Two-level Locality-Aware Parallel Delaunay Image-to-Mesh Conversion

Daming Feng, Andrey N. Chernikov, Nikos P. Chrisochoides*

Department of Computer Science, Old Dominion University, Norfolk, Virginia 23569, USA

Abstract

In this paper, we propose a three dimensional two-level Locality-Aware Parallel Delaunay image-to-mesh conversion algorithm (LAPD). The algorithm exploits two levels of parallelism at different granularities: coarse-grain parallelism at the region level (which is mapped to a node with multiple cores) and medium-grain parallelism at the cavity level (which is mapped to a single core). We employ a data locality-aware mesh refinement process to reduce the latency caused by the remote memory access. We evaluated LAPD on Blacklight, a cache-coherent NUMA distributed shared memory (DSM) machine in the Pittsburgh Supercomputing Center, and observed a weak scaling efficiency of almost 70% for roughly 200 cores, compared to only 30% for the previous algorithm, Parallel Optimistic Mesh Generation algorithm (PODM).

Keywords: Parallel Mesh Generation, Parallel Computing, Locality-Aware, Image-to-Mesh Conversion, High Performance Computing

1. Introduction

Mesh generation is a critical component for many (bio-)engineering applications. Delaunay mesh refinement is one of the most popular techniques for generating triangular and tetrahedral meshes for use in finite element analysis

*Corresponding author

Email addresses: dfeng@cs.odu.edu (Daming Feng), achernik@cs.odu.edu (Andrey N. Chernikov), nikos@cs.odu.edu (Nikos P. Chrisochoides)

and interpolation in various numeric computing areas because it can mathematically guarantee the quality of the mesh [1, 2, 3, 4]. The parallelization of Delaunay mesh refinement codes can be achieved by inserting multiple points simultaneously [5, 6, 7, 8, 9, 10]. The mesh generation of the ultra-high-resolution three dimensional images, such as BigBrain [11], brings new challenges to parallel mesh generation considering the required memory space and the execution time [12]. The NASA CFD vision 2030 study [13] reports that mesh generation is the critical bottleneck in the computational fluid dynamics (CFD) field. However, most current mesh generation algorithms are desktop-based, either sequential or parallel, developed for a small number of cores. Such mesh generation algorithms, when run on supercomputers, are either conservative in leveraging available concurrency [5, 7, 14] or require the solution of the domain decomposition which is still an open problem for three dimensional domains [6, 15]. There is no doubt that the scalability of mesh generation algorithms will continue to be critical for many (bio-)engineering applications, such as CFD simulations [16] and image discretization in bioinformatics [17].

In order to solve the problem and create high quality meshes of the desired resolution, it is necessary to apply a supercomputer-based parallel mesh generation algorithm which scales well on modern non-uniform (i.e., distributed) memory machines. In this paper, we describe a three dimensional two-level locality-aware parallel Delaunay image-to-mesh conversion algorithm that (1) applies a three dimensional *data decomposition* instead of *domain decomposition* according to the circum-centers of tetrahedra (2) employs two-level parallelism and a data locality optimization scheme to reduce the communication overhead caused by a large number of remote memory accesses. The algorithm is more aggressive in leveraging concurrency compared to other parallel Delaunay refinement algorithms. It shows good scalability for up to 200 cores on a NUMA machine. The long term goal is to exploit the performance potential of modern supercomputers and to deliver sufficient scalability for applications that require exascale computing [12]. We plan to achieve this by leveraging concurrency at different granularity levels using hierarchical hybrid mesh generation algo-

rithms. The algorithm we present in this paper is the first step as its good performance indicates that the blueprint of our long term goal is feasible and within our reach.

The previous parallel unstructured mesh generation and refinement algorithms are not suitable for NUMA DSM architecture supercomputers. These algorithms rely on irregular communication patterns and lack data locality due to a large number of remote memory accesses. Oliker and Biswas [18] concluded that the performance of unstructured mesh refinement deteriorates for some cases on just a 4-core cc-NUMA architecture. Chowdhury et al. [19] presented multicore-oblivious algorithms and run-time scheduler for several fundamental problems including matrix transposition, FFT, sorting, Gaussian elimination, and others, however, mesh generation has not been addressed. In this paper, we conduct an application-level exploration that will help design general locality-aware mesh generation algorithms and supporting run-time systems. In the previous work [8, 9], we implemented a parallel Image-to-Mesh (I2M) refinement algorithm, Parallel Optimistic Delaunay Mesh generator (PODM), which works on multi-label segmented three dimensional images. The algorithm works well for a low core count (less than 128 cores). However, it exhibits considerable performance deterioration for a higher core count (144 cores or more) because of the intensive and multi-hop communication. The communication not only causes contention in the communication links and makes the available bandwidth decrease, but also causes high latency due to remote memory accesses.

The locality-aware meshing algorithm LAPD described in this paper leverages concurrency at two granularity levels, and matches these two levels to the non-uniform memory architecture to increase the data-locality. In the coarse-grain level parallelism, a node that contains multiple cores is mapped to a subregion, and multiple subregions can be refined in parallel. In the medium-grain level, the algorithm takes advantage of the previous parallel image-to-mesh conversion approach, PODM, to insert or delete multiple points of a subregion in parallel. The experimental evaluation was performed on Blacklight, a cache-coherent NUMA shared memory machine in the Pittsburgh Supercomputing

Center. We observed a weak scaling efficiency of more than 67% on 192 cores, compared to only about 30% of that of PODM [8, 9]. Fig. 1 shows an example mesh created by the algorithm.

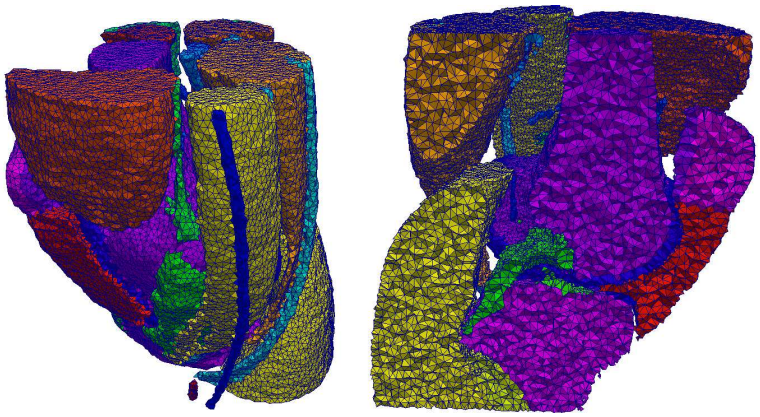


Figure 1: A mesh created by LAPD algorithm. The input image is the knee atlas obtained from Brigham & Women's Hospital Surgical Planning Laboratory [20]. The left figure demostrates the fidelity of the mesh. The boundaries of all tissues are well recovered. The cut-through figure in the right shows the quality of the mesh.

The rest of the paper is organized as follows: Section 2 presents the background of Delaunay mesh refinement and reviews the related prior work; Section 3 describes the main idea of PODM algorithm and analyzes the reasons of its performance deterioration on distributed shared memory architecture. Section 4 describes the two-level locality-aware parallel Delaunay mesh generation algorithm (LAPD); Section 5 outlines the analysis of the experimental results of our approach; Section 6 concludes the paper.

2. Background and Related Work

Delaunay refinement algorithms work by inserting additional (often called Steiner) points into an existing mesh to improve the quality of the elements. There are two kinds of Delaunay refinement algorithms. In the PLC-based algorithms, the surface of the objects is represented as a Piecewise Linear Complex (PLC) [21, 22]. The limitation of the PLC-based method is that the quality of

the input PLC affects the quality of the final volume mesh. The Isosurface-based algorithms [23, 4] recover the isosurface of the object in the image and mesh the volume simultaneously. These methods do not suffer from the small input angle constraint introduced by the given PLCs or the initial conversion to PLCs. In the case of bio-engineering applications, since we start with images, it is better to avoid the initial generation of the PLC and immediately proceed with volume mesh generation. The LAPD method proposed in this paper is an isosurface-based algorithm. It directly takes segmented images as input, and then recovers the surface and meshes the volume simultaneously.

The basic operation of Delaunay refinement is the insertion and deletion of points, which then leads to the removal of poor quality tetrahedra and of their adjacent tetrahedra from the mesh and the insertion of new tetrahedra. The new tetrahedra may or may not be of poor quality, and therefore may or may not require further point insertions. It is proven [22, 24] that the algorithm terminates after having eliminated all poor quality tetrahedra, and in addition, the termination does not depend on the order of processing of poor quality tetrahedra, even though the structure of the final meshes may vary. The insertion of a point is often implemented according to the well-known Bowyer-Watson kernel [25, 26, 27]. The parallel insertion of points by different threads needs to be synchronized. The problem of parallel Delaunay triangulation of a specified and fixed point set has been solved by Blelloch et al. [28]. They describe a divide-and-conquer projection-based algorithm for constructing Delaunay triangulations of pre-defined point sets in parallel.

One approach, Parallel Delaunay Refinement (PDR) [7, 14], is based on a theoretically proven method to choose the points for the insertion. This approach is based on the analysis of the dependencies between the inserted points and requires neither the runtime checks nor the geometry decomposition. The work was then extended to three dimensions [5]. PDR works as a wrapper code around the open source sequential mesh generators Triangle [29](2D) and Tetgen [30](3D). It can guarantee the independence of inserted points and thus avoid runtime data dependencies during the parallel refinement process.

Linardakis [15] presented a two dimensional Parallel Delaunay Domain Decoupling (PD^3) method. The PD^3 method is based on the idea of decoupling the individual subdomains so that they can be meshed independently with zero communication and synchronization by reusing the existing state-of-the-art sequential mesh generation codes. In order to eliminate the communication and synchronization costs, a proper decomposition that can decouple the mesh is required. However, the construction of such a decomposition is an equally challenging problem because its solution is based on Medial Axis [31, 32] which is very expensive and difficult to construct (even to approximate) for complex three dimensional geometries.

An idea of updating partition boundaries when inserted points happen to be close to them was presented [33] and extended [6] as a Parallel Constrained Delaunay Meshing (PCDM) algorithm. In PCDM, the edges on the boundaries of submeshes are fixed (constrained), and if a new point encroaches upon a constrained edge, another point is inserted in the middle of this edge instead. As a result, a split message is sent to the neighboring core, notifying that it also has to insert the midpoint of the shared edge. This approach requires the construction of the separators that will not compromise the quality of the final mesh, which is still an open problem for three dimensional domains.

The previous parallel mesh generation method, PODM [8, 9], is a tightly-coupled parallel optimistic Delaunay mesh generation algorithm. PODM suffers from communication overhead caused by a large number of remote memory accesses, and its performance deteriorates for a core count beyond 144 because of the network congestion caused by the communication among threads. The best weak scaling efficiency for 176 continuous cores is only about 49% on Blacklight.

Besides the Delaunay-based algorithms, a number of other parallel mesh generation algorithms have been published. De Cougny, Shephard and Ozturan [34] base the parallel mesh construction on an underlying octree. Lohner and Cebral [35], and Ito et al. [36] developed parallel advancing front schemes. Globisch [37, 38] presented a parallel mesh generator which uses a sequential frontier algorithm. A detailed review of many more methods appears in [39].

3. Remote Memory Access of PODM on NUMA Architecture

In distributed shared memory (DSM) systems, memory is physically distributed while it is accessible to and shared by all cores. However, a memory block is physically located at various distances (hops) from the cores. As a result, the memory access times vary and depend on the distances (hops) from a core to a memory block. When the parallel applications are running on such NUMA machines, a thread running on a core might access its own local memory or its non-local (remote) memory (memory local to another node).

The experimental platform, Blacklight [40], is a cc-NUMA shared-memory system consisting of 256 blades. Each blade holds 2 Intel Xeon X7560 (Nehalem) eight-core CPUs, for a total of 4096 cores across the whole machine. The 16 cores on each blade share 128 Gbytes of local memory. The dashline rectangle in Fig. 2 shows one individual rack unit (IRU) of 16 blades and 256 cores on Blacklight. Each rectangle represents a blade. The 16-port NL5 router is used to connect blades located internally to each IRU. Each of these routers connects to eight compute blades within the IRU. The remaining eight ports of the internal router are used to connect to other NL5 router blades [41]. The total 4096 cores have 32 TB of memory.

Each thread running on a core in PODM maintains its own Poor Element List (PEL) [9]. The PEL contains the poor elements that violate the quality criteria [4] and are assigned to be processed by this thread. If PEL_i is not empty, thread T_i will get the first element in the list, compute the cavity, delete the tetrahedra in the cavity and create new tetrahedra according to the Bowyer-Watson kernel [25, 26]. Additionally, a global load balancing list stores the IDs of threads whose poor element list is empty. When a thread T_i runs out of work (its PEL is empty), it will push back its ID to the global load balancing list and start waiting. If another thread T_j creates new elements, it checks whether the load balancing list is empty. If the list is empty, it means all the other threads are busy. Thread T_j adds the newly created elements to its own PEL. If the list is not empty, thread T_j adds the newly created elements to the PEL

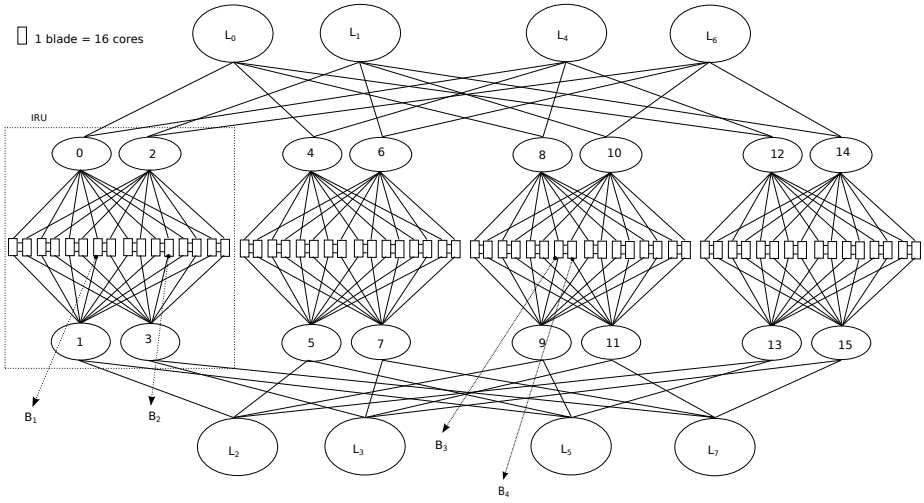


Figure 2: One possible allocation of 4 blades (64 cores) on Blacklight. The dashline rectangle represents the individual rack unit (IRU). Each IRU includes 16 blades and each blade has 16 cores that share 128 GB local memory. The blades B_1 and B_2 are in the same individual rack unit (IRU) and the blades B_3 and B_4 are in the same IRU. The maximum number of hops between two blades of the same IRU is 3 (B_1 and B_2) and that of different IRUs is 5 (B_2 and B_3).

of the first thread, suppose it is thread T_i, in the load balancing list. The new poor elements in thread $T_i's$ poor element list created by thread T_j still reside in thread $T_j's$ local memory. When T_i needs to refine these poor elements, it needs to access thread $T_j's$ local memory to fetch them. In the case that these two threads run on the cores that belong to the same blade, the ask-for-work operation between them is a local memory access since all cores in the same blade share memory without any switches. However, If thread T_i and thread T_j are not in the same blade, one thread needs to fetch a poor element from the local memory of another thread. This leads to a remote memory access. Moreover, if they are not in the same IRU, the time latency is much longer because of the increase in hops (about a 2000 cycle latency penalty on Blacklight [25] for each hop) and the traffic contention. Fig. 2 shows one possible case when we reserve 64 cores on Blacklight. It illustrates that the maximum number of hops between two blades of the same IRU is three and that of different IRUs is five.

As most DSM supercomputers, the experimental platform, Blacklight, is

shared by many users. The scheduling and reservation of cores (blades) is managed by the system. A user has no mechanism to decide which blades he can get to run his job. The system determines which blades are given to the user's job based on the available blades. In most cases, the job will get several non-adjacent blades among all blades in the system. For a data communication intensive application, such as parallel mesh generation, the performance will suffer on high core counts because of a large number of remote memory accesses. The performance of PODM was indeed poor when the allocated cores (>128) are non-consecutive. The ideal case is to make a thread finish all of its work on its local memory. However, this is almost impossible for unstructured parallel mesh generation because of the irregular and unpredictable communication during run-time. In this paper, we describe a two-level locality-aware parallel Delaunay mesh refinement algorithm LAPD. It divides the image into subregions and each subregion is refined by a parallel mesh generator (PODM, in our case). In each of the subregions, a thread of a PODM mesh generator has the flexibility to communicate with any other thread in the same PODM mesh generator in order to maximize the concurrency of this PODM mesh generator. The communication between different PODM mesh generators is confined, and only happens when the poor element is near the partition boundary. This two-level locality-aware parallel strategy eliminates a large number of remote memory accesses as well as alleviates the pressure of the network routers caused by the intensive communication among threads.

4. Two-level Locality-Aware Parallel Delaunay Mesh Generation

4.1. Two-level Parallel Mesh Refinement

The algorithm explores concurrency at two levels of granularity: coarse-grain parallelism at the subregion level (which is mapped to a node with multiple cores) and medium-grain parallelism at the cavity level (which is mapped to a single core).

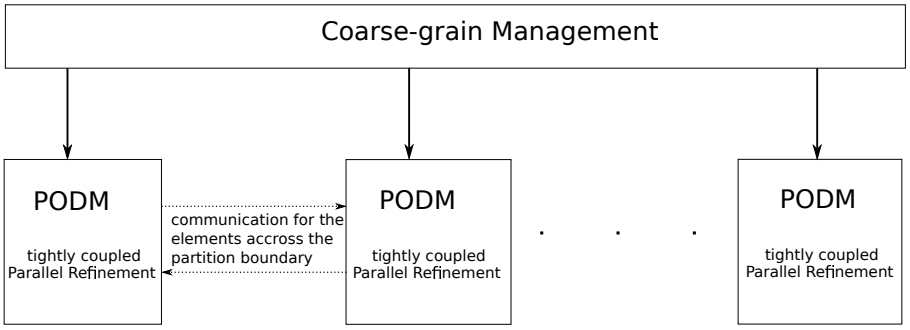


Figure 3: A diagram that illustrates the design of the two-level parallel Delaunay mesh generation algorithm. The boxes that are marked $PODM$ represent parallel tightly coupled Delaunay mesh generator. The block $Coarse\text{-}grain\ Management$ represents the management and distribution of PODM mesh generators on different subregions. In PODM, each thread has the flexibility to communicate with any other thread at any time. In LAPD, the communication between two PODM nodes is confined, and happens when one node needs to refine an element across the partition boundary between these two subregions.

In the coarse-grain parallel implementation, the input image (domain) is decomposed into subregions and each node is responsible for the refinement work of one subregion. The communication between two adjacent subregions happens only near the partition boundary. In the medium-grain parallel implementation, the threads running in the cores of a node follow the refinement rules of PODM to insert or delete multiple points in parallel. The work load balancing among the threads of each node is performed by the load balancing scheme of PODM mesh generator. Fig. 3 illustrates a diagram of the two-level parallel mesh generation design.

In the two-level parallel locality-aware refinement algorithm, we combine two different communication types in two granularity levels, partially coupled communication at the node level and tightly coupled communication at the core level. It quickly leverages high concurrency due to the aggressive speculative approach employed by tightly coupled PODM of each subregion, uses the partially coupled communication to ensure the conformity of elements across the partition boundary, and employs data partitioning to improve data locality gradually during the mesh refinement procedure. Fig. 4 depicts the pseudo-code of the algorithm.

```
1  Algorithm: LAPD (I,r,n,b)
   Input  : I is the input segmented image,
            r is the circum-radius upper bounds vector of length n
            /* r_i in the vector r defines the circum-radius upper bounds of elements created in step i.  */
          2 b is the number of blades.
   Output: A Delaunay Mesh M that is conforming to the size upper bound r_n.
3  Generate Initial Mesh that is conforming to the size upper bound r_1;
4  for i = 2 to n do
5      M_i = StepMesh(r_i, i, M_{i-1}, b);
       /* M_{i-1} is the mesh generated in the previous step and M_i is the new mesh created in step i.  */
6  end

7  Algorithm: StepMesh (r̄,i,M, b)
   Input  : i is the current step
            r̄ is the current size upper bound,
            M is the mesh created in the previous step,
            b is the number of blades.
   Output: A Delaunay Mesh M' that is conforming to the current size upper bound r̄.
8  Divide the 2^{i-2} subregions of the previous step into 2^{i-1} subregions by the bisection plane
   of the previous subregion along one dimension;
9  Assign the elements of M to subregion PELs based on the circumcenter coordinates;
10 Divide the b blades into 2^{i-1} nodes based on their physical locations;
11 Assign each node a PEL of a subregion;
12 for each node do
13     SM = GenerateMesh(r̄, PEL);
14 end

15 Algorithm: GenerateMesh (r̄,PEL)
   Input  : r̄ is the size upper bound of the current step,
            PEL is the poor element list that need to be refined.
   Output: A submesh SM of a subregion.
16 while PEL! = NULL do
17     Get the first poor element e in PEL;
18     Refine e and create new elements;
19     for each newly created element e' do
20         if e' is a poor element according to the size upper bound r̄ and fidelity bounds
           then
21             add e' to PEL;
22         else
23             add e' to output SM of this subregion;
24         end
25     end
26 end
```

Figure 4: Pseudocode of the locality-aware parallel Delaunay mesh algorithm. It consists of three sub-algorithms. Sub-algorithm $StepMesh()$ is to divide the mesh of the previous step into to different subregions and divide blades into different nodes. Sub-algorithm $GenerateMesh()$ is to generate mesh for each subregion.

4.2. Data Locality-Aware Implementation

For illustration purposes, in this subsection we exhibit a simplified two dimensional example showing the locality-aware property of LAPD algorithm. The parameter r_i is the circumradius upper bound that we use to control the size and the number of elements that are created in step i. The smaller r_i is, the more elements are created. Empirically we found that setting $r_i = r_{i-1}/2$ leades to the best balance between available concurrency and overheads.

In the beginning of mesh process, an appropriate isosurface is recovered and an initial tetrahedral mesh is constructed and refined in parallel using the PODM mesh generator until all elements satisfy the user-defined size upper bound r_1 determined by the theory we developed in the previous work [7, 14, 5]. The elements in this initial mesh are distributed among all memory blocks since PODM uses all available threads to jump-start the computation with maximum concurrency. Fig. 5a exhibits a two dimensional illustration of an initial mesh and its distribution in memory after the first step. We use four different colors to distinguish the elements in different memory of four blades. The left subfigure shows the mesh, and the right subfigure illustrates the element distribution on four different memory blocks. The different colors represent different memory blocks in which elements are stored.

In the second step the whole region (image) is divided into two subregions and the initial mesh is divided into two sub-meshes based on the coordinates of centers of element circum-spheres. A PODM mesh generator refines the sub-mesh of each subregion. The threads of the PODM mesh generator will work in the subregion and refine the sub-mesh in parallel to get the mesh that is conforming to size upper bound r_2 using similar criteria as before. As illustrated in Fig. 5b, the whole region, i.e., the bounding box and the underling image, is divided by its bisection line (bisection plane in three dimensional space) along one dimension, and the initial mesh is divided into two sub-meshes, M_1 and M_2, according to the coordinates of circum-centers of elements in the mesh. In this example, there are two nodes in this step. We assume that node G_1 contains blades B_1 and B_2 while node G_2 contains blades B_3 and B_4. Node G_1 is only

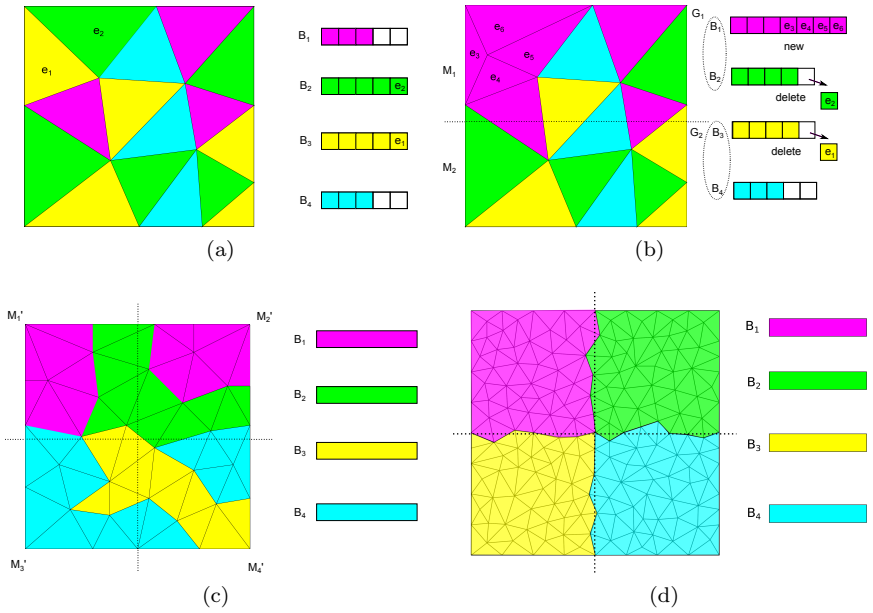


Figure 5: (a) A simplified two dimensional illustration of an initial mesh and its allocation in memory. (b) During the mesh refinement of the second step. (c) A new mesh was created after the second step and the mesh is divided into four sub-meshes in the third step. (d) After the refinement of the third step. In the next step, each node will only need to access its own local memory to finish the refinement work except the elements on the boundary.

responsible for refining the elements belonging to the top half subregion and node G_2 is responsible for the elements in the bottom half subregion.

During the refinement, B_1 gets the poor element e_1 that was stored in $B_3's$ local memory and adds it to its own poor element list. Element e_1 is triangulated and deleted. To keep the Delaunay property, element e_2 is part of the cavity and is also triangulated. The new elements e_3, e_4, e_5 and e_6 will be stored in the local memory of B_1 because the new elements will be stored in the local memory of the blade that created the elements, i.e., B_1, as shown in Fig. 5b. The same process is used to refine the other elements in the mesh. The elements of the first subregion, i.e. sub-mesh M_1, will be refined by node G_1 and the newly created elements will be in the local memory of G_1, i.e., in the local memory of blades of this node. Those elements of M_2 will be refined by G_2 and the newly created elements will be in the local memory of G_2.

Therefore, after the second step, a new mesh that is conforming to size upper bound r_2 is generated. Fig. 5c shows the mesh and its storage configuration in memory. The elements that are on the top half subregion were created by blades B_1 and B_2 of node G_1 and stored in the local memory of B_1 and B_2 while those that are on the bottom half subregion were created by blades B_3 and B_4 of node G_2 and stored in the local memory of B_3 and B_4.

In the third step, the whole region is divided into four subregions as shown in Fig. 5c. Each of the four sub-meshes will be assigned to one node to refine. Each blade in node G_1 of the second step, i.e., B_1 or B_2, can only be assigned a subregion that is in the top half part because the elements, the green and pink ones, of this part were stored in the local memory of B_1 and B_2. Similarly, B_3 or B_4 can only be assigned a subregion that is in the bottom half part because the elements, the green and pink ones, of this part were stored in the local memory of B_3 and B_4. Under this assignment, during the refinement of the third step, blades B_1 and B_2 can only refine the elements stored in either $B_1's$ or $B_2's$ local memory while blades B_3 and B_4 can only refine the elements stored in either $B_3's$ or $B_4's$ local memory. The communication happens between subegions if and only if a thread in one subregion wants to refine the elements that are adjacent to the elements of the other subregion along the partition boundary.

After the third step, a mesh is created and its storage configuration in memory is shown in Fig. 5d. The newly created elements of each subregion will be only stored in the local memory of the blade that is responsible for refining the subregion.

Finally, we go on to the next step. In this step, each blade will only need to access its own local memory to finish the refinement work except the elements on the boundary shown in Fig. 5d. The refinement continues until all elements in the mesh are Delaunay and conforming to the target size upper bound r_t.

In this data locality-aware mesh refinement algorithm, we reduce the number of remote memory accesses by controlling the inter-node communication in each step. A blade can only ask for work or give work to the blades that are in the same node, as B_1 can only communicate with B_2 in the second step shown in

Fig. 5b because they are in the same node and physically close to each other. In the last step, there will be only a few remote memory accesses (when the elements are near the partition boundary between subregions) because most of the poor elements that one blade needs to refine are in the local memory of this blade.

4.3. Over-decomposition and Block-based Partition for Load Balance

In this subsection, we describe an over-decomposed block-based partition approach to alleviate the load balancing problem.

The over-decomposed block-based partition proceeds in three main stages: (1) over-decompose the bounding box that overlaps the input image and the coarse mesh (i.e., the number of blocks is much greater than the number of cores), (2) mark the blocks that contains elements as active blocks, (3) partition the active blocks into N subregions to make each of subregions contain a roughly equal number of blocks, where N is the number of basic computing nodes that share the local memory (For example, a blade of 16 cores that shared 128GB memory is a basic computing node on Blacklight). In this case, each subregion ends up with a roughly equal number of elements and the refinement is well balanced among the mutlticore PODM mesh generators working on each subregion.

Fig. 6 gives a demonstration of how this static work load partition strategy makes LAPD ensure both data locality and load balance. We use the static block-based partition approach to roughly balance the load among different subregions at each step. For each subregion, we utilize the load balancing approach of PODM. Taking advantage of this two level load balancing scheme, LAPD ensures the data locality during the parallel refinement procedure and does not suffer from severe load imbalance for the complex input images. The case of dynamic load balancing problem in the context of adaptive mesh refinement is out of scope of this paper. We developed a run time system [42] to address this problem.

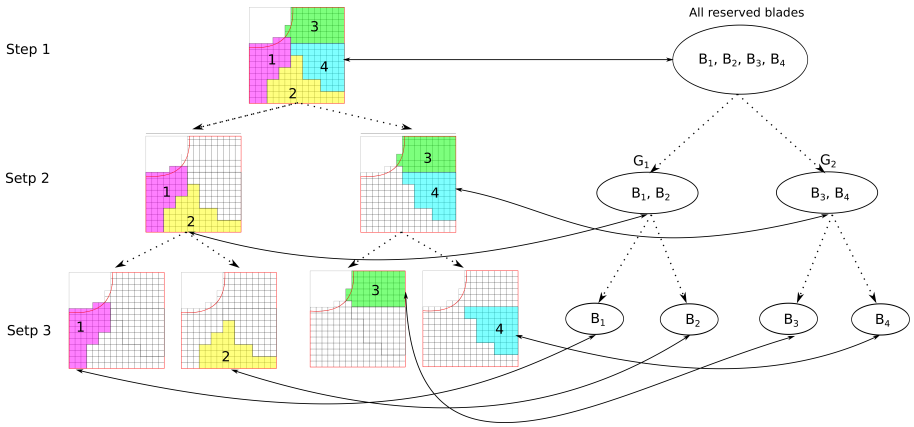


Figure 6: Block partition and computing node mapping illustration of LAPD

5. Experimental Evaluation

In this section, we present the weak scaling performance of the two-level locality-aware parallel mesh generation algorithm LAPD as well as that of PODM for comparison. The input images we used in the experiments are the 3D CT abdominal atlas obtained from IRCAD Laparoscopic Center [43] and the 3D Brain atlas [44]. We tested both LAPD and PODM on Blacklight using up to 192 cores. See Table 1 and Table 2 for detailed results of both approaches.

5.1. Performance Evaluation Metrics

We use the following metrics to evaluate and study the performance of parallel mesh generation algorithms [45, 46].

Speedup *S*: The ratio of the serial execution time of the fastest known serial algorithm (T_s) to the parallel execution time of the parallel algorithm (T_p).

Efficiency *E*: The ratio of speedup (S) to the number of cores (p): $E = S/p = T_s/(pT_p)$.

In the weak scaling case, the number of elements per thread (we use one thread per core) remains approximately constant. In other words, the problem size (i.e., the number of elements created) is increased proportionally to the number of threads. The number of elements generated equals approximately 3

million on a single Blacklight core. The problem size gradually increases from 3 million to 559 million tetrahedra for 1 to 192 cores on Blacklight. In practice, because of the irregular nature of the unstructured mesh, it is impossible to control the problem size (number of elements) exactly while the number of cores is increased by p times. So we use an alternative definition of speedup which is more precise for a parallel mesh generation algorithm.

We measure the number of elements (tetrahedra) generated every second in the experiment. Let us denote by $Elements(p)$ and $Time(p)$ the number of tetrahedra generated and the meshing time respectively, where p is the number of threads. Then we can use the following equation to compute the speedup:

$$S(p) = \frac{elements\_per\_sec(p)}{elements\_per\_sec(1)} = \frac{elements(p) \cdot time(1)}{time(p) \cdot elements(1)} \tag{1}$$

In equation (1), $elements\_per\_sec(p)$ represents the number of elements (tetrahedra) created per second using p threads (cores); $elements\_per\_sec(1)$ represents the number of elements (tetrahedra) created per second by the best sequential mesh generation algorithm. Since the PODM maintains the best single-threaded performance compared to other sequential three dimensioanl mesh generation software, such as Tetgen [30] and CGAL [47], we use the sequential rate of PODM, i.e., the number of elements generated per second by single-threaded PODM, as a reference when we compute the speedup and present the performance of multi-threaded LAPD.

5.2. Experimental Results and Analysis

Table 1 show the weak scaling performance of PODM and the two-level locality-aware parallel mesh generation algorithm LAPD respectively. The input image is the 3D abdominal atlas. We observe that both PODM and LAPD performed well on Blacklight for up to 64 cores. However, the speed-up of PODM deteriorates significantly for 128 or more cores. In fact, the speed-up on 144 cores is about 80.8, which is smaller than that on 128 cores (about 94.5), and it is down to only 62.9 and 54.8 for 176 and 192 cores, respectively. The

Table 1: Weak scaling performance comparison of PODM and LAPD. The input image is a 3D abdominal atlas. The number of elements remains approximately linear with respect to the number of threads in both PODM and LAPD.

(a) Weak scaling performance of PODM

Threads	1	32	64	128	144	160	176	192
Elements (millions)	3.09	96.96	186.80	374.09	419.65	467.01	513.81	559.20
Time(s)	27.78	26.07	27.02	35.97	47.24	57.24	74.20	92.77
Elements per second (millions)	0.11	3.75	6.91	10.41	8.89	8.14	6.92	6.03
Speedup	1.0	34.1	62.8	94.5	80.8	74.0	62.9	54.8
Efficiency	1.00	1.06	0.98	0.74	0.56	0.46	0.36	0.29

(b) Weak scaling performance of LAPD

Threads	1	32	64	128	144	160	176	192
Elements (millions)	3.09	96.96	186.80	374.09	419.65	467.01	513.81	559.20
Time(s)	27.81	26.18	26.90	31.26	34.04	36.27	37.82	39.63
Elements per second (millions)	0.11	3.73	6.95	12.01	12.30	12.70	13.61	14.12
Speedup	1.0	33.9	63.2	109.1	111.8	115.5	123.6	128.2
Efficiency	1.00	1.06	0.99	0.85	0.78	0.72	0.70	0.67

blue line in Fig. 7a shows clearly this speed-up degradation of PODM when core count is above 128.

The main reason of this performance deterioration of PODM is the increase of communication time due to the large number of remote memory accesses and the congested network. In PODM, each thread has the flexibility to communicate with any other thread during the refinement. This approach works well on a medium number of cores (threads) and exhibits impressive scalability. However, when core count is beyond a certain number, 128 on Blacklight for example, the communication overhead becomes the bottleneck that hinders the performance of PODM because it exerts too much pressure on the network routers.

In LAPD algorithm, we confine the tightly coupled communication among cores in each node, i.e., the threads running on the cores of this node have the flexibility to communicate with each other to maximize the concurrency of this node. As we explain bellow, the communication between two nodes happens when one node needs to refine an element across the partition boundary between these two subregions and at least one of element is in the local memory of the other node during the cavity expansion. In order to guarantee the con-

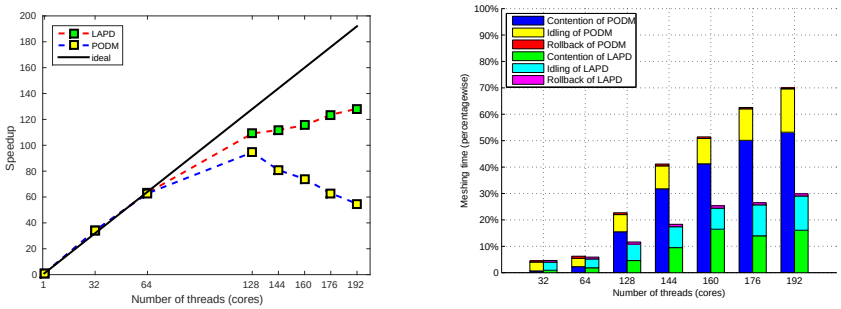
formity of the created mesh, this inter-node communication is necessary and unavoidable. In other words, the inter-node communication is forbidden unless it is unavoidable.

Table 1b shows that the speed-up and efficiency of LAPD on 128 Blacklight cores is 109.1 and 85% respectively, which is better than those of PODM, 94.5 and 74% respectively. The red line in Fig. 7a illustrates that each time we increase the number of cores by 16 (a blade), the approach gains some speedup increase for up to 192 cores. The efficiencies of LAPD on 176 and 192 cores are 1.9 and 2.3 times better than those of PODM. The previous image-to-mesh conversion algorithm, PODM, scales well up to only 128 cores on Blacklight. The locality-aware approach LAPD scales well up to 192 cores on Blacklight.

The overhead time of both PODM and LAPD mainly consists of three parts:

- Rollback overhead time: this is the time that threads spend on completing partial cavity expansions before they detect a conflict with some other thread and discard the expansion.
- Idling time overhead: this is the total time that threads had no poor elements to refine, and they were idling and waiting for more work from other threads.
- Communication (Contention) overhead time: this is the total time that threads spent on fetching elements that were not in the local memory.

See Fig. 7b for the details of overhead time percentage of PODM and LAPD. The figure shows that the total overhead time of LAPD is less than that of PODM for all numbers of cores from 64 to 192. We observe that for core counts beyond 128 the communication overhead time (the blue bar in Fig. 7b) contributes the main part of the total overhead time. The communication overhead time takes a higher percentage of the total overhead time with the increase in the number of cores. The percentage of communication overhead on 144 cores is about 32% while this number is already increasing to 50% for 176 cores and 53% for 192 cores. Since the problem size increases linearly with respect to



(a) (b)

Figure 7: Speedup and overhead comparison of PODM and LAPD for 3D abdominal image. (a) Weak scaling speedup of PODM and LAPD upto to 192 cores on Blacklight. Three million tetrahedra are created by each thread running on a core. The black line depicts the ideal linear speedup. The red dash line with green markers shows the speedup of LAPD and the blue one with yellow markers is the speedup of PODM. (b) Overhead percentage of PODM and LAPD. The left stacked bar shows the overhead percentage of PODM and the right one shows the overhead of LAPD.

the number of threads (cores), the communication traffic per network router increases during the refinement process. Besides, the risk of contention with other users' jobs running on Blacklight also increases with the core count increasing. Because of these overhead, the performance of PODM deteriorates on Blacklight for high core count.

The green bar in Fig. 7b illustrates that the communication overhead time of LAPD is less than half that of PODM. The idling time and rollback overhead time of each thread in LAPD stays approximately the same percentage as those in PODM. Since the communication overhead time is the main part of the total overhead time in PODM, the total overhead time in LAPD is reduced by a large percentage after the communication overhead time is reduced by the locality-aware optimization.

Table 2 shows the performance comparison of PODM and LAPD algorithms for the 3D Brain atlas. Again, we can see clearly the significant performance improvement of LAPD compared to that of PODM.

Table 2: Weak scaling performance comparison of PODM and LAPD. The input image is a 3D Brain atlas. The number of elements remains approximately linear with respect to the number of threads in both PODM and LAPD.

(a) Weak scaling performance of PODM

Threads	1	32	64	128	144	160	176	192
Elements (millions)	2.01	64.32	128.61	257.30	289.46	321.72	353.78	385.94
Time(s)	18.29	17.73	18.72	22.88	29.91	39.73	49.69	57.78
Elements per second (millions)	0.11	3.62	6.87	11.24	9.68	8.10	7.12	6.68
Speedup	1.0	33.0	62.5	102.2	87.97	73.60	64.72	60.72
Efficiency	1.00	1.03	0.98	0.80	0.61	0.46	0.37	0.32

(b) Weak scaling performance of LAPD

Threads	1	32	64	128	144	160	176	192
Elements (millions)	2.01	64.32	128.61	257.30	289.46	321.72	353.78	385.94
Time(s)	18.29	18.48	18.83	21.66	24.01	25.18	27.08	28.48
Elements per second (millions)	0.11	3.48	6.83	11.88	12.06	12.77	13.06	13.55
Speedup	1.0	31.6	62.1	109.6	111.8	116.1	118.8	123.2
Efficiency	1.00	0.99	0.97	0.84	0.76	0.73	0.68	0.65

6. Conclusion and Future Work

In this paper, we presented a three dimensional two-level locality-aware parallel Delaunay image-to-mesh conversion algorithm, LAPD. LAPD employs a combination of two-level parallelism and data locality-aware scheme to reduce the communication overhead caused by a large number of remote memory accesses in a NUMA architecture. It quickly leverages high concurrency due to the aggressive speculative approach employed by PODM, and uses data partitioning to improve data locality. We tested LAPD algorithm on Blacklight, a cc-NUMA shared memory machine in the Pittsburgh Supercomputing Center. By reducing the communication overhead caused by remote memory accesses, LAPD is scalable with the increase of core count on Blacklight. After the data locality optimization, LAPD reaches a more than 67% weak scaling efficiency for up to 192 cores in contrast with PODM which is only about 30%.

Our feature work includes extending the LAPD implementation to non-shared distributed memory architecture and comparing the performance with the implementation presented in this paper. We think that similar optimizations will apply to cache memory, however they will require a number of specialized

techniques that can be studied in a separate comprehensive exploration. The long term plan is to exploit the performance potential of modern muticore systems (both shared and distributed memory architecture) and deliver sufficient scalability for applications that require exascale computing. We will achieve this by leveraging concurrency at different granularity levels using hierarchical hybrid mesh generation algorithms.

Acknowledgment

This work is supported in part by NSF grants CCF-1139864 and CCF-1439079 and by the Richard T.Cheng Endowment. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575. We thank the systems group in the Pittsburgh Supercomputing Center for their great help and support. We especially thank Panagiotis Foteinos and Juliette Kelly Pardue 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 (4) (2014) 539–562.

[5] A. Chernikov, N. Chrisochoides, Three-dimensional Delaunay refinement for multi-core processors, ACM International Conference on Supercomputing (2008) 214–224.

[6] A. Chernikov, N. Chrisochoides, Algorithm 872: parallel 2D constrained Delaunay mesh generation, ACM Transactions on Mathematical Software 34 (2008) 6–25.

[7] A. Chernikov, N. Chrisochoides, Parallel guaranteed quality Delaunay uniform mesh refinement, SIAM Journal on Scientific Computing 28 (2006) 1907–1926.

[8] 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.

[9] P. Foteinos, N. Chrisochoides, High quality real-time image-to-mesh conversion for finite element simulations, Journal on Parallel and Distributed Computing 74 (2) (2014) 2123–2140.

[10] P. Foteinos, N. Chrisochoides, Dynamic parallel 3D Delaunay triangulation, in: International Meshing Roundtable, 2011, pp. 9–26.

[11] K. Amunts, C. Lepage, L. Borgeat, H. Mohlberg, T. Dickscheid, M.-t. Rousseau, S. Bludau, P.-L. Bazin, L. B. Lewis, A.-M. Oros-Peusquens, N. J. Shah, T. Lippert, K. Zilles, A. C. Evans, Bigbrain: An ultrahigh-resolution 3d human brain model, Science 340 (6139) (2013) 1472–1475. doi:10.1126/science.1235381.

[12] N. Chrisochoides, A. Chernikov, A. Fedorov, A. Kot, L. Linardakis, P. Foteinos, Towards exascale parallel Delaunay mesh generation, in: International Meshing Roundtable, Springer, 2009, pp. 319–336.

[13] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, D. Mavriplis, Cfd vision 2030 study: A path to revolutionary computational aerosciences, Tech. rep. (March 2014).

[14] A. Chernikov, N. Chrisochoides, Practical and efficient point insertion scheduling method for parallel guaranteed quality delaunay refinement, in: ACM International Conference on Supercomputing, 2004, pp. 48–57.

[15] L. Linardakis, N. Chrisochoides, Delaunay decoupling method for parallel guaranteed quality planar mesh refinement, SIAM Journal on Scientific Computing 27 (4) (2006) 1394–1423.

[16] O. C. Zienkiewicz, R. L. Taylor, P. Nithiarasu, The Finite Element Method for Fluid Dynamics, seventh Edition, Butterworth-Heinemann, 2013.

[17] J. Xu, A. Chernikov, Curvilinear Triangular Discretization of Biomedical Images with Smooth Boundaries, in: International Symposium on Bioinformatics Research and Applications, Springer, Norfolk, VA, 2015, to appear.

[18] L. Oliker, R. Biswas, Parallelization of a dynamic unstructured algorithm using three leading programming paradigms, IEEE Transactions on Parallel and Distributed Systems (TPDS) 11 (9) (2000) 931–940.

[19] R. A. Chowdhury, F. Silvestri, B. Blakeley, V. Ramachandran, Oblivious algorithms for multicores and networks of processors, Journal of Parallel and Distributed Computing 73 (7) (2013) 911–925.

[20] SPL Knee Atlas, http://www.spl.harvard.edu/publications/item/view/1953 (11 2012).

[21] H. Si, Constrained Delaunay tetrahedral mesh generation and refinement, Finite Elements in Analysis and Design 46 (2010) 33–46.

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

[23] P. Foteinos, A. Chernikov, N. Chrisochoides, Guaranteed quality tetrahedral Delaunay meshing for medical images, in: International Symposium on Voronoi Diagrams in Science and Engineering, 2010, pp. 215–223.

[24] L. P. Chew, Guaranteed-quality Delaunay meshing in 3D, in: Proceedings of the 13th ACM Symposium on Computational Geometry, 1997, pp. 391–393.

[25] D. F. Watson, Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes, Computer Journal 24 (1981) 167–172.

[26] A. Bowyer, Computing Dirichlet tesselations, Computer Journal 24 (1981) 162–166.

[27] N. Chrisochoides, D. Nave, Parallel Delaunay mesh generation kernel, International Journal for Numerical Methods in Engineering 58 (2003) 161–176.

[28] G. E. Blelloch, G. L. Miller, J. C. Hardwick, D. Talmor, Design and implementation of a practical parallel Delaunay algorithm, Algorithmica 24 (3) (1999) 243–269.

[29] J. Shewchuk, Triangle: Engineering a 2d quality mesh generator and delaunay triangulator, in: M. Lin, D. Manocha (Eds.), Applied Computational Geometry Towards Geometric Engineering, Vol. 1148 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1996, pp. 203–222. doi:10.1007/BFb0014497.

[30] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2) (2015) 11:1–11:36. doi:10.1145/2629697.

[31] C. Armstrong, D. Robinson, R. McKeag, T. Li, S. Bridgett, R. Donaghy, C. MCGleenan, Medials for meshing and more, in: 4th International Meshing Roundtable, 1995, pp. 277–288.

[32] H. N. Gursoy, N. M. Patrikalakis, An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part i algorithms, Engineering With Computers 8 (1992) 121–137.

[33] L. P. Chew, N. Chrisochoides, F. Sukup, Parallel constrained Delaunay meshing, in: ASME/ASCE/SES Summer Meeting,Special Symposium on Trends in Unstructured Mesh Generation, 1997, pp. 89–96.

[34] H. L. de Cougny, M. S. Shephard, C. Ozturan, 3rd national symposium on large-scale structural analysis for high-performance computers and workstations parallel three-dimensional mesh generation, Computing Systems in Engineering 5 (4) (1994) 311 – 323. doi:http://dx.doi.org/10.1016/0956-0521(94)90014-0.

[35] R. Löhner, J. R. Cebral, Parallel advancing front grid generation, in: Proceedings of the 8th International Meshing Roundtable, South Lake Tahoe, CA, 1999, pp. 67–74.

[36] Y. Ito, A. Shih, A. Erukala, B. Soni, A. Chernikov, N. Chrisochoides, K. Nakahashi, Generation of unstructured meshes in parallel using an advancing front method, in: International Conference on Numerical Grid Generation in Computational Field Simulations, San Jose, CA, 2005.

[37] G. Globisch, Parmesh – a parallel mesh generator, Parallel Computing 21 (3) (1995) 509–524. doi:http://dx.doi.org/10.1016/0167-8191(94)00085-0.

[38] G. Globisch, On an automatically parallel generation technique for tetrahedral meshes, Parallel Computing 21 (12) (1995) 1979–1995.

[39] N. P. Chrisochoides, A survey of parallel mesh generation methods, Tech. Rep. BrownSC-2005-09, Brown University, also appears as a chapter in Numerical Solution of Partial Differential Equations on Parallel Computers (eds. Are Magnus Bruaset and Aslak Tveito), Springer, 2006 (2005).

[40] Blacklight, https://portal.xsede.org/web/xup/psc-blacklight.

[41] Sgi altix uv 1000 system users guide, Report (2011).

[42] K. Barker, A. Chernikov, N. Chrisochoides, K. Pingali, A load balancing framework for adaptive and asynchronous applications, IEEE Transactions on Parallel and Distributed Systems 15 (2) (2004) 183–192.

[43] Ircad Laparoscopic Center, http://www.ircad.fr/softwares/3Dircadb/3Dircadb2 (2013).

[44] Multi-modality MRI-based Atlas of the Brain, http://www.spl.harvard.edu/publications/item/view/2037 (04 2015).

[45] J. L. Gustanfson, G. R. Montry, R. E. Benner, Development of parallel methods for a 1024-processor hypercube, SIAM Journal on Scientific and Statistical Computing 9 (4) (1988) 609–638.

[46] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Addison Wesley, 2003.

[47] Cgal, computational geometry algorithms library, `http://www.cgal.org`.