Elsevier Editorial System(tm) for Applied Numerical Mathematics
Manuscript Draft

Manuscript Number: APNUM-D-10-00228R2

Title: Multithread parallelization of lepp-bisection algorithms

Article Type: Special Issue - WONAPDE 2010

Keywords: Longest edge bisection, Triangulation refinement, Lepp-bisection algorithm, Parallel multithread refinement, Parallel lepp-bisection algorithm, Finite element method

Corresponding Author: Maria-Cecilia Rivara, Ph.D.

Corresponding Author's Institution: Universidad de Chile

First Author: Maria-Cecilia Rivara, Ph.D.

Order of Authors: Maria-Cecilia Rivara, Ph.D.; Pedro Rodriguez; Rafael Montenegro, Ph.D.; Gaston Jorquera

Manuscript Region of Origin: CHILE

Abstract: Abstract
Longest edge (nested) algorithms for triangulation refinement in two dimensions are able to produce hierarchies of quality and nested irregular triangulations as needed both for adaptive finite element methods and for multigrid methods. They can be formulated in terms of the longest edge propagation path (Lepp) and terminal edge concepts, to refine the target triangles and some related neighbors. We discuss a parallel multithread algorithm, where every thread is in charge of refining a triangle t and its associated Lepp neighbors. The thread manages a changing Lepp(t) (ordered set of increasing triangles) both to find a last longest (terminal) edge and to refine the pair of triangles sharing this edge. The process is repeated until triangle t is destroyed. We discuss the algorithm, related synchronization issues, and the properties inherited from the serial algorithm. We present an empirical study that shows that a reasonably efficient parallel method with good scalability was obtained.

Multithread parallelization of lepp-bisection algorithms

Maria-Cecilia Rivara[a], Pedro Rodriguez[a,b], Rafael Montenegro[c], Gaston Jorquera[a]

[a] *Department of Computer Science, University of Chile, Santiago, Chile*
[b] *Department of Information Systems, University of Bio-Bio, Concepción, Chile*
[c] *University of Las Palmas de Gran Canaria, University Institute for Intelligent Systems and Numerical Applications for Engineering, Spain*

Abstract

Longest edge (nested) algorithms for triangulation refinement in two dimensions are able to produce hierarchies of quality and nested irregular triangulations as needed both for adaptive finite element methods and for multigrid methods. They can be formulated in terms of the longest edge propagation path (Lepp) and terminal edge concepts, to refine the target triangles and some related neighbors. We discuss a parallel multithread algorithm, where every thread is in charge of refining a triangle t and its associated Lepp neighbors. The thread manages a changing Lepp(t) (ordered set of increasing triangles) both to find a last longest (terminal) edge and to refine the pair of triangles sharing this edge. The process is repeated until triangle t is destroyed. We discuss the algorithm, related synchronization issues, and the properties inherited from the serial algorithm. We present an empirical study that shows that a reasonably efficient parallel method with good scalability was obtained.

Keywords:
Longest Edge Bisection, Triangulation Refinement, Parallel Multithread Refinement, Lepp-bisection Algorithm, Finite Element Method

1. Introduction

Triangular mesh generation for finite element methods has been extensively studied and addressed by engineers and numerical analysts since the seventies. Finite element methods are widely used numerical techniques for the practical analysis of complex physical problems modeled by partial differential equations, which require appropriate discretizations of the associated geometries. Because of their flexibility, triangulations are preferred practical tools. Pioneer works on triangulations for finite element methods and related issues are due to Babuska

Email addresses: mcrivara@dcc.uchile.cl (Maria-Cecilia Rivara), prodrigu@dcc.uchile.cl (Pedro Rodriguez), rafa@dma.ulpg.es (Rafael Montenegro), gjorquera@dcc.uchile.cl (Gaston Jorquera)

and Aziz [4], Lawson [27], Sibson [51]. Since then, intensive research on practical mesh generation has been performed. See e.g. the papers of Baker [6, 7], Bouraki and George [9], Jones and Plassmann [23, 24], Williams [55].

Computational methods for generating and refining triangular and tetrahedral finite element meshes in 2 and 3-dimensions can be roughly classified as Delaunay based methods, [6, 19, 7, 17, 50, 47] and methods based on the partition of triangles and tetrahedra [39, 33, 34, 28].

Longest edge refinement algorithms for triangulations, based on the longest edge bisection of triangles (obtained by joining the midpoint of the longest edge with the opposite vertex) were especially designed to deal with adaptive multigrid finite element methods [33, 34, 35, 36]. They are able to perform iterative local refinement by essentially maintaining the geometric quality of the input mesh as needed in finite element applications; they produce hierarchies of quality, smooth and nested irregular triangulations as required for non-structured multigrid methods. The properties of these algorithms are inherited from the non-degeneracy properties of the iterative longest edge bisection of triangles, [45, 52, 53, 1, 21] and are summarized in section 2.2.

The longest edge algorithms were generalized for 3-dimensional mesh refinement [38, 30], as well as for the derefinement or coarsening of the mesh [37]. Improved longest edge algorithms based on using the concepts of terminal edges and longest edge propagating paths were also developed [40, 39, 43]. The algorithms have been used for developing software for partial differential equations. See e.g. the applications of Nambiar et al [31], Muthukrishnan et al. [30]. Based on the longest edge idea over Delaunay meshes, Lepp-Delaunay algorithms for triangulation improvement have been also developed [39, 41, 44]

2. Longest edge refinement algorithms in 2-dimensions.

Roughly speaking the problem is the following: *given a conforming, non-degenerate triangulation, construct a locally refined triangulation, with a desired resolution and such that the smallest (or the largest angle) is bounded.* To simplify we introduce a subregion R to define the refinement area; and a condition over the longest-edge of the triangles to fix the desired resolution. We can consider the following subproblems:

Area Refinement. Given a quality acceptable triangulation (a triangulation with angles greater than or equal to an angle α) of a polygonal region D, construct a locally refined triangulation such that the longest edge of the triangles that intersect the refinement region R are less than δ.

Point / Edge Refinement. Here the refinement is performed around one vertex or along a boundary side, until all the triangles that intersect the vertex or the boundary edge are less than δ.

Adaptive finite element refinement. In the adaptive finite element context, the refinement region is repeatedly defined as a subset of triangles S_{ref} of the current triangulation (not necessarily connected) where the error of the finite element solution is too big to be acceptable [36].

Given the input mesh, the algorithm locally and iteratively refines the triangles of a changing S_{ref} set (or those intersecting the refinement region R) and some neighboring triangles. The new points introduced in the mesh are midpoints of the longest edge of some triangles of either of the input mesh or of some refined nested meshes. The longest edge bisection guarantees in a natural way the construction of non-degenerate and smooth irregular triangulations whose geometrical properties only depend on the initial mesh. In practice, for adaptive triangulation refinement, at each step longest-edge algorithms produce a refined conforming and guaranteed-quality output triangulation by performing selective longest edge bisection of the triangles of S_{ref} and some (longest edge) related neighbors.

2.1. A serial Lepp-bisection algorithm

An edge E is called a terminal edge in triangulation τ if E is the longest edge of every triangle that shares E, while the triangles that share E are called terminal triangles [39, 40]. Note that in 2-dimensions either E is shared by two terminal triangles t_1, t_2 if E is an interior edge, or E is contained in a single terminal triangle t_1 if E is a boundary (constrained) edge. See Figure 1(a) where edge AB is an interior terminal edge shared by two terminal triangles t_2, t_3.

For any triangle t_0 in τ, the longest edge propagating path of t_0, called *Lepp(t_0)*, is the ordered sequence $\{t_j\}_0^{N+1}$, where t_j is the neighbor triangle on a longest edge of t_{j-1}, and longest-edge (t_j) > longest-edge (t_{j-1}), for *j=1,... N*. Edge E = longest-edge(t_{N+1}) = longest-edge(t_N) is an interior terminal edge in τ and this condition determines N. Consequently either E is shared by a couple of terminal triangles (t_N, t_{N+1}) if E is an interior edge in τ, or E is shared by a unique terminal triangle t_N with boundary (constrained) longest edge. See Figure 1(a) for an illustration of these ideas, where Lepp(t_0)=(t_0, t_1, t_2, t_3).

The Lepp-bisection algorithm can be simply described as follows: each triangle t in S_{ref} is refined by finding Lepp(t), a pair of terminal triangles t_1, t_2 and associated terminal edge l. Then the longest edge bisection of t_1, t_2 is performed by the midpoint of l. The process is repeated until t is destroyed (refined) in the mesh. An efficient formulation of the algorithm where Lepp(t_0) is not repeatedly recomputed, but repeatedly updated starting from the non-modified part of the previous Lepp(t_0), is presented below. To this end we use a dynamic ordered list that stores pointers to the increasing triangles of (partial and full) Lepp(t_0), while t_0 remains in the changing mesh. This will be the basis to develop a parallel algorithm in section 4.

Lepp-Bisection Algorithm
Input : a quality triangulation, τ, and a set S_{ref} of triangles to be refined
for each t in S_{ref} **do**
  Insert-Lepp-Points(τ, t)
end for

Insert-Lepp-Points(τ, t_0)
Initialize Ordered-List (associated dynamically to Lepp(t_0)) with t_0

while Ordered-List is not empty **do**
Find last triangle t_N in Ordered-List
Find longest edge neighbor t_{N+1} of t_N and add it to Ordered-List (t_{N+1} can be null if longest edge of t_N is over the boundary)
if t_N, t_{N+1} share a terminal edge **or** t_{N+1} is null **then**
Perform longest-edge bisection of t_N, t_{N+1} by midpoint of common terminal edge
Eliminate t_N, t_{N+1} from Ordered-List
end if
end while

Note that the refinement task is performed when each Lepp(t_0) is fully computed and a terminal edge is identified, by using a very local refinement operation (the bisection of pairs of terminal triangles that share a common longest edge). This guarantees that the mesh is conforming throughout the whole refinement process. This improves previous longest edge algorithms [34] that produced intermediate non-conforming meshes. Also the algorithm is free of non-robustness issues, since this do not depend of complex computations, and the selected points are midpoints of existing previous edges.

Figure 1 illustrates the refinement of triangle t_0 in the input triangulation (a). Triangulation (b) shows the first point inserted, while triangulation (c) corresponds to the final triangulation obtained where the vertices are numbered in the creation order. In order to achieve this work, the full Lepp computation is performed three times to respectively insert points P_1, P_2, P_3. The first Lepp computation includes triangles t_0, t_1, t_2, t_3, being t_2, t_3 terminal triangles. Once the refinement of these triangles is performed, Lepp(t_0) is partially recomputed starting from t_1 (Figure 1(b)). This now includes triangles $t_0, t_1, \tilde{t}_2$ being $t_1, \tilde{t}_2$ terminal triangles, which are then refined. This time the new Lepp (t_0) computation starts from t_0 and includes $t_0, \tilde{t}_1$ which are in turn terminal triangles. The processing of t_0 concludes after refinement of $t_0, \tilde{t}_1$.

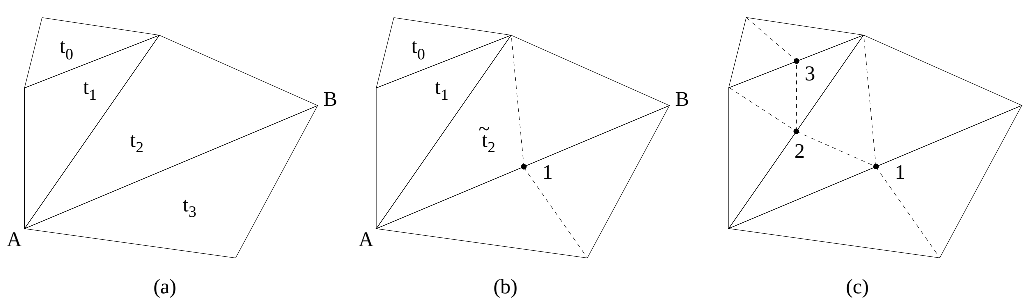

Figure 1: (a) Lepp(t_0) $= \{t_0, t_1, t_2, t_3\}$ and AB is terminal edge; (b) For refining triangle t, a first vertex 1 is added by bisection of the terminal triangles sharing AB. (c) Final triangulation obtained for refining t.

2.2. Properties of the 2-dimensional refinement algorithms

The non-degeneracy properties of the longest edge algorithms are summarized in lemmas 1 to 4 [21, 43]

Lemma 1 *(a) The iterative and arbitrary use of the algorithms only produces triangles whose smallest interior angles are always greater than or equal to $\alpha/2$, where α is the smallest interior angle of the initial triangulation. Also every triangle generated is similar to one of a finite number of reference triangles. (b) Furthermore, for any triangle t generated throughout the refinement process, its smallest angle α_t is greater than or equal to $\alpha_0/2$ where α_0 is the smallest angle of the triangle t_0 of the initial triangulation which contains t. Also t belongs to a finite number of similar triangles associated to t_0.*

Lemma 2 *Longest-edge refinement algorithms always terminate in a finite number of steps with the construction of a conforming triangulation.*

Lemma 3 *Any triangulation τ generated by means of the iterative use of the algorithms satisfies the following smoothness condition: for any pair of side-adjacent triangles $t_1, t_2 \in \tau$ (with respective diameters h_1, h_2) it holds that $\frac{min(h_1,h_2)}{max(h_1,h_2)} \geq k > 0$, where k depends on the smallest angle of the initial triangulation.*

Lemma 4 *For any triangulation τ, the global iterative application of the algorithm (the refinement of all the triangles in the preceding iteration) covers, in a monotonically increasing form, the area of τ with quasi-equilateral triangles (with smallest angles $\geq 30°$).*

The proof of Lemma 2 is based both on the fact that the refinement propagation is always performed towards bigger triangles in the current mesh, and on the fact that every mesh has bounded smallest angle. The smoothness property of Lemma 3 follows directly from the bound on the smallest angle of part (a) of Lemma 1. Lemma 4 states that the algorithm tends to isolate the worst angles.

2.3. Algorithm costs in two dimensions

The practical (adaptive) use of the refinement algorithm, requires of a number of K refinement iterations which produces a final refined mesh having a not a-priori known number of vertices $N = N_{ref} + N_{prop}$, where N_{ref} is the number of triangles iteratively marked for triangle refinement and N_{prop} is the sum of the number of points introduced by refinement propagation throughout the K iterations [43]. Thus the cost study requires of an amortized cost analysis [54] based on asymptotically studying the behavior of the algorithm throughout the refinement iterations, instead of the classical worst case study [29]. The amortized cost analysis must take into account the fact that, in one iteration the algorithm can introduce propagation points over all the triangles of the mesh (usually when the mesh is small), while for the remaining iterations a very small number of propagation points is introduced by each iteration.

To illustrate these ideas consider the mesh of Figure 2(a), where the single longest edge refinement of one triangle (triangle ABC) reaches the complete mesh as shown in Figure 2(b), which corresponds to a worst case behavior for a single refinement step. More importantly, note that after this step, the arbitrary iterative refinement of the triangles of vertex C, produces a very local refinement (1 or 2 vertices by refinement step) that approaches vertex C. Figure 2(c) shows the subtriangulation $ADBC$ obtained after 4 iterative refinement of the triangles of vertex C.

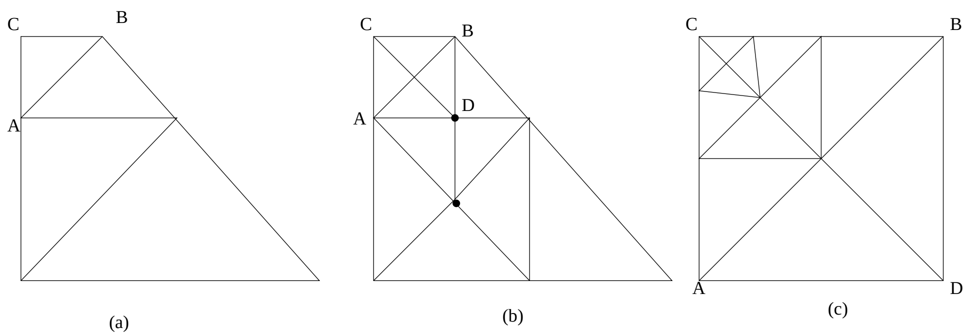

Figure 2: (a) Initial triangulation; (b) Refinement of triangle ABC reaches the complete mesh in one refinement step; (c) Further refinement around C is very local (triangulation detail $ADBC$)

Thus the iterative refinement of sets of target triangles introduces both a number N_{ref} of new vertices mandatorily generated to get the required triangle size, and a number N_{prop} of new propagation vertices which are introduced to keep the geometric quality and the smoothness of the refined mesh, as stated in lemmas 1 and 3.

We need: (1) To bound the numbers N_{ref} and N_{prop}, of new vertices inserted in the mesh; and (2) To study the computational cost of inserting them. The following lemma deals with the second item, which is a non obvious result since an algorithm can be able to select an optimal number of points K for point insertion, but the cost of inserting them can be higher than linear as happens with Delaunay point insertion strategy.

Lemma 5 *Consider any mesh M_1 obtained by iterative application of the Lepp-bisection algorithm over an initial mesh M_0, implying the introduction of K new vertices. Then the computational cost of finding and inserting the K vertices is linear in K, independently of the size of the triangulations and the number of iterations performed.*

The proof of this lemma follows directly from the fact that each triangle is added one time to the Ordered-List of the Insert-Lepp-Points function, and that the triangle refinement is performed in constant time.

Bounds on the number of points N_{ref} and N_{prop} introduced by the algorithm, as a function of L, the longest interior distance in the geometry D, and the required triangle size δ, are presented in Lemmas 6, 7, 8. We will consider the following two simple problems:

(P1) **Vertex refinement problem:** Iteratively refine the mesh around a vertex Q until the adjacent triangles have longest edge less than or equal to a length parameter δ.

(P2) **Circle area refinement:** Iteratively refine the triangles that intersect a circular region R_c until every triangle in R_c has longest edge less than or equal to a length parameter δ.

Lemma 6 *For solving (P1), a finite number of points N is added to the mesh, by longest edge bisection of pairs of terminal triangles, where* $N < K(Log(L/\delta))$, K *is a constant such that* $K = 2\pi/\alpha$, *and* L *is the longest interior distance in the polygonal geometry* D *measured over the smallest rectangle that contains* D.

The following lemma states that, for a big enough refinement area and for small δ, the number of propagation vertices is smaller than the number of vertices needed to get the desired mesh refinement.

Lemma 7 *For solving (P2), finite number of points* N_i *and* N_e *need to be respectively added in the interior and the exterior of* R_c *where*

$N_i < K_1((\frac{r}{\delta})^2)$, $N_e < K_2(\frac{r}{\delta})Log(\frac{L}{\delta})$, *$L$ is equal to the longest distance from the boundary of* R_c *to the boundary of* D, *r is the radius of* R_c, *and the constants are* $K_1 = 4\pi$ *and* $K_2 = 2\pi$.

Lemma 8 *(a) For solving (P1), the algorithm is linear in N defined in Lemma 6. (b) For solving (P2), the algorithm is linear in* $(N_i + N_e)$, *the number of points inserted in the mesh. In addition if* $r >> \delta$, *then the algorithm is linear in* N_i.

3. Previous results on parallel refinement algorithms

Distributed memory parallel algorithms for the refinement of huge meshes have been studied and used for complex practical applications for the last 20 years [55, 13, 23]. They are mainly based on the partition of basic geometric elements which produce nested refined meshes: algorithms based on the longest edge bisection of triangles / tetrahedra, or algorithms based on quadtree / octree techniques [48, 49]. As far as we know, parallel Delaunay algorithms have not been used in practice, because the serial Delaunay algorithms are less robust and more difficult to parallelize. However, research on the study of different aspects of parallel Delaunay methods, centered in the reuse of serial Delaunay codes, have been performed [2, 3, 15, 16] in recent years.

Distributed memory algorithms are based on partitioning the mesh and distributing its pieces to the computers of the cluster. To develop efficient and scalable distributed memory algorithms for mesh refinement, it is necessary to deal with the following issues: (a) to use efficient methods for mesh partitioning and related strategies such as dynamic mesh redistribution to equilibrate the load between processors throughout the computations; (b) to develop wise and efficient strategies for assuring coherent mesh refinement in the submeshes interfaces; and (c) to develop efficient methods for the movement of information between processors, which should be minimized.

The development of distributed parallel longest edge algorithms for the parallel refinement of triangulations have been studied and used for complex practical applications related with finite element methods. In a first review paper, Williams [55] recommends the use of parallel 4-triangles longest edge algorithm for the refinement of huge triangulations, for fluid dynamics applications. Later Jones and Plassmann [23, 24, 22] discussed parallel 4-triangles refinement algorithms for distributed memory finite element computations. They find independent sets of triangles to ensure that neighboring triangles are never simultaneously refined on different processors. The independent sets are chosen in parallel by a randomized strategy. A technique that dynamically repartitions the mesh to maintain adequate load distribution, which require to move triangles between processors, is also used. More recently Castaños and Savage [12, 14, 13] proposed a distributed memory parallelization of the original longest edge algorithm in 3-dimensions, and use this method for developing a parallel finite element code; they use a tree data structure to allow the refinement and derefinement of the meshes. Finally, Rivara et al [42] studied a Lepp-based algorithm for uniform refinement of tetrahedral meshes.

4. Multithread Lepp-bisection algorithm

In what follows we discuss a shared memory multithread algorithm that takes advantage both of the properties of the Lepp-bisection algorithm, and of the current multicore computers, which have several cores (light processors that perform the reading and instructions execution tasks) and dispose of a great amount of available memory to deal with big meshes.

To discuss the multithread Lepp-bisection algorithm, consider one step of the (adaptive finite element) refinement problem introduced in section 2. Given an input triangulation τ and a set $S \subset \tau$ of N triangles to be refined, we want to produce a conforming refined triangulation τ_f such that all the triangles of S are refined in τ_f. To this end we use a shared memory multicore computer having p physical cores with $p << N$. To perform this task each core $P_i (i = 1, ..p)$ is in charge of the processing of an individual triangle t in S and its associated changing Lepp sequence until the triangle t is refined in the mesh. Once the refinement of t is performed, the associated core will pick up another triangle of S to continue the refinement task.

To design the multithread algorithm, we need to deal with the following synchronization problems:

- S1 To avoid processing collisions associated to the parallel processing of triangles whose Lepp polygons overlap.

S2 To avoid data structure inconsistencies due to the parallel refinement of adjacent triangles that belong to different pairs of terminal triangles.

Problem $S1$ refers to the case where for different triangles t_0, t_0^* in S, their associated Lepp sequences overlap. Figure 3 illustrates the idea to avoid Lepp collision: cores 1 and 2 are respectively processing triangles t_0 and t_0^* which have overlapping Lepp sequences Lepp(t_0) $\cap$ Lepp(t^*) $= \{t_2, t_3, t_4, t_5\}$; core 1 reaches first triangle t_2, marks it as "busy" and proceeds to capture the full associated Lepp(t_0), while core 2 processing triangle t_0^* must suspend its work when marked triangle t_2 is reached.

To deal with overlapping Lepps, the algorithm takes advantage of the following result:

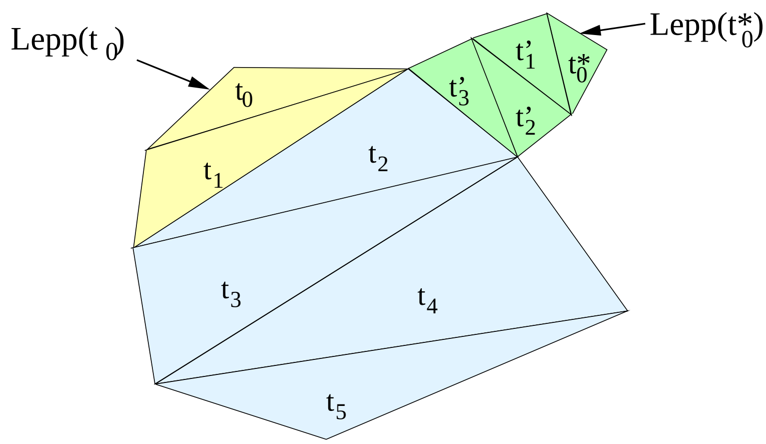

Figure 3: Lepp Collision: Lepp(t_0) $= \{t_0, t_1, t_2, t_3, t_4, t_5\}$ and Lepp(t_0^*) $= \{t^*, t'_1, t'_2, t'_3, t_2, t_3, t_4, t_5\}$

Proposition 1 *(a) Each triangle t_0 has an associated submesh Lepp(t_0) and a unique terminal edge which is the longest edge among all the edges of the submesh Lepp(t_0). (b) Any pair of triangles t_0, t_1 such that Lepp(t_0) $\bigcap$ Lepp(t_1)$\neq \phi$ have a common terminal edge. Furthermore, the smallest common triangle t_s allows to separate both involved Lepps in a full Lepp including the terminal triangles (let say Lepp(t_0)), and a partial Lepp that includes the smallest triangles of Lepp(t_1) until the predecessor of t_s.*

Proof. Part (a) follows directly from the definitions of Lepp and terminal edge. Part (b) follows from the fact that every Lepp is formed by an ordered set of increasing (longest edge) triangles. For the example of Figure 3, $t_s = t_2$ $\odot$

Problem $S2$ refers to the case where two threads attempt to refine neighboring triangles that do not belong to the same pair of terminal triangles as shown in Figure 4. Here the parallel refinement of triangles t_0 and t_0^*, and the updating of the data structure can introduce erroneous neighboring information. To deal with this issue, it is not allowed to simultaneously refine neighboring triangles associated to different cores.

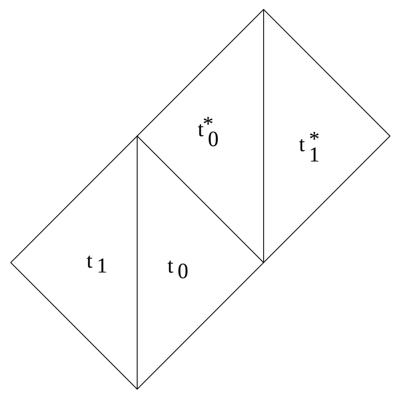

Figure 4: t_1, t_0 are terminal triangles; t_0^*, t_1^* are terminal triangles. Parallel refinement of triangles t_0, t_0^* is not allowed

Implementation details

1. To avoid Lepp processing collision, for each triangle t_0 being processed, the triangle t_0 and each triangle in the sequence of Lepp triangles is marked as Lepp-occupied, and can not be accessed by other thread until this triangle is refined. In exchange, the new refined triangles are marked as non-occupied.
2. For two threads p_0, p_1 processing in parallel triangles t_0, t_1 with Lepp collision, the first thread (let say p_0) that finds t_s in Proposition 1, will be in charge of performing the refinement associated to t_0. The second thread p_1 will be freed instead of waiting until the triangle t_s is refined. In the case that a partial Lepp is computed, the triangles are unmarked and the triangle t_1 is again added to S_{ref}
3. To assure data structure consistency we only perform refinement of a terminal triangle when the involved neighbors are non-marked triangles.
4. If a pair of terminal triangles with marked neighbors are found, the associated thread is freed and the complete marked Lepp is stored.

The parallel algorithm is summarized below:

Multithread-Lepp-Bisection Algorithm
Input: a quality triangulation τ, a set S_{ref} of N triangles to be refined, p threads (N>>p).
Output: a refined and conforming triangulation τ_f

Initialize S_{aux} as an empty set (set of triangles whose Lepp computation was blocked since a pair of terminal triangles with marked neighbor was found).
while $S_{ref} \cup S_{aux} \neq \phi$ **do**
  (process in parallel using the p threads)
  for each free processor p_i **do**
    select one triangle t of S_{ref} or one triangle t of S_{aux}. Eliminate t from S_{ref} or S_{aux}. Mark t as Lepp-occupied
    if t belonged to S_{aux} **then**
      Recover associated Ordered-List (storing fully computed Lepp) from Pending-Ordered-Lists
    else
      Initialize Ordered-List as an empty list
    end if
    Thread-Points-Insertion (τ, t, flag, Ordered-List)
    if flag indicates blocked terminal triangles **then**
      Add t to S_{aux}, and store Ordered-List in set of Pending-Ordered-Lists
    end if
    if flag indicates unfinished Lepp **then**
      Add t_0 to S_{ref} and unmark it
    end if
    Free thread P_i
    Update S_{ref} eliminating refined triangles
  end for
end while

Thread-Points-Insertion (τ, t_0, flag, Ordered-List)
if Ordered-List is empty **then**
  initialize Ordered-List with t_0
else
  Find terminal triangles in Ordered-List
  if terminal triangles have unmarked neighbors **then**
    perform longest edge bisection of terminal
    triangles and eliminate them from Ordered-List
  else
    set flag indicating blocked terminal edge and return
  end if
end if
while Ordered-List is not empty **do**
  Find last triangle t_N in Ordered-List
  Find longest edge neighbor t_{N+1} of t_N
  if t_{N+1} is Lepp-occupied **then**
    Set flag indicating unfinished Lepp
    Unmark all the triangles of Ordered-List
    Return
  end if
  Add t_{N+1} to Ordered-List and mark it as Lepp occupied

if t_N, t_{N+1} share a terminal edge or t_{N+1} is null **then**
  if t_N, t_{N+1} have unmarked neighbors **then**
    Perform longest edge bisection of t_N, t_{N+1}, by
    midpoint of common terminal edge
    Eliminate t_N, t_{N+1} from Ordered-List
  else
    set flag indicating blocked terminal edges and return
  end if
end if
end while

For the parallel algorithm the following properties hold:

Lemma 9 *(a) The parallel algorithm produces the same triangulations than the serial algorithm if the refinement of triangles having more than one longest edge is consistently performed by selecting the same longest edge. (b) The properties described in Lemma 1 to 8 for the serial algorithm extend to the multithread Lepp-bisection algorithm.*

5. Performance measures for parallel algorithms

The performance of a parallel algorithm is usually measured by using the speedup and the efficiency measures. The speedup S is defined as $S = T_s/T_p$, where T_s is the time taken by the sequential algorithm to solve the problem, while T_p is the time spent by the parallel algorithm by using p processors to solve the same problem.

The efficiency E is defined as $E = S/p$, where S is the speedup with p processors and p is the number of processors used to solve the associated problem.

The ideal speedup is equal to p, while the ideal efficiency is equal to 1. Note however that in practice it is common that a parallel implementation does not achieve linear speedup ($S = p$) since the parallel implementation usually requires additional overhead for the management of parallelism [32, 20].

Note also that the scalability of the parallel code can be observed by studying how the speedup changes as more cores are available. For an application that scales well, the speedup should increase at (or close to) the same rate as the amount of cores increases. That is if you double the number of cores, the speedup should also double [11].

Thus for an ideal and scalable parallel algorithm, the graph of the speedup versus the number of processors corresponds to a 45 degrees straight line (this behavior is called linear), while a good and scalable behavior corresponds to an approximate straight line with angle slightly less than 45°.

6. Empirical testing

For the testing work we have considered the following testing problems:

T1. Refinement of different initial Delaunay triangulations of sets of randomly generated data over a rectangle.

T2. Refinement of an L-shaped region around the reentrant corner to simulate adaptive finite element refinement.

We have used a computer with 4 physical cores (Intel Core (TM) i7 CPU, and 4GB of memory) to run the test problems.

6.1. Refinement of triangulations of randomly generated data over a rectangle

We consider randomly generated data over a rectangle. The CGAL library [10] was used to obtain the initial Delaunay triangulations. Note that due to the strategy used for generating the triangulation vertices, the initial triangulations include small sets of poor quality triangles. Our goal is to evaluate the performance of the iterative application of the multithread algorithm for triangulation refinement, going from triangulations of 200000 triangles to triangulations of 4-5 millions of triangles.

For testing the behavior of iterative refinement, we have considered three strategies for selecting sets of triangles to be refined:

Largest triangles refinement. Here we repeatedly select a fixed percentage of the triangles with largest (longest) edges in the current mesh. Note that this testing strategy means the selection of an important set of terminal triangles of each current mesh, so we expect that the refinement propagation be minimized.

Smallest triangles refinement. Here we repeatedly select a fixed percentage of the triangles with smallest (longest) edges in the current mesh. Consequently the refinement is repetitively concentrated around the smallest triangles of the initial mesh, and we expect that the propagation refinement be maximized.

Random triangles refinement. Here we repeatedly and randomly select a fixed percentage of the triangles of the current mesh.

For the three refinement strategies, we have repeatedly refined the 5% of the triangles of the current mesh (and so on with the 10%, and 25% of the triangles) until achieving meshes of around 5 millions of triangles.

Tables 1 to 6 present results for six testing problems (10% refinement of smallest, largest and random triangles, 25% refinement of smallest largest and random triangles). These include some statistics on the iterative parallel refinement by using 4 threads. Each row j associated to the jth refinement step, includes the size of the initial mesh, the number of triangles to be refined, the size of the refined mesh, the execution time spent at the current iteration for the 4-threads case, and the accumulated time until the jth iteration for the 4-threads case. For the final mesh of the current iteration, the row also includes the length of the average Lepp in the mesh, and the length of the longest Lepp in the mesh.

For the same testing problems, Tables 7 to 12 summarize the execution time for the serial case and for using 2, 3 and 4 cores, their associated speedup and their associated efficiency, throughout the refinement iterations.

As expected, the refinement of the largest triangles, which involves a big subset of the terminal triangles in the refinement process, produces the smallest size refined meshes, while the refinement of the smallest triangles produces the biggest size refined meshes. Note also that as expected the meshes size increase less in percentage as the refinement proceeds. On the other hand, we can see that the refinement of random triangles represents better the average behavior of the algorithm (in between of the results obtained for largest triangles refinement and smallest triangles refinement).

In order to evaluate the practical performance of the algorithm we have computed the speedup, which is the time of the serial algorithm divided by the p-processors algorithm time, and the efficiency measure, which is computed as the speedup divided by the number of processors, For a discussion on these concepts see section 5 and references [25, 20]. Note that according to the discussion of section 5, the algorithm presents acceptable efficiency (above 0.75 for most of the iterations and the different cases) and good scalable behavior. In general the efficiency increases as the size of the input mesh increases, while the efficiency remains almost constant as the number of cores increases. Note that the worst efficiency corresponds to some isolated iterations of the 10% largest triangle refinement case (0.58 for the refinement iterations 3,5,8,11). We believe that this is due to the overhead of the parallel Lepp processing, since for this case most of the work is performed over pairs of terminal triangles. On the contrary the 10% random triangle selection shows a superlinear behavior which happens rarely with some algorithms. This suggests that the random processing of the triangles of S_{ref} should reduce Lepp collisions and should improve the algorithm efficiency. Note that both for the largest and smallest triangle refinement cases, the triangles of S_{ref} were processed in order (from largest to smallest triangles, and from smallest to largest triangles, respectively).

Finally note that, since the randomly generated data point produces initial triangulations with a percentage of bad quality triangles, these are also good examples for studying the algorithm behavior with respect to the smallest angle, for big refined meshes. For all the test cases, the distribution of smallest angle was obtained, showing that the mesh improvement behavior of the Lemma 4 holds as expected. This can be seen in Table 13 for 10% random triangle refinement case. The initial mesh has 6.43% of triangles with smallest angles less than 10 degrees, 23.43% of triangles with smallest angles less than 20° and 25% of triangles with angles between 30 and 40 degrees. In exchange, the final mesh has only 1.93% of triangles with angles less than 10°, 5.45% of triangles with angles less than 20° and 48,30% of triangles with smallest angles between 30 and 40 degrees. Note that the worst angles are not eliminated but isolated in the refined meshes. These results are also presented graphically in Figure 5 (initial mesh, mesh 4 and mesh 8).

Finally, Figures 6 and 7 show the initial and a refined triangulation for a small example (3000 triangles in the initial mesh).

6.2. Refinement of an L-shaped domain

In order to simulate the adaptive refinement associated to finite element methods, we have considered the L-shaped domain of the Figure 8, with reentrant vertex B of coordinates (5,5). We have performed refinement by using a circle refinement region of center B and radius r (see Figure 8), for different values of the parameter r. The initial mesh is shown in Figure 8. We have performed iterative refinement of all the triangles that intersect R_c until obtaining triangles of size $\delta = 0.001$ (longest edge $\leq \delta$), by starting with the initial mesh of 6 triangles of Figure 8. Note that all the refined meshes only include right isosceles triangles.

Tables 14,15,16 show some statistics obtained for the iterative refinement for the last refinement steps, while Tables 17,18,19 summarize the computing times and the values of speedup and efficiency associated to these problems. We can see that the algorithm shows an almost ideal behavior with efficiency higher than 0.86 for all the iterations included.

Note that for these problems the refinement concentrates in the interior of R_c where the number of triangles refined by propagation remains very low throughout the refinement iterations. This is in complete agreement with the results of Lemmas 7 and 8. A small refined mesh of the L shaped domain is shown in Figure 9.

Table 1: Statistics on iterative refinement, 10% smallest triangles, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
199967	19997	390056	444	444	3.62137	17
390056	39006	648271	596	1040	3.62780	16
648271	64827	929824	665	1705	3.71453	17
929824	92982	1244975	856	2561	3.75525	18
1244975	124498	1601045	850	3411	3.75182	19
1601045	160105	2016463	1145	4556	3.71457	21
2016463	201646	2499121	1388	5944	3.67123	23
2499113	249911	3078594	1534	7478	3.61935	23

Table 2: Statistics on iterative refinement, 10% biggest triangles, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
199967	19997	229556	123	123	3.09404	12
229556	22956	255764	66	189	2.94647	14
255764	25576	284528	160	349	2.84749	11
284528	28453	316069	91	440	2.77763	11
316069	31607	350533	85	525	2.72792	12
350533	35053	388300	99	624	2.68983	13
388300	38830	429846	187	811	2.66200	13
429846	42985	475448	113	924	2.64107	13
475448	47545	525769	126	1050	2.62206	12
525769	52577	581251	137	1187	2.60169	10
581251	58125	642463	194	1381	2.58612	10
642463	64246	710053	264	1645	2.57197	10
710053	71005	784723	183	1828	2.56060	10
784723	78472	867121	261	2089	2.55001	11

Table 3: Statistics on iterative refinement, 10% random selection, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
199967	19996	294738	254	254	3.27653	19
294738	29473	420972	303	557	3.19591	16
420972	42097	595838	426	983	3.14954	15
595838	59583	839651	594	1577	3.13082	15
839651	83965	1180479	840	2417	3.11291	14
1180479	118047	1658733	1207	3624	3.10386	15
1658733	165873	2327815	1602	5226	3.09373	13
2327815	232781	3261653	2237	7463	3.08542	14
3261653	326165	4569294	3167	10630	3.07860	17

Table 4: Statistics on iterative refinement, 25% smallest triangles, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum. Time	Avrge Lepp	Longest Lepp
199967	49991	495340	755	755	3.41820	17
495340	123835	1035383	1296	2051	3.28188	16
1035383	258845	1881482	2124	4175	3.26975	17
1881482	470370	3164480	3132	7307	3.27757	17
3164480	791120	5066810	4684	11991	3.26006	19

Table 5: Statistics on iterative refinement, 25% largest triangles, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum. Time	Avrge Lepp	Longest Lepp
199967	49991	276067	197	197	2.85749	12
276067	69016	356806	250	447	2.71562	12
356806	89201	459687	315	762	2.64974	12
459687	114921	590847	372	1134	2.60073	10
590847	147711	757825	505	1639	2.56403	10
757825	189456	970193	578	2217	2.53998	11
970193	242548	1240472	785	3002	2.52329	10
1240472	310118	1585070	955	3957	2.51098	10
1585070	396267	2023093	1108	5065	2.49905	10
2023093	505773	2580320	1573	6638	2.48278	9
2580320	645080	3288422	1761	8399	2.47567	9
3288422	822105	4188240	2254	10653	2.47121	10

Table 6: Statistics on iterative refinement, 25% random selection, 4 threads case.

Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum. Time	Avrge Lepp	Longest Lepp
199967	49992	389426	532	532	3.16258	15
389426	97357	707114	819	1351	2.99168	13
707114	176779	1255906	1316	2667	2.89928	13
1255906	313977	2205602	2416	5083	2.84503	13
2205602	551401	3847925	4252	9335	2.80838	12

Table 7: Execution time and efficiency measures, 10% smallest triangles.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	1508	1007	657	451	1,50	2,30	3,34	0,75	0,77	0,84
2	2030	1226	800	604	1,66	2,54	3,36	0,83	0,85	0,84
3	2206	1449	876	759	1,52	2,52	2,91	0,76	0,84	0,73
4	2475	1449	1006	747	1,71	2,46	3,31	0,85	0,82	0,83
5	2805	1695	1256	877	1,65	2,23	3,20	0,83	0,74	0,80
6	3280	1860	1329	1134	1,76	2,47	2,89	0,88	0,82	0,72
7	3835	2249	1552	1386	1,71	2,47	2,77	0,85	0,82	0,69
8	4588	2649	1846	1508	1,73	2,49	3,04	0,87	0,83	0,76

Table 8: Execution time and efficiency measures, 10% largest triangles.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	239	133	139	92	1.80	1.72	2.60	0.90	0.57	0.65
2	212	120	88	72	1.77	2.41	2.94	0.88	0.80	0.74
3	233	130	95	100	1.79	2.45	2.33	0.90	0.82	0.58
4	256	143	104	84	1.79	2.46	3.05	0.90	0.82	0.76
5	281	157	113	123	1.79	2.49	2.28	0.89	0.83	0.57
6	309	171	124	99	1.81	2.49	3.12	0.90	0.83	0.78
7	338	188	136	116	1.80	2.49	2.91	0.90	0.83	0.73
8	372	225	150	159	1.65	2.48	2.34	0.83	0.83	0.58
9	411	230	166	131	1.79	2.48	3.14	0.89	0.83	0.78
10	453	267	197	138	1.70	2.30	3.28	0.85	0.77	0.82
11	500	279	233	216	1.79	2.15	2.31	0.90	0.72	0.58
12	552	307	229	173	1.80	2.41	3.19	0.90	0.80	0.80

Table 9: Execution Time and efficiency measures, 10% random selection.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	759	455	307	253	1.67	2.47	3	0.83	0.82	0.75
2	1085	590	397	294	1.84	2.73	3.69	0.92	0.91	0.92
3	1650	858	532	423	1.92	3.1	3.9	0.96	1.03	0.98
4	2438	1066	743	595	2.29	3.28	4.1	1.14	1.09	1.02
5	3504	1548	1032	839	2.26	3.4	4.18	1.13	1.13	1.04
6	5034	2153	1439	1295	2.34	3.5	3.89	1.17	1.17	0.97
7	7191	3070	2023	1583	2.34	3.55	4.54	1.17	1.18	1.14
8	10650	4147	2870	2221	2.57	3.71	4.8	1.28	1.24	1.2
9	14628	5903	3978	3190	2.48	3.68	4.59	1.24	1.23	1.15

Table 10: Execution time and efficiency measures, 25% smallest triangles.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	2351	1527	929	755	1,54	2,53	3,11	0,77	0,84	0,78
2	4236	2424	1626	1296	1,75	2,61	3,27	0,87	0,87	0,82
3	6691	3805	2593	2124	1,76	2,58	3,15	0,88	0,86	0,79
4	9988	5828	3979	3132	1,71	2,51	3,19	0,86	0,84	0,80
5	15008	8729	6306	4684	1,72	2,38	3,20	0,86	0,79	0,80

Table 11: Execution time and efficiency measures, 25% largest triangles.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	595	342	249	197	1,74	2,39	3,02	0,87	0,80	0,76
2	628	371	266	250	1,69	2,36	2,51	0,85	0,79	0,63
3	799	489	354	315	1,63	2,26	2,54	0,82	0,75	0,63
4	1017	676	472	372	1,50	2,15	2,73	0,75	0,72	0,68
5	1291	769	536	505	1,68	2,41	2,56	0,84	0,80	0,64
6	1639	977	679	578	1,68	2,41	2,84	0,84	0,80	0,71
7	2081	1232	858	785	1,69	2,43	2,65	0,84	0,81	0,66
8	2718	1557	1093	955	1,75	2,49	2,85	0,87	0,83	0,71
9	3370	1999	1392	1108	1,69	2,42	3,04	0,84	0,81	0,76
10	4402	2547	1780	1573	1,73	2,47	2,80	0,86	0,82	0,70
11	5449	3227	2560	1761	1,69	2,13	3,09	0,84	0,71	0,77
12	7091	4144	2877	2254	1,71	2,46	3,15	0,86	0,82	0,79

Table 12: Execution time and efficiency measures, 25% random selection.

	Execution Time (ms)				Speed-Up			Efficiency		
Mesh	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
1	1513	980	597	472	1.54	2.53	3.21	0.77	0.84	0.80
2	2522	1579	987	782	1.60	2.56	3.23	0.80	0.85	0.81
3	4160	2564	1713	1354	1.62	2.43	3.07	0.81	0.81	0.77
4	7401	4606	2955	2312	1.61	2.50	3.20	0.80	0.83	0.80
5	12940	7719	5143	4002	1.68	2.52	3.23	0.84	0.84	0.81

Table 13: Distribution of smallest angles throughout the iterations. Random triangles selection, 10% refinement.

		Distribution of smallest angles (in %)					
Mesh	Mesh Size	$0^o - 10^o$	$10^o - 20^o$	$20^o - 30^o$	$30^o - 40^o$	$40^o - 50^o$	$50^o - 60^o$
M0	199967	6.43	17.00	24.43	25.51	19.37	7.26
M1	294738	5.55	13.41	21.97	30.05	20.50	8.55
M2	420972	4.76	10.98	20.01	33.76	21.38	9.10
M3	595838	4.11	9.15	18.34	36.85	22.12	9.43
M4	839651	3.58	7.70	16.84	39.55	22.71	9.61
M5	1180479	3.14	6.50	15.54	41.84	23.24	9.77
M6	1658733	2.77	5.54	14.39	43.85	23.63	9.86
M7	2327815	2.46	4.73	13.40	45.57	23.94	9.95
M8	3261653	2.18	4.07	12.54	47.02	24.19	10.04
M9	4700254	1.93	3.52	11.88	48.30	24.29	10.08

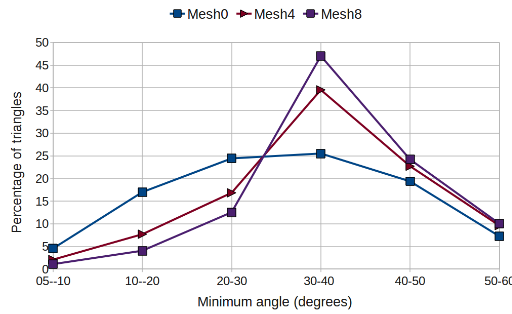

Figure 5: Distribution of smallest angles, 10% random refinement, 4 threads

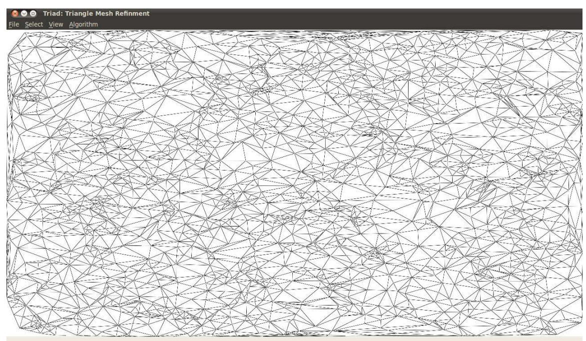

Figure 6: Initial triangulation

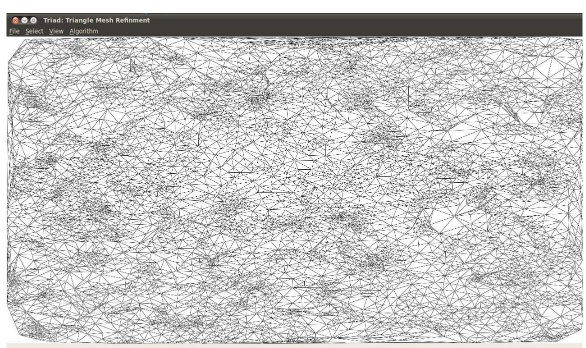

Figure 7: 25% random refinement; second refined mesh

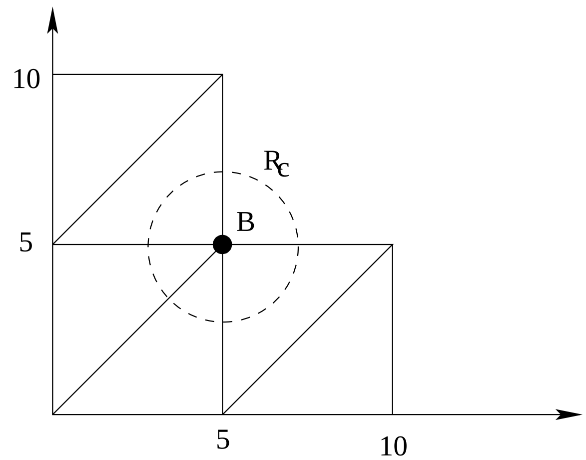

Figure 8: L-shaped domain with refinement region R_c

Table 14: Statistics on iterative refinement, L domain, circle refinement region, r=0.3, triangle size δ=0.001, 4 threads case.

Refinement Iteration	Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
22	39234	37530	77130	222	222	2.05207	14
23	77130	74760	152478	439	661	2.03722	14
24	152478	149094	302370	867	1528	2.02305	14
25	302370	297612	600990	1788	3316	2.01720	14
26	600990	594474	1197294	3557	6873	2.01720	14

Table 15: Statistics on iterative refinement, L domain, circle refinement region, r=0.5, triangle size δ=0.001, 4 threads case.

Refinement Iteration	Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
22	106452	103662	210738	677	677	2.03220	12
23	210738	206844	418512	1236	1913	2.02137	13
24	418512	413082	833040	2440	4353	2.01533	14
25	833040	825294	1660182	5001	9354	2.00983	11
26	1660182	1649400	3312420	10067	19421	2.00983	11

Table 16: Statistics on iterative refinement, L domain, circle refinement region, r=1.2, 24 iterations, 4 threads case.

Refinement Iteration	Mesh size	Triangles to be refined	Final Mesh Size	Execution Time [ms]	Accum Time	Avrge Lepp	Longest Lepp
20	152406	149094	302298	873	873	2.03703	14
21	302298	297612	600918	1791	2664	2.02295	14
22	600918	594468	1197198	3470	6134	2.01711	14
23	1197198	1188012	2387346	7132	13266	2.01245	14
24	2387346	2374542	4764960	14227	27493	2.00870	14

Table 17: Execution time and efficiency measures, L domain, circle refinement region, r=0.3, δ=0.001.

	Execution Time (ms)				Speed-Up			Efficiency		
Refinement Iteration	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
22	768	439	281	222	1,75	2,73	3,46	0,87	0,91	0,86
23	1559	823	571	439	1,89	2,73	3,55	0,95	0,91	0,89
24	3062	1666	1123	867	1,84	2,73	3,53	0,92	0,91	0,88
25	6343	3427	2300	1788	1,85	2,76	3,55	0,93	0,92	0,89
26	12543	6838	4600	3557	1,83	2,73	3,53	0,92	0,91	0,88

Table 18: Execution time and efficiency measures, L domain, circle refinement region, r = 0.5, δ=0.001

	Execution Time (ms)				Speed-Up			Efficiency		
Refinement Iteration	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
22	2112	1205	784	677	1,75	2,69	3,12	0,88	0,90	0,78
23	4556	2321	1627	1236	1,96	2,80	3,69	0,98	0,93	0,92
24	8461	4650	3135	2440	1,82	2,70	3,47	0,91	0,90	0,87
25	17367	9593	6396	5001	1,81	2,72	3,47	0,91	0,91	0,87
26	34668	19002	12804	10067	1,82	2,71	3,44	0,91	0,90	0,86

Table 19: Execution time and efficiency measures, L domain, circle refinement region, r=1.2, 24 iterations

	Execution Time (ms)				Speed-Up			Efficiency		
Refinement Iteration	Serial	2P	3P	4P	2P	3P	4P	E2	E3	E4
20	3038	1645	1144	873	1,85	2,66	3,48	0,92	0,89	0,87
21	6213	3425	2312	1791	1,81	2,69	3,47	0,91	0,90	0,87
22	12152	6673	4481	3470	1,82	2,71	3,50	0,91	0,90	0,88
23	24777	13910	9231	7132	1,78	2,68	3,47	0,89	0,89	0,87
24	49303	27328	18363	14227	1,80	2,68	3,47	0,90	0,89	0,87

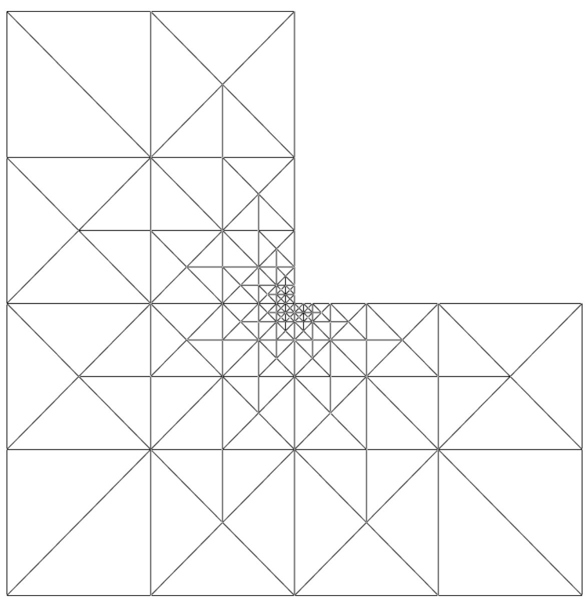

Figure 9:

7. Conclusions

We have presented a reasonably efficient and good scalable multithread parallel Lepp-bisection algorithm for the refinement of triangulations, with efficiency higher than 0.75 for most of the cases of randomly generated data, and with efficiency higher than 0.86 for the L shaped domain. The analysis of the experiments performed suggests that the random processing of the triangles to be refined should improve the algorithm efficiency. In the near future we will test the algorithm with different architectures and bigger multicore computers. We plan to generalize the algorithm to 3-dimensions, where each thread will take in charge the multidirectional Lepp points insertion task involved with each (to be refined) target tetrahedron. We will also study Lepp-bisection distributed memory algorithms, as well as mixed multithread / distributed refinement methods.

Acknowledgements. This work was partially supported by the Department of Computer Science , University of Chile; and by the Spanish Government, "Secretaría de Estado de Universidades e Investigación", "Ministerio de Ciencia e Innovación", and FEDER, grant contracts: CGL2008-06003-C03 and UNLP08-3E-010. We thank the referees who contributed to the improvement of this paper.

References

[1] A. Adler, On the Bisection Method for Triangles, Mathematics of Computation, 40 (1983) 571-574.

[2] C. Antonopoulos, F. Blagajevic, A. Chernikov, N. Chrisochoides, and D. Nikolopoulos. Algorithm, software, and hardware optimizations for delaunay mesh generation on simultaneous multithreaded architectures. Journal on Parallel and Distributed Computing, 69, 2009.

[3] C. Antonopoulos, F. Blagajevic, A. Chernikov, N. Chrisochoides, and D. Nikolopoulos. A multigrain delaunay mesh generation method for multicore smt-based architectures. Journal of Parallel and Distributed Computing, 69(7), 2009.

[4] I. Babuska, A.K. Aziz, On the angle condition in the finite elemen method, SIAM J. Numer. Anal 13 (1976) 214-226.

[5] I. Babuska, O. C. Zienkiewicz, J. Gago and E.R. de A. Oliveira, (Eds.). Accuracy estimates and adaptive refinements in finite element computations, John Wiley, 1986.

[6] T. Baker (1989), Automatic mesh generation for complex three dimensional regions using a constrained Delaunay triangulation. Engineering with Computers, 5(1989), 161-175.

[7] T. J. Baker (1994). Triangulations, mesh generation and point placement strategies. Computing the Future, ed. D Caughey, John Wiley, 1994, 61-75.

[8] R. E. Bank, PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 8.0. SIAM, 1998.

[9] H. Borouchaki and P. L. George (1997), Aspects of 2-D Delaunay Mesh Generation. International Journal for Numerical Methods in Engineering, 40, 1997, 1957-1975.

[10] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. *Triangulations in CGAL*. /Comput. Geom. Theory Appl./, 22:5-19, 2002.

[11] C. Breshears, The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications. O'Reilly Media, Inc. 2009.

[12] J.G. Castaños, J.E. Savage, Pared: a framework for the adaptive solution of pdes. In 8th IEEE Symposium on High Performance Distributed Computing, 1999.

[13] José G. Cataños and John E. Savage. Parallel refinement of unstructured meshes. Procs IASTED Conference on Parallel and Distributed Computing and Systems (PDCS'99), Boston, 1999.

[14] José G. Cataños and John E. Savage. Repartitioning unstructured adaptive meshes. In IPDPS, pages 823-832. IEEE Computer Society, 2000.

[15] A. Chernikov and N. Chrisochoides. Generalized two-dimensional delaunay mesh refinement. SIAM Journal on Scientific Computing, 31:3387-3403, 2009.

[16] A. Chernikov and N. Chrisochoides. Algorithm 872: Parallel 2d constrained delaunay mesh generation. ACM Trans. Math. Softw. 34(1):1-20, 2008.

[17] L. P Chew (1989a). Constrained Delaunay triangulations. Algorithmica 4 (1989) 97-108.

[18] L.P. Chew, Guaranteed-quality triangular meshes. Technique Report TR-89-983 Cornell University, 1989.

[19] P. L. George, F. Hecht, and E. Saltel (1991). Automatic mesh generator with specified boundary. Source, Computer Methods in Applied Mechanics and Engineering, 92 (1991) 269 V 288.

[20] A. Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing, 2nd. Ed., Addison Wesley, 2003.

[21] C. Gutierrez, F. Gutierrez, M.C. Rivara, Complexity on the bisection method. Theoretical Computer Science 382 (2007), 131-138.

[22] Mark T. Jones and Paul E. Plassmann. Parallel algorithms for adaptive mesh refinement. SIAM Journal on Scientific Computing, 18(3):686-708, 1997.

[23] M. T. Jones, P.E. Plassman, Computational results for parallel unstructured mesh computations, Computing Systems in Engineering, 5 (1994) 297–309.

[24] M. T. Jones, E. Plassmann, Adaptive refinement of unstructured finite element meshes, Finite Elements in Analysis and Design, 25 (1997) 41–60.

[25] A. H. Karp, H. P. Flatt. Measuring parallel processor performance. Communications of the ACM, 33 (1990), 539-543.

[26] B. Kearfott, A Proof of Convergence and an Error Bound for the Method of Bisection in R^n, Mathematics of Computation, 32 (1978) 1147–1153.

[27] C.L. Lawson, Software for C^1 surface interpolation, In Mathematical Software III, John R. Rice (editor), Academic Press 1977, 161-194.

[28] A. Liu, B. Joe, Quality local refinement of tetrahedral meshes based on bisection, SIAM Journal on Scientific Computing, 16 (1995) 1269–1291.

[29] U. Manber, Introduction to algorithms. A creative Approach, Addison Wesley, 1991.

[30] S. N. Muthukrishnan, P. S. Shiakolas, R. V. Nambiar, K. L. Lawrence, Simple algorithm for adaptative refinement of three-dimensional finite element tetrahedral meshes, AIAA Journal, 33 (1995) 928–932.

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

[32] T. Rauber, and G.Runger, Parallel programming for multicore and cluester systems. Springer, 2010.

[33] M. C. Rivara, Design and data structure for fully adaptive, multigrid finite-element software, ACM Transactions on Mathematical Software, 10 (1984) 242–264.

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

[35] M. C. Rivara, A dynamic multigrid algorithm suitable for partial differential equations with singular solutions, In Recent Advances in Systems Modelling and Optimization, L. Contesse, R. Correa, A. Weintraub (Eds), Lecture Notes in Control and Information Sciences, Springer-Verlag, (1986) 190–199.

[36] M. C. Rivara, Adaptive finite element refinement and fully irregular and conforming triangulations, Chapter 20 in Accuracy Estimates and Adaptive Refinements in Finite Element Computations, I. Babuska, J. Gago. E.R. de A. Oliveira, O.C. Zienkiewicz (Eds.), John Wiley (1986) 359–370.

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

[38] M. C. Rivara and C. Levin, A 3D refinement algorithm suitable for adaptive and multigrid techniques, Communications in Applied Numerical Methods, 8 (1992) 281–290.

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

[40] M. C. Rivara, M. Palma, New LEPP Algorithms for Quality Polygon and Volume Triangulation: Implementation Issues and Practical Behavior, In Trends in unstructured mesh generation, A. Cannan . Saigal (Eds.), AMD 220 (1997) 1–8.

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

[42] M. C. Rivara, C. Calderón, A. Federov, N. Chrisochoides, Parallel decoupled terminal-edge bisection method for 3D mesh generation, Engineering with Computers, 22 (2006) 536-544.

[43] M.C. Rivara, Lepp-bisection algorithms, applications and mathematical properties, Applied Numerical Mathematics, 59(2009) 2218-2235.

[44] M.C. Rivara, C. Calderon, Lepp terminal centroid method for quality triangulation, Computer-Aided Design 42(2010) 58-66.

[45] I. G. Rosenberg, F. Stenger, A Lower Bound on the Angles of Triangles Constructed by Bisecting the Longest Side, Mathematics of Computation, 29 (1975) 390–395.

[46] J. Ruppert, A Delaunay refinement algorithm for quality 2-dimensional mesh generation, Journal of Algorithms, 18 (1995) 548–585.

[47] W. J. Schroeder, M. S. and Shephard (1990). A combined octree/ Delaunay method for fully automatic 3-D mesh generation, International Journal for Numerical Methods in Engineering, John Wiley, Num 29, pp.37-55, 1990

[48] M.S. Shephard, F. Guerinoni, J.E. Flaherty, R.A. Ludwig, P.L. Baehmann (1988), Finite octree mesh generation for three-dimensional flow analysis, In Numerical Grid Generation in Computational Fluid Mechanics, Pineridge Press, pp.709-718, 1988

[49] M.S. Shephard, J.E. Flaherty, C.L. Bottasso, H. L. de Cougny, C. Ozturan, and M.L. Simone (1997). Parallel automatic adaptive analysis. Parallel Computing 23(9): 1327-1347, 1997.

[50] J.R. Shewchuk (2002), Delaunay refinement algorithms for triangular mesh generation. Computational Geometry. Theory and Applications 22(2002), 21-74.

[51] R. Sibson (1978). Locally equiangular triangulations. Computer Journal, 21(1978) 243–245, 1978

[52] M. Stynes, On Faster Convergence of the Bisection Method for certain Triangles, Mathematics of Computation, 33 (1979) 1195–1202.

[53] M. Stynes, On Faster Convergence of the Bisection Method for all Triangles, Mathematics of Computation, 35 (1980) 1195–1202.

[54] M. A. Weiss, Data structures and algorithm analysis in C++, 3rd edition, Addison Wesley, 2006.

[55] R. Williams, Adaptive parallel meshes with complex geometry, In Numerical Grid Generation in Computational Fluid Dynamics and related Fields, AS Arcilla, J. Hauser, P.R. Eiseman, J.F. Thompson (Eds) Elsevier Science Publishers. (1991) 201-213.

***Response to Reviewers**

Comments on revised paper

Ref. APNUM-D-10-00228

Title: Multithread parallelization of Lepp-bisection algorithms
Applied Numerical Mathematics
Corresponding author: Maria-Cecilia Rivara

The revised paper version considers all the reviewers' comments as enumerated below:

1. The typos and minor errors enumerated by both Reviewers were corrected. We spell-checked the complete manuscript.
2. In section 4, second paragraph, I eliminated the word "parallel" according to the suggestion of Referee #3.
3. The acknowledgments section was completed.

Maria-Cecilia Rivara
mcrivara@dcc.uchile.cl

Comments on revised paper

Ref. APNUM-D-10-00228

Title: Multithread parallelization of Lepp-bisection algorithms
Applied Numerical Mathematics
Corresponding author: Maria-Cecilia Rivara

The revised paper version considers all the reviewers' comments as enumerated below:

1. The presentations of the serial and parallel algorithms have been improved and balanced as suggested by Reviewer #1. The introduction and discussion of the serial algorithm was reduced to 7 pages, while the presentation of the parallel algorithm was extended and improved. This includes a more precise discussion of the synchronization issues considered in the algorithm design, with illustrations that clarify the ideas, as well as more precise implementation details.
2. We have included experiments that illustrate the practical refinement strategies typical in adaptive finite element method as suggested by Reviewer #1. This corresponds to an L-shaped domain with refinement around the re-entrant corner.
3. The performance analysis was improved (as suggested by Reviewer #3) in the following senses:
   - i) A new section 5 discussing the performance measures for parallel computations, with adequate references to parallel literature it is included. Note that in the literature it is known that it is difficult to achieve ideal efficiency in practice, due to the overhead of parallel implementations. We also clarify the important concept of scalability.
   - ii) The paper at present includes two sets of testing problems: (1) refinement of initial triangulations of random points, and (2) refinement of an L-shaped domain around the reentrant corner (asked for Reviewer #1), mimicking adaptive finite element refinement. The algorithm shows better performance for the new (more practical) problem.
   - iii) A more precise discussion on the algorithm performance is presented. This refers to the new section 5 (discussing practical parallel performance) and to the better results of the L-shaped domain. We conclude that the method is reasonably efficient and that shows good scalability: the efficiency is approximately maintained as the number of cores increases.
   - iv) The analysis of the experiments performed over the rectangular region allows to conclude that the random processing of the triangles to be refined should reduce Lepp collisions and improve the algorithm efficiency. This is proposed as a future improvement of the algorithm.

4. The quality of the refined meshes is only presented for one case (Reviewer #1 and Reviewer #3)
5. The execution times, speed up and efficiency statistics throughout the refinement iterations are presented together in one table. The serial fraction was omitted (Reviewer #3).
6. The hardware is fully described. We have used the physical cores, not the virtual ones, included in the computer.
7. The presentations of Lemmas 1, 4 and 5 were improved by considering the comments of Reviewer #1.
8. The typos and minor errors were corrected (Reviewer #1)

Maria-Cecilia Rivara
mcrivara@dcc.uchile.cl