Algorithm TOMS-2006-0003: Parallel 2D Constrained Delaunay Mesh Generation

ANDREY N. CHERNIKOV
and
NIKOS P. CHRISOCHOIDES
Department of Computer Science
College of William and Mary

Delaunay refinement is a widely used method for the construction of guaranteed quality triangular and tetrahedral meshes. We present an algorithm and a software for the parallel constrained Delaunay mesh generation in two dimensions. Our approach is based on the decomposition of the original mesh generation problem into N smaller subproblems which are meshed in parallel. The parallel algorithm is asynchronous with small messages which can be aggregated and exhibits low communication costs. On a heterogeneous cluster of more than 100 processors our implementation can generate over one billion triangles in less than 3 minutes, while the single-node performance is comparable to that of the fastest to our knowledge sequential guaranteed quality Delaunay meshing library (the Triangle).

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; G.4 [**Mathematical Software**]: Parallel and Vector Implementations; I.3.5 [**Computing Methodologies**]: Computer Graphics—*Computational Geometry and Object Modeling*

General Terms: Algorithms, Design, Performance, Theory

Additional Key Words and Phrases: Delaunay triangulation, mesh generation, parallel refinement

1. INTRODUCTION

A number of applications which model complex physical and biological phenomena require fast construction of very large two-dimensional meshes. Sometimes even three-dimensional simulations make use of two-dimensional meshes, as is the case with the direct numerical simulations of turbulence in cylinder flows with very large Reynolds numbers [Dong et al. 2004] and the coastal ocean modeling for predicting storm surge and beach erosion in real-time [Walters 2005]. In both instances, planar meshes are generated in the xy-plane and replicated in the third dimension, which is the z-direction in the case of cylinder flows and bathemetric contours in the case of coastal ocean modeling. With the increase of the Reynolds number, the

Authors' addresses: Computer Science Department, College of William and Mary, Williamsburg, VA 23185, email: {ancher;nikos}@cs.wm.edu.

size of the mesh grows in the order of $Re^{9/4}$ [Karniadakis and Orszag 1993], which motivates the use of parallel distributed memory mesh generation algorithms. The accuracy and the convergence of the numerical solution strongly depends on the size and the shape of mesh elements, see e.g. [Shewchuk 2002b] and the references therein. In this paper, we present a parallel guaranteed quality Delaunay mesh generation algorithm, which allows to satisfy user-defined constraints on triangle size and shape.

The parallel mesh generation procedure we present in this paper decomposes the original mesh generation problem into N smaller subproblems which are solved in parallel, see [Chrisochoides 2005] for a survey on parallel mesh generation. The coupling of the subproblems determines the intensity of the communication and the degree of dependency (or synchronization) between them. Our algorithm is weakly coupled, i.e., asynchronous with small messages, and exhibits low communication costs. This paper completes a series of four different parallel mesh generation classes of methods that were developed in our group: (1) tightly coupled published in [Chrisochoides and Nave 2003; Nave et al. 2004], (2) partially coupled in [Chernikov and Chrisochoides 2004b; 2005; 2006a; 2006b], (3) weakly coupled to be presented in this paper, and (4) decoupled in [Linardakis and Chrisochoides 2006; 2005].

The tightly coupled method [Chrisochoides and Nave 2003; Nave et al. 2004] is the first to our knowledge practical provably-good parallel mesh refinement algorithm for polyhedral domains. This approach is based on the speculative execution model, i.e., rollbacks can occur whenever the concurrent insertion of two or more points can lead to an invalid mesh. Although more than 80% of the overhead due to remote data gather operations is masked, the experimental data suggest only $\mathcal{O}(\log P)$ speedup, P being the number of processors. In contrast, the speedup of the Parallel Constrained Delaunay Meshing (PCDM) algorithm presented in this paper is close to $\mathcal{O}(P)$.

In order to eliminate communication and synchronization, Linardakis and Chrisochoides [Linardakis and Chrisochoides 2006] developed a Parallel Domain Decoupling Delaunay (PD^3) method for 2-dimensional domains. The PD^3 is based on the idea of decoupling the individual submeshes so that they can be meshed independently with zero communication and synchronization. In the past similar attempts to parallelize Delaunay triangulations and implement Delaunay based mesh generation were presented in [Blelloch et al. 1996; Galtier and George 1997]. However, in [Linardakis and Chrisochoides 2006; 2005] the authors solve some of the drawbacks and improve upon the previously published methods, e.g., with respect to the existence of the domain decomposition. The PD^3 algorithm consists of two major components: the medial axis domain decomposition (MADD) and the boundary discretization. The domain decomposition step employs a discrete medial axis transform to divide the original domain Ω into a set of subdomains $\{\Omega_i \mid i = 1, \ldots, N\}$, such that the artificially created boundaries of subdomains are suitable for the subsequent guaranteed quality parallel meshing. In particular, the boundaries do not create very small angles and allow to achieve fairly small surface-to-volume ratio. During the boundary discretization step, additional (Steiner) points are inserted on the boundaries with the goal of satisfying the ter-

mination condition of Ruppert's Delaunay refinement algorithm [Ruppert 1995]. The spacing between the points is chosen to meet the worst-case theoretical bound which assures the termination of the Ruppert's algorithm without introducing any additional points on the boundaries. In the present paper, we describe an algorithm which refines the boundaries simultaneously with the interiors of the subdomains at the cost of introducing tolerable asynchronous communication. This allows to insert only those points along the boundaries that are dictated by the generation of the triangular elements inside the subdomains and to achieve optimal size meshes for the given quality criteria.

In [Chernikov and Chrisochoides 2004b; 2006b] we presented a theoretical framework and the experimental evaluation of a partially coupled parallel Delaunay refinement (PDR) algorithm for the construction of the uniform guaranteed quality Delaunay meshes. We then extended the PDR approach [Chernikov and Chrisochoides 2004a; 2005; 2006a] for the non-uniform case, when the element size is controlled by a user-defined function. The PDR algorithm allows to avoid solving the difficult domain decomposition problem and to use a simple block data distribution scheme instead. However, this induces noticeable communication costs, which can be significantly lowered if an efficient domain decomposition procedure is available. For the two dimensions, the domain decomposition problem has been solved [Linardakis and Chrisochoides 2006]. The PCDM method we present in this paper uses the MADD software [Linardakis and Chrisochoides 2005] to develop a parallel mesh refinement code with lower communication costs than the PDR approach and smaller practical mesh size than the complete PD^3 algorithm. Table III in Section 5 shows the quantitative differences in the performance of the PCDM and the PDR algorithms.

Blelloch, Hardwick, Miller, and Talmor [Blelloch et al. 1999] described a divide-and-conquer projection-based algorithm for constructing Delaunay triangulations of pre-defined point sets in parallel. The work by Kadow and Walkington [Kadow and Walkington 2003; Kadow 2004a; 2004b] extended [Blelloch et al. 1996; Blelloch et al. 1999] for parallel mesh generation and further eliminated the sequential step for constructing an initial mesh, however, all potential conflicts among concurrently inserted points are resolved sequentially by a dedicated processor [Kadow 2004b]. The approach we present here eliminates this bottleneck by introducing only messages between the nodes that refine adjacent subdomains.

Spielman, Teng, and Üngör [Spielman et al. 2001; Üngör 2002] presented the first theoretical analysis of the complexity of parallel Delaunay refinement algorithms. In [Spielman et al. 2004] the authors developed a more practical algorithm. However, the assumption is that the whole point set is completely retriangulated after each parallel iteration [Teng 2004].

Chew [Chew 1987] presented an optimal $\mathcal{O}(n \log n)$ time divide-and-conquer algorithm for the construction of constrained Delaunay triangulations for a given set of points and segments. Later, Chew, Chrisochoides, and Sukup [Chew et al. 1997] used the idea of a constrained Delaunay triangulation to develop a parallel constrained Delaunay mesh generation algorithm. In this paper we present algorithm improvements like the elimination of global synchronization, message aggregation, termination detection algorithm described by Dijkstra [Dijkstra 1987], and effi-

cient representation of split points as well as a new implementation that utilizes a number of newly available software, such as the Medial Axis Domain Decomposition library [Linardakis and Chrisochoides 2005], the Metis graph partitioning library [Karypis and Kumar 1998], the Triangle Delaunay [Shewchuk 1996] triangulator, and the Robust Predicates [Shewchuk 1997a]. Moreover, we evaluate the algorithm and its most recent implementation on a large cluster of workstations with more than 100 nodes. As a result, more than one billion triangles can be constructed in less than three minutes.

In addition to the parallel Delaunay mesh generation methods that are directly related to this work, there are many more classes of parallel mesh generation methods: (1) octree/quadtree based methods [de Cougny et al. 1994; Löhner and Cebral 1999; Rypl and Bittnar 2001], (2) edge-subdivision based methods [Jones and Plassmann 1994; Rivara et al. 2006], and (3) a class of parallel out-of-core methods [Isenburg et al. 2006; Kot et al. 2006]. In [Isenburg et al. 2006] the authors compute a billion triangle mesh on a laptop in 48 minutes. Unfortunately, we cannot utilize their approach because they construct triangulations of predefined point sets, while we face a different problem, that of refining a mesh by computing and inserting new points which help improve the quality of the elements. There are also other parallel triangulation methods which include [Cignoni et al. 1995; Kohout et al. 2005]. A more extensive review of parallel mesh generation methods can be found in [Chrisochoides 2005].

In Section 2 we describe the guaranteed quality constrained Delaunay meshing problem. Then we present our parallel algorithm in Section 3. Section 4 contains selected software implementation details, and Section 5 presents our experimental results. We summarize in Section 6.

2. CONSTRAINED DELAUNAY MESHING

In this section, we present a very brief description of a constrained Delaunay mesh refinement kernel based on the Bowyer-Watson point insertion procedure, along with the notation we use throughout the paper.

We will denote point number i as p_i, the edge with endpoints p_i and p_j as $e\,(p_ip_j)$, and the triangle with vertices p_i, p_j, and p_k as $\triangle\,(p_ip_jp_k)$. When the vertices of a triangle are irrelevant, we will denote it by a single letter t. Let us call the open disk corresponding to a triangle's circumscribed circle its *circumdisk*.

Let V be a set of points in the domain $\Omega \subset \mathbb{R}^2$, and T be a set of triangles whose vertices are in V. Then the triangulation (mesh) $\mathcal{M}$ of V is said to be *Delaunay* if every triangle's circumdisk does not contain points from V. Delaunay triangulations have been studied extensively; see, for instance, [Shewchuk 1997b; 2002a; Ruppert 1995; George and Borouchaki 1998; Chew 1989] and the references therein. Among the reasons of the popularity of Delaunay triangulations are useful optimality properties (e.g., the maximization of the minimal angle) and the amenability to the rigorous mathematical analysis.

Typically, a Delaunay mesh generation procedure starts by constructing an initial Delaunay mesh, which conforms to the input vertices and segments, and then refines this mesh until the element quality constraints are met. The applications that use Delaunay meshes often impose two constraints on the quality of mesh elements: an

upper bound $\bar{\rho}$ on the circumradius-to-shortest edge ratio (which is equivalent to a lower bound on the minimal angle [Miller et al. 1995; Shewchuk 1997b]) and an upper bound $\bar{\Delta}$ on the element area.

The input to a planar triangular mesh generation algorithm includes a description of domain $\Omega \subset \mathbb{R}^2$, which is permitted to contain holes or have more than one connected component. We will use a *Planar Straight Line Graph* (PSLG), see [Shewchuk 1996], to delimit Ω from the rest of the plane. Each segment in the PSLG is considered *constrained* and must appear (possibly as a union of smaller segments) in the final mesh.

In this paper, we focus on parallelizing the Delaunay refinement stage, which is usually the most memory- and computation-expensive. The general idea of Delaunay refinement is to insert points in the circumcenters of triangles that violate the required bounds, until there are no such triangles left. We will extensively use the notion of *cavity* [George and Borouchaki 1998] which is the set of triangles in the mesh whose circumdisks include a given point p_i. We will use the symbol $\mathcal{C}(p_i)$ to represent the cavity of p_i and $\partial\mathcal{C}(p_i)$ to represent the set of edges which belong to only one triangle in $\mathcal{C}(p_i)$, i.e., external edges.

For our analysis, we will use the Bowyer-Watson (B-W) point insertion algorithm [Bowyer 1981; Watson 1981], which can be written as:

$$\begin{aligned} &V' \leftarrow V \cup \{p_i\}, \\ &T' \leftarrow T \setminus \mathcal{C}(p_i) \cup \{\triangle(p_i p_j p_k) \mid e(p_j p_k) \in \partial\mathcal{C}(p_i)\}, \end{aligned} \tag{1}$$

where $\mathcal{M} = (V, T)$ and $\mathcal{M}' = (V', T')$ represent the mesh before and after the insertion of p_i, respectively. The set of newly created triangles forms a *ball* [George and Borouchaki 1998] of point p_i, which is the set of triangles in the mesh that have p_i as a vertex.

Sequential Delaunay algorithms treat *constrained* segments differently from triangle edges [Shewchuk 1997b; Ruppert 1995]. A vertex p is said to *encroach upon* a segment s, if it lies within the open diametral lens of s [Shewchuk 1997b]. The diametral lens of a segment s is the intersection of two disks whose centers lie on each other's boundaries and whose boundaries pass through the endpoints of s. When a new point is about to be inserted and it happens to encroach upon a constrained segment s, another point is inserted in the middle of s instead [Ruppert 1995], and a cavity of the segment's midpoint is constructed and triangulated as before.

A triangulation is said to be *constrained Delaunay* if the open circumdisks of its elements do not contain any points visible from the interior of these elements [Shewchuk 1997b; Chew 1987; George and Borouchaki 1998]. The visibility between two points is considered to be obstructed if the straight line segment connecting these points intersects a pre-specified fixed (i.e., *constrained*) segment. For example, in Figure 1a the segment $e(p_1 p_3)$ is constrained and the triangle $\triangle(p_1 p_2 p_3)$ is constrained Delaunay, even though it does not satisfy the original Delaunay criterion since the point p_4 is inside its circumdisk.

Our parallel constrained Delaunay meshing algorithm uses constrained segments to separate the subdomains (Fig. 1b). Obviously, if the mesh inside each subdomain is Delaunay, then the global mesh is constrained Delaunay. Hence, provided that the domain decomposition is constructed, our task is to create a guaranteed quality

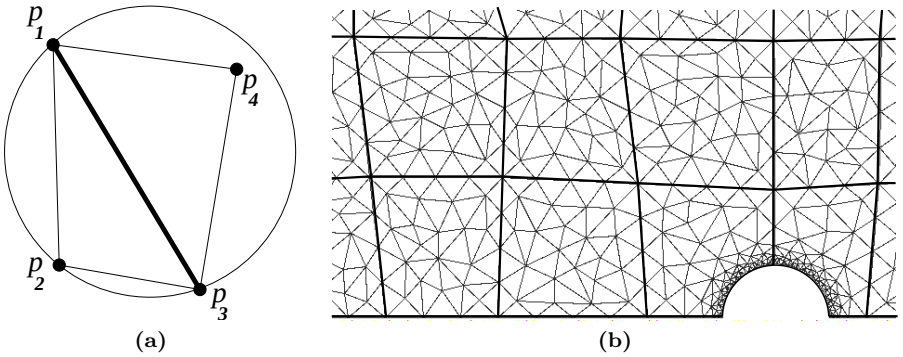

Fig. 1. **(a)** $\triangle(p_1p_2p_3)$ is constrained Delaunay, but not Delaunay. **(b)** A part of the mesh for the cylinder flow problem (see Fig. 8). The boundaries of the subdomains are composed of constrained segments (bold lines). If the mesh inside each subdomain is Delaunay then the global mesh is constrained Delaunay.

mesh for each of the subdomains and to ensure the conformity of the mesh along the interfaces. Our parallel algorithm executes the B-W mesh refinement procedure concurrently in each of the subdomains and maintains the set of constrained segments along the boundaries. The boundary segments may be subdivided by the insertion of their midpoints, in which case the neighboring subdomains are updated by means of SPLIT messages. This way, the global mesh preserves its conformity and the constrained Delaunay property.

3. THE PARALLEL ALGORITHM

3.1 Domain Decomposition

Chew et al. [Chew et al. 1997] list the following three requirements on the generation of the artificial boundaries between the subdomains:

(i) The amount of work assigned to different processes should be approximately equal[1].

(ii) There should be no small gaps between the artificial boundaries, which can lead to unnecessarily small triangles.

(iii) The artificial boundaries should not create small angles, which will unavoidably be forced into the final mesh and cause the creation of poorly shaped triangles.

We would like to add two more requirements:

(iv) The total length of the boundaries which are shared between different processors should be as small as possible, so that the amount of the communication among the subdomains could be decreased.

[1]However, with the development of the load balancing libraries (e.g., LBL [Barker et al. 2004]), a good initial equidistribution of the workload has become less crucial. Such libraries shift the task of mapping subdomains to processors from the initialization step to runtime, which allows the parallel system to adapt its behavior in response to changing levels of refinement and processor performance.

```
PCDM({(X_i, M_i) | i = 1,...,N}, Δ̄, ρ̄, P, p)
Input: X_i are PSLGs that define the subdomains Ω_i
       M_i are initial coarse meshes of Ω_i
       Δ̄ is the upper bound on triangle area
       ρ̄ is the upper bound on triangle circumradius-to-shortest edge ratio
       P is the total number of processes
       p is the index of the current process
Output: Modified Delaunay meshes {M_i} which respect
        the bounds Δ̄ and ρ̄
1   Compute the mapping κ : {1,...,N} → {1,...,P}
        of subdomains to processes
2   Distribute subdomains to processes
3   Let {Ω_{i_1},...,Ω_{i_{N_p}}} be the set of local subdomains
4   for j = 1,...,N_p
5       DelaunayRefinement(X_{i_j}, M_{i_j}, Δ̄, ρ̄, κ)
6   endfor
7   Terminate()

DelaunayRefinement(X, M, Δ̄, ρ̄, κ)
1   Q ← {t ∈ M | (ρ(t) ≥ ρ̄) ∨ (Δ(t) ≥ Δ̄)}
2   while Q ≠ ∅
3       Let t ∈ Q
4       BadTriangleElimination(X, M, t, κ)
5       Update Q
6   endwhile

BadTriangleElimination(X, M, t, κ)
1   p_i ← Circumcenter(t)
2   if p_i encroaches upon a segment s ∈ X
3       p_i ← Midpoint(s)
4       RemoteSplitRequest(κ(Neighbor(s)), p_i)
5   endif
6   C(p_i) ← {t ∈ M | p_i ∈ ◯(t)}
7   M ← M \ C(p_i) ∪ {△(p_i p_m p_n) | e(p_m p_n) ∈ ∂C(p_i)}
```

Fig. 2. A high level description of the algorithm.

(v) The number of subdomains (N) has to be much larger than the number of processes (P) to provide sufficient flexibility for a load balancing library (e.g., LBL [Barker et al. 2004]). When $N \gg P$, we say that the domain is *overdecomposed*.

The purpose of requirement (iv) above is to minimize the communication of both the parallel mesh generation and parallel finite element solver. However, in the case of the parallel mesh generation only it can be re-phrased as: minimize split messages which implies minimize the chances of encroachment which implies minimize (or eliminate) the chances to introduce new points on the subdomain boundary. This approach is followed in [Linardakis and Chrisochoides 2006], however requires a priory and proper discretization of the subdomain boundary which leads to about 2% over-refinement.

All of the above mentioned requirements can be satisfied by the use of the MADD algorithm [Linardakis and Chrisochoides 2005] followed by the mapping of the re-

Fig. 3. The PSLG representing the outline of the Chesapeake bay.

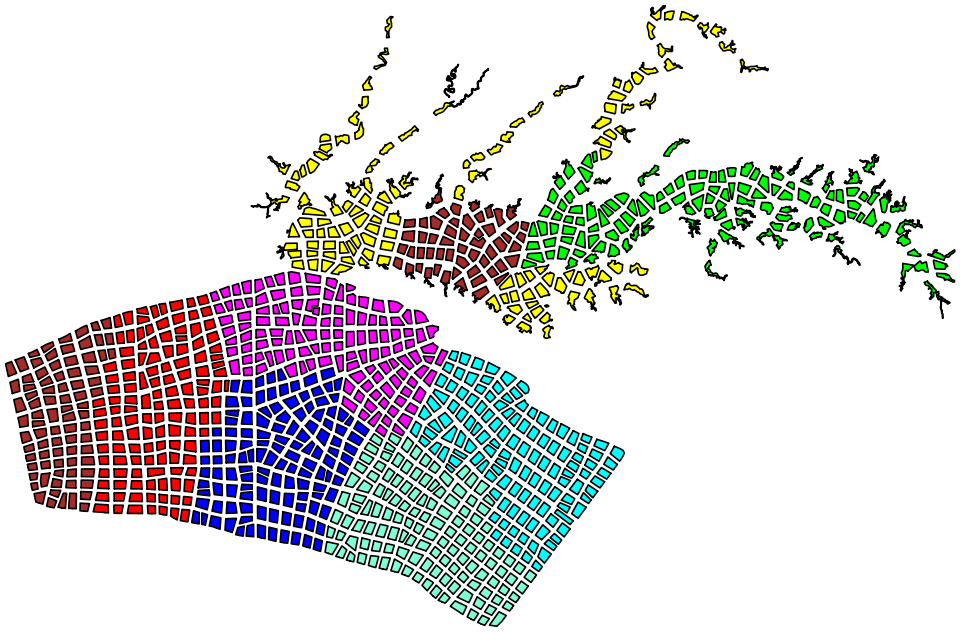

Fig. 4. The Chesapeake bay model decomposed into 1024 subdomains, which are mapped onto 8 processes. The assignment of subdomains to processes is shown with different colors.

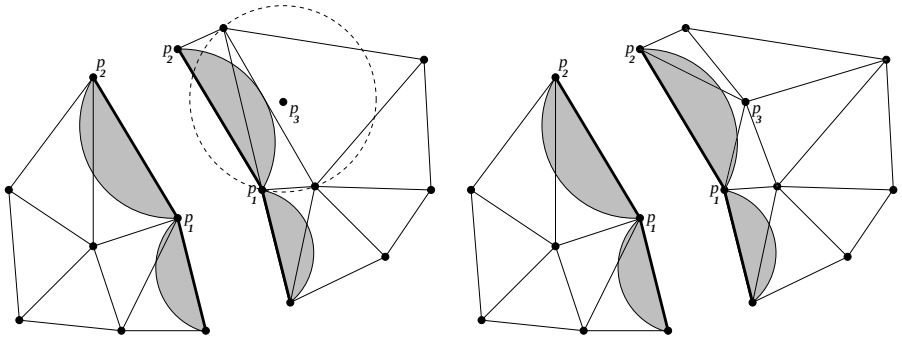

Fig. 5. If the inserted point p_3 does not encroach upon any constrained edge, a cavity is constructed and triangulated.

sulting subdomains to processes using a generic graph partitioning library.

First, we use the MADD to overdecompose the initial domain Ω into N subdomains. In practice, it is sufficient to satisfy $N \geq 20P$. Then, we solve the following graph partitioning problem with the goal of distributing the subdomains among the processes, so that the sum of the weights of the subdomains assigned to each process is approximately equal, and the total length of the subdomain boundaries which are shared between the processes is minimized. We construct a weighted graph $G = (V, E)$ such that

—every subdomain Ω_i $(i = 1, \ldots, N)$ is represented by a vertex $v_i \in V$, and the weight of v_i is set equal to the area of Ω_i;

—every piecewise-linear boundary line which is shared by two subdomains Ω_i and Ω_j is represented by an edge $e_{ij} \in E$, and the weight of e_{ij} is set equal to the length of the line.

Graph G is then sequentially partitioned into P parts using the Metis graph partitioning library [Karypis and Kumar 1998] and the subdomains are distributed among the corresponding processes. Since the number of vertices in the graph is not very large, the use of a parallel graph partitioning procedure is not required.

3.2 Mesh Conformity and Message Passing

The conformity of the mesh along the boundaries of the neighboring subdomains is assured by sending SPLIT messages each time a boundary edge is encroached and subdivided (see Figures 5 and 6).

When a subdomain receives a SPLIT message from its neighbor, it is necessary to identify the edge which has to be split and the triangle which is incident upon this edge in the local data structure. This task is complicated by the fact that SPLIT messages may arrive out of order due to subdomain migration performed by a load balancing library (e.g., LBL [Barker et al. 2004]) and/or the underlying communication substrate (e.g., CLAM [Fedorov and Chrisochoides 2004]).

For example, consider an initial constrained edge $e\,(p_0p_1)$, which is shared by subdomains A and B. In the beginning, all points on subdomain boundaries are assigned unique identificators, such that there is no ambiguity whenever the refinement of A or the refinement of B causes the split of $e\,(p_0p_1)$ by a new point p_2; it is

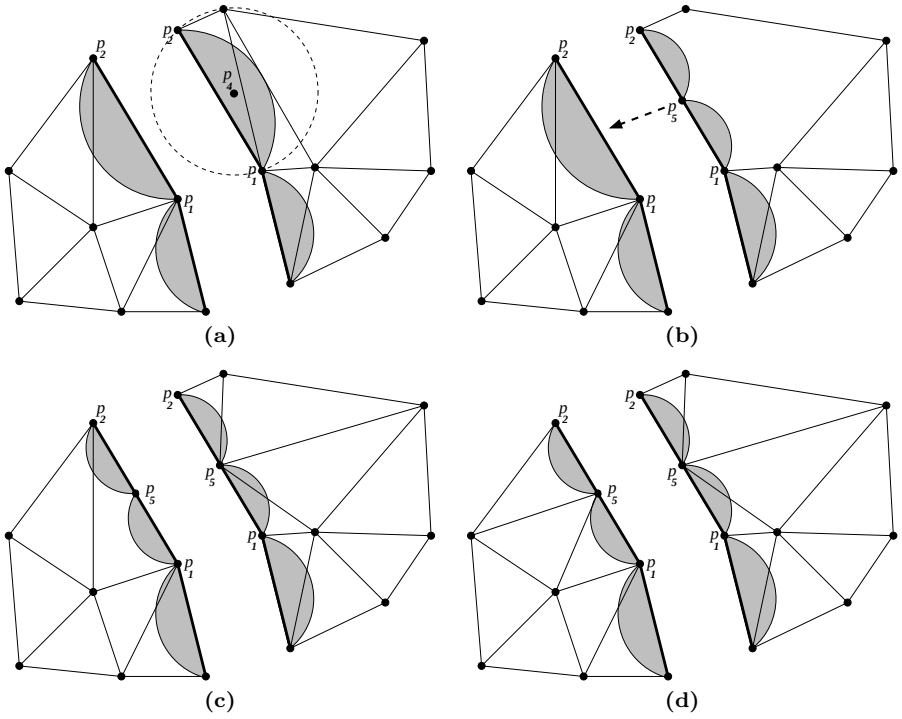

Fig. 6. If the candidate point p_4 encroaches upon a constrained edge $e\,(p_1p_2)$ **(a)** another point p_5 is inserted in the middle of $e\,(p_1p_2)$ instead, and a message is sent to the neighboring process **(b)**; the local cavity of p_5 is constructed and triangulated **(c)**; the neighboring process also constructs and triangulates its local cavity of p_5 **(d)**.

enough to send the identificators of p_0, p_1, and p_2, we can denote such message as SPLIT(p_0, p_1, p_2). However, complications may arise if the refinement of A leads to the insertion of point p_2 in the middle of $e\,(p_0p_1)$, and then to the insertion of point p_3 in the middle of $e\,(p_0p_2)$. Naturally, first SPLIT(p_0, p_1, p_2) and then SPLIT(p_0, p_2, p_3) are sent from A to B. If the second message is received by B before the first one, B will not have the edge $e\,(p_0p_2)$ in its local data structure and will not be able to service the SPLIT request properly.

The related issue is the processing of the incoming SPLIT requests with respect to the points which already have been independently inserted locally. It is necessary to reliably identify duplicate points. For the purpose of point comparison, the use of floating-point coordinates can lead to errors. Hence, it is desirable to develop a point identification scheme which satisfies the following conditions:

(1) The position of a point with respect to a shared constrained edge can be unambiguously resolved without the knowledge of the other points which split this edge.
(2) The use of the floating point arithmetic should be avoided.
(3) The size of the messages and the amount of the associated computation should be minimized.

We adopted the following scheme which is similar in spirit to rational arithmetic. The position (x, y) of every split point is computed with respect to the corresponding edge $e\,(p_0p_1)$ as:

$$(x, y) = (x_0, y_0) + \frac{n}{2^m}((x_1, y_1) - (x_0, y_0)),$$

where n and m are integers. This scheme allows to represent each split point by only two integers (n, m) and have the SPLIT messages be of the form SPLIT(p_0, p_1, n, m). which avoids using imprecise floating point arithmetic for maintaining point ordering along the boundary edges. In addition, it enables us to determine the sequence of edge splits performed in the neighboring subdomain, which has lead to the insertion of a given point, in the case of out-of-order message arrival[2] and to duplicate this sequence. One possible implementation solution is to customize a rational number type provided by one of the available libraries, e.g., CGAL [CGAL Editorial Board 2006], and the other is to encode an application-specific solution. We chose the latter approach.

The termination of the algorithm is detected by a circular token transmission scheme described in [Dijkstra 1987]. Our parallel mesh refinement algorithm satisfies all the assumptions made in [Dijkstra 1987]:

(1) Each of the P machines (processes) can be either active (refining a subdomain) or passive (all subdomains have been refined).
(2) Only active machines send (SPLIT) messages to other machines.
(3) Message transmission takes a finite period of time.
(4) When a machine receives a message, it becomes active.
(5) Only the receipt of a SPLIT message can transition a machine to activity (with the exception of the start of the algorithm).
(6) The transition of a machine from active to passive state can occur without the receipt of a message (when the refinement finishes).

4. IMPLEMENTATION

The PCDM software is written using the C++ programming language. It uses the message passing interface (MPI) for the communication among the processes. Additionally, the code needs to be linked with the Triangle [Shewchuk 1996], the Robust Predicates [Shewchuk 1997a], and the Metis [Karypis and Kumar 1998] libraries. Triangle is used only for the initial triangulation of each subdomain. This task does not require inter-process communication, and we can use off-the-shelf sequential codes like the Triangle. The initial triangulation is passed to the PCDM code to perform the actual mesh generation which requires some communication.

4.1 Data Structures

The units of work are represented by the subdomain objects. These objects are initially created by a selected process, which reads the input file, partitions the subdomains into P groups and distributes them to the remaining processes.

[2]Although MPI guarantees the ordering of the messages, our implementation is designed with LBL and CLAM in mind.

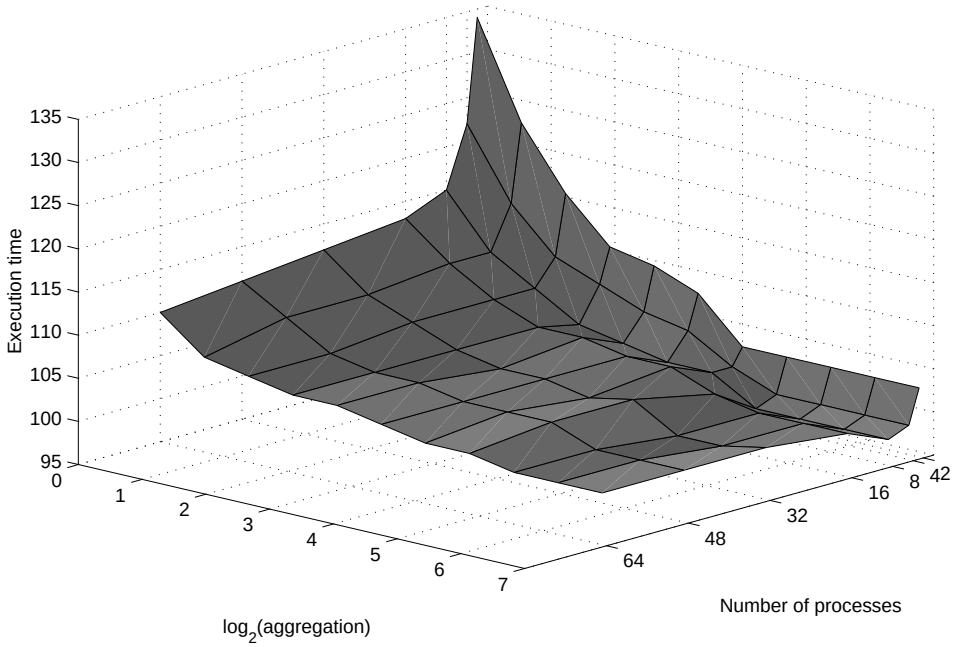

Fig. 7. The influence of message aggregation on the execution time, the pipe model.

Each subdomain contains the collections of the constrained edges, the triangles, and the points. The triangles are classified into those that satisfy the quality constrains and those that violate either the ratio or the area requirement. Similar to the Triangle [Shewchuk 1996] implementation, the poor quality triangles are distributed over a number of buckets, based on the measure of their minimal angle. Each triangle contains three pointers to its vertices and three pointers to the neighboring triangles. For the point insertion, we use the B-W algorithm (1). The constrained (boundary) segments are protected by diametral lenses, and each time a segment is encroached, it is split in the middle; as a result, a SPLIT message is sent to the neighboring subdomain. To ensure the correct message addressing, with each segment we store the identificators of the subdomains it separates, and each process keeps a map from a subdomain identificator to process number.

When a SPLIT message arrives, we need to find out (i) whether or not the specified split point has already been inserted locally and, if the answer is negative, (ii) which local triangle contains the edge that the new point has to be inserted on. If the point already exists locally, the message is simply discarded. Otherwise, we insert the point and triangulate the cavity according to the B-W algorithm. In order to speed up the search of the initial cavity triangle, we maintain a binary search map from the set of pairs of points along the constrained segment to the corresponding adjacent triangles.

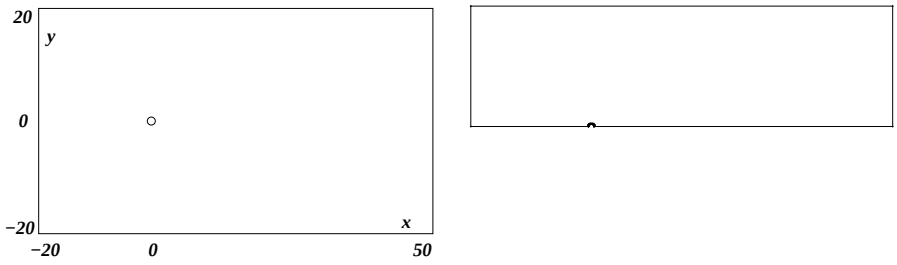

Fig. 8. **(Left)** Two-dimensional "z-slice" of the computational domain used for the simulation of the turbulent flow past a stationary cylinder at $Re = 10000$ [Dong et al. 2004]. **(Right)** The upper half of the domain used in our tests.

4.2 Message Aggregation

One of the most noticeable improvements in the performance of the code was achieved by aggregating the SPLIT messages, which are sent between neighboring subdomains as a result of inserting points on the common boundaries. The time to send a message is [Kumar et al. 1994]:

$$\text{Communication time} = \text{Startup time} + \frac{\text{Message size}}{\text{Network bandwidth}}.$$

If we send n messages separately, we incur the startup time cost for every message. However, if the messages are aggregated and sent together atomically, this cost can be paid only once. Figure 7 shows the influence of the message aggregation on the execution time. The *aggregation* parameter indicates the number of SPLIT messages that are accumulated before sending them atomically to a given subdomain. We evaluated the effects of message aggregation on a subcluster of up 64 processors, which is the largest homogeneous configuration available at our site. In these and all further tests, each MPI process was executed on a single physical CPU, so we use the terms process and processor interchangeably, depending on the context. We can observe two trends. First, the increase of the *aggregation* causes an improvement in the running time. The running time flattens out after the 512 aggregation value, which we adopted for the further scalability evaluation. Second, the time corresponding to low aggregation decreases as we increase the number of processors; this can be explained by the growth of the utilized network and, consequently, the aggregate bandwidth.

5. EXPERIMENTAL EVALUATION

For the experimental evaluation of our code, we used the SciClone cluster at the College of William and Mary[3]. In particular, we employed its "whirlwind" (64 single-cpu Sun Fire V120 servers at 650 MHz with 1 GB memory), "hurricane"

[3]http://compsci.wm.edu/SciClone/

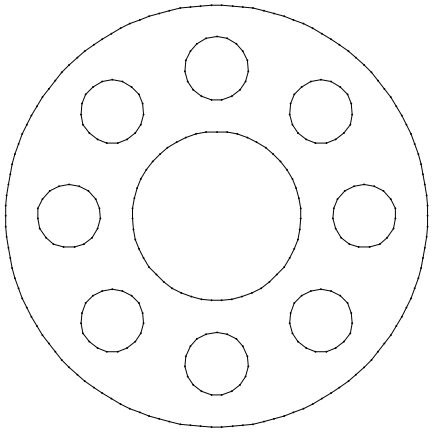

Fig. 9. A cross-section and a cartoon pipe of a regenerative cooled pipe geometry which came from testing of a rocket engine at NASA Marshall.

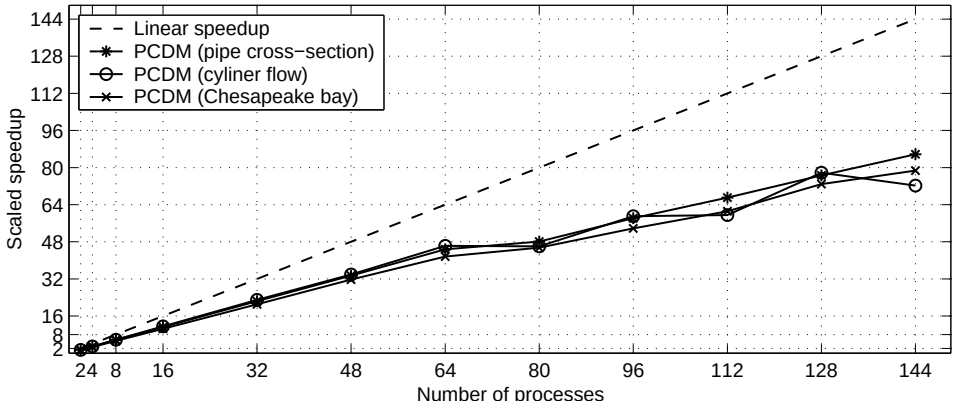

Fig. 10. The scaled speedup measurements. The number of triangles is approximately 10P, i.e., for 2 processors 20M, and for 144 processors about 1.4B.

(4 quad-cpu Sun Enterprise 420R servers at 450 MHz with 4 GB memory), and "twister" (32 dual-cpu Sun Fire 280R servers at 900 MHz with 2 GB memory) subclusters. In all tests, the first 64 processors were selected from the "whirlwind" subcluster, the next 16 — from the "hurricane", and the remaining — from the "twister" subcluster. The experiments involving sequential software were performed on one of the "whirlwind" nodes.

Among other geometries, we tested our software using the Chesapeake bay model (Fig. 3), the cylinder flow model (Fig. 8), and the pipe cross-section model (Fig. 9).

Figure 10 presents the scaled speedup evaluation using these three models. We computed the scaled speedup as

$$\text{Scaled speedup} = \frac{P \times T_s(W)}{T_{MADD} + T_{CDT} + T_P(W \times P)},$$

Table I. The comparison of the sequential and the parallel execution times, sec.

Model	T_s	T_{MADD}	T_{CDT}	T_{64}	T_{80}	T_{144}
Pipe cross-section	74.7	5.4	0.34	101.1	118.5	120.3
Cylinder flow	75.4	4.1	0.31	99.8	126.6	146.3
Chesapeake bay	77.4	13.9	0.59	104.5	121.7	127.1

Table II. For 64 processes, the time spent on computation (i.e., mesh refinement) and communication, sec.

Model	Computation			Communication		
	min	average	max	min	average	max
Pipe cross-section	85.8	90.0	94.0	0.08	2.89	2.98
Cylinder flow	77.5	89.2	93.1	0.07	2.78	2.87
Chesapeake bay	81.9	91.2	95.2	0.19	3.92	4.09

Table III. For 64 processes, the time spent on computation (i.e., mesh refinement) and communication, per one million of triangles, sec. Pipe geometry.

Method	Computation			Communication		
	min	average	max	min	average	max
PCDM	0.134	0.141	0.147	0.000125	0.004515	0.004656
PDR	0.006	0.606	0.907	0.265479	0.566294	1.166690

where T_s is the time taken by the fastest to our knowledge sequential mesh generator the Triangle [Shewchuk 1996], T_{MADD} is the domain decomposition time, T_{CDT} is the time to construct the initial coarse triangulation using the Triangle, T_P is the time taken by our parallel implementation on P processors, and W is a fixed amount of work. We used Triangle version 1.6 which is the latest release of this software and implements the efficient off-center point insertion method [Üngör 2004]. T_P includes the time to read the input data files and broadcast them to all processes, the time to refine the subdomains (i.e., the computation time presented in Table II), the communication time, the idle polling time, and miscellaneous overheads, but excludes the time to write the resulting meshes on disc, since they are intended to stay in the memory to be used by the solvers. Since the mesh generation time in practice is linear with respect to the number of the resulting triangles, for convenience we chose W to be the amount of work required to create 10 million triangles. Then $W \times P$ represents the amount of work required to produce $10 \times P$ million triangles. The number of elements is roughly inversely proportional to the required triangle area bound, and can be controlled by selecting the area bound correspondingly. This estimation is not an exact prediction of the size of the final mesh, but it worked very well in all of our experiments. In all tests we used the same triangle shape constraint which is equivalent to 20° minimal angle. From Figure 10 we can see that the code scales up linearly with the parallel efficiency close to 0.6. Table I summarizes the costs for the three models, and Table II lists the computation and the communication time ranges for the 64 processor configuration. Table III compares the computation and communication costs of the PCDM and the PDR methods, per one million of triangles using 64 processors. We can see that the PCDM methods is much less expensive. However, as we pointed out in the Introduction, the PDR method does not rely on domain decomposition which

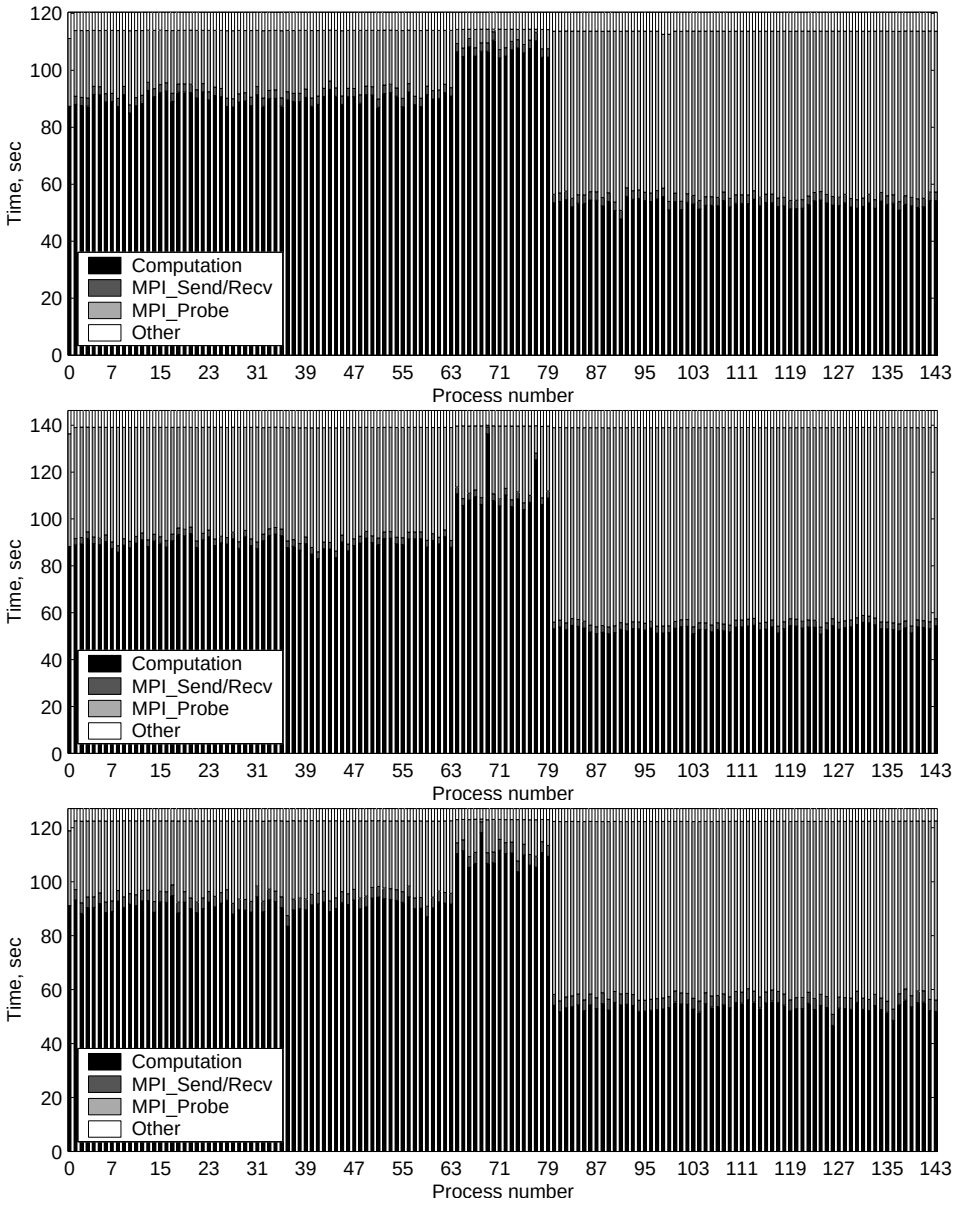

Fig. 11. The breakdown of the total parallel execution time for the pipe (top), cylinder flow (middle), and Chesapeake bay (bottom) models. Each process is represented by a single stacked bar. 'Computation' is the time spent refining the meshes. 'MPI_Send/Recv' and 'MPI_Probe' represent the aggregate time spent on the corresponding function calls. When a process is idle, it calls MPI_Probe repeatedly. MPI_Probe is also called in between a certain number of refinement calls.

makes it more amenable for the three-dimensional geometries.

Figure 11 shows the breakdown of the parallel execution time for each of the 144

processes. We can observe the ratio of the total time spent by each of the processes on the computation and the communication. The difference in the hardware performance of the subclusters can also be clearly seen. This difference caused some workload imbalance, which leaves room for improvement from load balancing.

6. CONCLUSIONS

We presented an algorithm and an implementation for the parallel two-dimensional guaranteed quality Delaunay mesh generation. Our algorithm is based on the idea we presented in [Chew et al. 1997]. However, we improved many aspects of the algorithm and its implementation, utilized newly available software, and made a number of enhancements. Our experimental results show very good scalability of the code and its ability to generate over a billion triangles on a heterogeneous cluster with more than one hundred nodes. The difference in processor performance causes some imbalance in the workload and slight speedup degradation. The use of the Load Balancing Library (LBL) [Barker et al. 2004] allows to alleviate this problem as shown in [Fedorov and Chrisochoides 2004]. However, in this paper we present and submit the MPI-only version of our software, which can be used without the LBL, since the LBL is not a standard library available on all clusters as is the case with the MPI. The PCDM software can be used in conjunction with parallel problem-solving environments, e.g., PELLPACK [Houstis et al. 1998].

The development of an analogous algorithm for the three dimensions will require the solution of the three-dimensional domain decomposition problem and the design of appropriate boundary face splitting strategies. Both are non-trivial extensions of the tools and methods we present in this paper.

7. ACKNOWLEDGMENTS

We thank Leonidas Linardakis for providing the domain decomposition software.

We thank Filip Blagojevic for the numerous insights on improving the single node performance of the code. We also acknowledge Paul Chew, Florian Sukup, Démian Nave, and Brian Holinka for their contributions in earlier implementations of the method.

This work was supported by NSF grants: ACI-0085969, EIA-9972853, EIA-0203974, and ACI-0312980.

This work was performed using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

We thank the anonymous reviewers for useful suggestions which helped us improve the presentation of this paper.

REFERENCES

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

Blelloch, G. E., Hardwick, J., Miller, G. L., and Talmor, D. 1999. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica 24*, 243–269.

Blelloch, G. E., Miller, G. L., and Talmor, D. 1996. Developing a practical projection-based parallel Delaunay algorithm. In *12th Annual Symposium on Computational Geometry.* 186–195.

Bowyer, A. 1981. Computing Dirichlet tesselations. *Computer Journal 24*, 162–166.

CGAL Editorial Board. 2006. *CGAL-3.2 User and Reference Manual*.

Chernikov, A. N. and Chrisochoides, N. P. 2004a. Parallel guaranteed quality planar Delaunay mesh generation by concurrent point insertion. In *14th Annual Fall Workshop on Computational Geometry*. MIT, 55–56.

Chernikov, A. N. and Chrisochoides, N. P. 2004b. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. In *Proceedings of the 18th Annual International Conference on Supercomputing*. ACM Press, 48–57.

Chernikov, A. N. and Chrisochoides, N. P. 2005. Parallel 2D graded guaranteed quality Delaunay mesh refinement. In *Proceedings of the 14th International Meshing Roundtable*. Springer, 505–517.

Chernikov, A. N. and Chrisochoides, N. P. 2006a. Generalized Delaunay mesh refinement: From scalar to parallel. In *Proceedings of the 15th International Meshing Roundtable*. Springer, 563–580.

Chernikov, A. N. and Chrisochoides, N. P. in press, May 2006b. Parallel guaranteed quality Delaunay uniform mesh refinement. *SIAM Journal on Scientific Computing*.

Chew, L. P. 1987. Constrained Delaunay triangulations. In *SCG '87: Proceedings of the third annual symposium on Computational geometry*. ACM Press, 215–222.

Chew, L. P. 1989. Guaranteed-quality triangular meshes. Tech. Rep. TR89983, Cornell University, Computer Science Department.

Chew, L. P., Chrisochoides, N., and Sukup, F. 1997. Parallel constrained Delaunay meshing. In *ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation*. Northwestern University, Evanston, IL, 89–96.

Chrisochoides, N. and Nave, D. 2003. Parallel Delaunay mesh generation kernel. *Int. J. Numer. Meth. Engng. 58*, 161–176.

Chrisochoides, N. P. 2005. 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, Petter Bjorstad, Aslak Tveito), Springer Verlag.

Cignoni, P., Laforenza, D., Montani, C., Perego, R., and Scopigno, R. 1995. Evaluation of parallelization strategies for an incremental Delaunay triangulator in E^3. *Concurrency: Practice and Experience 7,* 1, 61–80.

de Cougny, H. L., Shephard, M. S., and Ozturan, C. 1994. Parallel three-dimensional mesh generation. *Computing Systems in Engineering 5*, 311–323.

Dijkstra, E. W. 1987. Shmuel Safra's version of termination detection. Circulated privately, http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD998.PDF.

Dong, S., Lucor, D., and Karniadakis, G. E. 2004. Flow past a stationary and moving cylinder: DNS at Re=10,000. In *2004 Users Group Conference (DOD_UGC'04)*. 88–95.

Fedorov, A. and Chrisochoides, N. 2004. Communication support for dynamic load balancing of irregular adaptive applications. In *Proceedings of the International Conference on Parallel Processing Workshops*. Montreal, Quebec, Canada, 555–562.

Galtier, J. and George, P. L. 1997. Prepartitioning as a way to mesh subdomains in parallel. In *Special Symposium on Trends in Unstructured Mesh Generation*. ASME/ASCE/SES, 107–122.

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

Houstis, E. N., Rice, J. R., Weerawarana, S., Catlin, A. C., Papachiou, P., Wang, K.-Y., and Gaitatzes, M. 1998. Pellpack: a problem-solving environment for pde-based applications on multicomputer platforms. *ACM Trans. Math. Softw. 24,* 1, 30–73.

Isenburg, M., Liu, Y., Shewchuk, J., and Snoeyink, J. 2006. Streaming computation of Delaunay triangulations. accepted to SIGGRAPH.

Jones, M. T. and Plassmann, P. E. 1994. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In *Scalable High Performance Computing Conference*. IEEE Computer Society Press, 478–485.

KADOW, C. 2004a. Adaptive dynamic projection-based partitioning for parallel Delaunay mesh generation algorithms. In *SIAM Workshop on Combinatorial Scientific Computing*.

KADOW, C. 2004b. Parallel Delaunay refinement mesh generation. Ph.D. thesis, Carnegie Mellon University.

KADOW, C. AND WALKINGTON, N. 2003. Design of a projection-based parallel Delaunay mesh generation and refinement algorithm. In *Fourth Symposium on Trends in Unstructured Mesh Generation*. http://www.andrew.cmu.edu/user/sowen/usnccm03/agenda.html.

KARNIADAKIS, G. AND ORSZAG, S. 1993. Nodes, modes, and flow codes. *Physics Today 46*, 34–42.

KARYPIS, G. AND KUMAR, V. 1998. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 4.0*. University of Minnesota.

KOHOUT, J., KOLINGEROVÁ, I., AND ŽÁRA, J. 2005. Parallel Delaunay triangulation in E^2 and E^3 for computers with shared memory. *Parallel Computing 31,* 5, 491–522.

KOT, A., CHERNIKOV, A., AND CHRISOCHOIDES, N. 2006. Effective out-of-core parallel delaunay mesh refinement using off-the-shelf software. In *20th IEEE International Parallel and Distributed Processing Symposium*.

KUMAR, V., GRAMA, A., GUPTA, A., AND KARYPIS, G. 1994. *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*. The Benjamin/Cummings Publishing Company, Inc.

LINARDAKIS, L. AND CHRISOCHOIDES, N. 2005. A static medial axis domain decomposition for 2d geometries. *ACM Transactions on Mathematical Software*. Submitted.

LINARDAKIS, L. AND CHRISOCHOIDES, N. 2006. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Journal on Scientific Computing 27,* 4, 1394–1423.

LÖHNER, R. AND CEBRAL, J. R. 1999. Parallel advancing front grid generation. In *Proceedings of the Eighth International Meshing Roundtable*. 67–74.

MILLER, G. L., TALMOR, D., TENG, S.-H., AND WALKINGTON, N. 1995. A Delaunay based numerical method for three dimensions: Generation, formulation, and partition. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*. ACM Press, 683–692.

NAVE, D., CHRISOCHOIDES, N., AND CHEW, L. P. 2004. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. *Computational Geometry: Theory and Applications 28*, 191–215.

RIVARA, M.-C., CALDERON, C., FEDOROV, A., AND CHRISOCHOIDES, N. 2006. Parallel decoupled terminal-edge bisection method for 3d meshes. *Engineering with Computers 22,* 2 (Sept.), 111–119.

RUPPERT, J. 1995. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms 18(3)*, 548–585.

RYPL, D. AND BITTNAR, Z. 2001. Parallel 3D mesh generator: Performance comparison. In *Computational Fluid and Solid Mechanics*, K. Bathe, Ed. Vol. 2. Elsevier, 1644–1646.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds. Lecture Notes in Computer Science, vol. 1148. Springer-Verlag, 203–222. From the First ACM Workshop on Applied Computational Geometry.

SHEWCHUK, J. R. 1997a. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry 18,* 3 (Oct.), 305–363.

SHEWCHUK, J. R. 1997b. Delaunay refinement mesh generation. Ph.D. thesis, Carnegie Mellon University.

SHEWCHUK, J. R. 2002a. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications 22,* 1–3 (May), 21–74.

SHEWCHUK, J. R. 2002b. What is a good linear element? In *Proceedings of the 11th International Meshing Roundtable*. Sandia National Laboratories, 115–126.

SPIELMAN, D. A., TENG, S.-H., AND ÜNGÖR, A. 2001. Parallel Delaunay refinement: Algorithms and analyses. In *Proceedings of the Eleventh International Meshing Roundtable*. 205–217.

Spielman, D. A., Teng, S.-H., and Üngör, A. 2004. Time complexity of practical parallel Steiner point insertion algorithms. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM Press, 267–268.

Teng, S.-H. 2004. Personal Communication, February 2004.

Üngör, A. 2002. Parallel Delaunay refinement and space-time meshing. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Üngör, A. 2004. Off-centers: A new type of Steiner points for computing size-optimal guaranteed-quality Delaunay triangulations. In *Proceedings of LATIN*. 152–161.

Walters, R. A. 2005. Coastal ocean models: Two useful finite element methods. *Recent Developments in Physical Oceanographic Modelling: Part II 25*, 775–793.

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