Parallel Mesh Generation

Nikos Chrisochoides

Computer Science Department
College of William and Mary
Williamsburg, VA 23185
and
Division of Applied Mathematics
Brown University
182 George Street
Providence, RI 02912

1 Abstract

Parallel mesh generation is a relatively new research area between the boundaries of two scientific computing disciplines: computational geometry and parallel computing. In this chapter we present a survey of parallel unstructured mesh generation methods. Parallel mesh generation methods decompose the original mesh generation problem into smaller subproblems which are meshed in parallel. We organize the parallel mesh generation methods in terms of two basic attributes: (1) the sequential technique used for meshing the individual subproblems and (2) the degree of coupling between the subproblems. This survey shows that without compromising in the stability of parallel mesh generation methods it is possible to develop parallel meshing software using off-the-shelf sequential meshing codes. However, more research is required for the efficient use of the state-of-the-art codes which can scale from emerging chip multiprocessors (CMPs) to clusters built from CMPs.

2 Introduction

This chapter presents a survey of parallel unstructured mesh generation methods based on three widely used techniques: Delaunay [41], Advancing Front [67], and Edge Subdivision [60]. Parallel methods for quadrilateral [7] and hexahedral [55] mesh generation as well as block structured [93, 23, 90] and structured adaptive mesh refinement [2] methods are not reviewed in this chapter.

Parallel mesh generation procedures in general decompose the original 2-dimensional (2D) or 3-dimensional (3D) mesh generation problem into N_s

smaller subproblems which are solved (i.e., meshed) concurrently using P processors. The subproblems can be formulated to be either tightly coupled [61, 57, 79], partially coupled [56, 33, 21] or even decoupled [40, 80, 53]. The coupling of the subproblems determines the intensity of the communication and the amount/type of synchronization required between the subproblems.

The challenges in parallel mesh generation methods are: to maintain *stability* of the parallel mesher (i.e., retain the quality of finite elements generated by state-of-the-art sequential codes) and at the same time achieve 100% *code re-use* (i.e., leverage the continuously evolving and fully functional off-the-shelf sequential meshers) without substantial deterioration of the *scalability* of the parallel mesher. In this chapter we review parallel mesh generation methods having in mind these three requirements.

We build on top of previous work [32, 41] where parallel mesh generation methods are classified in terms of the *way* and the *order* the artificial boundary surfaces (interfaces) of the subproblems are meshed. Specifically, in [33, 41] existing parallel methods are classified in three categories: (i) methods that first mesh (either in parallel [56] or sequentially [80]) the interfaces of the subproblems and then mesh in parallel the individual subproblems, (ii) methods that first solve the meshing problem in each of the subproblems in parallel and then mesh the interfaces so that the global mesh is conforming [38], and (iii) methods that simultaneously mesh and improve the interfaces as they mesh the individual subproblems [26, 21, 27].

In this chapter we organize the parallel mesh generation methods in terms of two basic attributes. First, the sequential technique used for meshing the individual subproblems: *(1) Delaunay, (2) Advancing Front, and (3) Edge Subdivision*. Second, the degree of coupling between the subproblems: *(a) tightly-coupled, (b) partially-coupled, and (c) decoupled methods*.

3 Domain Decomposition Approaches

Parallel mesh generation methods use a sequential pre-processing step for the data partitioning problem with the exception of [48, 49]. The data are partitioned using either the continuous domain which is decomposed into *subdomains* (see Figure 1, left) or a discrete approximation (i.e., an initial coarser mesh) of the domain which is decomposed into *submeshes* (see Figure 1, right). The internal boundaries between the subdomains or submeshes (S_i) are called *interfaces* or *separators* (∂S_i). In both cases the number of generated subdomains or submeshes (N_s) can be significantly greater than the number of processors P (*over-decomposition*). Over-decomposition was introduced in parallel computing in mid 80s. It is used to hide communication latency in message passing [51] and to mask information dissemination, decision making and data migration costs in dynamic load balancing [28].

The domain decomposition (DD) problem in parallel mesh generation is defined as follows:

$$lfs(\Omega) \leq lfs(S_i) \quad i = 1, N_s \tag{1}$$

$$\min_{i=1,N_s} \frac{|\partial S_i|}{|S_i|} \tag{2}$$

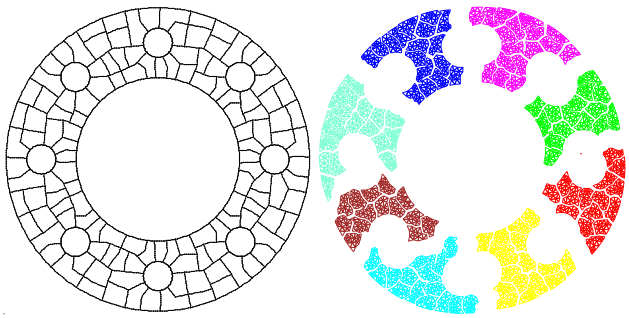
∂S_i form "good" angles between each other and the boundary $\partial\Omega$. (3)

where $lfs(\Omega)$ and $lfs(S_i)$ are the local feature size [84] of the original domain Ω and the subdomains (or submeshes) S_i, respectively. The $|\partial S_i|$ denotes the length (in 2D) and surface (in 3D) of the interfaces while the $|S_i|$ denotes the area (in 2D) and volume (in 3D) of the subdomains (or submeshes) S_i.

Continuous Domain Decomposition

The continuous domain decomposition methods partition the region Ω into subdomains $\Omega_i, i = 1, N_s$. There are two types of continuous DD methods. The first and most popular approach is based on quadtree/octree methods [33, 57, 59]. The octree methods utilize an octree structure for the decomposition of Ω into blocks (octants). The octants along with a description of the external boundary $\partial\Omega$ define the subdomains. Another class of continuous DD methods [53] is based on auxiliary structures like the Medial Axis [10, 69, 91] so that the subdomains Ω_i have no new features like small angles between the separators and the separators and external boundary [53].

Continuous DD approaches are attractive because they refine the individual subdomains by re-using existing well tested and fine-tuned sequential



DD of continuous geometry DD of discrete geometry

Fig. 1. Domain decomposition of the continuous geometry [53] and the discrete geometry [19] of a cross section of a rocket pipe.

codes on each subdomain independently. However, independence in mesh refinement and high code re-use in some cases come at a price. The polyhedral surfaces which arise due to the decomposition of the initial mesh impose additional constraints on the execution of sequential meshing algorithms in each of the subdomains. Poorly generated interface surfaces can affect the termination of meshing algorithms and the quality of the elements. Moreover, the artificially imposed interfaces can affect the mesh gradation.

Discrete Domain Decomposition

The Discrete DD methods partition an initial coarse mesh (usually a boundary conforming mesh), D into a number of simply-connected submeshes $D_i, i = 1, N_s$ while they try to minimize the surface-to-volume ratio for each of the submeshes. Usually a coarse mesh is generated on a high-performance workstation using sequential mesh generators. The partitioning of a coarse mesh is performed either sequentially or in parallel using generic graph partitioning libraries like Metis/Parallel Metis [81] and Chaco [43]. Also, there are mesh partitioning libraries like Domain Decomposer [24, 22], Zoltan [35], Drama [4], Plum [65], and Jove [87] (to mention a few) which extend and customize the generic data partitioning techniques for FEM calculations.

4 Parallel Mesh Generation Methods

In this section we review parallel mesh generation methods which are based on Delaunay triangulation in Section 4.1, Advancing Front Technique in Section 4.2, and Edge Subdivision methods in Section 4.3.

4.1 Delaunay Based Methods

There are many approaches to generate Delaunay meshes [41], we focus on methods based on Bowyer-Watson [11, 96] kernel which can lead to: (1) more efficient parallel implementations due to easier optimizations for improving data locality and (2) simpler and more efficient data structures. The Bowyer-Watson (BW) kernel is described in Figure 2 and the loop bellow:

Algorithm 1 (BW($M_o, p_1, ..., p_n$)).

1. **Input:** Mesh M_o an initial mesh and a set of n points
2. **for** i = 1, n
3. Compute the cavity of $\mathcal{C}_i$ of the point p_i
4. Compute the ball B_i of point p_i
5. $M_{i+1} = M_i - \mathcal{C}_i + B_i$
6. **endfor**
7. **Output:** A new mesh M = M_{n+1}

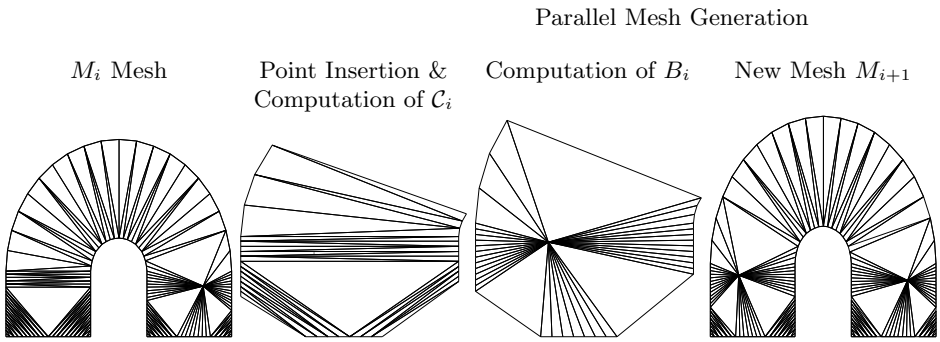


Fig. 2. Bowyer-Watson kernel starts with a mesh M_i (left), computes the cavity (center left) of a newly inserted point, triangulates the cavity (center right) and updates the mesh into M_{i+1} (right).

where the cavity $\mathcal{C}$ of a point p is defined as the set of all triangles whose circumcircle includes p; the ball B of a point p is defined as set of new triangles defined by the point p and the vertices of the boundary of its cavity [41].

The challenge, for parallel mesh generation methods based on the BW kernel, is to maintain the following loop invariant: *M_i is conformal and Delaunay, for $i = 1, n$.* Figure 3 depicts two cases where the concurrent point insertion violates the loop invariant. First, the cavities intersect i.e., there is a triangle $\triangle p_3p_6p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$, then concurrent insertion of p_8 and p_9 results in a non-conformal mesh. Second, the cavities share an edge in 2D (or a face in 3D), an edge p_3p_6 is shared by $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ and $\mathcal{C}(p_{10}) = \{\triangle p_3p_5p_6, \triangle p_3p_4p_5\}$, then the new triangle $\triangle p_3p_{10}p_6$ can have point p_8 inside its circircle, thus, violating the Delaunay property.

The focus of this section is on parallel mesh generation methods that address this challenge. There is a number of parallel Delaunay and triangulation methods like the MIMD method in [92] and the HPF implementation in [16] which target parallel programming paradigms no longer in use for practical purposes. Other methods [29, 63, 64, 12] also contributed in shaping up this author's directions and work in parallel mesh generation and implicitly contribute in this chapter.

In [8] the authors describe a divide-and-conquer projection-based algorithm for constructing in parallel 2D Delaunay triangulations of a set of given points. The method extends to 3D, but its implementation is quite complex. The goal in parallel mesh generation, though, is to refine an existing mesh by inserting new points i.e., the set of points in the final mesh is not known in advance.

In [47, 49] the authors extended [8] for parallel 2D mesh generation which further eliminates the sequential step for an initial mesh, but does not address the issue of code re-use. The method in [47, 49] is partially coupled.

In [37] the authors define the points x and y as independent if the closures of their *prestars* (or *cavities*) are disjoint. The approach in [37] does not

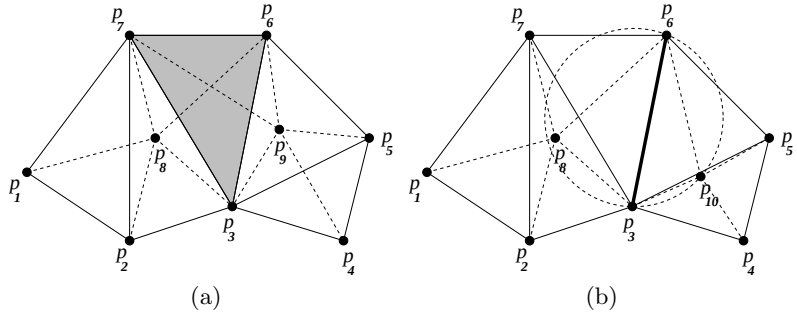


Fig. 3. (a) Intersection of two cavities and (b) two cavities share an edge; solid lines represent the edges of the initial triangulation, and dashed lines edges created by the insertion of p_8, p_9, and p_{10}.

provide a way to schedule the concurrent insertion of points whose cavity closures are disjoint.

In [88] the authors presented the first theoretical analysis of the complexity of parallel Delaunay refinement algorithms. However, the assumption in [88] is that the global mesh is completely retriangulated each time a set of independent points is inserted. In [89] the authors developed a more practical algorithm.

In the rest of this section we describe five different practical (i.e., they have been implemented) parallel Delaunay mesh generation methods. These methods formulate the subproblems to be: (1) tightly coupled, (2) decoupled, and (3) partially coupled.

Tightly Coupled Methods

A straight forward approach to parallel computing is based on identifying some partial order among the computations of well understood and successful sequential kernels and then in a brute-force fashion use message passing or threads to implement the computations on distributed and shared memory parallel machines, respectively. This approach leads to the tightly coupled method presented in [61] for parallel guaranteed quality Delaunay mesh generation.

Parallel Optimistic Delaunay Meshing (PODM) Method

In [61] the authors presented the first provable 3D parallel guaranteed quality Delaunay mesh generation method for polyhedral domains. PODM is based on discrete domain decomposition, but it is not constrained by the interfaces of the submeshes. The algorithm guarantees the stability by simultaneously re-partitioning and refining the interface surfaces and volume of the submeshes [27] —refinement due to a point insertion might extend across

subproblem (or submesh) boundaries. The extension of a cavity beyond the interfaces is a source of intensive communication. However, PODM can tolerate most of the communication by concurrently refining other regions of the submeshes while it waits for remote data to arrive. Unfortunately, the concurrent refinement can create a number of inconsistencies in the mesh (see Figure 3). These inconsistencies are resolved at the cost of setbacks (or rollbacks [45]) and thus we call this method Parallel Optimistic Delaunay Meshing method. Setbacks is a source of major algorithm and code re-structuring (due to overlapping cavities) and they lead to zero code re-use. Unfortunately, the overlapping of the cavities becomes even more complex when they are near the external boundary, where a certain order of inserted points needs to be maintained due to encroachment rules that are used to maintain and prove the quality of the elements and thus satisfy the stability requirement.

Figure 4a depicts a cavity which extends beyond the submesh interfaces (because two of the cavity BHGFAC triangles $t \in M_1$ and $t^* \in M_2$ are non-local to submesh M_0) in order to guarantee the quality of the mesh. The extension of the cavity beyond the interfaces is a source of intensive communication. However, as Figure 4b shows PODM can tolerate the communication by concurrently refining other regions (e.g. compute a new cavity ABCDE) of the submeshes while it waits for remote data (e.g. the partially completed cavity BCAF) to arrive (eg. rest of the cavity BFGH). Unfortunately, the concurrent refinement can lead the violation of the loop invariant by creating non-conforming meshes and/or the violation of the Delaunay property as is the case in Figure 4a where the point P_j is within the circumcenter of $\triangle P_l CA$ which is a newly created triangle from the triangulation of the cavity (BHGFAC) that corresponds to the point P_1. These violations are resolved at the cost of setbacks and frequent message polling shown in the performance graph of Figure 4c. With some additional communication cost the PODM becomes domain decomposition independent and moreover re-distributes new elements as they are generated (Figure 4d).

In summary, PODM does not depend on good domain decompositions before, during and after parallel meshing at the cost of being labor intensive approach. PODM is a stable and tightly coupled method, with zero code re-use.

Decoupled Methods

In [40, 53] the authors present two approaches which achieve 100% code re-use and eliminate communication and synchronization. Both approaches rely on continuous domain decomposition and decouple the individual subdomains (subproblems) so that they can be meshed independently. Earlier, in [9] the authors presented similar approach for the parallel triangulation of a set of fixed points.

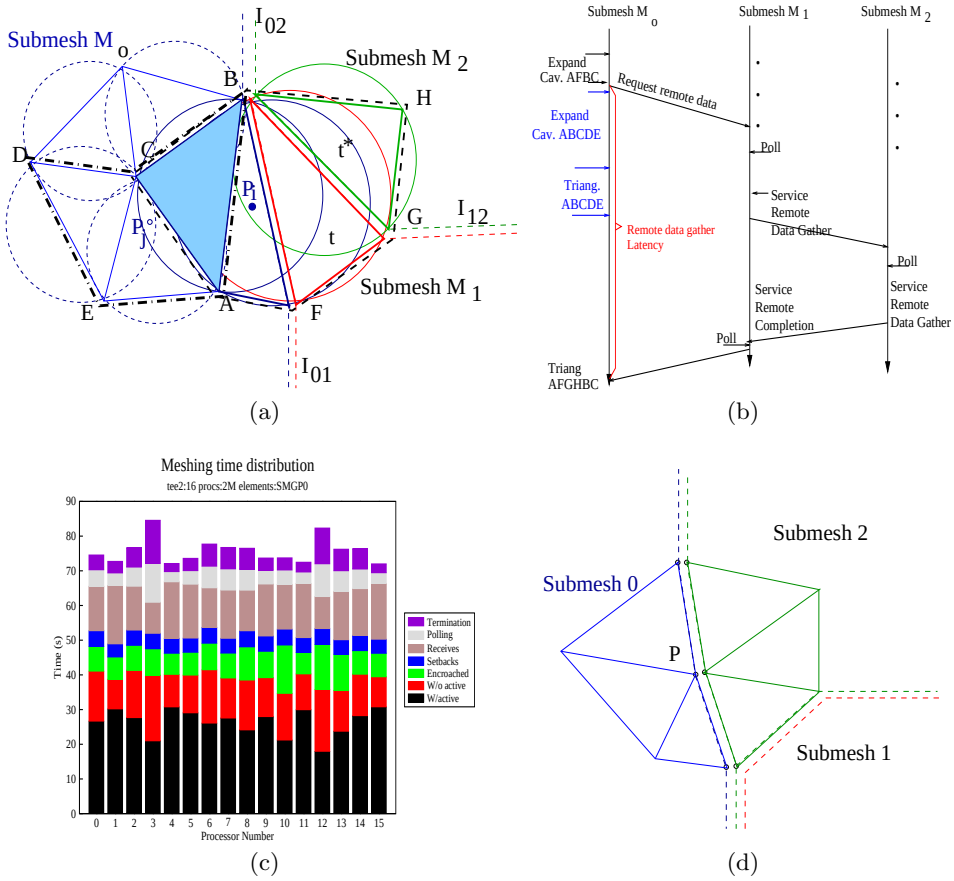


Fig. 4. a) cavity extension beyond submesh interfaces, b) time diagram with concurrent point insertion, c) a breakdown of execution time for PODM, and finally d) the refinement of a cavity with simultaneous distribution of the newly created elements.

Parallel Projective Delaunay Meshing

The Parallel Projective Delaunay Meshing (P^2DM) method [40] starts by sequentially meshing the external surfaces of the geometry and by precomputing domain separators whose facets are Delaunay-admissible (i.e., the precomputed interface faces of the separators will appear in the final Delaunay mesh). The separators decompose the continuous domain into subdomains which are meshed in parallel using a sequential Delaunay mesh generation method on each of the processors.

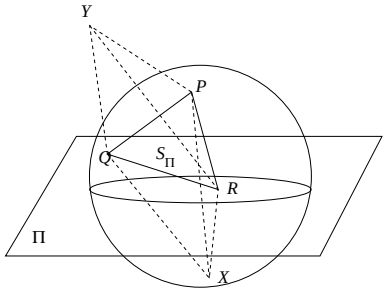


Fig. 5. The circumcenter of the face △PQR lies on the plane Π which helps define a separator S_Π. Note that △PQR $\in$ S_Π.

The basic idea for computing Delaunay-admissible separators can be explained easier in the context of the parallel triangulation of a convex hull for a set of points $S \in \mathcal{R}^3$ [38, 32]. The convex hull of a set of points S is decomposed in two subdomains by computing a Delaunay admissible separator as follows: First, the position of a surface (in practice a plane Π) is computed using an Inertia Axis Decomposition method [35]. The plane Π decomposes the convex hull of S into two almost equal pieces (in terms of points). Then the algorithm finds all faces (P,Q,R) $\in R^3$ (see Figure 5) for which there is an empty sphere whose center lies on the plane Π and passes through the points P,Q,R. These faces constitute a polyhedral separator S_Π which decomposes the domain into two subdomains assuming that the corresponding tetrahedra PQRX and PQRY contain the centers of their respective circumscribed spheres i.e., the quality of the initial mesh around the separators is very good which requires substantial refinement around the separators. In [40] it is shown that the faces of the polyhedral separator S_Π will appear in the final Delaunay triangulation of the convex hull. The generalization of the idea to complex geometries is possible, however it is much more difficult and it is explained in [40].

It is possible that the pre-constructed separators can not be Delaunay-admissible [40] and the whole process has to start from the beginning. This is a very difficult problem which for 2D has been solved in [53] using a different approach.

Parallel Delaunay Domain Decoupling PD^3 Method

The PD^3 method [53] like P^2DM is based on continuous domain decomposition. PD^3, for the domain decomposition of 2D geometries, uses medial axis of the domain and relies on the following simple geometric property [53]:

Lemma 1. *Let $MA(\Omega)$ be the medial axis of Ω and b a contact point of $c \in MA(\Omega)$. The angles formed by the segment cb and the tangent of the boundary $\partial\Omega$ at b are at least $\pi/2$.*

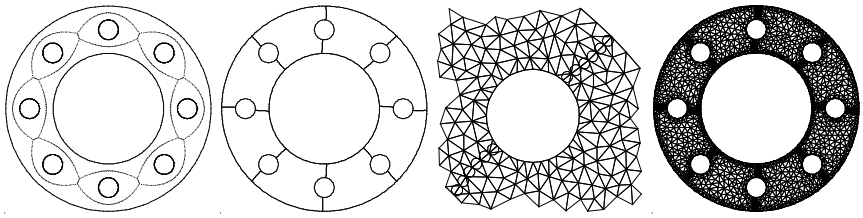


Fig. 6. The Medial Axis Transformation (a) which in turn is used to achieve high quality domain decomposition (b). For PD^3 the interfaces of the subdomains are refined (c) in a pre-processing step in order to decouple the subdomains which are refined independently (d).

The medial axis of a domain Ω is approximated by Voronoi points of a discretization of the domain. Figure 6a depicts the medial axis approximation and a 8-way partition (b) for the same geometry. The level of the discretization of the boundary determines the quality of the approximation of the medial axis. However, the goal in [53] is not to approximate accurately the medial axis, but to obtain good angles from the separator. Therefore, the criteria for the discretization of the domain are determined from the quality of the angles formed between the separators and the external boundary of the domain [54].

After the decomposition of the domain (see Figure 6b), PD^3 constructs a "zone" around the interfaces of the submeshes. The "zone" consist of the union of all diametral circles of the interface edges (see Figure 6c). The interfaces of the subdomains are refined using the lfs of the original domain. This leads into an overrefinement of the final distributed mesh. Experimental data from PD^3 (see Table 1) suggest that the overrefinement is not as high as one could expect. However, the authors of [53] are working on a new approach which will use adaptive domain decomposition [54] and different lfs for different interfaces of the subdomains. This method is expected to reduce overrefinement of the interfaces and produce well graded meshes [52].

Table 1. Overrefinement data as we increase the number of subdomains for the decomposition of a cross section of a rocket pipe model.

Subs	1	16	32	64	128
Elms :	21,0**16,403**	21,0**16,857**	21,0**18,522**	21,0**30,711**	21,0**44,689**
ORef.Elms/Sub	0	28	66	379	299

In [53] the authors prove that sequential Delaunay meshers will not insert any new points within a zone around the subdomain interfaces i.e., the sequential Delaunay meshing on the individual submeshes can terminate without

inserting any new points on the interfaces and thus eliminate communication and modifications of the sequential codes. This way, the problem of parallel meshing is reduced into a "proper" domain decomposition and a discretization of interfaces. However, the construction of decompositions that can decouple the mesh is a challenging problem, since its solution is based on medial axis which is very expensive and difficult to construct (even to approximate) for complex 3-dimensional geometries [42, 83, 31].

Partially Coupled Methods

The parallel tightly-coupled and decoupled methods we have seen so far address some of the parallel mesh generation requirements we described in Section 2. For example, PODM is a 3D stable and domain decomposition independent, but it is zero code re-use with high communication method; while P^2DM and PD^3 address the code re-use and communication issues, but their applicability in 3D is limited by the Delaunay-admissible and domain decomposition problem, respectively. In the rest of this section we present two partially coupled methods that make an attempt to balance trade-offs between all three requirements and the domain decomposition problem at the cost of some communication.

Parallel Constrained Delaunay Meshing (PCDM) Method

In order to address the communication and synchronization problem in [21], the authors developed the PCDM which is asynchronous and can reduce the variable and unpredictable communication patterns to irregular but bulk communication.

The PCDM [21] is based on the Constrained Delaunay Triangulation [20] and a discrete DD method. Each submesh is treated as an independent mesh defined by external boundary (if any) and/or constrained edges which are the edges of the interfaces between any pair of adjacent submeshes.

Intuitively, the constrained Delaunay triangulation is as close as one can get to the Delaunay triangulation given that one needs to preserve certain (constrained) edges and internal boundaries. It has been shown in [20] that the constrained internal edges do not affect the quality of the resulting mesh more than the edges and faces that define the external boundary. However, one might be able to identify such boundaries (interfaces for the PDCM) in the resulting mesh by noting the way in which triangle edges are aligned. Using the idea of a constrained Delaunay mesh generation one can introduce in the mesh artificial constrained edges (interfaces) which decompose the mesh into submeshes and can be meshed almost independently.

By the definition of the constrained Delaunay mesh, points inserted on one side of an interface have no effect on triangles on the other side; thus, no synchronization is required during the element creation process. In addition, communication between submeshes is tremendously simplified: the only

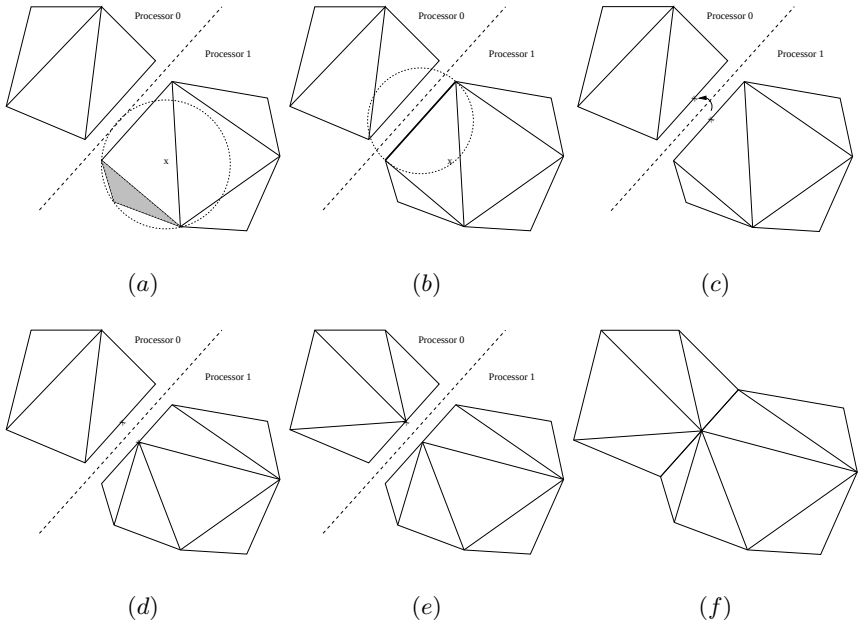


Fig. 7. Processor P1 inserts a new point (a) which is encroaching upon an interface edge (b). Then P1 discards the new point and inserts the midpoint of the encroached edge (c) while at the same time it sends a request to split the same interface edge on processor P0. Processor P0 computes the cavity of the midpoint (d). The triangulation of the cavities (e) and (f) of the midpoint of the interface edge results in a new conforming and distributed Delaunay (in the CDT sense) triangulation which guarantees the quality of the elements.

message between adjacent processes is of the form [21]: "Split this interface (i.e., constrained) edge" if a newly inserted point encroaches (see Fig. 7) upon an interface edge. Since interface edges are always split exactly in half, no additional information needs to be communicated.

The PCDM is an asynchronous with bulk communication and thus partially coupled method. Moreover, the number and size of messages can be reduced by message aggregation [19]. Although this optimization improves the performance of the PCDM it has its own problems when many "Split this interface edge" messages are delayed. This causes performance degradation due to: (1) the large number of accumulated messages which can consume memory, (2) redundant computation (by delaying messages), from neighboring processors which are unaware of each other's interface splits. In [19] these problems are addressed by a mechanism which adaptively changes the number of messages allowed to be aggregated before a low-level message is send.

However, code re-use remains a problem due to "Split this interface edge" message and optimizations required for reducing the fine-grain communication to a bulk and asynchronous message passing.

Parallel Delaunay Refinement (PDR) Method

The above tightly coupled (PODM) and partially coupled (PCDM) methods [62, 25, 21] require algorithm re-structuring and thus lead to completely new implementations for parallel Delaunay mesh generation. The implementation of sequential mesh generation codes is labor intensive and requires multi-disciplinary effort; it takes about ten to fifteen years to develop the algorithmic and software infrastructure for sequential industrial strength mesh generation libraries. Moreover, improvements in terms of quality, speed, and functionality are open ended and permanent which makes the task of delivering state-of-the-art parallel mesh generation codes much more difficult.

This problem is addressed by P^2DM and PD^3 in [40, 53], where two decoupling methods are presented in order to use (without modifications) optimized and fully functional sequential codes on each of the subproblems and eliminate communication and synchronization. However, P^2DM can suffer setbacks due to difficulty of constructing Delaunay-admissible separators and PD^3, for 3D geometries, is expected to be suffer high pre-processing overhead due the construction (or approximation) of the medial axis.

With PDR in [18, 17] the authors try to balance trade-offs between the data decomposition, communication and code re-use i.e., maintain stability and achieve high code re-use using a simple domain decomposition method at the cost of some communication. The key idea of the PDR method is based on the concurrent point insertion of more than two points without calculating their corresponding cavities ahead of time in order to decide whether they violate the conformity and Delaunay properties of the mesh. PDR accomplishes this objective by introducing for the first time a practical Delaunay-independence criterion for concurrent point insertion [18]:

Theorem 1 *Let $\bar{r}$ be the upper bound on triangle circumradius in the mesh and $p_i, p_j \in \Omega \subset \mathbb{R}^2$. Then if $\|p_i - p_j\| \geq 4\bar{r}$, then independent insertion of p_i and p_j will result in a mesh which is both conformal and Delaunay.*

Theorem 1 is applicable throughout the run of the algorithm, since the execution of the Bowyer-Watson kernel, either sequentially [11, 96] or in parallel [62], does not violate the condition that $\bar{r}$ is the upper bound on triangle circumradius in the entire mesh [18]. However, checking the inequality of the theorem, for every pair of candidate points, would be quite expensive task. In [18] the authors present a simple block domain decomposition scheme[1] which guarantees that any pair of points in non-adjacent cells are far apart

[1]This scheme is based on a simple block decomposition for uniform mesh refinement [18] and octree decomposition for graded mesh refinement [17].

no less than $4\bar{r}$. To enforce the $\bar{r}$ circumradius bound in the mesh they derive the following relation which allows the use of a standard sequential Delaunay refinement algorithm/software like Triangle [85] for preprocessing [18]:

Theorem 2 *If $\bar{\rho}$ and $\bar{\Delta}$ are upper bounds on triangle circumradius-to-shortest edge ratio and area, respectively, then $\bar{r} = 2(\bar{\rho})^{3/2}\sqrt{\bar{\Delta}}$ is an upper bound on triangle circumradius.*

4.2 Advancing Front Based Methods

All five parallel Delaunay methods we present in Section 4.1 maintain the stability of the parallel mesher. However, parallel finite element codes require only "good" quality of elements and the definition of quality depends on the field solver and varies from code to code. For example, in [80] although the stability is not guaranteed, it appears that the generated meshes are practical and of "good" quality. This raises the following two questions: Is the stability of parallel mesher important? Does the parallel mesh generation without the stability requirement become easier?

The answer to the first question depends on the upstream solver. Regarding the second question, our experience[2] suggests that even if we relax the stability criterion the problem of parallel mesh generation does not become easier. In fact, the termination problem (which is even more fundamental than the stability) becomes, for some cases, a very important issue. In some cases, subdomains or submeshes obtained from state-of-the-art partitioning libraries can not be meshed even by industrial strength advancing front sequential meshers. Parallel mesh smoothing techniques [58] are helpful, but do not work always.

There is a trade-off between the domain decomposition and the capability of the sequential mesher required to mesh the individual subdomains. A balance between the two is important not only for stability but even for termination. Two successful Parallel Advancing Front Techniques [57, 30] address this issue by what we refer to as *guided re-partitioning or shifting of the separators*. In [57] the authors present a tightly coupled method for shared memory machines and in [30] the authors present a partially coupled method for distributed memory machines. We review both methods in the rest of this section.

Tightly Coupled Methods

Lóhner et al. in [57] revisit a partially coupled Parallel Advancing Front Technique (PAFT) they developed in [56] (see bellow) in order to address the termination, stability, and code re-use requirements. In [57] they address these issues by developing a PAFT for shared memory computers (PAFT_{SM}). However, instead of generating and partitioning a very fine-grain octree as in [59]

[2] From the implementation of a method similar to one appeared in [80].

on a single processor, for the whole geometry, they use an octree to identify the zones where elements can be introduced concurrently. They set the edge length of the smallest octree box to be an order of magnitude larger than the specified size of elements and they use the "shift and regrid" technique, but in a completely different way from the method in [34]. The PAFT$_{SM}$ is broken into two phases: (1) the AFT phases and (2) "shift" or as we call it here guided re-partitioning phase. At each AFT phase the active front expands and a new one is created. The process continues until the whole domain is meshed. The PAFT$_{SM}$ method synchronizes at the beginning of each AFT phase in order to sequentially refine and re-partition the global octree, for the new active front, whose leaves will be refined in parallel. The method is suitable for shared memory machines but can not be used in large-scale distributed memory parallel platforms, because of the global synchronization required between the mesh generation and re-partitioning phases.

The PAFT$_{SM}$ is stable and code re-use is achieved at the cost of global synchronization which is not expensive on shared memory machines.

Partially Coupled Methods

In [56] Lóhner et al. introduced the first 2D PAFT. The initial mesh is subdivided into submeshes using a discrete domain decomposition approach. Each submesh is further separated into an interior region and interface regions, where interface regions of a submesh are defined to be the set of elements that are adjacent to elements that belong to different submehses. The interior regions of each submesh are refined independently. The interface regions and then the corners are refined once all the interior and interface regions are meshed, respectively (a posteriori approach). The order of meshing interface and interior regions can change i.e., interfaces can be refined first (a priori approach) and the interior regions refined last [56]. The submeshes synchronize locally, because no new elements can be inserted in the interfaces and corner regions before the meshing of adjacent interior and interface regions, respectively. The pre-computed interface regions work well for AFT because they create buffer zones which fully decouple the interior regions of the submeshes.

Parallel Octree AFT (POAFT) Method

The 3D POAFT in [30], contrary to the PAFT in [56], is based on continuous domain decomposition method. The POAFT method generates a distributed coarse-grain octree using a divide-and-conquer algorithm. The terminal octants and the geometric model of a domain define the subdomains. The terminal octants of the octree are classified into: interior, interface, boundary, and complete. Interface octants have at least one adjacent octant which is not local. Boundary octants include mesh entities from the input surface mesh. Complete octants have no front faces in their volumes. The subdomains represented as subtrees (on each processor) which are refined further until their

leaves reach to a predefined size to use tetrahedral meshing templates. The new octrees are repartitioned (using stop-and-repartition methods) in order to guarantee load balancing during the execution of meshing templates. After the re-distribution of interior octants, mesh templates are applied so that the triangulations conform on both sides of interface-octant faces. The interior octants of one processor are independent of the interior octants in other subdomains and thus can be meshed in parallel. At this step code re-use is high, since the meshing templates of sequential octree meshing code [82] can be used on each processor. Existing scalar meshing templates for the interface octants can be used, but some communication will be required during the meshing process. Instead in [30] meshing templates were re-designed in order to guarantee conformity without compromising stability and eliminating communication. The potential for ambiguous splits of faces is addressed and resolved in [70].

Before the boundary octants are refined and meshed a re-partitioning might take place if it is necessary. Any parallel partitioning algorithm can be used; in [30] the parallel recursive inertia bisection method is applied. The meshing of boundary octants is a challenging task. Every processor applies a tree-based face removal procedure [30] in order to connect the input surface mesh with the mesh of the interior octants. The face removal (from the active front) is a basic operation in AFT and it consists of connecting a front mesh face to a target mesh vertex which is drawn from a "neighborhood" of the face [30]. In the parallel face removal, portion of the "neighborhood" might be on a remote processor and a target vertex can not be found locally; in this case the face removal is postponed. This will create unmeshed regions between the terminal interface boundary octants and input surface mesh. In [30] active terminal and boundary interface octants are repartitioned so that the remaining unmeshed "neighborhoods" become local and thus the face removal becomes a local operation. This permits code re-use. This process is repeated until there are no unmeshed regions. The "guided" repartitioning is a very challenging problem.

4.3 Edge Subdivision Based Methods

Parallel Edge Subdivision (PES) methods have been used successfully for both 2D domains [97, 46] and 3D geometries [15, 30, 66, 79]. PES methods use discrete DD for data decomposition and their termination and stability does not depend on the geometric properties of the submeshes. Once a coarse mesh is partitioned into submeshes, the individual submeshes are refined in parallel by splitting tetrahedra using sequential subdivision techniques. The longest-edge bisection method [73, 76] is the most commonly used for parallel refinement/derefinement [97, 46, 15, 79]. In 2D an element is refined into two triangles by adding an edge defined by the longest-edge midpoint and its opposite vertex, while in 3D an element is refined into two tetrahedra by adding a triangle defined by the longest-edge midpoint and its two opposite

vertices. The longest-edge bisection technique is attractive because it simplifies the management of intermediate non conforming points throughout the process. With the introduction of terminal-edges in [79] this management is localized in a similar way the cavity localizes the computation of Delaunay based methods.

Like all parallel mesh generation methods PES refinement methods should satisfy all three requirements we listed in the introduction of this chapter. Existing PES methods address some of these requirements successfully and have the potential to meet all the requirements in the future. In [13] the authors present a termination proof, for parallel longest-edge bisection algorithms, using Dijkstra's general termination algorithm [36, 6]. Moreover, they prove the stability and even show that the mesh refined in parallel is identical to a sequentially generated mesh.

The scalability of PES methods depends on the way they address the *refinement collision*: *more than one processor split concurrently two different copies of the same interface edge*. Other factors that affect the scalability is the choice of dynamic load balancing methods and the degree of code reuse. For example, frequent use of stop-and-repartition methods due to global barrier operations can deteriorate the scalability of computationally inexpensive parallel mesh generation methods [3]. In general parallel mesh generation methods that do not take advantage of highly optimized sequential codes have difficulty to demonstrate good scalability against the best sequential codes.

In [66] it has been shown that 100% code re-use is possible at the cost of 10% overhead by putting a wrapper around the sequential data structure in order to handle data distribution and remote memory accesses. Communication is another aspect of parallel codes that affects scalability. In [30] the authors present a number of subdivision templates that can be used to decouple the refinement on different processors and thus eliminate communication completely.

The main challenge in PES methods is the *collision refinement* problem. In order to achieve mesh conformity and correctness the interface faces between the submeshes should be subdivided the same way from all submeshes that share them. Thus interface edges that are subdivided in one submesh are marked to be subdivided from all other submeshes that share them. This causes communication which is handled by sending, at the end of the refinement of the interface faces, a message to submeshes that share refined faces and edges. Based on the communication and synchronization requirements for handling the refinement collision problem, the PES methods are classified into three categories: tightly coupled methods [79], partially coupled methods [46, 30, 15, 66] and decoupled methods [78].

Tightly Coupled Methods

The 3D Parallel Terminal-Edge (PTE) method described in [79] is an inherently decoupled method. However, the PTE method in [79] is implemented as

a coupled method. In [78] a new design and implementation is presented so that the stability and code re-use requirements are satisfied while the global synchronization for maintaining the global name of all bisected edges is eliminated.

Parallel Coupled Terminal-Edge (PCTE) Method

In this paragraph we will refer to the tightly coupled implementation of the PTE method as PCTE. The PCTE method is based on longest-edge bisection approach introduced by Rivara [72, 73, 76]. Triangles/ tetrahedra are refined by bisecting their longest-edge. The longest-edge bisection algorithm requires the management of sequences of intermediate non conforming mesh points throughout the refinement process. This complicates its parallel implementation because it requires some synchronization in order to handle the collision refinement and global name of newly inserted vertices, both are required to maintain the conformity of the distributed mesh. The PCTE method [79], although it requires zero communication between processors, relies on a central processor for global name-assignment of new mesh points. The use of the central processor limits the scalability of the method for more than 60 processors and reduces the speed (tetrahedra per second) of the method by an order of magnitude. However, in [78] the authors present a decoupled method and implementation which takes full advantage of the terminal-edge algorithm introduced in [79]. The terminal-edge of a longest edge propagation path of t, Lepp(t), is the longest-edge between all the edges involved in Lepp(t) including the boundary of the Lepp polygon [74, 75, 77]. We review this method at the end of this Section.

Partially Coupled Methods

Partially coupled methods resolve inconsistencies during the collision refinement by processing interface edges in 2D (or faces in 3D) using independent sets of elements [46] and by breaking the mesh refinement process into two phases [30, 15, 66]: computation (actual refinement of elements) and communication (exchange of information about the newly created points and elements due to refinement of interfaces).

Parallel Independent Set Method

In [46] the refinement of a 2D mesh takes place in phases (refinement of one independent set at a time). This guarantees the conformity of the mesh and the elimination of the collision refinement problem, since non-local adjacent elements never refine interface edges concurrently and the processors are always aware of bisections of their interface edges. Specifically, the authors in [46] use a vertex-based partition of a 2D mesh to generate P submeshes, where P is the number of processors. Then all non-local adjacent elements (i.e., elements that share an edge) and adjacent vertices to the elements and

vertices of submeshes are computed to create a layer of "ghost" mesh entities which are used to minimize communication in the independent set (IS) phase. The distributed memory implementation of the IS phase in [46] computes a distributed independent set $I = \cup_{i=1}^{P} I_{M_i}$, where $I_{M_i} = I \cap M_i$ and M_i is a submesh, as follows: a triangle $t_a \in I_{M_i}$ if: $\forall t_b \in adj(t_a)$ and one of the following three holds (1) $t_a, t_b \in M_i$, (2) $\rho(t_a) > \rho(t_b)$, and (3) t_b is not a marked triangle for refinement, where adj(t) is the set of adjacent triangles of t, and $\rho(t)$ is a unique random number assigned to each element in the mesh M_i, $i = 1, P$. Note that due to the ghost elements, the communication for checking the above conditions is eliminated. The algorithm requires communication only for: (a) the update of the bisections of ghost elements, and (b) a global reduction operation for termination. Both take place at the end of the refinement of an independent set. These two types of communication make the algorithm partially coupled, since experimental data in [46] indicate that the number of refinement phases (or loop iterations) is small (10 to 20) as the number of processors and the size of the mesh increase to 200 processors and a million elements, respectively.

Parallel Alternate Bisection Method

DeCougny et. al [30] addresses the collision refinement problem by using, first, alternate bisection on the interface faces then by applying region subdivision templates on the rest of the tetrahedra. After the mesh faces are subdivided, it is possible to create non-conforming interface edges on the interfaces. The non-conforming interface edges are sent to the corresponding adjacent submeshes that are refined by different processors. This will start a new mesh face subdivision followed by a communication phase, until no mesh faces need to be subdivided. Upon termination of face subdivisions, the mesh is conforming across the interfaces and then a region subdivision using sequential templates is applied in parallel to the rest of the interior tetrahedra.

Parallel Nested Elements Method

Castaños and Savage [15] have parallelized the non-conforming longest edge bisection algorithm both in 2D and 3D. In this case the refinement propagation implies the creation of sequences of non-conforming edges that can cross several submeshes involving several processors. This also means the creation of non-conforming interface edges which is particularly complex to deal with in 3D. To perform this task each processor P_i iterates between a no-communication phase (where refinement propagation between processors is delayed) and an interprocessor communication phase. Different processors can be in different phases during the refinement process, their termination is coordinated by a central processor P_0. The subdivision of an interface edge might leads to either a non conforming edge or to a conforming edge, but the creation of different copies (one per subdomain) of its midpoint. However, after the communication phase a remote cross reference for each newly created

interface edge midpoint along with *nested elements* information guarantee a unique logical name for these newly created vertices [14].

Decoupled Methods

The PTE method [78] in addition to the terminal-edge of a Lepp(t) takes full advantage of the terminal-star, which is the set of tetrahedra that share a terminal-edge. The terminal-star can play the same role in PTE the cavity plays in PCDM. Contrary to the method in [15] the terminal-star refinement algorithm completely avoids the management of non-conforming edges both in the interior of the submeshes and in the inter-subdomain interface. This eliminates the communication among subdomains and thus processors. Similarly to Castaños et al. the terminal-star method can terminate using a single processor as coordinator for adaptive mesh refinement i.e., when a global stopping criterion like the minimum-edge length of terminal-edges is not used.

The decoupled PTE algorithm and its implementation lead to an order of magnitude performance improvements compared to a previous tightly coupled implementation [78] of the same algorithm. Although the algorithm is theoretically scalable, our performance data indicate the contrary; the reason is the work-load imbalances and heterogeneity of the clusters we use. We will address these two issues in Section 6.

5 Taxonomy

	Mesh Generation Technique		
Coupling	Deluanay	Advancing Front	Edge Bisection
Tight	PODM	$PAFT_{SM}$	PCTE
Partial	PCDT, PDR	POAFT	PIS, PNE
None	P^2DM, PD^3	PAFT	PTE

Fig. 8. Taxonomy of Parallel Mesh Generation Methods.

The taxonomy in Figure 8 helps to clarify basic similarities and differences between parallel tetrahedral meshing methods. The taxonomy is based on the two attributes we used to classify the methods reviewed in this chapter: (i) the basic sequential meshing technique used for each subproblem and (ii) the degree of coupling between the subproblems. The coupling (i.e., the amount of

communication and synchronization between the subproblems) is determined by the degree of dependency between the subproblems.

6 Implementation

The complexity of implementing efficient parallel mesh generation codes arises from the dynamic, data-dependent, and irregular computation and communication patterns of the algorithms. This inherent complexity, when combined with challenges from using primitive tools for communication like message-passing libraries [86, 5], makes the development of parallel mesh generation codes even more time-consuming and error-prone.

In the rest of the section we focus on dynamic load balancing issue. The scientific computing community has developed application-specific runtime libraries and software systems [4, 35, 50, 65, 68, 98] for dynamic load balancing. These systems are designed to support the development of parallel multi-phase applications which are computationally-intensive and consist of phases that are separated by computations such as the global error estimation. In these cases the load-balancing is accomplished by dynamically repartitioning the data after a global synchronization [95]. Throughout this chapter we call this approach to load balancing the *stop-and-repartition* method.

The stop-and-repartition approaches are good for loosely-synchronous applications like iterative PDE solvers, however they are not well-suited for applications such as adaptive mesh generation and refinement. Because for asynchronous and not computation-intensive applications the global synchronization overhead can overwhelm the benefits from load balancing. This problem is exacerbated as the number of processors in the parallel system grows. In order to address this issue, the authors in [3] developed a *Parallel Runtime Environment for Multi-computer Applications* (PREMA).

PREMA is a software library which provides a set of tools to application developers via a concise and intuitive interface. It supports single-sided communication primitives which conform to the *active messages* paradigm [94], a global namespace, message forwarding mechanisms to cope with object/data migration and a preemptive dynamic load balancing [3].

Performance Evaluation

In the rest of this section we present some performance data that show the effects of two sources of imbalance: (1) work-load due to geometric complexity of the subdomains/submeshes, and (2) processor heterogeneity. The experimental study was performed on Sciclone [1] cluster at the College of William and Mary which consists of many different heterogeneous subclusters. We have used three subclusters: (1) Whirlwind subcluster which consists of 64 single-cpu Sun Fire V120 nodes (650 MHz, 1 GB RAM), (2) Tornado which consists of 32 dual-cpu Sun Ultra 60 nodes (360 MHz, 512 MB RAM) and (3)

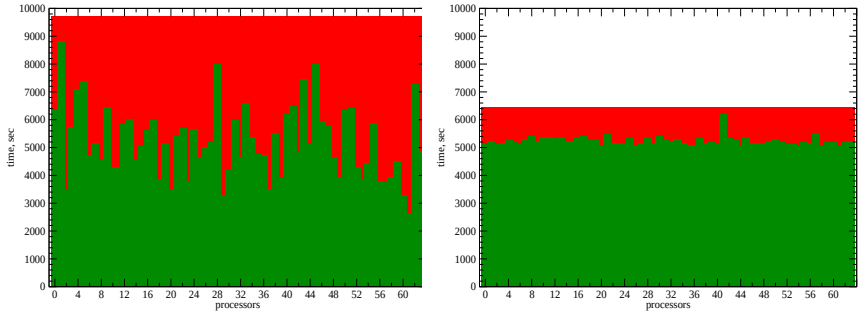


Fig. 9. Execution time of PAFT on Whirlwind subcluster with 64 homogeneous processors for the simplified human brain model without load balancing (left) and with load balancing using PREMA (right). The final mesh in both cases is 1.2 billion tetrahedrons.

Typhoon which consists of 64 single-cpu Sun Ultra 5 nodes (333 MHz, 256 MB RAM). The models we used are: (i) a cross-section of the rocket pipe (see Figure 1) and (ii) a simplified model of a human brain (see Figure 11, left).

Figure 9 shows the impact of dynamic load balancing on the performance of PAFT on the human brain model. The work-load imbalances are due to differences in the geometric complexity of the submeshes. The PAFT with dynamic load balancing (using PREMA) took 1.7 hours to generate the 1.2 billion elements while without dynamic load balancing it took 2.7 hours. The dynamic load balancing improved the performance of PAFT by more than 30%. Sequentially using Solidmesh [39], it takes three days one hour and 27 minutes, by executing the subdomains one at a time.

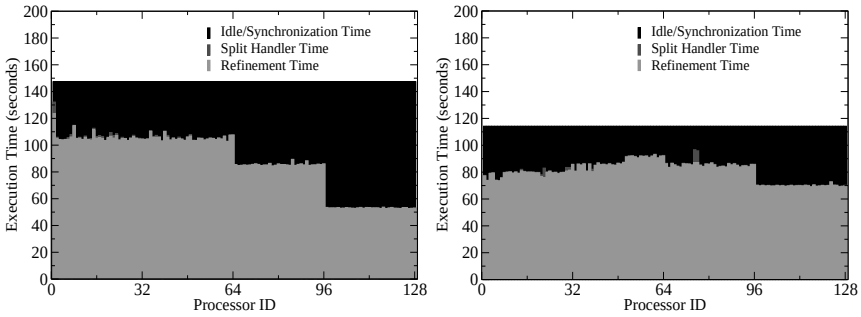


Fig. 10. The execution time of PCDM for the cross section of the rocket pipe whose data are equidistributed on 128 heterogenous processors; without load balancing (left) and with load balancing using PREMA (right).

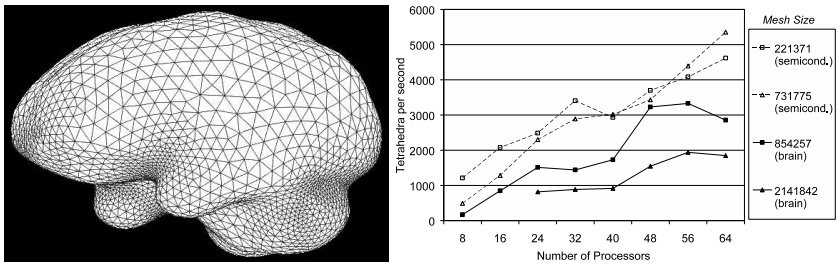


Fig. 11. Surface of the tetrahedral mesh for a simplified model of a human brain generated from an advancing front method [44].

Figure 10 shows the impact of dynamic load balancing (using PREMA) on the performance of the PCDM for the cross section of rocket pipe. Although we used state-of-the-art ab-initio data partition methods for equidistributing the data and thus the computation among all 128 processors, the imbalances are due to heterogeneity of the three different clusters; the first 64 processors are from Typhoon (slowest cluster), the next 32 processors are from Tornado and the last 32 processors are from Whirlwind (the fastest cluster). Again, the dynamic load balancing (using PREMA) improved the performance of parallel mesh generation by 23%.

Finally, the data from Figure 11 and Table 2 indicate the impact of work load imbalances due to: (1) the differences in the work-load of submeshes and (2) heterogeneity of processors using the PTE method. Figure 11(right), shows that the speed of the PTE method is substantially lower, for the brain model (see Figure 11, left), due to work-load imbalances; while for a more regular geometry (the semiconductor test case [78]), the PTE speed is almost twice higher, because of better load balancing due to more uniform point distribution. Also, Table 2 indicates a 19% slowdown in the PTE's speed once we increase the number of processors from 64 to 96 using additional 32 slower processors, despite the fact the PTE is a scalable method. Finally, a comparison between the speed data from the Figure 11 (right) and Table 2, for the brain model, indicate that the coupling (i.e., global synchronization) in the PCTE method slows down the speed of the code by an order of magnitude.

Table 2. PTE speed (in tetrahedra per second) for the simplified human brain model using min-edge = 2.0. The final mesh is about 2.4 million tetrahedra.

Processors	8	16	32	48	64	96
Whirlwind	5427	9920	16195	21890	29035	23571
Tornado	3852	7025	11408	15526	20312	23571

These data (and some more from [25, 3, 19, 78]) suggest that the tightly coupling methods should be used as a last resort. In addition, these data suggest that work-load imbalances are no longer a problem and it should not limit our creativity in the second round of our search for practical and effective parallel mesh generation methods. Runtime software systems like PREMA [3] can handle work-load imbalances quite successfully.

7 Future Directions

It takes about ten to fifteen years to develop the algorithmic and software infrastructure for *sequential industrial strength mesh generation libraries*. Moreover, improvements in terms of quality, speed, and functionality are open ended and permanent which makes the task of delivering state-of-the-art parallel mesh generation codes even more difficult.

This survey demonstrates that without compromising in the stability of parallel mesh generation methods it is possible for all three mesh generation classes of techniques to develop parallel meshing software using off-the-shelf sequential meshing codes.

An area with immediate high benefits to parallel mesh generation is domain decomposition. The DD problem as it is posed in Section 3 is still open for 3D geometries and its solution will help to deliver stable and scalable methods that rely on off-the-shelf mesh generation codes for Delaunay and Advancing Front Techniques. The edge subdivision methods are independent off the domain decomposition.

A longer term goal should be the development of both theoretical and software frameworks like PDR to implement new mesh generation methods which can: (1) take advantage of multicore architectures with more than two hardware contexts for the next generation of high-end workstations and (2) scale without any substantial implementation costs for clusters of high-end workstations.

Finally, a long term investment to parallel mesh generation is to attract the attention of mathematicians with open problems in mesh generation and broader impact in mathematics. For example, develop theoretical frameworks able to prove the correctness of single threaded guaranteed quality Delaunay theory in the context of partial order [71].

Acknowledgment

This work is supported by the IBM professorship at Brown University, the Alumni Memorial Endowment at the College of William and Mary, and National Science Foundation (NSF) grants: CNS-0312980, ANI-0203974, ACI-0085969, Career Award CCR-0049086, EIA-9972853, EIA-9726388. The experimental work was performed using computational facilities at the College

of William and Mary which were enabled by grants from Sun Microsystems, the NSF, and Virginia's Commonwealth Technology Research Fund. Andrey Chernikov made many insightful recommendations and Leonidas Linardakis made useful comments for the Delaunay methods. Most of the figures and tables are the result of my collaboration with my students from the College of William and Mary Kevin Barker, Andrey Chernikov, Andriy Fedorov, Brian Holinka, Leonidas Linardakis, from Notre Dame Demian Nave, and from Universidad de Chile Carlo Calderon and Daniel Pizarro. The anonymous referee for his many useful comments.

References

1. http://www.compsci.wm.edu/sciclone/. Last accessed, March 2005.
2. S. Baden, N. Chrisochoides, D. Gannon, and M. Norman, editors. *Structured Adaptive Mesh Refinement Grid Methods*. Springer-Verlag, 1999.
3. Kevin Barker, Andrey Chernikov, Nikos Chrisochoides, and Keshav Pingali. A load balancing framework for adaptive and asynchronous applications. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):183–192, February 2004.
4. A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. Walshaw. Dynamic load-balancing of finite element applications with the drama library. *Applied Mathematical Modeling*, 25:83–98, 2000.
5. A. Belguelin, J. Dongarra, A. Geist, R. Manchek, S. Otto, and J. Walpore. Pvm: Experiences, current status, and future direction. In *Supercomputing '93 Proceedings*, pages 765–766, 1993.
6. Dimitri Bertsekas and John Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.
7. Topping B.H.V. and B Cheng. Parallel adaptive quadrilateral mesh generation. *Computers and Structures,*, 73:519–536, 1999.
8. G. E. Blelloch, J.C. Hardwick, G. L. Miller, and D. Talmor. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 24:243–269, 1999.
9. G. E. Blelloch, G. L. Miller, and D. Talmor. Developing a practical projection-based parallel Delaunay algorithm. In *12th Annual Symposium on Computational Geometry*, pages 186–195, 1996.
10. H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of speech and Visual Form*, pages 362–380. MIT Press, 1967.
11. Adrian Bowyer. Computing Dirichlet tesselations. *Computer Journal*, 24:162–166, 1981.
12. Hardwick Jonathan C. Implementation and evaluation of an efficient 2d parallel Delaunay triangulation algorithm. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures,*, 1997.
13. J.G. Castanos and J.E. Savage. Parallel refinement of unstructured meshes. In *Proceedings of the IASTED, International Conference of Parallel and Distributed Computing and Systems*, 1999.

14. José G. Castaños and John E. Savage. The dynamic adaptation of parallel mesh-based computation. In *SIAM 7th Symposium on Parallel and Scientific Computation*, 1997.
15. José G. Castaños and John E. Savage. Pared: a framework for the adaptive solution of PDEs. In *8th IEEE Symposium on High Performance Distributed Computing*, 1999.
16. Min-Bin Chen, Tyng Ruey Chuang, and Jan-Jan Wu. Efficient parallel implementations of near Delaunay triangulation with high performance Fortran. *Concurrency: Practice and Experience*, 16(12), 2004.
17. Andrey N. Chernikov and Nikos P. Chrisochoides. Parallel guaranteed quality planar Delaunay mesh generation by concurrent point insertion. In *14th Annual Fall Workshop on Computational Geometry*, pages 55–56. MIT, November 2004.
18. Andrey N. Chernikov and Nikos P. Chrisochoides. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 48–57. ACM Press, 2004.
19. Andrey N. Chernikov, Nikos P. Chrisochoides, and L. Paul Chew. Design of a parallel constrained Delaunay meshing algorithm, 2005.
20. L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.
21. L. Paul Chew, Nikos Chrisochoides, and Florian Sukup. Parallel constrained Delaunay meshing. In *ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation*, pages 89–96, Northwestern University, Evanston, IL, 1997.
22. N. P. Chrisochoides, Elias Houstis, and John Rice. Mapping algorithms and software environment for data parallel PDE iterative solvers. *Special issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming*, 21(1):75–95, 1994.
23. Nikos Chrisochoides. An alternative to data-mapping for parallel iterative PDE solvers: Parallel grid generation. In *Scalable Parallel Libraries Conference*, pages 36–44. IEEE, 1993.
24. Nikos Chrisochoides, C.E. Houstis, E.N.Houstis, P.N. Papachiou, S.K. Kortesis, and J.R. Rice. Domain decomposer: A software tool for partitioning and allocation of PDE computations based on geometry decomposition strategies. In *4th International Symposium on Domain Decomposition Methods*, pages 341–357. SIAM, 1991.
25. Nikos Chrisochoides and Démian Nave. Parallel Delaunay mesh generation kernel. *Int. J. Numer. Meth. Engng.*, 58:161–176, 2003.
26. Nikos Chrisochoides and Florian Sukup. Task parallel implementation of the BOWYER-WATSON algorithm. In *Proceedings of Fifth International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, 1996.
27. Nikos P. Chrisochoides. A new approach to parallel mesh generation and partitioning problems. *Computational Science, Mathematics and Software*, pages 335–359, 2002.
28. N.P. Chrisochoides. Multithreaded model for load balancing parallel adaptive computations. *Applied Numerical Mathematics*, 6:1–17, 1996.
29. P. Cignoni, D. Laforenza, C. Montani, R. Perego, and R. Scopigno. Evaluation of parallelization strategies for an incremental Delaunay triangulator in E^3. *Concurrency: Practice and Experience*, 7(1):61–80, 1995.

30. H.L. De Cougny and M.S. Shephard. Parallel refinement and coarsening of tetrahedral meshes. *Int. J. Meth. Eng.*, 46(1101-1125), 1999.
31. Tim Culver. *Computing the Medial Axis of a Polyhedron Reliably and Efficiently.* PhD thesis, The University of North Carolina at Chapel Hill, 2000.
32. H. de Cougny and M. Shephard. *CRC Handbook of Grid Generation*, chapter Parallel unstructured grid generation, pages 24.1–24.18. CRC Press, Inc., 1999.
33. H. de Cougny and M. Shephard. Parallel volume meshing using face removals and hierarchical repartitioning. *Comp. Meth. Appl. Mech. Engng.*, 174(3-4):275–298, 1999.
34. Hugues L. de Cougny, Mark S. Shephard, and Can Ozturan. Parallel three-dimensional mesh generation on distributed memory mimd computers. *Engineering with Computers*, 12:94–106, 1995.
35. K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan. Design of dynamic load-balancing tools for parallel applications. In *Proc. of the Int. Conf. on Supercomputing*, Santa Fe, May 2000.
36. E. W. Dijkstra and C.S Sholten. Termination detection for diffusing computations. *Inf. Proc. Lettres*, 11, 1980.
37. Herbert Edelsbrunner and Damrong Guoy. Sink-insertion for mesh improvement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pages 115–123. ACM Press, 2001.
38. Pascal Jean Frey and P. L. George. *Mesh Generation: Applications to Finite Element.* Hermis; Paris, 2000.
39. J. Gaither, D. Marcum, and B. Mitchell. Solidmesh: A solid modeling approach to unstructured grid generation. In *7th International Conference on Numerical Grid Generation in Computational Field Simulations*, 2000.
40. J. Galtier and P. L. George. Prepartitioning as a way to mesh subdomains in parallel. In *Special Symposium on Trends in Unstructured Mesh Generation*, pages 107–122. ASME/ASCE/SES, 1997.
41. P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing: Applications to Finite Element.* Hermis; Paris, 1998.
42. Halit Nebi Gürsoy. *Shape interrogation by medial axis transform for automated analysis.* PhD thesis, Massachusetts Institute of Technology, 1989.
43. B. Hendrickson and R. Leland. The chaco user's guide, version 2.0. Technical Report SAND94-2692, Sandia National Laboratories., 1994.
44. Alan M. Shih Ito, Yasushi and Bharat K. Soni. Reliable isotropic tetrahedral mesh generation based on an advancing front method. In *Proceedings 13th International Meshing Roundtable, Williamsburg, VA, Sandia National Laboratories*, pages 95–106, 2004.
45. D. A. Jefferson. Virtual time. In *ACM Transactions on Programming Languages and Systems*, volume 7, pages 404–425, July 1985.
46. Mark T. Jones and Paul E. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In *Proceedings of the Scalable High-Performance Computing Conference*, 1994.
47. Clemens Kadow. Adaptive dynamic projection-based partitioning for parallel Delaunay mesh generation algorithms. In *SIAM Workshop on Combinatorial Scientific Computing*, February 2004.
48. Clemens Kadow and Noel Walkington. Design of a projection-based parallel Delaunay mesh generation and refinement algorithm. In *Fourth Symposium on Trends in Unstructured Mesh Generation*, July 2003. http://www.andrew.cmu.edu/user/sowen/usnccm03/agenda.html.

49. Clemens Martin Kadow. *Parallel Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, May 2004.
50. S. Kohn and S. Baden. Parallel software abstractions for structured adaptive mesh methods. *Journal of Par. and Dist. Comp.*, 61(6):713–736, 2001.
51. Ravi Konuru, Jeremy Casas, Robert Prouty, Steve Oto, and Jonathan Walpore. A user level process package for pvm. In *Proceedings of Scalable High-Performance Computing Conferene*, pages 48–55. IEEE, 1997.
52. Leonidas Linardakis and Nikos Chrisochoides. Parallel Delaunay domain decoupling method for non-uniform mesh generation. *SIAM Journal on Scientific Computing*, 2005.
53. Leonidas Linardakis and Nikos Chrisochoides. Parallel domain decoupling Delaunay method. *SIAM Journal on Scientific Computing*, in print, accepted Nov. 2004.
54. Leonidas Linardakis and Nikos Chrisochoides. Medial axis domain decomposition method. *ACM Trans. Math. Software*, To be submited, 2005.
55. R. Lober, T.J. Tautges, and R.A. Cairncross. The parallelization of an advancing-front, all-quadrilateral meshing algorithm for adaptive analysis. In *4th International Meshing Roundtable*, pages 59–70, October 1995.
56. R. Löhner, J. Camberos, and M. Marshal. *Unstructured Scientific Computation on Scalable Multiprocessors (Eds. Piyush Mehrotra and Joel Saltz)*, chapter Parallel Unstructured Grid Generation, pages 31–64. MIT Press, 1990.
57. Reinald Löhner and Juan R. Cebral. Parallel advancing front grid generation. In *Proceedings of the Eighth International Meshing Roundtable*, pages 67–74, 1999.
58. Freitag Lori, Mark Jones, and Paul Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proceedings 4th International Meshing Roundtable*, pages 47–58, 1995.
59. R. Perucchio M. Saxena. Parallel FEM algorithm based on recursive spatial decomposition. *Computers and Structures*, 45(9-6):817–831, 1992.
60. S. N. Muthukrishnan, P. S. Shiakolos, R. V. Nambiar, and K. L. Lawrence. Simple algorithm for adaptative refinement of three-dimensionalfinite element tetrahedral meshes. *AIAA Journal*, 33:928–932, 1995.
61. Démian Nave, Nikos Chrisochoides, and L. Paul Chew. Guaranteed: quality parallel delaunay refinement for restricted polyhedral domains. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 135–144. ACM Press, 2002.
62. Démian Nave, Nikos Chrisochoides, and L. Paul Chew. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications*, 28:191–215, 2004.
63. T. Okusanya and J. Peraire. Parallel unstructured mesh generation. In *5th International Conference on Numerical Grid Generation on Computational Field Simmulations*, pages 719–729, April 1996.
64. T. Okusanya and J. Peraire. 3D parallel unstructured mesh generation. In S. A. Canann and S. Saigal, editors, *Trends in Unstructured Mesh Generation*, pages 109–116, 1997.
65. L. Oliker and R. Biswas. Plum: Parallel load balancing for adaptive unstructured meshes. *Journal of Par. and Dist. Comp.*, 52(2):150–177, 1998.
66. Leonid Oliker, R. Biswas, and H. Gabow. Parallel tetrahedral mesh adaptation with dynamic load balancing. *Parallel Computing Journal*, pages 1583–1608, 2000.

67. S. Owen. A survey of unstructured mesh generation. Technical report, ANSYS Inc., 2000.
68. M. Parashar and J. Browne. Dagh: A data-management infrastructure for parallel adaptive mesh refinement techniques. Technical report, Dept. of Comp. Sci., Univ. of Texas at Austin, 1995.
69. N.M. Patrikalakis and H.N. Gürsoy. Shape interrogation by medial axis transform. In *Design Automation Conference (ASME)*, pages 77–88, 1990.
70. Pilippe Pebay and David Thompson. Parallel mesh refinement without communication. In *Proceedings of International Meshing Roundtable*, pages 437–443, 2004.
71. Stratos Prassidis and Nikos Chrisochoides. A categorical approach for parallel Delaunay mesh generation, July 2004.
72. M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.
73. M. C. Rivara. Selective refinement/derefinement algorithms for sequences of nested triangulations. *International Journal for Numerical Methods in Engineering*, 28:2889–2906, 1989.
74. M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40:3313–3324, 1997.
75. M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, 33:263–277, 2001.
76. M. C. Rivara and C. Levin. A 3d refinement algorithm for adaptive and multigrid techniques. *Communications in Applied Numerical Methods*, 8:281–290, 1992.
77. M. C. Rivara and M. Palma. New lepp algorithms for quality polygon and volume triangulation: Implementation issues and practical behavior. In *In Trends unstructured mesh generationi Eds: S. A. Cannan . Saigal, AMD,*, volume 220, pages 1–8, 1997.
78. Maria-Cecilia Rivara, Carlo Calderon, Daniel Pizarro, Andriy Fedorov, and Nikos Chrisochoides. Parallel decoupled terminal-edge bisection algorithm for 3d meshes. *(Invited) Engineering with Computers*, 2005.
79. Maria-Cecilia Rivara, Daniel Pizarro, and Nikos Chrisochoides. Parallel refinement of tetrahedral meshes using terminal-edge bisection algorithm. In *13th International Meshing Roundtable*, September 2004.
80. R. Said, N.P. Weatherill, K. Morgan, and N.A. Verhoeven. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering*, (177):109–125, 1999.
81. K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel diffusion schemes for repartitioning of adaptive meshes. Technical Report 97-014, Univ. of Minnesota, 1997.
82. M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 32:709–749, 1991.
83. Evan Conway Sherbrooke. *3-D shape interrogation by medial axial transform.* PhD thesis, Massachusetts Institute of Technology, 1995.
84. Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation.* PhD thesis, Carnegie Mellon University, 1997.

85. J.R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Proceedings of the First workshop on Applied Computational Geometry*, pages 123–133, Philadelphia, PA, 1996.
86. M. Snir, S. Otto, S. Huss-Lederman, and D. Walker. Mpi the complete reference. MIT Press, 1996.
87. A. Sohn and H. Simon. Jove: A dynamic load balancing framework for adaptive computations on an SP-2 distributed memory multiprocessor, 1994. Technical Report 94-60, Dept. of Comp. and Inf. Sci., New Jersey Institute of Technology, 1994.
88. Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings of the Eleventh International Meshing Roundtable*, pages 205–217, 2001.
89. Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör. Time complexity of practical parallel Steiner point insertion algorithms. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 267–268. ACM Press, 2004.
90. Moitra Stuti and Anutosh Moitra. Considerations of computational optimality in parallel algorithms for grid generation. In *5th International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 753–762, 1996.
91. T.K.H. Tam, M.A. Price, C.G. Armstrong, and R.M. McKeag. Computing the critical points on the medial axis of a planar object using a Delaunay point triangulation algorithm.
92. Y. Ansel Teng, Francis Sullivan, Isabel Beichl, and Enrico Puppo. A data-parallel algorithm for three-dimensional Delaunay triangulation and its implementation. In *SuperComputing*, pages 112–121. ACM, 1993.
93. Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill. *Handbook of Grid Generation*. CRC Press, 1999.
94. T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th Int. Symp. on Comp. Arch.*, pages 256–266. ACM Press, May 1992.
95. C. Walshaw, M. Cross, and M. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Par. and Dist. Comp.*, 47:102–108, 1997.
96. David F. Watson. Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. *Computer Journal*, 24:167–172, 1981.
97. Roy Williams. *Adaptive parallel meshes with complex geometry*. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, 1991.
98. X. Yuan, C. Salisbury, D. Balsara, and R. Melhem. Load balancing package on distributed memory systems and its application particle-particle and particle-mesh (P3M) methods. *Parallel Computing*, 23(10):1525–1544, 1997.