Parallel Decoupled Terminal-Edge Bisection Method For 3D Mesh Generation

Maria-Cecilia Rivara* Carlo Calderon*
Universidad de Chile,
Chile {mcrivara, carcalde}@dcc.uchile.cl

Andriy Fedorov† and Nikos Chrisochoides‡
College of William and Mary,
Williamsburg VA, U.S.A. {fedorov, nikos}@cs.wm.edu

October 28, 2005

Abstract

We present a practical and stable algorithm for the parallel refinement of tetrahedral meshes. The algorithm is based on the refinement of terminal-edges and associated terminal stars. A terminal-edge is a special edge in the mesh which is the longest edge of every element that shares such an edge, while the elements that share a terminal-edge form a terminal star. We prove that the algorithm is inherently decoupled and thus scalable. Our experimental data show that we have a stable implementation able to deal with hundreds of millions of tetraedra and whose speed is in between one and two order of magnitude higher from the method and implementation we presented in [33].

Keywords: parallel mesh generation, 3-dimesnional (3D), longest-edge, terminal-edge, Lepp.

Introduction

Parallel mesh generation methods should satisfy the following four practical criteria: (1) *stability* in order to guarantee termination and good quality elements for parallel finite element methods, (2) *simple domain decomposition* in order to reduce unnecessary pre-processing overheads, (3) *code re-use* in order to benefit from fully functional, highly optimized,

* *This work was partially supported by Fondecyt 1040713*
† *This author's work is supported in part by ITR #ACI-0085969, and NGS #ANI-0203974*
‡ *This author's work is supported in part by NSF Career Award #CCR-0049086, ITR #ACI-0085969, NGS #ANI-0203974, and ITR #CNS-0312980*

and fine tuned sequential codes, and (4) *scalability*. Our parallel mesh generation algorithm satisfies all four requirements.

Parallel mesh generation procedures in general overdecompose the original mesh generation problem into N smaller subproblems which are meshed concurrently using P ($\ll N$) processors [9]. The subproblems can be formulated to be either tightly [23, 3] or partially coupled [19, 10, 4] or even decoupled [2, 34, 18]. The coupling of the subproblems determines the intensity of the communication and the degree of dependency (or synchronization) between the subproblems.

The parallel mesh generation and refinement method we present in this paper is *a decoupled* method (i.e., requires zero communication and synchronization between the subproblems). This is an improved version of the method we presented in [33] which again required zero communication between the subproblems, but it used a central processor for synchronization in order to maintain the conformity of the distributed mesh. In this paper we eliminate the synchronization i.e., there is no need for a central processor and we prove the correctness of our algorithm.

There are two classes of parallel tetrahedral mesh generation methods: (1) Delaunay and (2) non-Delaunay. There are few parallel implementations [11, 26, 23] for 3D Delaunay mesh generation due to the inherent complexity of the Delaunay algorithms. Moreover, there are (to the best of our knowledge) only two parallel stable Delaunay mesh generation algorithms and software [11, 23]. The parallel Delaunay mesh generator in [11] is for general 3D geometries. It starts by sequentially meshing the external surfaces of the geometry and pre-computes domain separators whose facets are Delaunay-admissible (i.e., the precomputed interface faces of the separators will appear in the final Delaunay mesh). The separators decompose the continuous domain into subdomains which are meshed in parallel using a sequential Delaunay mesh generation method on each subdomain.

The algorithm in [23] works only for polyhedral geometries. It maintains the stability of the mesher by simultaneously partitioning and refining the interface surfaces and volume of the subdomains [7] —a refinement due to a point insertion might extend across subproblem (or subdomain) boundaries (or interfaces). The extension of a cavity beyond subdomain interfaces is a source of irregular and intensive communication with variable and unpredictable patterns. Although the method in [23] can tolerate up to 90% of the communication—by concurrently refining other regions of the subdomain while it waits for remote data to arrive—its scalability is of the order of $O(\log P)$, where P is the number of processors. Unfortunately, the concurrent refinement can lead to a non-conforming and/or non-Delaunay mesh [23]. These problems are solved at the cost of setbacks which require algorithm/code re-structuring [8, 3] or at the cost of mesh re-triangulation [36].

On the other hand, longest-edge bisection algorithms, introduced by Rivara [27, 28, 31] are much simpler and easier to implement on both sequential [20, 22, 21] and parallel [38, 16, 25] platforms. The algorithms are based on the bisection of triangles/tetrahedra by its longest-edge as follows: in two-dimensions this is performed by adding an edge defined by the longest-edge midpoint and its opposite vertex, while in three di-

mensions the tetrahedron is bisected by adding a triangle defined by the longest-edge midpoint and its two opposite vertices.

Jones and Plassman in [16] have proposed a 2-dimensional, 4-triangles parallel algorithm for the refinement / derefinement of triangulations. In order to avoid synchronization, the algorithm uses a Monte Carlo rule to determine a sequence of independent sets of triangles which are refined in parallel. In order to minimize communication costs, a mesh partitioning algorithm based on an imbalanced recursive bisection strategy is also used. Castaños and Savage in [25] have parallelized the non-conforming longest edge bisection algorithm both in 2 and 3 dimensions. In this case the refinement propagation implies the creation of sequences of non-conforming edges that can cross several submeshes involving several processors. This also means the creation of non-conforming interface edges which is particularly complex to deal with in 3-dimensions. To perform this task each processor P_i iterates between a no-communication phase (where refinement propagation between processors is delayed) and an interprocessor communication phase. Different processors can be in different phases during the refinement process, their termination is coordinated by a central processor P_0. Duplicated vertices can be created at the non conforming interface edges. A remote cross reference of newly created interface vertices during the interprocessor communication phase along with the concept of *nested elements* [24] guarantees the assignment of the same logical name for these vertices. The load balancing problem is addressed by using mesh repartitioning based on an incremental partitioning heuristic.

The method we present here, contrary to the algorithm in [25], completely avoids the management of non-conforming edges both in the interior of the submeshes and in the inter-subdomain interface. This property completely eliminates the communication and synchronization between the subdomains both for global mesh refinement and for refinement guided by an edge-size density function.

Background

In 2-dimensions the longest-edge bisection algorithm essentially guarantees the construction of refined, nested and unstructured conforming triangulations (where the intersection of pairs of neighbor triangles is either a common vertex, or a common edge) of analogous quality as the input triangulation. More specifically the repetitive use of the algorithms produce triangulations such that: (a) The smallest angle α_t of any triangle t obtained throughout this process, satisfies that $\alpha_t \geq \alpha_0/2$, where α_0 is the smallest angle of the initial triangulation. (b) A finite number of similarly distinct triangles is generated in the process. (c) For any conforming triangulation the percentage of bad quality triangles diminishes as the refinement proceeds. Even when analogous properties have not been fully proved in 3-dimensions yet, both empirical evidence [31, 21] and mathematical results on the finite number of similar tetrahedra generated over a set of tetrahedra [12] allow to conjecture that a lower bound onthe tetrahedra quality can be also stated in the 3-dimensional setting.

The serial pure longest-edge bisection algorithm [27], works as follows:

for any target triangle t to be refined both the longest-edge bisection of t and the longest-edge bisection of some longest-edge neighbors are performed in order to produce a conforming triangulation. This task usually involves the management of sequences of intermediate non conforming points throughout the process.

Alternative longest-edge based algorithms (i.e., the 4-triangles longest-edge algorithm), which use a fixed number of partition patterns have been also proposed in [27, 20]. The 4-triangles algorithms maintain only one non-conforming vertex as the refinement propagates toward larger triangles; however its generalization to 3-dimensions is rather cumbersome.

More recently, the use of two new and related mathematical concepts - the longest-edge propagation path (Lepp) of a triangle t and its associated terminal edge, have allowed the development of improved Lepp based algorithms for the longest edge refinement / derefinement of triangulations both in 2- and 3-dimensions [29, 30, 32] which completely avoids the management of non conforming meshes. Moreover, the application of Lepp / terminal edge concepts to the Delaunay context have also allowed the development of algorithms for the quality triangulation of PSLG geometries [30], for the improvement of obtuse triangulations [13, 14], and for approximate quality triangulation [35].

Either for improving or refining a mesh, the Lepp based algorithms use a terminal-edge point selection criterion as follows. For any target element to be improved or refined, a Lepp searching method is used for finding the midpoint of an associated terminal-edge which is selected for point insertion. Each terminal-edge is a special edge in the mesh which is the common longest edge of all the elements (triangles or tetrahedra) that share this edge in the mesh. Once the point is selected, this is inserted in the mesh. In the case of the terminal-edge refinement algorithm, this is done by longest-edge bisection of all the elements that share the terminal-edge, which is a very local operation that simplifies both the algorithm implementation and its parallelization. The process is repeated until the target element is destroyed in the mesh.

In 2-dimensions a Lepp based algorithm for the quality refinement of any triangulation was introduced in [29], where the refinement of a target triangle t_0 essentially means the repetitive longest-edge bisection of pairs of terminal triangles sharing the terminal-edge associated with the current Lepp(t_0), until the triangle t_0 itself is refined. Lepp (t_0) is defined as the longest edge propagation path associated to t_0 and corresponds to the sequence of increasing neighbor longest edge triangles that finishes when a terminal-edge is found in the mesh. For an illustration see Figure 1, where Lepp(t_0)=$\{t_0, t_1, t_2, t_3\}$ over the triangulation (a), and the associated terminal edge is the edge shared by triangles t_2, t_3. Triangulations (b) and (c) respectively illustrate the first and second refinement steps, while triangulation (d) corresponds to the final mesh when the Lepp Bisection procedure is applied to t_0. Note that the new vertices were enumerated in the order they were created. The generalization of this algorithm to 3-dimensions is formulated in the next section.

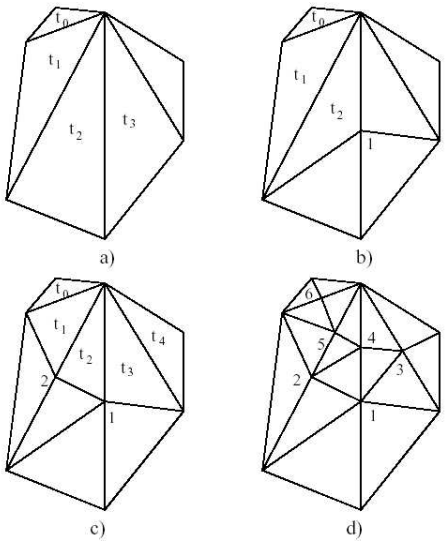

Figure 1: Lepp refinement of target triangle t_0 (a) initial mesh; (b) first refinement step; (c) second refinement step; (d) final mesh where triangle t_0 was refined.

Serial 3D Lepp and Terminal-edge Algorithms

As discussed in [29, 32], the 3-dimensional algorithm implies a multi-directional Lepp searching task, involving a set of terminal-edges. In this case each terminal-edge in the mesh is the common longest-edge of every tetrahedron that shares such an edge; and the refinement operation involves a terminal-star (the set of tetrahedra that share a terminal-edge) refinement. It is worth noting that the refinement is confined in the interior of the terminal-star.

Definition 1. Any edge E in a valid tetrahedral mesh M is a terminal-edge if E is the longest edge of every tetrahedron that shares E. In addition the set of tetrahedra that share E define a terminal-star in 3-dimensions, while that every tetrahedron in a terminal star is called a terminal tetrahedron.

Definition 2. For any tetrahedron t_0 in M, the Lepp (t_0) is a 3-dimensional submesh (a set of contiguous tetrahedra) recursively defined as follows:

(a) Lepp (t_0) contains every tethatedron t that shares the longest edge of t_0 with t, and such that longest edge (t) $>$ longest edge (t_0).

(b) For any tetrahedron t' in Lepp(t_0), the submesh Lepp(t_0) also contains every tetrahedron t not contained yet in Lepp (t_0), such that t shares the longest edge of t' and where longest edge (t) $>$ longest edge (t').

Proposition 1. *In 3-dimensions, for any tetrahedron t in M, Lepp (t) has a finite, variable number of associated terminal-edges.*

Proof The proof follows from the fact that every tetrahedron t in any Lepp (t_0) has a finite, non fixed number of neighbor tetrahedra sharing the longest edge of t. So in the general case more than one of these tetrahedra has longest edge greater than the longest edge of t, which implies that the searching task involved in Definition 4 is multidirectional, and stops when a finite number of terminal edges (which are local longest edges in the mesh), are found in M.

Note that, the Lepp(t_0) corresponds to a submesh of an associated Lepp polyhedron, which captures the local point distribution around t_0 in the direction of the longest edge.

Proposition 2. *Every terminal-edge l associated to any Lepp(t_0) is the longest edge between all the edges involved in the chain of tetrahedra traversed to reach l in the Lepp searching path.*

Proof. The proof relies in the Lepp definition which involves finding a set of increasing longest edge tetrahedra until a terminal edge is found. So, every terminal edge is a last longest edge in a sequence of increasing longest edge tetrahedra.

A high level 3-dimensional refinement algorithm for the local refinement of a tetrahedral mesh follows:

3D Lepp Bisection Refinement Algorithm
Input { a mesh M and set S of tetrahedra to be refined in M}
for each t in S **do**
  while t remains in M **do**
    Find Lepp (t) and associated set of terminal edges (TE)
    Refine each terminal star associated to every terminal edge in TE
  end while
end for
Output {mesh M}

In this paper we focus on the parallelization of a global terminal edge refinement algorithm which makes implicit use of the Lepp concept. This serial algorithm performs the repetitive refinement of every terminal edge greater than a given tolerance on the size of the terminal edges as follows:

Terminal-edge Refinement Algorithm
Input {a mesh M and a tolerance parameter b(M)}
Perform successive refinement of the terminal stars associated to terminal edges greater than b(M) in M until no terminal edge greater than b(M) remains in the mesh.
Output {refined M}

The following theorem assures that the final mesh has every edge less than b(M):

Theorem 1. *The use of the Terminal-edge Refinement Algorithm with tolerance parameter b(M) produces a refined mesh M_F with every edge in M_F less than or equal to b(M).*

Proof The existence of a (non-terminal) edge E in M_F with length(E) $>$ b(M) would imply the existence of at least one tetrahedron t with longest edge greater than b(M), which in turn implies the existence of Lepp(t) with at least one terminal-edge greater than b(M), which contradicts the definition of M_F.

Parallel Terminal-Edge Bisection Algorithm

In this section we consider a simple Parallel Terminal-Edge (PTE) bisection algorithm which globally refines any tetrahedral mesh as follows: for each submesh, the parallel refinement of terminal-edges is performed until their size is less than or equal to a global user defined tolerance *b(M)*. A high level description of the method follows:

Parallel Terminal Edge (PTE) Refinement Algorithm

1. *Read Input* {a mesh M and edge tolerance b(M)},
2. *Partition the mesh M* in submeshes $M_i, i = 1...N$,
3. Distribute the submeshes M_i among the processors,
4. *Perform submesh PTE refinement over each M_i.*
5. Output

Each submesh $M_i(S_i, V_i)$ is defined by its interface surface S_i and corresponding set of tetrahedra V_i. The distribution of the submeshes M_i to processors takes place by traditional ab-initio data mapping methods [6]. Each submesh assigns a unique identifier (ID) to each new vertex created in the submesh M_i. Based on the expected size of the mesh, a 32 or 64 bit word is used to store each ID. The mesh refinement task, based on edge refinement in each submesh M_i is performed as follows:

Submesh PTE Refinement Algorithm

while there is a terminal edge $>$ b(M) in M_i **do**
  (Step1) Perform repetitive refinement of every interior terminal edge greater than b(M) in the interior V_i of M_i
  (Step2) Perform repetitive refinement of every interface terminal edge greater than b(M) in S_i of M_i
end while

It is worth noting that step 2 can be accomplished without any communication since for each interface terminal-edge E greater than b(M), every processor that shares E knows that E must be bisected (in its associated submesh).

At termination a new finer mesh $M'(S', V')$ is constructed by simply computing the union of S_i and V_i, $i = 1...N$. According to Theorem 1, the resulting mesh will be a conforming mesh with every edge less than b(M). The interface vertices, and edges are replicated among the subdomains that share them. Consequentely a simple post-processing step allows to compute a global vertex ID if this is required, as well as adjacency information for the finite element application.

Theoretical Framework

Even when there not exists yet a theoretical bound on the geometrical quality of the tetrahedra by longest edge bisection in 3-dimensions, two comments are in order: (a) The 3-dimensional algorithm behaves in practice as the 2-dimensional algorithm does, in the sense that, at the first global refinement steps, the mesh quality can show some quality decreasement (by approximately 1/3) for a small percentage of elements, while the mesh quality distribution quickly stabilizes according to a Gaussian distribution. After this point the refinement algorithm tends to improve the mesh distribution. (b) In [12] it has been proved that the symmetric longest edge bisection of a regular tetrahedron produces a finite number of similary different tetrahedra, the fist step to state a theoretical bound on the mesh quality.

By assuming the conjecture that the algorithm do not significally deteriorathe the elements, which holds in practice, termination results can be stated. The termination of the PTE algorithm is based on the Proposition 2 and the Lemmas bellow while. Theorem 2 proves that the PTE algorithm is decoupled.

Lemma 1. *Let E be any interior terminal edge in M_i with length(E) > b(M) and for which there exists at least one tetrahedron t in M_i such that E belongs to Lepp(t) and b(M) < longest edge (t) < length(E). Then the processing of E in the step 1 of PTE algorithm implies the successive processing of a sequence of new terminal edges in the submesh M_i including the longest edge of t which also becomes a terminal edge in the mesh. Furthermore, both the refinement of these terminal edges and their associated terminal stars is performed in the same step 1.*

Proof The existence of t implies that there exists a sequence of interior edges in Lepp(t) which need to be traversed in order to reach the associated terminal edge E in M_i. Thus in the same step 1 of PTE algorithm, and in decreasing edge size order, each one of these edges becomes a terminal edge in M_i greater than $b(M)$ which is refined in the same step 1.

Lemma 2. *Consider a set S_B of interface terminal edges to be processed in step 2 of the algorithm in submesh M_i. If M_i has interior edges greater than b(M), then the refinement of the terminal-edges of S_B (in step 2) introduces a set S_A of interior terminal edges greater than (b(M) in M_i. Furthermore, the processing of each E in S_A in the next step 1 introduces a sequence of interior terminal edges greater than b(M) which are also processed in the same step 1.*

Proof: The refinement of each interface terminal edge $E_{interface}$ introduces a set S_{tet} of tetrahedra sharing a bisected edge. For each t in S_{tet} with interior longest edge E greater than b(M) two cases arise: (a) If E is an interior terminal edge in S_A, then Lemma 3.1 applies and the processing of E in step 1 can introduce a sequence of interior terminal edges which are processed in the same step 1; (b) If E is not a terminal edge then E becomes a terminal edge by processing another terminal edge in S_A in the same step 1. To prove this consider that submesh M_i was only modified by refinement of the terminal edge $E_{interface}$ which produced

the set S_{ted} and tetrahedron t with interior longest edge E. By definition of step 1, Lepp (t) can only finish in an interior terminal edge due to the refinement of $E_{interface}$ which according to Lemma 1 implies that E will become a refined terminal edge in the same step 1. Otherwise, Lepp(t) over the current mesh M_i only has interface terminal edges. So the processing of any of this interface terminal edges will reach E (which will become a refined terminal edge) in the same step 1 of the PTE algorithm.

Theorem 2. *In the PTE algorithm the use of a global edge-size tolerance b(M) eliminates interprocessor communication.*

Proof Consider any interface edge $L > b(M)$. Then there are three cases:

case 1: If L is not a terminal edge in any submesh M_i that contains L, then, according to Theorem 1, throughout the refinement process, L will become a terminal edge in one of the interface surfaces S_i that contains L. Since every existing interface terminal edge greater than $b(M)$ will be refined (according to Lemma 3.1 and Lemma 3.2), then L will be handled by the *case 3*.

case 2: An interface edge L can be a terminal edge in a submesh M_i but not a terminal edge in at least one adjacent submesh M_j. However, since the edge L is greater than $b(M)$, by performing interior refinement in M_j, according to Lemma 3.1 and Lemma 3.2, the edge L will become a terminal edge in M_j, too, which will be handled by *case 3*. The submesh M_j is refined independently of the submesh M_i and at the end we will have a conforming final mesh where all edges are less than or equal to $b(M)$.

case 3: An interface edge L is a terminal edge in the global mesh i.e., $\cup_{i=1}^{N} M_i$. This implies that L is in turn a terminal edge in each interface S_i of the submesh M_i that contains it. So after L is independently refined in each submesh that contain L, we will get a conforming refined mesh since all the submeshes will bisect L by its midpoint.

Remark. Note that for the same b(M) value, both the TE and PTE algorithms produce the same final nested meshes whenever that, for every tetrahedra t, its longest edge is unique. In the case that there exists tetrahedra which do not have unique longest edge, the longest edge selection can be made unique by defining the tetrahedra in a consistent, oriented way. Consequentely from a theoretical point of view, the method is fully stable and deterministic in the sense that, with adequate tetrahedron oriented convention and unique longest edge selection, for a fixed quality measure and a fixed mesh refinement criterion (b(M) value), both the sequencially generated mesh and the parallel one, are indentical refined meshes.

Future extensions

The PTE algorithm can be generalized for a variable edge size (density) function as follows:

Submesh VPTE Algorithm for Variable Terminal-edge Size
(1) Non-communication Phase
while the set S_E of terminal edges E such that
length (E) $>$ b(E) is not empty **do**
(Step 1) Perform repetitive refinement of every interior terminal edge $\tilde{E}$ greater than b($\tilde{E}$) in the interior of M_i
(Step 2) Perform repetitive refinement of every interface terminal edge $\tilde{E}$ greater than b($\tilde{E}$) in the interface of M_i
end while
(2) Comunication Phase
Use Lepp concept and edge interface communication to refine non-terminal edges E greater than b(E) (which implies refinement of some neighbor edges)

In the phase (1) of the VPTE algorithm, each interface edge E has a unique b(E) associated value, which implies that the terminal-edge refinement task (non-communication phase) is a direct generalization of the PTE algorithm. It is worth noting however that, once finished phase (1), the refined mesh can have a set S of non-terminal-edges L greater than b(L). In the case we need all edges L in the final mesh to be greater than b(L), a communication phase (2) to refine the edges of S_L will be required. This will refine the edges of S_L and some neighbor edges by making explicit use of the Lepp concept. To perform this task a limited interprocessor communication, similar to that required in adaptive refinement but for a small number of edges, is needed.

In the case that an adaptive refinement is needed based on an error estimator of a finite element solver, a limited communication between adjacent submeshes is required to communicate that an edge L needs to be refined by all submeshes that share L (no message reply is required and thus the communication phase is asynchronous). This is subject of an ongoing research in the University of Chile.

Performance Evaluation

The experimental study was performed in the Sciclone cluster in the College of William and Mary which consists of several different heterogeneous subclusters. We have used Whirlwind subcluster which consists of 64 single-cpu Sun Fire V120 nodes (650 MHz, 1 GB RAM). Also, we have used two geometric models with different needs for refinement: (i) a real human artery bifurcation model (Figure 1, left) and (ii) a simplified model for a human brain (Figure 1, right). The initial mesh of 92K tets for brain human model was decomposed into 503 subdomains while an initial mesh of 91K tets for the artery bifurcation model was decomposed into 504 subdomains. We used static cyclic assignment of subdomains to processors (i.e., $processor_{Pid}\ of\ a\ subdomain_{Sid} = subdomain_{Sid}\ \%\ number\ of\ processors$).

All the timing data reported in this paper use an optimized $C++$ implementation of the PTE algorithm, which in turn uses RemGO3D code which is a C++ serial implementation of the Terminal Edge Refinement Algorithm developed at the University of Chile, while the timing data

reported in [33] corresponds to a Java prototype which used a coordinator processor based implementation. The stopping criterion is a predefined bound (size of minimum edge) b for the terminal edges.

The stability of the PTE algorithm like any parallel longest-edge bisection algorithm is proved in [24, 25] and thus we do not present any data regarding this issue. Moreover, empirical studies on the quality of longest-edge subdivision for 3-dimensional meshes are addressed in detail in [20, 21].

Table 1: Time (seconds) for bifurcation example.

Mesh	Processors						
Size	1	2	4	8	16	32	64
3M	284.1	143.6	79.8	46.0	28.1	17.3	12.4
6M	650.4	330.5	184.2	107.7	66.0	40.3	27.6
15M	1889.0	955.2	531.6	302.8	184.4	114.4	81.8
30M	3816.8	1929.1	1060.5	611.8	369.9	229.5	156.2
72M	9483.4	4817.9	2703.4	1561.7	959.7	588.2	404.7
242M	22172.0	11228.1	6212.6	3595.8	2194.9	1352.0	921.3

Table 1 shows the execution time which is equal to the time it takes to process all of the subdomains assigned to a number of specifics processors. We report the maximum processing time (T_{max}) of all processors i.e., the time of the processor that dominates the parallel execution time. The speed of the C++ code varies from about 10.5K tets per second to 10.9K tets per second while the speed of the same algorithm using Java is about 500 to 800 tets per second on the same cluster. An improvement of more than an order of magnitude.

Table 2 shows the load imbalance measured in terms of the ($T_{max} - T_{min}$), where T_{min} is the execution time of the processor that completes the mesh generation of its subdomains first and waits for the termination of the rest of the processors. This table shows that work-load imbalance is

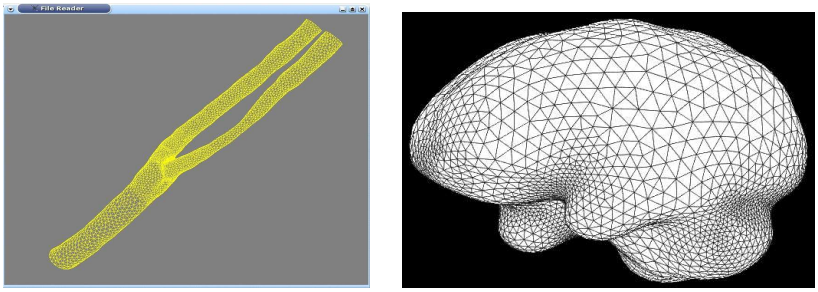

Figure 2: Surface of the tetrahedra mesh for an artery bifurcation model and simplified model of a human brain generated from MRI images [15].

Table 2: Imbalance meassured as the T_{max} - T_{min} (seconds).

Mesh	Processors						
Size	1	2	4	8	16	32	64
3M	0.0	3.0	16.0	21.1	17.0	13.0	11.4
6M	0.0	10.6	37.9	50.7	41.5	30.4	25.5
15M	0.0	21.4	108.0	135.0	109.7	84.5	74.4
30M	0.0	41.4	191.8	270.0	220.3	169.5	141.9
72M	0.0	152.4	589.0	727.9	594.5	437.7	369.5
242M	0.0	284.2	1197.1	1633.2	1335.9	1008.9	838.9

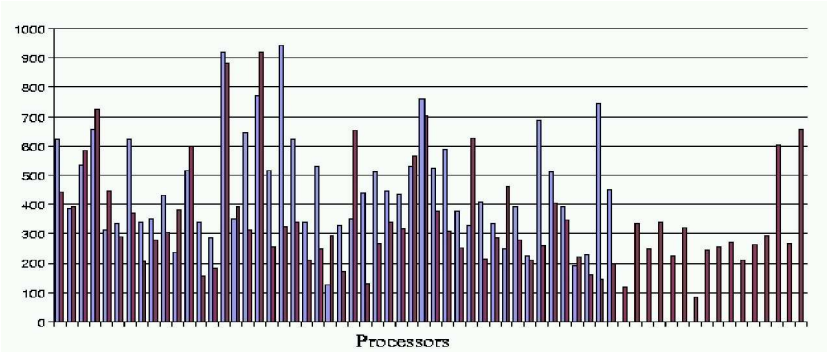

Figure 3: Execution time (in seconds) of 48 (light color bars) and 64 (dark color bars) processors for generating two tetrahedral meshes with 72M and 242M elements for the artery bifurcation model, respectively.

a very serious problem. The overdecomposition with the cyclic (ab-initio) assignment can not handle such sever imbalances. This suggests that the dynamic load balancing problem is important for mesh generation and refinement, for the PTE algorithm. Moreover, in [33] we have seen that the work-load balancing problem can be exaggerated due to heterogeneity of the clusters. The dynamic load balancing of the PTE method is out of the scope of this paper. Currently we are working to address this problem; we will use a parallel runtime system (PREMA [1]) which is developed at W&M for this purpose and the new C++ implementation of the PTE algorithm which is developed at the University of Chile.

Figure 3 shows the execution time of all processors for two configurations (48 and 64 processors). These data indicate that many processors are out of balance i.e., the work load imbalance is more serious than the fact that the (T_{max} - T_{min}) is very high. This explain the lack of scalability in the data of Table 1 despite the fact that there is no communication and global syncrhonization in PTE method and its new implementation. However, our data in [5] suggest that after dynamic load balancing the PTE method will scale well like the parallel advancing front which is de-

coupled method with zero communication and synchronization.

The data from Table 3 and Table 4 confirme our earlier conclusions on a different geometry (the human brain model). These data indicate that the behavior of PTE method in terms of load imbalance is geometry independent, since we have observed the same behavior in different geometries, too.

Table 3: Time (seconds) for the human brain model.

Mesh	Processors						
Size	1	2	4	8	16	32	64
21M	2650.4	1425.1	766.7	425.0	254.5	131.8	87.6
171M	15198.3	8169.7	4399.34	2437.6	1460.9	758.39	503.0

Table 4: Imbalance meassured as the T_{max} - T_{min} (seconds) for the human brain model.

Mesh	Processors						
Size	1	2	4	8	16	32	64
21M	0.0	199.8	174.9	148.1	164.3	93.1	76.6
171M	0.0	1141.1	1005.2	852.4	944.4	542.5	439.9

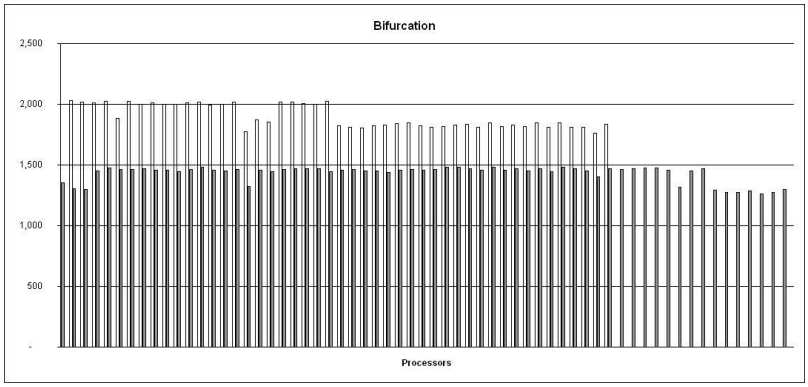

Figure 4: Number of tetrahedra of 48 (light color bars) and 64 (dark color bars) processors for artery bifurcation model for the input mesh

Conclusions

We have presented a parallel 3D Terminal-Edge mesh generation and refinement method which is stable with zero communication and synchronization. The new decoupled algorithm and its implementation lead to

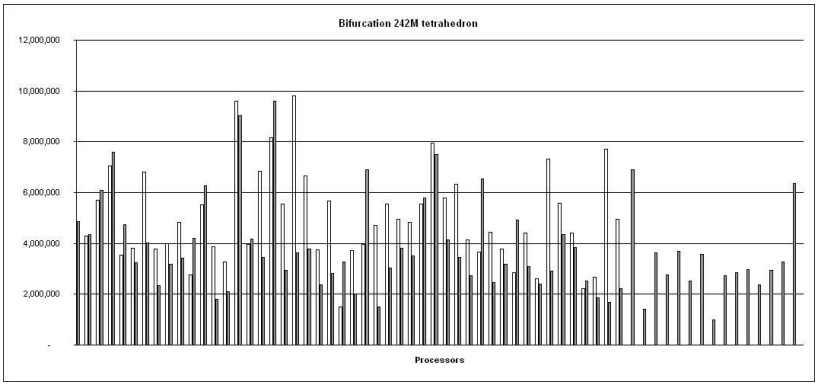

Figure 5: Number of tetrahedra of 48 (light color bars) and 64 (dark color bars) processors for artery bifurcation model for the final mesh (242 M tetrahedra)

one to two orders of magnitude performance improvements compared to an earlier implementation [33] of the same algorithm. The PTE method relies on 100% code re-use. We use a single code which can run both on single processor and many processors due to the decoupling nature of the PTE method. Usually with code re-use we take advantage of existing codes that target only single CPU computers. In this paper we developed a new method and its implementation that can be used both for single and multiple CPU computers. This will allow us to optimize a single code using well understood and familiar (sequential) programming methods and at the same time be able to generate faster larger meshes using multiple processors. The PTE algorithm and its current implementation are scalable (see Theorem 2). However, our performance data indicate the contrary due to processor work-load imbalances. This is due both to that PTE is a deterministic method, and to the fact that the overpartition of the initial mesh produced small submeshes with different size elements. To illustrate this imbalance behavior see Figures 4 and 5 that respectivaly show the number of tetrahedra for two configurations (48 and 64 processors) for the input and final meshes for the artery bifurcation problem.

Currently we are working in two fronts to address the scalability and overall performance (i.e., speed) of the PTE software by: (1) improving the performance of the Terminal-Edge method by further optimizing the new C++ implementation of the PTE method and (2) improving the work-load of processors using dynamic load balancing methods [1].

In the future we plan to use the new C++ implementation of the PTE algorithm within MRTS [17] in order to implement a parallel out-of-core terminal-edge mesh generation software capable to generate hundreds of millions of elements on relative small CoWs.

Acknowledgments

The C++ code RemGO3D was developed at the University of Chile under grant Fondecyt 1040713. The parallel results were obtained by using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund. We thank the referees whose comments helped to improve the paper and Daniel Pizarro who wrote the previous Java implementation.

References

[1] Kevin Barker, Andrey Chernikov, Nikos Chrisochoides, and Keshav Pingali. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):183–192, February 2004.

[2] G. Blelloch, J. Hardwick, G. Miller, and D. Talmor. Design and implementation of a practical parallel delaunay algorithm. *Algorithmica*, 24:243–269, 1999.

[3] D. Nave Nikos Chrisochoides P. Chew. Guaranteed–quality parallel delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28(2-3):191–215, June 2004.

[4] L. Paul Chew, Nikos Chrisochoides, and Florian Sukup. Parallel constrained Delaunay meshing. In *ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation*, pages 89–96, Northwestern University, Evanston, IL, 1997.

[5] Nikos Chrisochoides. *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume in print, chapter A Survey of Parallel Mesh Generation Methods. Springer Verla, 2006.

[6] Nikos Chrisochoides, Elias Houstis, and John Rice. Mapping algorithms and software environment for data parallel PDE iterative solvers. *Journal of Parallel and Distributed Computing*, 21(1):75–95, 1994.

[7] Nikos Chrisochoides and Demian Nave. Simultaneous mesh generation and partitioning. *Mathematics and Computers in Simulation*, 54(4-5):321–339, 2000.

[8] Nikos Chrisochoides and Démian Nave. Parallel Delaunay mesh generation kernel. *Int. J. Numer. Meth. Engng.*, 58:161–176, 2003.

[9] N.P. Chrisochoides. Multithreaded model for load balancing parallel adaptive computations. *Applied Numerical Mathematics*, 6:1–17, 1996.

[10] H. de Cougny and M. Shephard. Parallel volume meshing using face removals and hierarchical repartitioning. *Comp. Meth. Appl. Mech. Engng.*, 1999.

[11] J. Galtier and P. L. George. Prepartitioning as a way to mesh subdomains in parallel. In *Special Symposium on Trends in Unstructured Mesh Generation*. ASME/ASCE/SES, 1997.

[12] F. Gutierrez. The longest edge bisection of regular tetrahedron. In *Personal communication*, 2003.

[13] N. Hitschfeld and M. C. Rivara. Automatic construction of non-obtuse boundary and/or interface delaunay triangulations for control volume methods. *International Journal for Numerical Methods in Engineering*, 55:803–816, 2002.

[14] N. Hitschfeld, L. Villablanca, J. Krause, and M.C. Rivara. Improving the quality of meshes for the simulation of semiconductor devices using lepp-based algorithms. *to appear. International Journal for Numerical Methods in Engineering*, 2003.

[15] Yashusi Ito. Advance front mesh generator. Unpbulished Software, March 2004.

[16] Mark T. Jones and Paul E. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In *Proceedings of the Scalable High-Performance Computing Conference*, 1994.

[17] A. Kot and Nikos Chrisochoides. "green" multi-layered "smart" memory management system. *International Scientific Journal of Computing*, 2(3):91–97, 2004.

[18] L. Linardakis and N. Chrisochoides. Delaunay decoupling method for parallel guaranteed quality planar mesh generation. *(in print) SIAM Journal for Scientific Computing*, 2005.

[19] R. Lohner and J. Cebral. Parallel advancing front grid generation. In *International Meshing Roundtable*. Sandia National Labs, 1999.

[20] M.C.Rivara. Design and data structure of fully adaptive multigrid, finite element software. *ACM Transactions on Mathematical Software*, 10:242–264, 1984.

[21] S. N. Muthukrishnan, P. S. Shiakolos, R. V. Nambiar, and K. L. Lawrence. Simple algorithm for adaptative refinement of three-dimensionalfinite element tetrahedral meshes. *AIAA Journal*, 33:928–932, 1995.

[22] N. Nambiar, R. Valera, K. L. Lawrence, R. B. Morgan, and D. Amil. An algorithm for adaptive refinement of triangular finite element meshes. *International Journal for Numerical Methods in Engineering*, 36:499–509, 1993.

[23] Démian Nave, Nikos Chrisochoides, and L. Paul Chew. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 135–144, 2002.

[24] José G. Casta nos and John E. Savage. The dynamic adaptation of parallel mesh-based computation. In *SIAM 7th Symposium on Parallel and Scientific Computation*, 1997.

[25] José G. Casta nos and John E. Savage. Pared: a framework for the adaptive solution of pdes. In *8th IEEE Symposium on High Performance Distributed Computing*, 1999.

[26] T. Okusanya and J. Peraire. 3D parallel unstructured mesh generation,. In *Trends in Unstructured Mesh Generation, 1997, pp. 109–116.*, 1997.

[27] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.

[28] M. C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *International Journal for Numerical Methods in Engineering*, 28:2889–2906, 1989.

[29] M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40:3313–3324, 1997.

[30] M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. In *Computer-Aided Design*, volume 33, pages 263–277, 2001.

[31] M. C. Rivara and C. Levin. A 3d refinement algorithm for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.

[32] M. C. Rivara and M. Palma. New lepp algorithms for quality polygon and volume triangulation: Implementation issues and practical behavior. In *In Trends unstructured mesh generationi Eds: S. A. Cannan . Saigal, AMD,*, volume 220, pages 1–8, 1997.

[33] M.C. Rivara, D. Pizarro, and N. Chrisochoides. Parallel refinement of tetrahedral meshes using terminal-edge bisection algorithm. In *proceedings 13th International Meshing Roundtable, Williamsburg, Virginia*, pages 427–436, 2004.

[34] R. Said, N. Weatherill, K. Morgan, and N. Verhoeven. Distributed parallel delaunay mesh generation. *Comp. Methods Appl. Mech. Engrg.*, 177:109–125, 1999.

[35] R.B. Simpson, N. Hitschfeld, and M.C. Rivara. Approximate quality mesh generation. *Engineering with computers*, 17:287–298, 2001.

[36] Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 205–217, 2001.

[37] Shang-Hua Teng. Personal communication, February 2004.

[38] Roy Williams. *Adaptive parallel meshes with complex geometry.* Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, 1991.