Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Engineering 00 (2015) 000–000
Procedia Engineering
www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Multithread Lepp-bisection algorithms in 2-dimensions

Pedro A. Rodriguez[a], Maria-Cecilia Rivara[b,*]

[a]*Departamento de Sistemas de Información, Universidad del Bio-Bio, Av. Collao 1202, Concepción, 4051381, Chile*
[b]*Departamento de Ciencias de la Computación, Universidad de Chile, Avenida Beauchef 851, Santiago, 8370456, Chile*

Abstract

We discuss and compare three multithread Lepp-bisection algorithms for the refinement of triangulations over multicore architectures. We have obtained an efficient and robust serial implementation, and a partially scalable and efficient multithread method.

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

Keywords: Collisions, efficiency, Lepp, Lepp-bisection, multicore, multithread, thread, speedup.

1. Introduction

Longest edge algorithms for local refinement of triangulations guarantee the construction of refined triangulations that maintain the quality of the input mesh [2,5,7]. Lepp-bisection algorithm is an efficient reformulation of the longest edge algorithm with the following advantages: (a) only local refinement operations are performed which always maintain a conforming mesh (the intersection of pairs of triangles is either a common vertex, or a common edge); (b) the use of the Lepp concept allows to easily design parallel algorithms.

Distributed longest edge based algorithms for the parallel refinement of triangulations have been discussed in the literature. In a review paper for fluid dynamics applications, Williams [10] recommends the use of parallel 4-triangles longest edge algorithm for the refinement of huge triangulations; Jones and Plassmann [4] discuss in detail a parallel distributed 4-triangles refinement algorithm; Castaños and Savage [3] proposed a distributed parallelization of the original longest edge algorithm in 3-dimensions; Rivara et al [6] proposed a simple algorithm for the global refinement of tetrahedral meshes. Balman in [1] proposed an algorithm that uses a 8-tetrahedra longest edge algorithm.

A previous simple multithread Lepp-bisection algorithm (PA2 algorithm in this paper) for two-dimensional triangulations over a simple architecture having 4 cores in one socket is studied in [8]. An algorithm for the refinement of tetrahedral meshes is presented in [9]. Here we study the behavior of three variants of multithread Lepp-bisection algorithm over an Intel E5-2660, 20 cores, 2 sockets, architecture. We present an efficient and robust serial implementation which is used to compute the speedup measures.

* Corresponding author. Tel.: +56-2-2978-4365 ; fax: +56-2-2689-5531.
E-mail address: mcrivara@dcc.uchile.cl (Maria-Cecilia Rivara)

1877-7058
Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

2. Serial Lepp-bisection algorithm

An edge E is a terminal edge in a triangular mesh τ if E is the longest edge of the triangles that share this edge.

In two dimensions, Lepp(t_0), the longest edge propagating path of a triangle t_0 [5,7], is a sequence of N increasing neighbor triangles such that t_i is the neighbor triangle of t_{i-1} by the longest edge of t_{i-1} that allows to find a unique terminal edge either shared by one boundary triangle (t_N) or two terminal triangles (t_{N-1}, t_N). Thus Lepp(t_0) is a 2D submesh that finishes when a terminal edge associated to triangle t_0 is found in the mesh (see Fig. 1 (a)).

Given a triangle t_0, the serial Lepp-bisection algorithm computes Lepp(t_0) and refines the couple of terminal triangles by longest edge bisection of these triangles. The process is repeated until the triangle t_0 is refined (see Fig. 1). This algorithm is formulated as follows [5]:

Serial-Lepp-Bisection-Algorithm(τ, S)
Input: τ, a quality triangulation, and $S \subset \tau$, set of triangles to be refined.
Output: τ_f, a refined and conforming final triangulation.
for (each triangle $t \in S$) **do**
  while (t remains without being bisected) **do**
    Find Lepp(t), terminal triangles t_{N-1}, t_N (triangle t_N can be null if AB is a boundary edge)
    Bisect the terminal triangles
  end while
end for

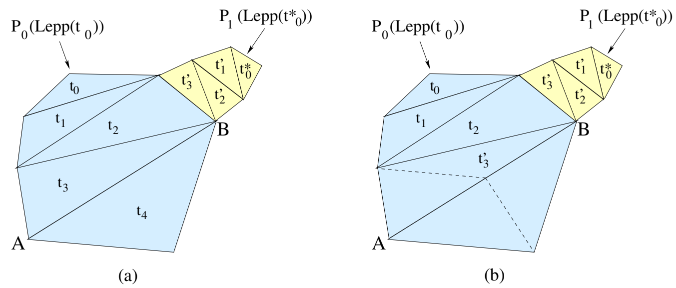

Fig. 1. (a) Lepp(t_0) = $\{t_0, t_1, t_2, t_3, t_4\}$ and Lepp(t_0^*) = $\{t_0^*, t_1', t_2', t_3', t_2, t_3, t_4\}$, AB is a terminal edge; (b) First step of Lepp-bisection refinement of t_0.

3. Multithread Lepp-bisection algorithms

We consider a shared memory multicore computer having p physical cores. To perform the refinement task, each core P_i (i=1,...,p) is in charge of the parallel 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, P_i will pick up another triangle of S to continue the refinement task.

To perform the parallel work, we need to deal with the following synchronization issues [8]: (a) To avoid collisions associated to the parallel processing of triangles whose Lepp sub-meshes overlap; (b) To avoid data structure inconsistencies due to the parallel refinement of adjacent terminal triangles that belong to adjacent (non-overlapping) Lepp sub-meshes. Mutexes are used as synchronization methods to control the access to shared data structure and to avoid inconsistencies. Fig. 1 (a) illustrates the case of overlapping Lepp sub-meshes for triangles t_0 and t_0^* in 2-dimensions, where Lepp(t_0) $\cap$ Lepp(t_0^*) = $\{t_2, t_3, t_4\}$.

We present three versions of multicore Lepp-bisection algorithms:

1. PA1. Marking Lepp Algorithm. If not collision is found, this algorithm locks the triangles of the full Lepp computed and performs refinement of terminal triangles. If collision is detected, the partial Lepp computed (until triangle t_3' in Lepp(t_0^*) in Fig. 1 (a)) is discarded and the thread proceeds to pick up another triangle.

2. PA2. Partial Lepp Storing Algorithm. This multicore algorithm additionally to the work performed by the PA1 algorithm, also stores the partial Lepp computed for being later processed.
3. PA3. Lepp Recomputation Algorithm. This multicore algorithm recomputes Lepp(t) when triangle t is again processed (neither the full Lepp is locked, neither the partial Lepps are stored). Only the terminal triangles and their immediate neighbors are locked.

In the case of the Partial Lepp Storing algorithm (PA2), if the Lepp(t) is successfully computed, the triangles that belong to Lepp(t) are marked as occupied and the terminal triangles are refined. On the contrary if a collision is detected, the partial Lepp(t) is saved into the list L to be later processed (function RefineListOfPartialLepp(L) is invoked at final of the algorithm shown below, to process pending partial Lepps).

1: **PartialLeppStoringAlgorithm(τ, S)**
2: Input: a quality triangulation τ; $S \subset \tau$ set of triangles to be refined
3: Output: a refined and conforming triangulation τ_f
4: Initialize a list L of pending partial Lepps
5: **while** $S \neq \phi$ **do**
6:     A free thread selects a triangle t from S; mark t as occupied.
7:     **while** t remains in mesh **do**
8:         Compute Lepp(t) while non-occupied triangle is found
9:         **if** collision is detected **then**
10:             Insert partial Lepp(t) into L
11:         **else**
12:             Mark all the triangles in Lepp(t) as occupied
13:             **if** neighbor triangles of terminal triangles are unmarked **then**
14:                 Refine terminal triangles and update S.
15:             **end if**
16:         **end if**
17:     **end while**
18: **end while**
19: RefineListOfPartialLepp(L)

The Marking Lepp Algorithm is the same than the previous algorithm without the instructions 4, 9, 10 and 19. For the Lepp Recomputation Algorithm, if two threads P_0 and P_1 refine in parallel triangles t_0, t_0^*, with overlapping Lepps (see Fig. 1 (a)), the thread that access a locked triangle is freed and allowed to refine another triangle. The triangles in the propagation path are not marked as occupied, and the terminal triangles and their immediate neighbors are locked. Thus the refinement of a pair of terminal triangles is not performed if at least one of its neighboring triangles is locked.

LeppRecomputationAlgorithm(τ, S)
Input: a quality triangulation τ; $S \subset \tau$ set of triangles to be refined
Output: a refined and conforming triangulation τ_f
while $S \neq \phi$ **do**
    A free thread P selects a non-locked triangle t from S
    Compute full Lepp(t) (find the terminal triangles)
    if Collision is not detected while computing Lepp(t) **then**
        if Neighbor triangle of terminal triangles is not locked **then**
            Lock terminal triangles and neighbor triangles
            Refine terminal triangles
            Unlock neighbor triangles and update S.
        end if
    end if
end while

A randomization technique was also used in the three algorithms. The random assignment of the triangles of S to the processors contributes to minimize the number of collisions produced by parallel processing of overlapping Lepps.

4. Empirical testing

Table 1 shows the behavior of the serial algorithm for an input mesh of approximately three millions of randomly generated input points. The triangles to be refined are randomly selected at each refinement iteration.

Table 1. Behavior of the serial algorithm. Random data (set of points randomly generated)

♯ Iteration	Mesh Size ♯ Triangles	Triangles to be refined	Final mesh size	♯ Added Triangles	Total time (ms)	Average Time by Triangle (ms)
0	5,999,945	0	0	0	0	0
1	5,999,945	1,199,495	10,789,679	4,789,734	27,955	0.0058
2	10,789,679	2,157,626	18,192,544	7,402,865	14,191	0.0019
3	18,192,544	3,637,167	29,741,177	11,548,633	23,906	0.0021
4	29,741,177	5,950,849	47,694,435	17,953,258	38,014	0.0021

Tables 2, 3 and 4 summarize statistics obtained for the four refinement steps of Table 1 by respectively using Marking Lepp algorithm (PA1), Partial Lepp Storing algorithm (PA2) and Lepp recomputation algorithm (PA3), over an Intel E5-2660 architecture. Figure 2 summarizes the speedup obtained for the three algorithms.

Table 2. Statistics on Marking Lepp algorithm (PA1), random selection. Intel Xeon E5-2660 (2 sockets, 20 cores).

		Execution Time (ms)							Speedup						
♯ Iter	Serial Time	2P	4P	8P	10P	12P	16P	20P	2P	4P	8P	10P	12P	16P	20P
1	27955	18681	10424	6721	6403	5993	5925	6077	1.5	2.7	4.2	4.4	4.7	4.7	4.6
2	70102	45561	28463	16945	17324	15409	15585	17032	1.5	2.5	4.1	4.0	4.5	4.5	4.1
3	136154	93503	54613	36627	32594	32496	32606	37655	1.5	2.5	3.7	4.2	4.2	4.2	3.6
4	240220	165247	95734	62095	64368	61135	62622	72920	1.5	2.5	3.9	3.7	3.9	3.8	3.3

Table 3. Statistics on Partial Lepp Storing algorithm (PA2), random selection. Intel Xeon E5-2660 (2 sockets, 20 cores).

		Execution Time (ms)							Speedup						
♯ Iter	Serial Time	2P	4P	8P	10P	12P	16P	20P	2P	4P	8P	10P	12P	16P	20P
1	27955	19512	11295	7651	7015	6532	6421	7317	1.4	2.8	3.7	3.9	4.3	4.4	3.8
2	70102	50834	30019	20525	19091	17330	17133	17846	1.4	2.3	3.4	3.7	4.1	4.1	3.9
3	136154	101037	60213	38725	34060	33457	35982	38046	1.4	2.3	3.5	4.0	4.1	3.8	3.6
4	240220	186306	104086	61763	58282	58567	64411	69762	1.3	2.3	3.9	4.1	4.1	3.7	3.4

Table 4. Statistics on Lepp Recomputation algorithm (PA3), random selection. Intel Xeon E5-2660 (2 sockets, 20 cores).

		Execution Time (ms)							Speedup						
♯ Iter	Serial Time	2P	4P	8P	10P	12P	16P	20P	2P	4P	8P	10P	12P	16P	20P
1	27955	18286	10622	6907	6671	6253	6310	6685	1.5	2.6	4.1	4.2	4.5	4.4	4.2
2	70102	49107	28083	18178	18036	16546	15000	18858	1.4	2.5	3.9	3.9	4.2	4.7	3.7
3	136154	100154	57974	32402	32471	32475	31240	37156	1.4	2.4	4.2	4.2	4.2	4.4	3.7
4	240220	181162	101169	58420	54109	54047	57430	65744	1.3	2.4	4.1	4.4	4.4	4.2	3.7

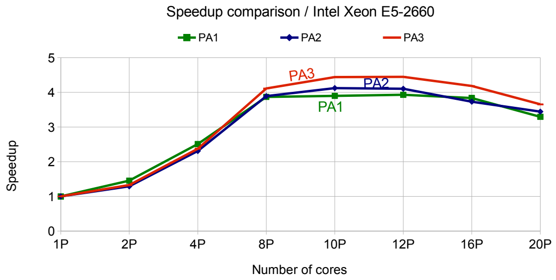

Fig. 2. Speedup behavior comparison. Intel Xeon E5-2660, random selection.

5. Conclusions

An efficient and linear serial implementation was obtained (see Table 1). This requires 0.002 ms for each new triangle introduced in the refined mesh, independently of the mesh size. For each multithread algorithm, the speedup is computed as T_s/T_p where T_s and T_p are the times of the serial and multithread algorithms, respectively.

The analysis of the speedup behavior shows that Lepp recomputation algorithm (PA3) shows a better partially scalable behavior until 12 cores, since this minimizes the number of blocked triangles. The PA1 and PA2 algorithms achieved similar behavior but by below of the algorithm PA3 for 8 to 12 cores.

Note that at each refinement step (Table 1) we have refined approximately 20% of the triangles in the current mesh. Thus an almost global refinement is performed at each iteration which tends to maximize the number of Lepp collisions. A next step in our research will consider dividing the mesh into two parts to be assigned to each socket to reduce inter socket communication.

Acknowledgements

Work partially supported by Departamento de Ciencias de la Computación, Univ. de Chile, Departamento de Sistemas de Información and Research Group GI150115/EF, Univ. del Bio-Bio. We used the supercomputing infrastructure of the NLHPC (ECM-02).

References

[1] M. Balman, In ICPP Workshops, Tetrahedral mesh refinement in distributed environments. pp 497–504. IEEE Computer Society, (2006).
[2] C. Bedregal C, M-C. Rivara, A study on size-optimal longest edge refinement algorithms. In: proceedings of the 21st International Meshing Roundtable, pp. 121–136, (2013).
[3] J. Castaños, J. Savage, Parallel refinement of unstructured meshes. Technical report cs-99-10, Department of Computer Science, Brown University, (1999).
[4] M. Jones, P. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In Proceedings of the Scalable High-Performance Computing Conference IEEE, pages 478–485, (1997).
[5] M-C. Rivara, New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations, International Journal for Numerical Methods in Engineering, vol. 40 (18), pp. 3313–3324, (1997).
[6] M-C. Rivara, C. Calderon, A. Fedorov, N. Chrisochoides. Parallel decoupled terminal-edge bisection method for 3d mesh generation. Eng. Comput. (Lond.), 22(2):111–119, (2006).
[7] M-C. Rivara, Lepp-bisection algorithms, applications and mathematical properties. Appl. Numer. Math., 59(9):2218–2235, (2009).
[8] M-C. Rivara, P. Rodriguez, R. Montenegro, G. Jorquera. Multithread parallelization of lepp-bisection algorithms. Appl. Numer. Math., 62(4):473–488, (2012).
[9] P. A. Rodriguez, M-C. Rivara, Multithread Lepp-Bisection Algorithm for Tetrahedral Meshes. In Proceedings of the 22nd International Meshing Roundtable, Sandia National Laboratories, pp 525-540, (2014).
[10] R. Williams, Adaptive parallel meshes with complex geometry. In Numerical Grid Generation in Computational Fluid Dynamics and related Fields. Elsevier Science Publishers, pages 201–213, (1991).