begin
   1. Construct forest of directed trees, $G_r$.
   2. Construct Euler Tour of each tree in $G_r$.
   3. Assign weights to each edge of $G_r$.
   4. Perform prefix summation of the weights on the edges from the beginning of each
      linked list by pointer jumping.
   5. Compute the number of marked descendants of each node.
   6. Refine an edge corresponding to a tree node if it is initially marked or it
      has positive number of marked descendants.
   7. Apply the appropriate refinement template to each triangle.
end
```

FIG. 4. *Steps of parallel refinement algorithm*

THEOREM 3.2. *Algorithm* `ParallelRefine` *has logarithmic worst case complexity.*

Proof. Step 1 of the algorithm takes $O(1)$ time since it involves assigning a processor to each triangle and having it compare the lengths of its edges. The degree of each node in graph G_r is at most 4. Therefore, step 2 can be done in $O(1)$ as follows: The directed edges in the graph G_r act as backward edges, $< v, parent(v) >$, in the Euler Tour. The backward edges and the forward edges, $< parent(v), v >$, can be assigned to one of the four locations associated with a node using exclusive writes. To do this, note that we can order the vertices of each triangle in such a way that we have a consistent (for example, clockwise) traversal. If we let (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the vertices of a triangle, the ordering can be done by arranging vertices in triangles such that the determinant formula, $(x_2y_3 - x_3y_2) - (x_1y_3 - x_3y_1) + (x_1y_2 - x_2y_1)$ has a consistent sign. Given an edge $e_i = (u_{i_0}, u_{i_1})$, we can have at most two triangles sharing this edge. From the consistent ordering, we can then state that the difference in the indices, will be positive for one triangle (e.g. $i_0 - i_1 > 0$) and negative for the other triangle (e.g. $i_1 - i_0 < 0$). Furthermore, we can classify edges within a single triangle according to the order they appear (taking the longest edge as a frame of reference). In this way, we can get a unique location in the range $0, \ldots, 3$ for each edge and hence make the assignment by exclusive writes. Weight assignment in step 3 takes $O(1)$. The prefix summation by pointer jumping at step 4 takes $O(\log n)$ time. Step 5 takes $O(1)$ time to compute the differences in prefix summation. Finally steps 6 and 7 take $O(1)$ time since it involves each processor assigned to a triangle check locally for the appropriate template to use and to apply the refinement using this template. The overall parallel refinement algorithm has logarithmic worst case complexity. $\square$

The next section gives an example mesh and shows examples of linearized tree constructions on this mesh.

4 Example and Tree Representations

We illustrate the steps of our parallel refinement algorithm on the example mesh given in Figure 4. Figure 4(a) shows the original mesh with labels on the edges which act as the longest edge of at least one triangle. It also shows the shaded face f_1 as being initially marked for refinement. Since edge 3 is the longest edge of face f_1, it has a weight of 1 and is drawn with a dark circle to indicate this weight. The rest of the labelled edges are not

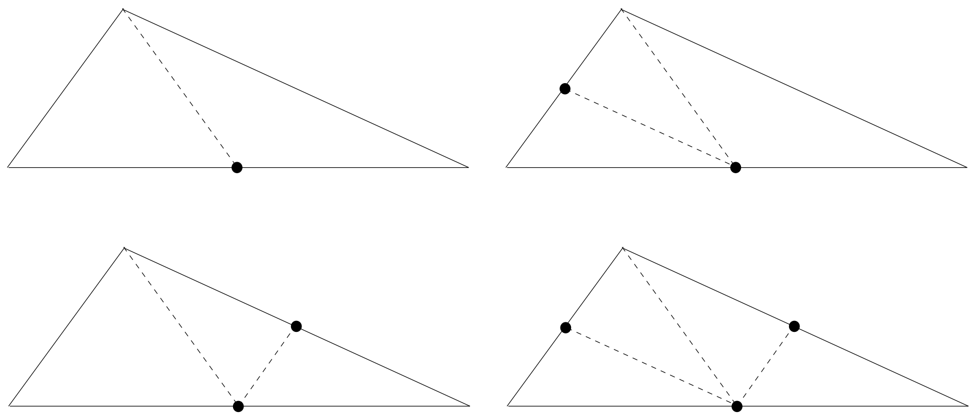

FIG. 3. *Rivara refinement templates*

vertex to the other if the refinement of the former induces the refinement of the latter. The theorem proves an important property of the graph constructed in this way.

THEOREM 3.1. *Let $G_r(V_r, E_r)$ be a directed graph constructed from a triangular mesh $T(V, E, F)$ with the vertex set,*

$$V_r = \{e_i : e_i \in E \text{ and } e_i \text{ is the longest edge of at least one triangle in } F\},$$

and the directed edge set,

$$E_r = \{< v_i, v_j > : v_i, v_j \in V_r \text{ and } length(v_i) < length(v_j)\};$$

then the graph G_r forms a forest of directed trees.

Proof. There are two observations. The lengths of mesh edges increase strictly as we follow a directed path. Therefore, there can be no cyles in G_r and hence G_r forms a directed acyclic graph (DAG). Secondly, a vertex $v_i \in V_r$ can have at most one outward directed edge. The only way a vertex could have more than one outward directed edge is if it was *not* the longest edge of any triangle ; but these types of edges are not included in the definition of V_r. Therefore, we have a DAG, with vertices having at most one outward edge. This type of graph forms a forest of directed trees. $\square$

Our algorithm which parallelizes the refinement process basically constructs Euler Tours of directed trees and does a computation similar to computing the number of descendants of a tree node by the pointer jumping technique [6, p. 118]. We assign a weight of 1 to each edge $< v, parent(v) >$, $v \in V_r$, in the Euler Tour if node v has been initially marked for refinement. All the other edges in the Euler Tour get a weight of 0. We compute prefix sums of these weights. Treating the root of the tree as a special case, we take the difference between the prefix sums of $< parent(v), v >$ and $< v, parent(v) >$ to get the number of marked descendants. If a node has positive number of marked descendants or the node itself has been initially marked for refinement, that node (i.e. the corresponding *mesh* edge) will be refined.

If the number of triangles is n, then the numbers of edges and nodes are also $O(n)$. Let us assume a concurrent read, exclusive read model of PRAM. We will use $O(n)$ number of processors. The following theorem establishes the complexity of the algorithm assuming this model.

2 Problem Specification

Let $T(V, E, F)$ denote a two-dimensional triangular mesh with V representing the set of vertices,

$$V = \{(x_i, y_i) : x_i, y_i \in \Re\},$$

E, the set of edges,

$$E = \{(v_i, v_j) : v_i, v_j \in V\},$$

and F, the set of triangular faces,

$$F = \{(e_i, e_j, e_k) : e_i, e_j, e_k \in E\}.$$

We define the $length(e_i)$ of an edge, $e_i \in E$, as its euclidean length and use the index i to break the ties when multiple edges have same the euclidean length.

In this paper, we assume that the input data supplied by the user specifies V, E and F. Some mesh generators may output the set of faces and express each face as a 3-tuple of vertices. In the discussion and conclusion section, we suggest a possible way of converting such a data set to the one assumed in this paper.

Rivara presents two algorithms for refining the mesh by longest edge bisection. These are illustrated in Figure 2. In the first algorithm, a triangle marked for refinement (either initially or as a result of satisfying the conformity requirement) is always divided into two by bisecting it by the longest edge. In the second algorithm, the triangle is first bisected by the longest edge into two and if the non-conformity still persists, one or both of the two refined triangles are further subdivided to maintain conformity.

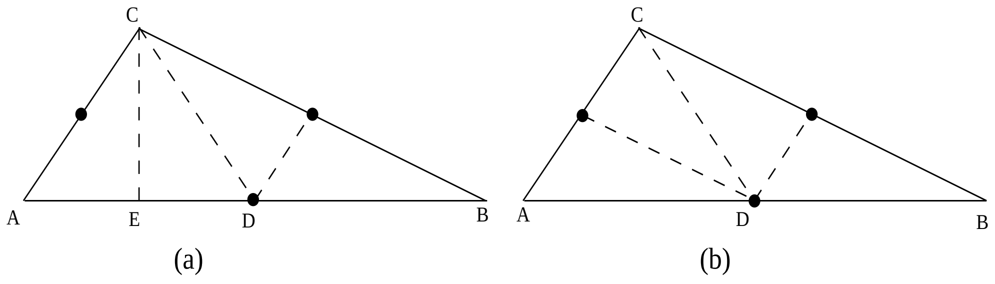

FIG. 2. *Illustration of Rivara's two algorithms to refine by longest edge bisection*

Rivara's second algorithm is simpler and more practical. It also has the same angle bound as the first one. Therefore, it is the choice of several implementations including that of Jones and Plassman. In this paper, the parallelization technique we present in the next section applies to this second algorithm. Given a triangle with one, two, or all of the edges marked for refinement, Figure 3 shows the four possible templates that are used to refine the triangle in the second algorithm.

3 Algorithm Description

Unlike the previous algorithms which apply the refinement templates and propagate refinement simultaneously, our approach first propagates the edge-markings to satisfy conformity and then applies the templates at the end. Application of the templates can be parallelized easily once appropriate edges have been marked for refinement. The issue that remains is how to parallelize the propagation step in an efficient way.

The following theorem constructs the directed data dependency graph of refinement propagation by treating each mesh edge as a vertex and drawing a directed edge from one

1 Introduction

Recently, adaptive mesh refinement (AMR) techniques for the solution of partial differential equations have gained importance due to their ability to concentrate computational as well as storage resources to regions where they are most needed, i.e. regions where the error in the computed solution is not within a prescribed tolerance. AMR methods are driven by automatic estimation and control of discretization errors and therefore need procedures to refine regions with high errors. Since the mesh needs to be refined selectively, specialized data structures and algorithms must be devised to refine and maintain the nonuniform mesh.

In this paper, we address the problem of parallelizing adaptive refinement of two-dimensional triangular meshes. A variety of techniques have been proposed to refine such meshes sequentially [2][3][10][11]. The following three properties are highly desirable in a refined mesh: (*i*) the resultant mesh should be *conforming*, i.e., the intersection of any two triangles should be either a single vertex, or an edge joining two vertices or the empty set, (*ii*) the mesh gradation should be smooth, i.e., the areas of neighboring triangles should not differ drastically, and finally (*iii*) the angles in the mesh should be neither too small nor too large [1][5].

Rivara's refinement algorithm [10] which is based on bisecting the triangles by their longest edge satisfies properties (*i*) and (*ii*). In relation to property (*iii*), it is proved that the smallest angle in the (succesively) refined mesh is bounded by at worst one-half the smallest angle in the original mesh.

Parallelization of mesh refinement procedures based on longest edge bisection have been considered in [4][7][14][16]. The refinement procedures implemented by Jones and Plassmann [7] use randomized graph coloring heuristics to resolve data structure update conflicts. The data structures used in [4] and [14] do *not* necessitate the use of any graph coloring heuristic to implement parallel mesh refinement. All these implementations, however, have a worst case linear complexity due to possible propagation of refinement to all other triangles in the mesh. A worst case example has been given by Jones and Plassmann [7] and is shown in Figure 1.

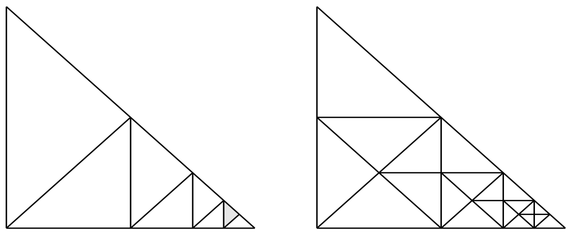

FIG. 1. *Example given by Jones and Plassmann showing linear worst case propagation of refinement.*

In this paper, we present a logarithmic data parallel algorithm to refine triangular meshes by Rivara's longest edge bisection procedure. We first present the problem statement and then describe the algorithm. We also present an example illustrating the steps of our algorithm and suggest data structures for possible practical implementation.

Worst Case Complexity of Parallel Triangular Mesh Refinement by Longest Edge Bisection

Can Özturan*
Assistant Professor
Computer Engineering Department
Bogazici University
Istanbul, Turkey

Abstract

We present a logarithmic algorithm for performing parallel refinement of triangular meshes by the widely used longest edge bisection procedure. We show that the refinement propagation forms a data dependency which can be expressed as a forest of directed trees. We solve a parallel Euler Tour problem on the trees to propagate the refinement. After propagation, we apply refinement templates. Our algorithm improves earlier reported results which had linear worst case complexity.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-0001.