Efficient Core Utilization in a Hybrid Parallel Delaunay Meshing Algorithm on Distributed-Memory Cluster

Daming Feng
Old Dominion University

Andrey N. Chernikov
Old Dominion University

Nikos P. Chrisochoides
Old Dominion University

Original Publication Citation

Feng, D., Chernikov, A. N., & Chrisochoides, N. P. (2017). *Efficient Core Utilization in a hybrid parallel Delaunay meshing algorithm on distributed-memory cluster*. Paper presented at the 26th International Meshing Roundtable, IMR26, Barcelona, Spain, September 18-21, 2017. https://imr.sandia.gov/papers/imr26/2024_imr26RN_Feng.pdf

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2017) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain

Efficient Core Utilization in a Hybrid Parallel Delaunay Meshing Algorithm on Distributed-Memory Cluster

Daming Feng[a], Andrey N. Chernikov[a], Nikos P. Chrisochoides[a,*]

[a]*Computer Science Department, Old Dominion University, Norfolk, VA 23508, USA*

Abstract

Most of the current supercomputer architectures consist of clusters of nodes that are used by many clients (users). A user wants his/her job submitted in the job queue to be scheduled promptly. However, the resource sharing and job scheduling policies that are used in the scheduling system to manage the jobs are usually beyond the control of users. Therefore, in order to reduce the waiting time of their jobs, it is becoming more and more crucial for the users to consider how to implement the algorithms that are suitable to the system scheduling policies and are able to effectively and efficiently utilize the available resources of the supercomputers. We proposed a hybrid MPI+Threads parallel mesh generation algorithm on distributed memory clusters with efficient core utilization. The algorithm takes the system scheduling information into account and is able to utilize the nodes that have been partially occupied by the jobs of other users. The experimental results demonstrated that the algorithm is effective and efficient to utilize available cores, which reduces the waiting time of the algorithm in the system job scheduling queue. It is up to 12.74 times faster than the traditional implementation without efficient core utilization when a mesh with 2.58 billion elements is created for 400 cores.

Keywords: Hybrid Programming; Parallel Mesh Generation; Two-Level Parallelization; Core Utilization

1. Introduction

Delaunay mesh refinement is a popular technique for generating triangular and tetrahedral meshes for use in finite element analysis and interpolation in various numeric computing areas because it can mathematically guarantee the quality of the mesh [1–4]. Scalable parallel mesh generation algorithms with quality and fidelity guarantees are demanding for the real world (bio-)engineering and medical applications. The quality of mesh refers to the quality of each element in the mesh which is usually measured in terms of its circumradius-to-shortest edge ratio (radius-edge ratio for short) and (dihedral) angle bound. Normally, an element is regarded as a good element when the radius-edge ratio is small [5–7] and the angles are in a reasonable range [8,9]. The fidelity is understood as how well the boundary of the created mesh represents the boundary (surface) of the real object. A mesh has good fidelity when its boundary is

*Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000.
E-mail address: {dfeng, achernik, nikos}@cs.odu.edu

a correct topological and geometrical representation of the real surface of the object. The hybrid MPI+Threads parallel mesh generation algorithm proposed in this note is a Delaunay algorithm that conforms to all of these requirements.

Most of the current supercomputer architectures consist of clusters of nodes, each of which contains multiple cores that share the in-node memory. A hybrid parallel programming model, which utilizes message passing interface (MPI) for the parallelization among distributed memory nodes and uses thread-based libraries (Pthread or OpenMP) to exploit the parallelization within the shared memory of a node, seems to be an excellent solution to take advantage of the resources of such architectures. This leads to a trend to write hybrid parallel programs that involve both process level and thread level parallelization. However, designing new hybrid parallel mesh generation algorithms or modifying existing algorithms that are suitable to the supercomputer architectures brings new challenges because of the data dependencies and the irregular and unpredictable behavior of mesh refinement. In addition, the high-performance computing (HPC) cluster is always shared by many users. The jobs of these users are always scheduled and managed by the grid computing computer cluster software system (also known as a batch-queuing system), such as Sun Grid Engine (SGE) and Slurm. This leads to the situation that some cores of a node may have been occupied by the jobs of other users due to the management and scheduling.

In the previous work [10], we proposed a parallel Delaunay image-to-mesh conversion algorithm which is the first three-dimensional hybrid MPI+Threads parallel meshing algorithm that involves both distributed memory parallelization and shared memory parallelization. However, the execution of the algorithm has very high resource requirements. It only works on clusters with homogeneous nodes and needs exclusive accesses of these nodes. In other words, it cannot execute on the nodes which have been partially occupied by the jobs of other users. As a result, the waiting time of the algorithm in the queue is large in order to get the resources it needs.

In this note, we proposed a hybrid MPI+Threads parallel mesh generation algorithm based on the previous work [10]. The major contributions are listed as follows.

- It creates meshes with quality and fidelity guarantees.
- It overcomes the limitations of the previous parallel meshing algorithm [10] which only works on clusters with homogeneous nodes and needs exclusive accesses of these nodes. As a result, it is up to 12.74 times faster than the previous algorithm without efficient core utilization for 400 cores and 2.58 billion element mesh as illustrated in Table 1.
- It takes the realtime system scheduling information as input parameter in the implementation, which always guarantees that the algorithm is able to utilize the available cores to do the two level parallel mesh refinement.
- It contributes to the understanding of the challenges of adaptive, irregular and realtime-computing applications that involve both distributed memory parallelization and shared memory parallelization on supercomputers.

The rest of the note is organized as follows. Section 2 describes the main idea of the hybrid MPI+Threads algorithm with efficient resource utilization; Section 3 presents preliminary experimental results and the quality analysis of meshes that our meshing algorithm creates; Section 4 concludes the note.

2. Algorithm

The algorithm has two stages: the pre-processing stage and the two level parallel refinement stage. In the pre-processing stage, the algorithm firstly uses the scheduling information from the system to create processes and threads. The scheduling information includes the hostname, also can be called node name, and the number of available cores (and their IDs) of each node. Based on the information, the algorithm creates a MPI process on each node. Each process creates threads based on the number of available cores of each node. The number of threads of each process (master and worker threads) is equal to the number of available cores of each node and is different from each other. The node with maximum number of available cores is chosen as the master node and the process running on it is the master process. Then, a coarse mesh is constructed by the multiple threads of the master process, and the whole region (the bounding box of the input image) is decomposed into subregions. Next, the bad quality elements of the coarse mesh is assigned into subregions based on the coordinates of circumcenters of the elements. If the circumcenter of an element is inside a subregion, the element is assigned to the subregion.

The second stage is the two level parallel mesh refinement stage. The algorithm simultaneously explores process-level parallelization and thread-level parallelization: inter-node communication using MPI and inter-core communication inside one node using threads. In the process level parallelization, the master process uses a task scheduler to schedule the tasks (subregions) to worker processes through MPI communication. Each subregion is considered as task and the submesh inside the subregion is the data. The worker processes communicate with the master process and with each other for task request and data migration. In the thread level parallelization, the process of each computing node creates multiple threads that follow the refinement rules of the Parallel Optimistic Delaunay Mesh generation algorithm (PODM) [11,12] to refine the bad elements of each subregion in parallel by inserting multiple points simultaneously. The parallel mesh refinement terminates until all the bad elements in the coarse mesh are eliminated according to the user-spedified quality criteria.

The MPI communication and local shared memory mesh refinement is separated in order to overlap the communication and computation in this two level parallelization model. The master thread of each process that runs on each computing node initializes the MPI environment, creates worker threads based on the number of available cores of each computing node. It communicates with the master thread of other processes that run on other nodes for data movement and task requests. The worker threads of each process do not make MPI calls and are only responsible for the local mesh refinement in the shared memory of each node. One thing we should mention is that the number of threads of each process should be at least two (a master thread for communication and a worker thread for mesh refinement) in order to separate the MPI communication and local mesh refinement. Therefore, we set a threshold for the number of available cores and discard the node that has only one available core.

3. Preliminary Experimental Results

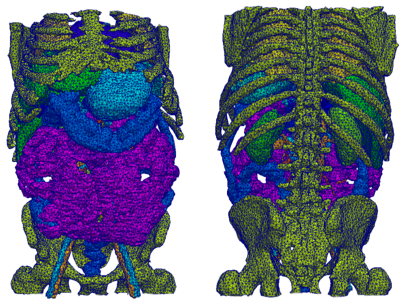
We conducted a set of experiments to test the performance of the algorithm. The experimental platform is the Turing cluster computing system at High Performance Computing Center of Old Dominion University. The Turing cluster contains 190 multi-core compute nodes each containing between 16, 20 and 32 cores and 128 Gb of RAM. We used the 3D multi-tissue image, the CT abdominal atlas obtained from IRCAD Laparoscopic Center [13], as input in the experiments. We use homogeneous algorithm to represent the previous algorithm which only works on cluster with homogeneous nodes and the word heterogeneous algorithm to represent the algorithm we proposed in this research note.

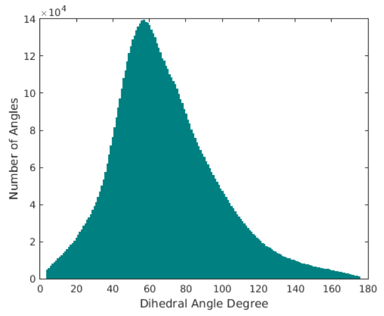
3.1. Total Time Comparison

The total time of the algorithm in the experiments has two parts: the execution time and the waiting time. The execution time is the time that the job executes on the cluster. The waiting time is the time that the job spends on waiting in the SGE job queue to get the resources from the scheduling system. It depends on the job scheduling of the system and the resources that a job requests, such as the number of cores and the amount of memory. In the experiments, the problem size (i.e., the number of elements created) in the experiments increases proportionally to the

Table 1: The performance comparison of the heterogeneous algorithm and the homogeneous algorithm. The input is abdominal atlas.

Cores	Hosts		Elements	Execution Time (s)		Waiting Time (s)		Total Time (s)		Total Time Ratio
	homo	hetero	(million)	homo	hetero	homo	hetero	homo	hetero	homo:hetero
1	1	1	6.64	64.70	64.35	0	0	64.70	64.35	1:1
20	1	1	133.93	76.47	77.27	0	0	76.47	77.27	1:1
40	2	3	261.54	102.63	107.09	0	0	102.63	107.09	0.96:1
60	3	4	390.61	110.35	115.22	0	0	110.35	115.22	0.96:1
80	4	6	520.31	115.02	121.44	0	0	115.02	121.44	0.95:1
100	5	10	650.16	125.96	204.79	0	0	125.96	204.79	0.62:1
200	10	22	1292.20	139.33	249.87	731	0	870.33	249.87	3.48:1
300	15	26	1937.00	131.53	231.44	3674	181	3805.53	412.44	9.23:1
400	20	36	2577.91	127.65	280.35	18622	1191	18749.65	1471.35	12.74:1


(a)



(b)

Fig. 1: (a) The final output mesh created by the algorithm. (b) Dihedral angle distribution of the final mesh.

number of cores and the number of elements per core remains approximately constant. The comparison of the total time of the heterogeneous algorithm and the previous homogeneous algorithm is demonstrated in Table 1.

The execution time of the heterogeneous algorithm is a bit longer than that of the homogeneous algorithm because more computing nodes involve in the communication and the execution of the job is disturbed by the jobs of other users running on the same node. As shown in Table 1, the execution time of the homogeneous algorithm of 100 cores is 125.95 seconds and five 20-core node are used while the execution time is 204.79 seconds of the heterogeneous algorithm and ten partially occupied nodes are used in order to get 100 cores. However, the heterogeneous algorithm reduces the waiting time which is the dominating part of the total time when the number of cores is large. As a consequence, the total time of the algorithm is much smaller than that of the homogeneous algorithm. As shown in Table 1, the waiting time for the homogeneous algorithm of 400 cores is about 18622 seconds in order to get the computing resources (the homogeneous nodes with exclusive accesses). In contrast, it only takes about 1191 seconds for the new algorithm waiting in the queue before it executes. As a result, the total time of the previous algorithm is 12.74 times more than that of the heterogeneous algorithm for the experiments of 400 cores.

3.2. *Output Mesh and Quality Analysis*

We analyzed the quality of the meshes created by heterogeneous algorithm. The quality of mesh refers to the quality of each element in the mesh which is measured by Delaunay methods in terms of its circumradius-to-shortest edge ratio (radius-edge ratio for short) and (dihedral) angle bound. The radius-edge ratio of a tetrahedron is defined as the ratio of its circumradius to the length of its shortest edge. The radius-edge ratio of the created mesh is theoretically guaranteed by the Delaunay refinement method and the actual edge ratio bound is less than 2. Because of the potential nearly flat tetrahedra, a more useful measure is the dihedral angle. We show the output mesh and the dihedral angle distribution of the final mesh created by the heterogeneous algorithm in Fig. 1a and Fig. 1b. We compared the quality of meshes that were created by our method with the quality of meshes that were created by the homogeneous algorithm on the multi-material 3D input image (abdominal atlas) as shown in Table 2. The results illustrate that the heterogeneous algorithm is able to generate meshes with the same quality guarantees as the homogeneous algotithm.

In summary, the hybrid MPI+Threads parallel meshing algorithm proposed in this research note (i) creates meshes with quality guarantees and (ii) is more flexible to take advantage of the available computing resources compared to the previous algorithm which only works on homogeneous nodes and needs exclusive accesses to these nodes.

Table 2: Mesh quality comparison of the two algorithms. The input image is abdominal atlas.

	number of elements	number of vertices	edge-radius ratio bound	min dihedral angle	max dihedral andle
Hetero	1,345,237	244,043	2	4.98°	174.34°
Homo	1,352,737	244,027	2	4.78°	174.47°

4. Conclusion

In this research note, we extended the previous hybrid MPI+Threads parallel mesh generation algorithm [10] on heterogeneous clusters. It simultaneously explores process-level parallelization and thread-level parallelization: inter-node communication using MPI and inter-core communication inside one node using threads. The algorithm overcomes the limitations of the previous algorithm which only works on clusters with homogeneous nodes and needs exclusive accesses of these nodes. It utilizes the nodes which have been partially occupied by the jobs of other users. In other words, it is able to utilize the available cores to do the parallel mesh refinement in order to reduce the waiting time in the queue.

We compared the total time of the algorithm with the previous homogeneous algorithm. The preliminary experimental results illustrated that the execution time of the algorithm is a bit longer than that of the previous homogeneous algorithm. However, it is able to take advantage of the available resources and reduce the waiting time in the queue dramatically. As a sequence, it is up to 12.74 times faster compared to the previous algorithm when a mesh with 2.58 billion elements is created for 400 cores. We also compared the quality of the final mesh with the mesh created by the previous algotithm. The results illustrate that the algorithm creates mesh with quality guarantees.

As we described above, the number of available threads (cores) of each process (computing node) are different from each other in the algorithm, which makes the computation ability of each process different from each other. Therefore, the future work is to explore dynamic load balance among processes with various computation ability to improve the performance of the algorithm on heterogeneous cluster.

Acknowledgements

This work is funded (in part) by NSF grants CCF-1439079 and CCF-1139864 and by the Richard T. Cheng Endowment. We thank the systems group in the High Performance Computing Center of Old Dominion University for their great help. We especially thank Min Dong and Terry R. Stilwell for many insightful discussions. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NSF.

References

[1] S.-W. Cheng, T. K. Dey, J. Shewchuk, Delaunay Mesh Generation, CRC Press, 2012.

[2] P.-L. George, H. Borouchaki, Delaunay Triangulation and Meshing. Application to Finite Elements, HERMES, 1998.

[3] A. Chernikov, N. Chrisochoides, Generalized insertion region guides for Delaunay mesh refinement, SIAM Journal on Scientific Computing 34 (2012) A1333–A1350.

[4] P. Foteinos, A. Chernikov, N. Chrisochoides, Guaranteed quality tetrahedral Delaunay meshing for medical images, Computational Geometry: Theory and Applications 47 (2014) 539–562.

[5] J. R. Shewchuk, Tetrahedral mesh generation by Delaunay refinement, in: Proceedings of the 14th ACM Symposium on Computational Geometry, 1998, pp. 86–95.

[6] H. Si, Tetgen: A quality tetrahedral mesh generator and a 3D Delaunay triangulator, `http://wias-berlin.de/software/tetgen/`, 2013.

[7] CGAL, computational geometry algorithms library, `http://www.cgal.org`, 2014.

[8] A. N. Chernikov, N. P. Chrisochoides, Multitissue tetrahedral image-to-mesh conversion with guaranteed quality and fidelity, SIAM Journal on Scientific Computing 33 (2011) 3491–3508.

[9] J. Bronson, J. Levine, R. Whitaker, Lattice cleaving: A multimaterial tetrahedral meshing algorithm with guarantees, Visualization and Computer Graphics, IEEE Transactions on 20 (2014) 223–237.

[10] D. Feng, A. Chernikov, N. Chrisochoides, A hybrid parallel delaunay image-to-mesh conversion algorithm scalable on distributed-memory clusters, in: International Meshing Roundtable, 2016.

[11] P. Foteinos, N. Chrisochoides, High quality real-time image-to-mesh conversion for finite element simulations, in: ACM International Conference on Supercomputing, ACM, 2013, pp. 233–242.

[12] P. Foteinos, N. Chrisochoides, High quality real-time image-to-mesh conversion for finite element simulations, Journal on Parallel and Distributed Computing 74 (2014) 2123–2140.

[13] Ircad Laparoscopic Center, `http://www.ircad.fr/softwares/3Dircadb/3Dircadb2`, 2013.