Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Comment [S1]: Elsevier to update with volume and page numbers.

GPU-Based Parallel Algorithms for Delaunay Mesh Refinement

Sushrut Pande[a]*, Supratim Biswas[b], Amitava De[b]

[a] *University of California, Berkeley, USA.*
[b]*Indian Institute of Technology (IIT) Bombay, India.*

Abstract

This research note reports the design and implementation of a novel, robust parallel algorithm using GPUs (graphics processing units) to generate triangular meshes over complex, unstructured domains or with an unsuitable initial discretization. Algorithms have been implemented using the CUDA architecture and compared with their serial counterparts for several benchmark case studies. Results demonstrate that the GPU-based parallel algorithms developed are several orders of magnitude faster than their serial equivalents.

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

Keywords: Graphics Processing Units(GPUs); Mesh Refinement; Parallel Algorithms; Delaunay Triangulation.

1. Introduction

In addition to an accurate representation of the geometry, meshes developed to approximate physical domains must also be suitable for analysis as distorted geometric shapes directly impact the accuracy of the solution of any governing partial differential equations solved on the domain. To address this problem, several algorithms have been proposed in literature which offer theoretical guarantees and are good in practice. Of these, the Delaunay Triangulation and its variants have been the most popular because they provide good quality control and have

* Corresponding author. Tel.: (1)-510-375-0137.
E-mail address: sushrut2005@gmail.com

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

localized effects. However, for highly complex domains, one may need a mesh with a large number of elements to satisfy these constraints. This process is very time consuming on a sequential CPU and a need exists to minimize it.

Researchers have attempted to parallelize these algorithms by using various efficient domain decomposition strategies. During the last few years, however, a class of multi-core processors called GPUs have become prominent because of their improved performance in the floating-point domain. With the advent of CUDA (Compute Unified Device Architecture), GPUs are increasingly being used in scientific computing. Literature documents scant research on using GPUs for mesh generation or refinement.

The preliminary results presented in this research note focus on developing GPU-based parallel algorithms for generating fast, high-quality unstructured triangular meshes.

2. Overview of Algorithms

There have been several algorithms developed to construct Delaunay Triangulated meshes over complex domains. Most involve refining an existing triangulation by inserting points in the interior of the domain to achieve certain quality measures. Ruppert's Algorithm [1] is popular as it offers theoretical guarantees on quality, gradation and size optimality. It is well documented that triangles with very large or very small angles suffer from large error. Thus, the first step in this algorithm is to identify them. Two measures are suggested to deal with them. For every edge in the mesh, if a point lies interior to its diametral circle it is bisected. This attempts to get rid of short, fat triangles and such edges are termed as "encroached segments". For long, skinny triangles the circumcentre is inserted and re-triangulation is carried out instead. Such triangles are termed as "poor triangles". In order to maintain the Delaunay property of the mesh, an "edge flip" step is necessary to ensure that every common edge between two triangles must have the sum of the two opposite angles being less than or equal to 180^0.

However, every triangular meshing algorithm has one drawback. If the input itself contains small input angles, then no meshing algorithm can generate a triangulation without creating any small angles that are not present in the input. Hence, for certain domains, every algorithm will consist of some poor quality triangles regardless of the termination condition used. A common reason why this phenomenon is observed is that of "mutual encroachment", which occurs when two sub-segments of unequal length are separated by an angle $< 45^0$ and leads to an infinite cycle of encroachment. An attempt has been made to correct for this by constructing concentric shells centered at the shared vertex with their radii being powers of two, and choosing the best balanced split. [2]

CUDA is a parallel architecture which integrates a high-level language such as C/C++ with the parallel memory elements in the GPUs. CUDA has three types of memory—global, shared and constant. Global memory is accessible to all threads, however, it suffers from limited bandwidth leading to very long latencies. Constant memory is fast access although read only. The task is broken down into small units – each unit can be assigned to one "thread". Threads are put into "blocks" which can be synchronized and can efficiently share data through a low latency (albeit limited quantity) shared memory.

As each thread in CUDA solves one independent problem, the parallelization strategy boils down to finding the independent sub-problems in each algorithm and assigning them to one thread. Classification of poor edges or triangles or of edges to be flipped is an embarrassingly data parallel problem. However, new points cannot be inserted in parallel if they belong to a common region. This leads to the definition of a Maximum Independent Set (MIS).

Assume the existence of a point set P, edge set E, and triangle set T.

For encroached segments, define MIS (ES) = {edges $e \in E$ such that e is encroached and $\forall$ e1, e2 $\in$ MIS (ES) there does not exist any triangle $t \in T$ to which both e1 and e2 belong}

For edge flipping, define MIS (EF) = {edges $e \in E$ such that e violates Delaunay property and $\forall$ e1, e2 $\in$ MIS (EF) there does not exist any triangle $t \in T$ to which both e1 and e2 belong}

For poor triangles, first define the cavity set of a triangle = {$t \in T$ such that t is of "poor quality" and all $t1 \in T$

such that circum center of t lies inside circum circle of t1}. Then, MIS (PT) = {triangles $t \in T$ such $\forall$ t1, t2 $\in$ MIS (PT) there does not exist any triangle $t \in T$ common to both cavity (t1) and cavity (t2)}

We attempted to devise a parallel algorithm to construct the MIS, followed by insertion of points belonging to the MIS in parallel. One possibility was to assign weights to each geometric entity (edge/triangle) and if the entity had a greater weight than its neighbours then to add it in parallel to the MIS. However, it was observed that this strategy or its variants led to the creation of a sub-optimal MIS whose cardinality was much smaller than the optimum.

The complete algorithm proposed and implemented in this work is given below:

Input: P, E, Initial Delaunay Triangulation T(P)

while (an edge is encroached or at least one triangle is poor or average quality $\leq$ threshold)
in parallel for each edge e $\in$ E
check whether e is encroached and if so, label it.
Serially find MIS (ES) of encroached edges es.
In parallel $\forall$ edges es $\in$ MIS (ES)
P= P $\cup$ {midpoint (es)}
Update affected triangles and edges.
Apply Parallel Edge-Flipping.
If (no edge is encroached)
In parallel for each triangle t $\in$ T
Check whether t is of "poor quality" and if so, label it.
Serially find MIS (PT) of poor triangles pt.
in parallel $\forall$ triangles pt $\in$ MIS (PT)
if (circum center (pt) encroaches an edge e $\in$ E)
mark e as encroached for next iteration.
Else if (circum center (pt) does not encroach any edge e $\in$ E)
P= P $\cup$ {circum center (t)}
Update affected triangles and edges.
Apply Parallel Edge-Flipping.

Output: High-quality Delaunay Triangulation T(P) of P, E.

The algorithm for parallel edge flipping is as follows:

Input: P, E, Triangulation T(P)

while (at least one edge is locally non-Delaunay)
in parallel for each edge e $\in$ E
determine t1 and t2 $\in$ T such that e $\in$ t1 && e $\in$ t2
calculate α and γ, angles opposite to edge for t1 and t2
if ($\alpha + \gamma > 180^0$)
mark edge for flipping
serially construct MIS (EF).
In parallel for each edge ef $\in$ MIS(EF)
swap coordinates of points for "flipped" edge and update t1 and t2 appropriately
Output: Delaunay Triangulation DT (P)

To deal with mutual encroachment, the Concentric Shell Algorithm was also parallelized. The strategy is as follows:

Input: Set of encroached segments ES, P, E, Triangulation T, MIS (ES)
In parallel for each encroached segment es =(p,q) $\in$ MIS (ES) where p, q $\in$ P
find e $\in$ E such that (p,r)=e or (q,r)=e
if (angle between es & e < 45^0)
choose newpoint such that length of one of the two new sub-segments formed is 2^i where i is an integer and the length of the sub-segments is nearly equal
else if (angle between es and all e >= 45^0)
newpoint = midpoint (es)
P= P $\cup$ newpoint
Update affected triangles & edges.
Apply Parallel Edge-Flipping.

A crucial detail in these algorithms is how to optimally use the various types of memory available. The strategy implemented loaded the data structures into the global memory. The global data structure was loaded block-wise so that it fit into the shared memory, followed by evaluation of whether the edge or triangle was poor. This data was then copied back to the global data structure. The MIS was then constructed serially. The global data structures were modified dynamically in parallel. In theory, there is data transfer happening and potential overheads are high. In practice however this is offset by the speed of access of the shared memory over the global memory.

3. Implementation and Testing

The algorithms were implemented on a 2.4 GHz CPU running Linux with 4 GB RAM and the NVIDIA TESLA C2070 GPU. Double precision calculations were carried out on the GPU. The algorithms were run for several benchmark case studies from literature. The stopping criterion used was either no poor triangles left, or an average quality of 0.8. This is expected to give fairly accurate results in engineering applications using techniques such as the Finite Element Method (FEM).

The serial and parallel codes were run on 6 typical benchmark case studies. Table 1 and 2 display the results, while Fig 1 displays the meshes output by the algorithm with the concentric shell modification.

Table 1. Results for the Parallel Algorithm Without Concentric Shell Modification

	Avg Initial Quality	Initial Triangles	Final Triangles	Serial (CPU) Time (ms)	Parallel (GPU) Time (ms)
Plate with Hole	0.415	12	4847	6550	24.096
Airfoil	0.0824	65	22690	145800	15.025
Africa	0.4251	54	2097	674	1.574
Guitar	0.5905	152	350	25	0.374
Key	0.5608	55	460	37	0.736
Lake Superior	0.3405	322	7734	9700	1.621

Table 2. Results for the Parallel Algorithm With Concentric Shell Modification

	Avg Initial Quality	Initial Triangles	Final No of Triangles	Serial(CPU) Time (ms)	Parallel(GPU) Time (ms)
Plate with Hole	0.415	12	3560	2850	3.41
Airfoil	0.0824	65	20165	99825	12.287
Africa	0.4251	54	286	25	0.552
Guitar	0.5905	152	253	12	0.188
Key	0.5608	55	3459	2025	2.034
Lake Superior	0.3405	322	8286	12913	1.965

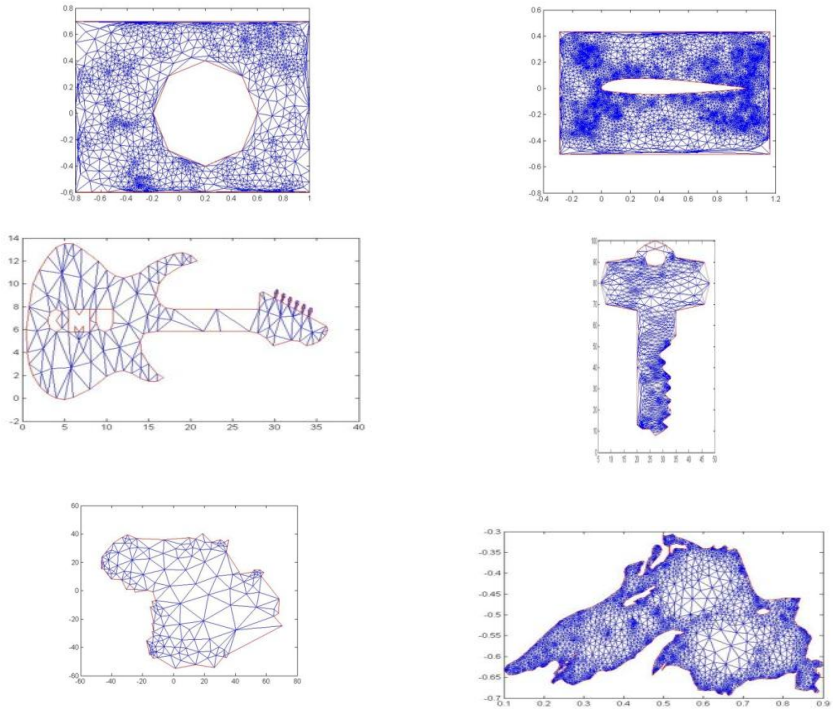

Fig 1: Meshes (Top Left: Plate with a Hole, Top Right: Airfoil, Middle Left: Guitar, Middle Right: Key, Bottom Left: Africa, Bottom Right: Lake Superior)

4. Conclusions and Future Directions

For both algorithms, the parallel times show a very significant speed up over the serial times to generate a mesh with the same number of elements. This makes these algorithms an attractive proposition to use for generating a large number of elements for certain extremely complicated domains. In general, the Concentric Shell Algorithm is observed to have a superior performance for both serial and parallel cases. The high variability in computation times for convergence seen in Tables 1 and 2 across various domains is a direct function of the quality of the initial triangulation. The initial triangulation is taken to be the Delaunay Triangulation of the boundary points of domain D.

This work will be extended further for domains with extremely small input angles, with provision of an automatic handle to control mesh gradation.

5. References

[1] J Ruppert, "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation", Journal of Algorithms, Vol. 18, pp 548-585, 1995.

[2] J.R. Shewchuk, "Delaunay Refinement Algorithms for Triangular Mesh Generation", Computational Geometry: Theory and Applications **22**(1–3):21–74, May 2002