# PARALLEL GUARANTEED QUALITY DELAUNAY UNIFORM MESH REFINEMENT

ANDREY N. CHERNIKOV* AND NIKOS P. CHRISOCHOIDES†

**Abstract.** We present a theoretical framework for developing parallel guaranteed quality Delaunay mesh generation software, that allows us to use commercial off-the-shelf sequential Delaunay meshers for two-dimensional geometries. In this paper, we describe our approach for constructing uniform meshes, in other words, the meshes in which all elements have approximately the same size. Our uniform distributed- and shared-memory implementations are based on a simple (block) coarse-grained mesh decomposition. Our method requires only local communication, which is bulk and structured as opposed to fine and unpredictable communication of the other existing practical parallel guaranteed quality mesh generation and refinement techniques. Our experimental data show that on a cluster of more than 100 workstations we can generate about 0.9 billion elements in less than 5 minutes in the absence of work-load imbalances. Preliminary results for this paper were presented in [6]. Our work in progress includes extending the presented approach, which can efficiently generate only uniform meshes, to nonuniform graded meshes.

**Key words.** Delaunay triangulation, mesh generation, parallel refinement

**AMS subject classifications.** 65D18, 68W05, 68W10, 68N19

## 1. Introduction.

With the development of very efficient sequential guaranteed quality mesh generators, e.g. the "Triangle" [33], the construction of very large meshes that cannot fit into the memory of a single machine still remains a challenging problem. These large meshes are required, for instance, to conduct direct numerical simulations of turbulence in cylinder flows with very large Reynolds numbers, see [14]. With the increase of the Reynolds number, the size of the mesh grows in the order of $Re^{9/4}$, see [23], which motivates the use of a parallel distributed memory mesh generation algorithm.

There are four important requirements that are usually imposed on the parallel mesh generation algorithms: *stability*, *scalability*, *efficient domain decomposition*, and *code reuse*. *Stability* refers to the fact that distributed meshes retain the same quality of the elements and partition properties as the sequentially generated and partitioned meshes, see [10]. *Scalability* is understood as the ability of the algorithm and software to achieve the speedup proportional to the number of processors, see [25]. The *efficiency of domain decomposition* can be measured in terms of its computation cost and the appropriateness of the resulting subdomains for the selected mesh generation algorithm. Finally, *code reuse* allows to take advantage of the ever evolving basic sequential meshing libraries which are now mature and highly optimized. In this paper, we present an algorithm that conforms to all four of these requirements, while the previously published results address only some of them.

The sophisticated domain decomposition methods like the Medial Axis Domain Decomposition (MADD) [27] allow to completely decouple the subproblems, and therefore eliminate the communication, but they are very expensive and difficult to construct for three-dimensional domains. In this paper, we settle for a simpler (block) domain decomposition at the cost of some bulk communication and work-load imbalances. Fortunately, the bulk communication in our method is structured and can be

*Department of Computer Science, College of William and Mary, Williamsburg, VA 23185 (ancher@cs.wm.edu).

†Department of Computer Science, College of William and Mary, Williamsburg, VA 23185 (nikos@cs.wm.edu).

tolerated. To cope with the work-load imbalances, we developed a runtime system [2] that successfully addresses the dynamic load balancing problem for mesh generation.

Nave, Chrisochoides, and Chew [30] presented the first practical provably-good parallel mesh refinement algorithm for polyhedral domains. They addressed the stability, to some success the scalability, but not the code reuse requirement. The algorithm in [30] guarantees stability by simultaneously partitioning and refining the interface surfaces and the volumes of the subdomains — the changes caused by a point insertion can propagate across the processor boundaries — at the cost of high communication and of the restructuring of the Delaunay kernel. In [10], Chrisochoides and Nave restructured the sequential Bowyer-Watson (B-W) Delaunay kernel [5, 39] using the speculative execution model, see [20], in order to tolerate high communication latencies. The speculative execution model allows setbacks to occur if the changes caused by the concurrent insertion of two or more points are incompatible. Although more than 80% of the overhead due to the remote data gather operations is masked, the experimental data in [10] suggest only $\mathcal{O}(\log P)$ speedup, $P$ being the number of processors. In this paper, we provide a sufficient condition, which (in two dimensions) guarantees the independence of the simultaneously inserted points, eliminates the setbacks and the need to restructure the mesh generation kernel, and does not induce intensive communication. Moreover, our condition is not based on the coloring techniques which require high pre-processing time, since they need first to construct a graph (cavity graph[1]).

Blelloch, Hardwick, Miller, and Talmor [4] describe a divide-and-conquer projection-based algorithm for constructing Delaunay triangulations of pre-defined point sets in parallel. The work by Kadow and Walkington [21, 22] extended [4] for parallel mesh generation and further eliminated the sequential step (contrary to our proposed method) for an initial mesh, but did not address the issue of code reuse.

An idea of updating partition boundaries when inserted points happen to be close to them was presented by Chew, Chrisochoides, and Sukup [8] as a Parallel Constrained Delaunay Meshing (PCDM) algorithm. In PCDM, the edges on the boundaries of submeshes are fixed (constrained), and if a new point encroaches upon a constrained edge, another point is inserted in the middle of this edge instead. As a result, a "split" message is sent to the neighboring processor, notifying that it also has to insert the midpoint of the shared edge. This approach requires the construction of the separators that will not compromise the quality of the final mesh, and permits to reuse sequential codes only to a limited degree, since one has to handle "split" messages and partial cavity expansions within the B-W kernel.

The theoretical analysis of the idea of the concurrent insertion of independent points has been started by several authors. Edelsbrunner and Guoy [15] define the prestar of point $x$ as the difference between the closure of the set of tetrahedra whose circumspheres enclose $x$ and the closure of the set of remaining tetrahedra. Further, they define the points $x$ and $y$ as independent if the closures of their prestars are disjoint. We prove a similar condition of point independence. The difference between the independence condition in [15] and in this paper is that our formulation is less restrictive: it allows the prestars (we use the word cavity) to share a point. However,

[1]When a new point is inserted, all tetrahedra that have this point within their circumspheres are eliminated. This creates a *cavity*. In the cavity graph each cavity is represented by a vertex and two adjacent cavities are represented by an edge. Chew et al. [9] presented a method to improve the efficiency of this method, but its computation requirements and possible optimizations are subject to additional research.

computing the prestars (cavities) and their intersections for all candidate points is very expensive. That is why we do not use coloring methods that are based on the cavity graphs and continue by proving a theorem, which allows to use only the distance between the points to check whether they are independent. The minimum separation distance argument in [15] is used to derive the upper bound on the number of inserted vertices and prove termination, but does not ensure point independence. In addition, we propose a simple block decomposition scheme for scheduling parallel point insertion for both the distributed and the shared memory implementations. In [15], a shared memory algorithm based on finding the maximal independent set of points is used. Also, Spielman, Teng, and Üngör [36] presented the first theoretical analysis of the complexity of the parallel Delaunay refinement algorithms. However, the assumption is that the global mesh is completely retriangulated each time a set of independent points is inserted [38]. In [37] the authors developed a more practical algorithm.

Besides the Delaunay-based algorithms, a number of other parallel mesh generation algorithms have been published. De Cougny, Shephard, and Ozturan [12] base the parallel mesh construction on an underlying octree. Löhner and Cebral [28], and Ito et al. [19] developed parallel advancing front schemes. Globisch [17, 18] presented a parallel mesh generator which uses a sequential frontier algorithm. A detailed review of many more methods appears in [11].

The rest of the paper is organized as follows. Section 2 presents the theoretical framework for our parallel guaranteed quality Delaunay refinement algorithm. After introducing the notation, we describe the sequential Delaunay refinement algorithm and emphasize the Delaunay triangulation loop invariant, which has to be maintained in order to ensure the correctness of the result. In subsection 2.3.1, we show how the concurrent point insertion can violate this invariant and lead to either non-conforming or non-Delaunay mesh. We prove a theorem which states that if the distance between two points is greater than four times the upper bound on triangle circumradius, then these points can be safely inserted in parallel. We continue by proving in subsection 2.3.2 that the maximal circumradius is not increased in the course of the execution of the B-W algorithm, which makes our condition applicable at all times during the mesh refinement. Next, we show in subsection 2.3.3 how to select the preprocessing parameters so that we can efficiently use all processors and at the same time have minimal sequential overhead. Our coarse-grained partitioning scheme is described in Section 3. Finally, in Section 4 we present the implementation issues and the experimental results. We conclude with a summary of our results and future work directions.

## 2. Parallel Delaunay Refinement.

**2.1. Notation.** We will denote a point with index $i$ as $p_i$ and the triangle with vertices $p_i$, $p_j$, and $p_k$ as $\triangle p_i p_j p_k$. When the vertices of a triangles are irrelevant, we will denote such a triangle with a single letter $t$. We will use the small letters $a$, $b$, and $c$ to refer to the side lengths of a triangle, adopting the convention $a \leq b \leq c$ and the capital letters $A$, $B$, and $C$ to represent the measures of the three angles in the triangle, where we suppose the relation $A \leq B \leq C$.

By the term *circle with center $p$ and radius $r$* we will refer to the set of points in $\mathbb{R}^2$, whose Euclidean distance to $p$ is exactly $r$. We will use the term *open disk* for the set of points in $\mathbb{R}^2$ whose distance from $p$ is strictly less than $r$. The symbol $\|\cdot\|$ will represent the conventional Euclidean distance (the $L_2$ norm).

Let the *circumcircle* of triangle $\triangle p_i p_j p_k$ be the unique circle that passes through $p_i$, $p_j$, and $p_k$. The center and the radius of a triangle's circumcircle are called its *circumcenter*, and *circumradius* respectively. Let us call the open disk corresponding to a triangle's circumcircle its *circumdisk*. We will use the symbols $\bigcirc(t)$, $r(t)$, and $\odot(t)$ to represent the circumdisk, circumradius, and the circumcenter of the triangle $t$, respectively.

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 [33], to delimit $\Omega$ from the rest of the plane. A PSLG consists of a set $P$ of points and a set $S$ of segments connecting the points in $P$. The segments are not allowed to intersect in points that are not in $P$. Each segment in $S$ is considered *constrained* and must appear (possibly, as a union of smaller segments) in the final mesh.

Definition 1. *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 $V$. We will call $\mathcal{T} = (V, T)$ a* conformal[2] triangulation*, see [16], if the following conditions hold:*

1. *The union of the vertices of all triangles in $T$ is exactly $V$.*
2. *The union of all triangles in $T$ is exactly $\Omega$.*
3. *There are no empty (degenerate) triangles in $T$.*
4. *The intersection of any two triangles is either the empty set, a vertex, or an edge.*

We will use the terms *triangulation* and *mesh* interchangeably, depending on the context.

Also, let $\bar{r}$, $\bar{\rho}$, and $\bar{\Delta}$ denote the upper bounds on the circumradius, the circumradius-to-shortest edge ratio, and the area of the triangles, respectively.

**2.2. Sequential Delaunay Refinement.** The applications that use Delaunay meshes usually impose two constraints on the quality of the mesh elements: the upper bound $\bar{\Delta}$ on the element area and the lower bound $\bar{A}$ on the smallest angle. As shown by Miller, Talmor, Teng, and Walkington [29] and Shewchuk [34], in two dimensions the circumradius-to-shortest edge ratio $\rho$ of a triangle can be expressed in terms of its minimal angle $A$ as

$$\rho = \frac{1}{2 \sin A}. \tag{2.1}$$

As the circumradius-to-shortest edge ratio is the metric that is naturally optimized by the Delaunay refinement, see [29], we will consider $\bar{\Delta}$ and $\bar{\rho}$, along with a PSLG $\mathcal{X}$, as the parameters to the mesh generation algorithm.

Typically, a mesh generation procedure starts with constructing an initial mesh which conforms to the input vertices and segments, and then refines this mesh until the constraints $\bar{\Delta}$ and $\bar{\rho}$ are met. For sufficiently small values of $\bar{\Delta}$, resulting in large fine meshes, most of the time is spent reducing the area of the triangles. Although asymptotically $\mathcal{O}(n \log n)$ time is required to triangulate $n$ points, while refining an existing mesh is on average linear in the number of the triangles produced, for fairly large final meshes the latter time significantly dominates. In this paper, we focus on parallelizing the Delaunay refinement stage, which is usually the most memory- and computation-expensive. The general idea of the Delaunay refinement is to insert

[2]The term which sounds similar, but has a different meaning, is the *conforming triangulation*, which is used in the context of preserving constrained edges [32].

```
DelaunayRefinement($\mathcal{X}$, $\mathcal{M}$, $\bar{\Delta}$, $\bar{\rho}$)
Input: $\mathcal{X}$ is a PSLG which defines $\Omega$
       $\mathcal{M}$ is some Delaunay mesh over $\Omega$
       $\bar{\Delta}$ and $\bar{\rho}$ are desired upper bounds
Output: a conforming Delaunay mesh respecting $\bar{\Delta}$ and $\bar{\rho}$
 1  $Q \leftarrow \{t \in \mathcal{M} \mid (\rho(t) \geq \bar{\rho}) \vee (\Delta(t) \geq \bar{\Delta})\}$
 2  while $Q \neq \emptyset$
 3      Let $t \in Q$
 4      $p \leftarrow \odot(t)$
 5      Find $S$, a set of segments encroached upon by $p$
 6      if $S \neq \emptyset$
 7          SplitSegments($S$)
 8      else
 9          $\mathcal{M} \leftarrow \mathcal{M} \setminus \mathcal{C}_{\mathcal{M}}(p) \cup \{\triangle puv \mid (u,v) \in \mathcal{B}_{\mathcal{M}}(p)\}$
10      endif
11      Update $Q$
12  endwhile
13  return $\mathcal{M}$
```

Fig. 2.1. *The sequential Delaunay refinement algorithm. A more detailed description along with the segment splitting strategies, which always guarantee termination, can be found in [35].*

points in the circumcenters of the triangles that violate the required bounds, until there are no such triangles left. There are two main variations of the point insertion procedure: the Lawson's algorithm [26], which uses edge flips, and the B-W algorithm [5, 39], which is based on deleting the triangles that are no longer Delaunay and inserting new triangles that satisfy the Delaunay property. These algorithms produce equivalent results, see [34], thus, we will use only the B-W algorithm for our analysis. The following definition, adapted from [16], plays a major role in the study of the B-W algorithm.

Definition 2. *Let the* cavity $\mathcal{C}_{\mathcal{M}}(p)$ of point $p$ with respect to mesh $\mathcal{M}$ *be the set of triangles in $\mathcal{M}$, whose circumdisks include $p$: $\mathcal{C}_{\mathcal{M}}(p) = \{t \in \mathcal{M} \mid p \in \bigcirc(t)\}$.* We will write simply $\mathcal{C}(p)$ when $\mathcal{M}$ is clear from the context. Also, we will denote $\mathcal{B}_{\mathcal{M}}(p)$ to be the set of the edges which belong to only one triangle in $\mathcal{C}_{\mathcal{M}}(p)$, i.e., the external edges of $\mathcal{C}_{\mathcal{M}}(p)$.

The sequential Delaunay algorithms treat constrained (boundary) segments differently than triangle edges in the interior of the mesh, see [31, 34]. For a segment $s$, the unique circle with $s$ as a diameter is called the *diametral circle* of $s$. A vertex $p$ is said to *encroach upon* $s$, if it lies within the open diametral disk 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, and the cavity of the segment's midpoint is constructed and triangulated in a similar way.

For a thorough description of the sequential Delaunay refinement algorithms and the proofs of their termination, as well as the proofs of good grading and size optimality of the result, the reader can consult [16, 31, 34, 35] and the references therein. For our analysis, it is sufficient to consider the algorithm in Figure 2.1. The while-loop of the algorithm maintains the following invariant:

Loop Invariant 1 (Delaunay Triangulation). *$\mathcal{M}$ is (i) conformal and (ii) Delaunay.*

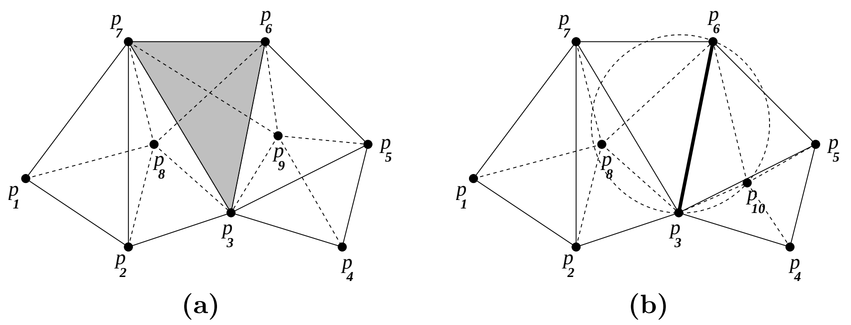

FIG. 2.2. **(a)** *If $\triangle p_3p_6p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$, where $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ and $\mathcal{C}(p_9) = \{\triangle p_3p_6p_7, \triangle p_3p_5p_6, \triangle p_3p_4p_5\}$, then the concurrent insertion of $p_8$ and $p_9$ results in a non-conformal mesh. The solid lines represent the edges of the initial triangulation, and the dashed lines represent the edges created by the insertion of $p_8$ and $p_9$.* **(b)** *If the 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 circumdisk, thus, violating the Delaunay property.*

**2.3. Theoretical Framework.** The parallelization of the sequential Delaunay refinement algorithm can be achieved by inserting the circumcenters of multiple triangles concurrently by several processors. In this subsection, we propose a theoretical framework which allows to select the candidate circumcenters in such a way that the Delaunay triangulation loop invariant holds throughout the run of the parallel algorithm, and, thus, all the quality guarantees of the sequential algorithm remain valid.

**2.3.1. Maintaining the Delaunay Triangulation Loop Invariant.** We expect a parallel Delaunay refinement algorithm to maintain Loop Invariant 1. The reason for a parallel algorithm to produce a non-conformal mesh can be a concurrent retriangulation of two or more cavities that have common triangles. Consider, for example, Figure 2.2(a). However, even if the cavities of two points do not have common triangles, Loop Invariant 1 can still be violated due to the creation of non-Delaunay triangles, i.e., triangles whose circumdisks include mesh vertices. This can happen if two cavities have common edges, see Figure 2.2(b). The following Lemma provides a sufficient condition for the simultaneous retriangulation of two cavities to yield a conformal Delaunay mesh.

LEMMA 1. *If $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ have no common triangles and do not share any triangle edges, then the independent insertion of $p_i$ and $p_j$ will result in a mesh which is both conformal and Delaunay.*

*Proof.* Since $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ do not intersect, the processors that are working independently on retriangulating $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ will not attempt to simultaneously delete the same triangles or create overlapping triangles, thus, the resulting mesh will be conformal.

To show that the mesh will also satisfy the Delaunay criterion, we need to recall the Delaunay Lemma, see [13, 16], which states that if the empty circumdisk criterion holds for every pair of adjacent triangles, the triangulation is globally Delaunay. The condition that $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ do not share any triangle edges implies that every external edge of $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ is incident upon some triangle which belongs neither to $\mathcal{C}(p_i)$ nor to $\mathcal{C}(p_j)$. These external triangles together with the ones created by the retriangulation of $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ will locally satisfy the Delaunay property, see [34]. Hence, the mesh will be globally Delaunay. $\square$

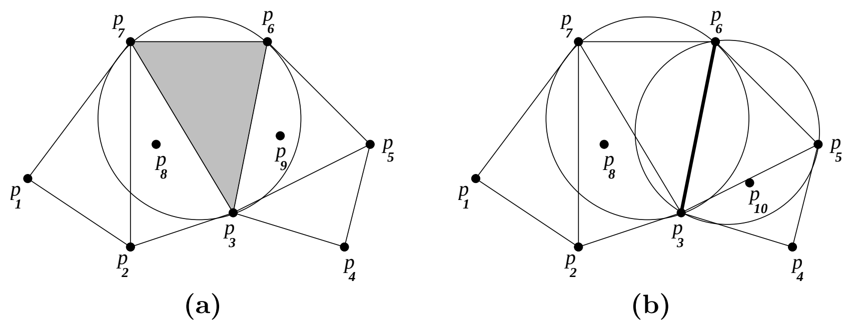

FIG. 2.3. **(a)** *If* $\triangle p_3p_6p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$*, then* $\|p_8 - p_9\| < 2r(\triangle p_3p_6p_7) < 2\bar{r}$*.* **(b)** *If edge* $p_3p_6$ *is shared by* $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ *and* $\mathcal{C}(p_{10}) = \{\triangle p_3p_5p_6, \triangle p_3p_4p_5\}$*, then* $\|p_8 - p_{10}\| < 2r(\triangle p_3p_6p_7) + 2r(\triangle p_3p_5p_6) < 4\bar{r}$*.*

In practice, though, this Lemma may not be very useful, since it requires the construction and the comparison of the cavities for all candidate points. The next Lemma achieves a more practical result.

LEMMA 2. *Let* $\bar{r}$ *be the upper bound imposed on all of the circumradii belonging to the triangles in the mesh* $\mathcal{M}$*. Considering points* $p_i, p_j \in \Omega$ *in the two-dimensional domain* $\Omega$ *we have the following. If the distance between them fulfills the estimation* $\|p_i - p_j\| \geq 4\bar{r}$*, then the corresponding cavities* $\mathcal{C}(p_i)$ *and* $\mathcal{C}(p_j)$ *do not intersect, and furthermore, do not have any triangle edge in common.*

*Proof.* We will prove this Lemma by contradiction by considering two cases:

(i) Suppose $\|p_i - p_j\| \geq 4\bar{r}$ and $\mathcal{C}(p_i) \cap \mathcal{C}(p_j) \neq \emptyset$. Let $t \in \mathcal{C}(p_i) \cap \mathcal{C}(p_j)$, see Figure 2.3(a). By the definition of the cavity, $p_i \in \bigcirc(t)$ and $p_j \in \bigcirc(t)$, hence $\|p_i - p_j\| < 2r(\triangle p_kp_lp_m) < 2\bar{r} < 4\bar{r}$, which contradicts our assumption.

(ii) Suppose $\|p_i - p_j\| \geq 4\bar{r}$ and there exists an edge $p_kp_l$ which is shared by $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$, see Figure 2.3(b). Let $\triangle p_kp_lp_m \in \mathcal{C}(p_i)$ and $\triangle p_kp_lp_n \in \mathcal{C}(p_j)$. Obviously, $\|p_k - p_i\| < 2r(\triangle p_kp_lp_m)$ and $\|p_k - p_j\| < 2r(\triangle p_kp_lp_n)$. From the triangle inequality it follows that $\|p_i - p_j\| < 2r(\triangle p_kp_lp_m) + 2r(\triangle p_kp_lp_n) < 4\bar{r}$, which again contradicts the assumption.

□

Theorem 1 is the immediate consequence of Lemmata 1 and 2 and can be used for an efficient constant-time verification of whether two points are independent, and, hence, can be inserted concurrently:

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

**2.3.2. The Circumradius Upper Bound Invariant.** For Theorem 1 to be applicable throughout the run of the algorithm, we need to prove another invariant:

LOOP INVARIANT 2 (Circumradius Upper Bound). *The condition that* $\bar{r}$ *is the upper bound on triangle circumradius in the entire mesh holds both before and after the insertion of a point.*

Next, we show that the execution of the B-W algorithm, either sequentially or in parallel, does not violate Loop Invariant 2.

Let the *reflection of disk* $\bigcirc(\triangle p_kp_lp_m)$ *about edge* $p_kp_l$ be the disk $\bigcirc'_{p_kp_l}(\triangle p_kp_lp_m)$ that has the same radius, whose circle passes through the points $p_k$ and $p_l$, and whose center lies on the other side of edge $p_kp_l$ from point $p_m$. See Figure 2.4.

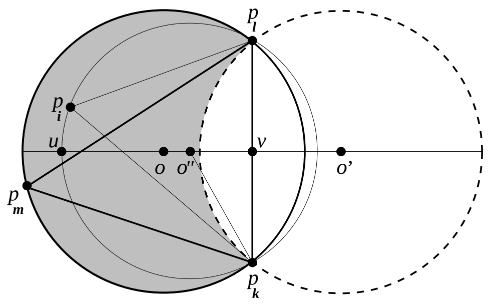

FIG. 2.4. *The solid circle corresponds to $\bigcirc(\triangle p_k p_l p_m)$ with the center in point $o$, and the dashed circle corresponds to $\bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$ with the center in point $o'$. Point $o''$ is the center of the variable-radius disk, whose circle passes through $p_k$ and $p_l$. We prove that for any point $p_i$ inside the shaded region, $r(\triangle p_k p_l p_i) < r(\triangle p_k p_l p_m)$.*

LEMMA 3. *For any point $p_i$ inside the region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$, the following condition is fulfilled having the assumption outlined in Figure 2.4: $r(\triangle p_k p_l p_i) < r(\triangle p_k p_l p_m)$.*

*Proof.* Let $o''$ be the center of the disk $\bigcirc(\triangle p_k p_l p_i)$, where $p_i$ is any point inside the shaded region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$. $o''$ has to lie on the straight line through $o$ and $o'$. Let $v$ be the point of intersection of this straight line with the edge $p_k p_l$ (i.e., the midpoint of $p_k p_l$), and $u$ be the point of intersection of the straight line with the circle of $\bigcirc(\triangle p_k p_l p_i)$ inside the shaded region. Let $x = \|u - v\|$. We will express the radius of $\bigcirc(\triangle p_k p_l p_i)$ as a function of $x$: $r(\triangle p_k p_l p_i) = f(x)$.

Let us consider the triangle $\triangle p_k v o''$ with the right angle at the vertex $v$. If we denote $\|p_k - p_l\| = a$, and $\|v - o\| = \|v - o'\| = b$, then by observing that $\|p_k - v\| = a/2$, $\|v - o''\| = x - f(x)$, and $\|o'' - p_k\| = f(x)$, we have:

$$f^2(x) = \left(\frac{a}{2}\right)^2 + (x - f(x))^2, \quad \text{or} \quad f(x) = \frac{a^2}{8x} + \frac{x}{2}, \quad \text{where } 0 < x < \infty.$$

$f(x)$ is convex everywhere on $0 < x < \infty$ since its second derivative is positive:

$$f''(x) = \frac{a^2}{4x^3} > 0 \quad \text{for } 0 < x < \infty.$$

In addition,

$$\begin{aligned} f(x)|_{o''=o'} &= f(r(\triangle p_k p_l p_m) - b) = r(\triangle p_k p_l p_m), \\ f(x)|_{o''=o} &= f(r(\triangle p_k p_l p_m) + b) = r(\triangle p_k p_l p_m) \end{aligned}$$

Using Jensen's inequality, see [40], for $0 \le \lambda \le 1$ and $r(\triangle p_k p_l p_m) - b < x < r(\triangle p_k p_l p_m) + b$, we can derive the following:

$$\begin{aligned} f(x) &< \lambda f(r(\triangle p_k p_l p_m) - b) + (1 - \lambda) f(r(\triangle p_k p_l p_m) + b) \\ &= \lambda r(\triangle p_k p_l p_m) + (1 - \lambda) r(\triangle p_k p_l p_m) \\ &= r(\triangle p_k p_l p_m). \end{aligned}$$

□

LEMMA 4. *Let $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$ and $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$. Then the following inequality holds: $r(\triangle p_k p_l p_i) < \max(r(\triangle p_k p_l p_m), r(\triangle p_k p_n p_l))$.*

*Proof.* There are two cases:

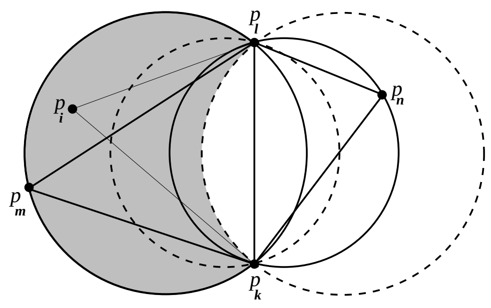

FIG. 2.5. $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$, $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, *and* $r(\triangle p_k p_l p_m) > r(\triangle p_k p_n p_l)$.

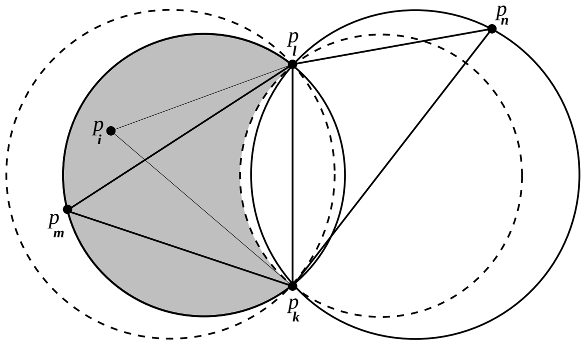

FIG. 2.6. $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$, $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, *and* $r(\triangle p_k p_l p_m) \leq r(\triangle p_k p_n p_l)$.

(i) $r(\triangle p_k p_l p_m) > r(\triangle p_k p_n p_l)$, see Figure 2.5. In this case, $p_i$ has to lie inside the region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc(\triangle p_k p_n p_l)$, which is a subset of the region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$, and, according to Lemma 3, $r(\triangle p_k p_l p_i) < r(\triangle p_k p_l p_m)$.

(ii) $r(\triangle p_k p_l p_m) \leq r(\triangle p_k p_n p_l)$, see Figure 2.6. Again, $p_i$ has to lie in the region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc(\triangle p_k p_n p_l)$, which in this case is a subset of $\bigcirc'_{p_k p_l}(\triangle p_k p_n p_l) \setminus \bigcirc(\triangle p_k p_n p_l)$, and, by Lemma 3, $r(\triangle p_k p_l p_i) < r(\triangle p_k p_n p_l)$.

$\square$

Thus, we may state the following theorem:

THEOREM 2. *Loop Invariant 2 holds.*

*Proof.* The B-W algorithm creates only triangles of the form $\triangle p_k p_l p_i$, where there exists $\triangle p_k p_l p_m \in \mathcal{M}$ such that $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$. There are two cases:

(i) There exists $\triangle p_k p_n p_l \in \mathcal{M}$ such that $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, in other words, $p_k p_l$ is not a boundary segment. In this case, by Lemma 4, $r(\triangle p_k p_l p_i) < \max(r(\triangle p_k p_l p_m), r(\triangle p_k p_n p_l))$.

(ii) $p_k p_l$ is a boundary segment. Then $p_i$ must be outside the diametral circle of $p_k p_l$, otherwise, $p_i$ wouldn't have been inserted. Consequently, $p_i$ lies in the region $\bigcirc(\triangle p_k p_l p_m) \setminus \bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$, and by Lemma 3, $r(\triangle p_i p_k p_l) < r(\triangle p_k p_l p_m)$.

$\square$

**2.3.3. Adjusting the Degree of Concurrency.** We have shown how with a simple and inexpensive test one can check whether two points (i.e., circumcenters) can be inserted independently. Now it remains to provide a way of finding enough independent circumcenters, so that all processors can be kept busy inserting them. This task can be accomplished by decreasing all triangle circumradii below some value $\bar{r}$ with the purpose of increasing the number of pairwise independent circumcenters. Fortunately, this can be easily done by applying the sequential Delaunay refinement procedure as a preprocessing step. In this step, we use the initial parameter $\bar{\rho}$ and select parameter $\bar{\Delta}$ as

$$\bar{\Delta} = \frac{\bar{r}^2}{4\bar{\rho}^3}, \tag{2.2}$$

where $\bar{r}$ is computed based on the number of available processors and the size of the domain (see Section 3).

Such preprocessing allows us to decrease the maximal circumradius of the triangles in a mesh automatically as a result of decreasing the maximal circumradius-to-shortest edge ratio and the maximal area. In the rest of this subsection, we prove equation (2.2).

We start by proving an auxiliary Lemma, which relates the length of any side of a triangle to the sum of the cotangents of the adjacent angles and the triangle's area.

LEMMA 5. *If $a$, $b$, $c$ and $A$, $B$, $C$ are the lengths of the sides and the measures of the angles of a triangle respectively, then*

$$a^2 = 2(\cot B + \cot C)\Delta, \tag{2.3}$$

*where $\Delta$ is the area of the triangle.*

*Proof.* The cyclical application of the law of cosines

$$\cos A = \frac{b^2 + c^2 - a^2}{2bc} \tag{2.4}$$

coupled with the formula computing the area of the triangle

$$\Delta = \frac{1}{2} bc \sin A \tag{2.5}$$

allows us to obtain the following formula:

$$\cot A + \cot B + \cot C = \frac{a^2 + b^2 + c^2}{4\Delta}. \tag{2.6}$$

Rearranging (2.5) and substituting into (2.4) yields

$$\cot A = \frac{b^2 + c^2 - a^2}{4\Delta}. \tag{2.7}$$

Finally, by coupling (2.6) and (2.7), we obtain (2.3). $\square$

Now, we show how an expression involving the smallest angle of a triangle bounds the sum of the cotangents of the other angles.

LEMMA 6. *If $A \leq B \leq C$ are the measures of the angles of a triangle, then*

$$\cot B + \cot C \leq \frac{1}{\sin A}.$$

*Proof.* Since $A$, $B$, $C$ are positive summing themselves up to $\pi$, we can define the following function:

$$f(B) = \cot B + \cot C = \cot B + \cot(\pi - A - B) = \cot B - \cot(A + B),$$

where

$$0 < A \le B \le \frac{\pi - A}{2} \le C \le \pi - 2A, \text{ and } A \le \frac{\pi}{3}. \tag{2.8}$$

$f(B)$ is continuous on $\left[A, \frac{\pi-A}{2}\right]$ and has no points of local maxima which satisfy (2.8), thus, it can reach its maxima only in the ends of the interval $\left[A, \frac{\pi-A}{2}\right]$. Direct substitution yields

$$f(A) = \frac{1}{2\sin A \cos A} \le \frac{1}{2\sin A\ \cdot \frac{1}{2}} = \frac{1}{\sin A}$$

and

$$f\left(\frac{\pi - A}{2}\right) = 2\frac{1 - \cos A}{\sin A} \le 2\frac{1 - \frac{1}{2}}{\sin A} = \frac{1}{\sin A}.$$

Hence,

$$f(B) \le \frac{1}{\sin A}, \quad \forall B \in \left[A, \frac{\pi - A}{2}\right].$$

□

Finally, assuming that the bounds $\bar{\rho}$ and $\bar{\Delta}$ on the triangle area and the circumradius-to-shortest edge ratio, respectively, hold for all triangles in the mesh, we find out the upper bound $\bar{r}$ on the circumradius of triangles.

LEMMA 7. *Let a triangle with the circumradius $r$ and the side lengths $a \le b \le c$ have the circumradius-to-shortest edge ratio $\rho$ bounded by $\bar{\rho}$: $\rho = r/a < \bar{\rho}$, and the area bounded by $\bar{\Delta}$: $\Delta < \bar{\Delta}$. Then*

$$r < 2(\bar{\rho})^{3/2}\sqrt{\bar{\Delta}}.$$

*Proof.* From $\rho = r/a$, or $r = \rho a$, using Lemma 5 it follows that

$$r = \rho a = \rho\sqrt{2(\cot B + \cot C)\Delta}$$

and Lemma 6 implies that

$$r \le \rho\sqrt{2\frac{1}{\sin A}\Delta} = \rho\sqrt{\frac{2\Delta}{\sin A}}. \tag{2.9}$$

By substituting $\sin A$ from (2.1) into (2.9), we have:

$$r \le 2\rho^{3/2}\sqrt{\Delta} < 2(\bar{\rho})^{3/2}\sqrt{\bar{\Delta}}.$$

□

The following theorem directly follows from Lemma 7:

THEOREM 3. *If $\bar{\rho}$ and $\bar{\Delta}$ are upper bounds on the triangle area and the circumradius-to-shortest edge ratio, respectively, then $\bar{r}$, computed using the relation*

$$\bar{r} = 2(\bar{\rho})^{3/2}\sqrt{\bar{\Delta}},$$

*is an upper bound on the triangle circumradius.*

```
ParameterizedDelaunayRefinement($\mathcal{X}$, $\mathcal{M}$, $\bar{\Delta}$, $\bar{\rho}$, $\sigma()$)
Input: $\mathcal{X}$ is a PSLG which defines $\Omega$
       $\mathcal{M}$ is some Delaunay mesh over $\Omega$
       $\bar{\Delta}$ and $\bar{\rho}$ are desired upper bounds
       $\sigma()$ is a boolean predicate which takes point coordinates as input
Output: a conforming Delaunay mesh $\mathcal{M}$ such that
            $\forall t \in \mathcal{M} \;\; \sigma(\odot(t)) \Rightarrow (\rho(t) < \bar{\rho}) \wedge (\Delta(t) < \bar{\Delta})$
1   $Q \leftarrow \{t \in \mathcal{M} \mid (\rho(t) \geq \bar{\rho}) \vee (\Delta(t) \geq \bar{\Delta})\}$
2   for each $t \in Q$
3       $p \leftarrow \odot(t)$
4       if $\sigma(p)$
5           Find $S$, a set of segments encroached upon by $p$
6           if $S \neq \emptyset$
7               SplitSegments($S$)
8           else
9               $\mathcal{M} \leftarrow \mathcal{M} \setminus \mathcal{C}_{\mathcal{M}}(p) \cup \{\triangle puv \mid (u, v) \in \mathcal{B}_{\mathcal{M}}(p)\}$
10          endif
11          Update $Q$
12      endif
13  endfor
14  return $\mathcal{M}$
```

Fig. 3.1. *An augmented sequential Delaunay refinement algorithm.*

**3. Coarse Grain Partitioning.** One possible way to break the meshing problem of the entire domain up into smaller independent subproblems is to partition the domain into subdomains in such a way that the circumcenters of the triangles in each subdomain can be inserted concurrently with the circumcenters in any other subdomain.

If we consider a selected spatial region as a subdomain, which is refined sequentially by one processor, then the layer of triangles whose circumcenters are within $2\bar{r}$ from any circumcenter in the region, forms the buffer zone of the entire region. The partitioning and the decoupling of the subdomains can be achieved by creating a special buffer zone around every subdomain, so that the insertion of a point in one subdomain can modify the triangles only inside this subdomain and its buffer zone, but the changes will not propagate to other subdomains and their buffer zones. After refining the subdomains, the buffer zones can be refined in a similar way.

We have chosen a simple and efficient way to partition the domain, which consists in dividing its bounding box into squares. One of the main advantages of this approach is the efficiency of assigning triangles to subdomains based on the coordinates of their circumcenters. A small adjustment (see Figure 3.1) to the sequential Delaunay refinement algorithm (Figure 2.1) allows the insertion of only those circumcenters which satisfy some user defined condition (i.e., being inside a given rectangle). The drawback is that the amount of work assigned to different processors can vary significantly. However, we developed a runtime system [2] for dynamic load balancing which solves this problem and thus we do not have to address it here.

The selection of subdomain box sizes is closely related to the scheduling of the refinement phases. First, Theorem 1 requires that the regions simultaneously refined by different processors are at least $4\bar{r}$ away. Second, the refinement of a region can create poor quality triangles near the boundary in a neighboring region, which has been refined in a previous iteration (see Fig. 3.2). To avoid performing multiple refinement iterations of the same region, some overlapping among the regions needs to be introduced. The degree of overlapping depends on the partitioning of $\Omega$. We partition the domain $\Omega$ into a two-dimensional lattice of squares, arrange the processors into logical rows and columns, and assign to each processor a fraction $(N/P)$ of the lattice, where $N$ is the number of cells and $P$ is the number of processors, which it is responsible to mesh. We ensure global mesh consistency, by forcing each processor to hold and exchange $(4\bar{r})$-wide layers of triangles near the boundaries of their assigned subdomains. Each processor in position $(i, j)$ communicates only with its neighbors in positions $(s, t)$, such that $|i - s| \leq 1$ and $|j - t| \leq 1$.

The entire parallel algorithm is summarized in Figure 3.2. The scheduling of the refinement and the communication phases is also schematically presented in Figure 4.3. A sequence of meshes created by four processors for the pipe cross-section model (Figure 4.1) is shown in Figure 4.2.

## 4. Parallel Implementation.

For the experimental evaluation of our method, we triangulated (i) 1000 random points uniformly distributed in a unit square and (ii) the pipe cross-section model (Figure 4.1). We used the “Triangle” [33] as a sequential mesher on each processor. The “Triangle”facilitates the use of a user-defined function for deciding which triangles should be considered “big” and queued for refinement. However, apparently, it doesn’t provide a mechanism for the user to decide at run-time which triangles are “bad” in terms of the circumradius-to-shortest edge ratio or minimal angle. This fact caused us to insert a test for the triangle circumcenter position inside the “Triangle” code and recompile it. We believe that in the future releases of sequential mesh generation codes the facility for defining user-supplied functions, which test the suitability of both types of triangles, can be easily provided.

Another technical detail that we ran into while running our tests, is the substantial difference in time required by the “Triangle” to refine an existing mesh and to construct a fine mesh without taking advantage of a previously constructed coarser mesh. Contrary to our expectations, building a fine mesh anew turned out to be much faster. For example, in the case of the contour of the letter “A” which is included in the “Triangle” distribution, the “Triangle” is about six times faster in retriangulating a given set of points than in reconstructing the mesh data structure from a list of triangles. That is why, when moving a part of the mesh from one processor to another, we decided to migrate only the set of points and boundary edges and retriangulate them as necessary. Since the Delaunay triangulation of a given set of points, is unique provided that there are no four cocircular points, see [34], this process will not destroy the boundary conformity. In the presence of cocircular points near the boundary, the enforcement of boundary segments causes the required edge flips. The additional advantage of this optimization is the decrease of the network traffic.

We have conducted our experiments on the Sciclone cluster computing system at the College of William and Mary using its two tightly-coupled subclusters: “whirlwind” — 64 single-cpu Sun Fire V120 servers (650 MHz, 1 GB RAM) and “twister” — 32 dual-cpu Sun Fire 280R servers (900 MHz, 2 GB RAM). The scaled and the fixed workload evaluation for a mesh of a unit square is shown in Table 4.1, and for the pipe model — in Table 4.3. We can see that in the pipe case the algorithm performs slightly worse, which happens because of the load imbalance due to the empty space inside pipe holes. Table 4.4 shows a comparison of a distributed and a shared memory implementations on the “hurricane” subcluster.

```
ParallelDelaunayMeshing($\mathcal{X}$, $\bar{\Delta}$, $\bar{\rho}$, $P$, $p$)
Input: $\mathcal{X}$ is a PSLG which defines $\Omega$
        $\bar{\Delta}$ and $\bar{\rho}$ are desired upper bounds
        $P$ is the total number of processors (we assume $\sqrt{P}$ is integer)
        $p$ is the index of the current processor, $0 \le p \le P-1$
Output: a distributed conforming Delaunay mesh respecting $\bar{\Delta}$ and $\bar{\rho}$
1   Calculate $row(p)$ and $column(p)$ of the current processor
        // $0 \le row(i), column(i) \le \sqrt{P}-1,\ 0 \le i \le P-1$
2   Let $l$ be the longest dimension of the bounding box of $\Omega$
3   $\bar{r} \leftarrow l/(4\sqrt{P})$
4   if $p = 0$
5       Construct $\mathcal{M}_1$, a constrained Delaunay mesh of $\Omega$
6       $\bar{\Delta}_1 \leftarrow \bar{r}^2/(4\bar{\rho}^3)$
7       $\mathcal{M}_1 \leftarrow$ ParameterizedDelaunayRefinement($\mathcal{X}$, $\mathcal{M}_1$, $\bar{\Delta}_1$, $\bar{\rho}$, true)
8       Assign triangles in $\mathcal{M}_1$ to cells $\{d_{ij} \mid 0 \le i, j \le 4\sqrt{P}-1\}$
            of side length $4\bar{r}$ based on the coordinates of their circumcenters
9       for each $i \in \{0, \ldots, P-1\}$
10          Send cells $\{d_{m,n} \mid 4row(i)-1 \le m \le 4(row(i)+1),$
                $4column(i)-1 \le n \le 4(column(i)+1)\}$ to processor $i$
11      endfor
12  endif
13  Receive cells $\{d_{ij} \mid 1 \le i, j \le 4\}$ of local mesh $\mathcal{M}$
        // Phase (0) is over
14  Send cells $\{d_{4,j} \mid 1 \le j \le 3\}$ to processor in $(row(p)+1,\ column(p))$
15  Send cells $\{d_{i,4} \mid 1 \le i \le 3\}$ to processor in $(row(p),\ column(p)+1)$
16  Send cell $d_{4,4}$ to processor in $(row(p)+1,\ column(p)+1)$
17  Receive cells $\{d_{0,j} \mid 1 \le j \le 3\}$ from processor in $(row(p)-1,\ column(p))$
18  Receive cells $\{d_{i,0} \mid 1 \le i \le 3\}$ from processor in $(row(p),\ column(p)-1)$
19  Receive cell $d_{0,0}$ from processor in $(row(p)-1,\ column(p)-1)$
        // Phase (1) is over
20  Let $(x_0, y_0)$ be the upper left coordinate of local cell $d_{0,0}$
21  $\sigma(q) \leftarrow (q \in [x_0+2\bar{r}, y_0+2\bar{r}] \times [x_0+14\bar{r}, y_0+14\bar{r}])$
22  $\mathcal{M} \leftarrow$ ParameterizedDelaunayRefinement($\mathcal{X}$, $\mathcal{M}$, $\bar{\Delta}$, $\bar{\rho}$, $\sigma()$)
        // Phase (2) is over
        // Phases (3)–(10) are performed by analogy (Figure 4.3)
23  return $\mathcal{M}$
```

Fig. 3.2. *A parallel Delaunay meshing algorithm. In order to simplify the presentation of the algorithm, we do not concern with the index out-of-range problems (e.g. $d_{-1}$), which we handle in the implementation.*

Finally, we didn't have to write any mesh generation code, since for both the preprocessing and the refinement it was sufficient to call the "Triangle" as a library.

**5. Conclusions.** We introduced a parallel coarse-grained mesh generation algorithm based on a block domain decomposition, which can be used in a variety of contexts. First, as we showed, the parallel algorithm can be implemented efficiently on distributed- and shared-memory memory systems by exploiting the fact that only local communication between processors is necessary, the data can be sent in large blocks at the beginning of each computation phase, and the number of the

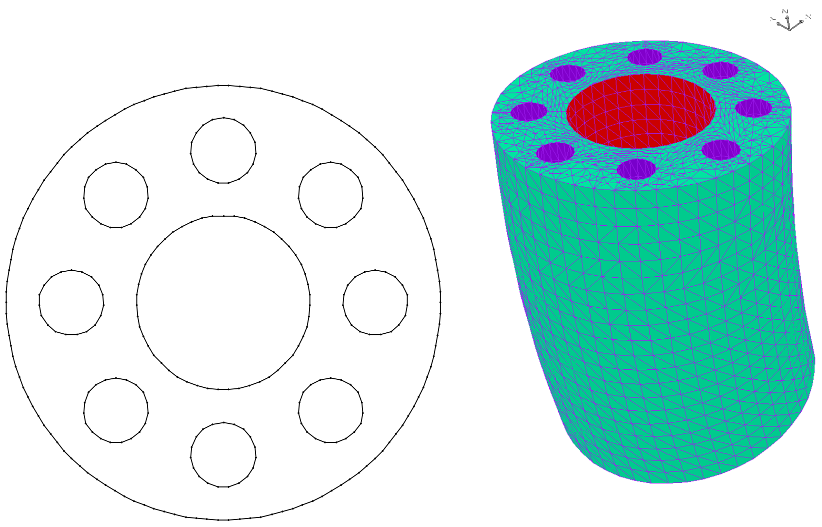

Fig. 4.1. *A cross section and a cartoon pipe of a regenerative cooled pipe geometry which came from the testing of a rocket engine at NASA Marshall. NASA had to slightly bend one of the regenerative cooled "pipes" that contained combustion gases in order to fit a rocket engine onto a test stand. In testing, this curved pipe failed due to an unanticipated increase of heating. Since this was a one-off design, NASA couldn't afford to do the extensive testing. The failure caused them to scrap that particular project since it destroyed the engine under test. This is an example of why simulation is useful in reducing the cost.*

| $P$ | Scaled workload | | | Fixed workload, 23.8M elements, $\bar{\Delta} = 5 \times 10^{-8}$, total time, sec. |
|---|---|---|---|---|
| | Total Time, sec. | Number of Elements, Millions | Area Bound $\bar{\Delta}$ $\times 10^{-8}$ | |
| 4 | 293.7 | 23.8 | 5.00 | 293.7 |
| 9 | 294.7 | 58.8 | 2.22 | 122.3 |
| 16 | 295.4 | 109.3 | 1.25 | 67.2 |
| 25 | 296.8 | 175.4 | 0.80 | 43.3 |
| 36 | 293.4 | 255.0 | 0.56 | 30.2 |
| 49 | 294.5 | 352.6 | 0.41 | 23.3 |
| 64 | 300.1 | 470.7 | 0.31 | 19.4 |
| 81 | 296.2 | 587.8 | 0.25 | 17.0 |
| 100 | 300.3 | 738.9 | 0.20 | 15.6 |
| 121 | 293.7 | 873.5 | 0.17 | 15.1 |

Table 4.1
*Parallel Delaunay refinement for a mesh of a unit square. For all tests,* $\bar{\rho} = 1.41$.

communication phases is a small constant (five). In this paper, we addressed the code reuse issue from the viewpoint of concurrency; the design of the generic application programming interface is out of the scope, see e.g. [3] for a discussion. Second, we designed an out-of-core algorithm [24] which utilizes the absence of communication during subdomain refinement phases. Third, a shared memory implementation can use the sufficient condition of the circumcenter independence to guide the fine-grained parallelism. However, our experimental evaluation [1] of fine-grain parallelism

| $P$ | Sequential Prerefinement | Parallel Refinement | Communication and Synchronization | Filling in and Merging Cells |
|---|---|---|---|---|
| 4 | 0.24 | 218.7 | 32.7 | 39.9 |
| 9 | 0.44 | 220.0 | 35.4 | 40.3 |
| 16 | 0.71 | 218.8 | 37.1 | 40.0 |
| 25 | 1.05 | 219.1 | 39.1 | 40.8 |
| 36 | 1.43 | 218.6 | 36.1 | 40.0 |
| 49 | 1.88 | 217.7 | 37.5 | 40.7 |
| 64 | 2.49 | 221.0 | 40.7 | 41.5 |
| 81 | 3.04 | 216.6 | 46.2 | 41.1 |
| 100 | 3.69 | 218.4 | 46.5 | 41.5 |
| 121 | 4.42 | 210.3 | 53.6 | 40.9 |

Table 4.2

*A breakdown of the execution time for a single processor in the case of the unit square, seconds. The Communication and Synchronization column shows the time to send the data to the neighboring processors and receive the data from them, which includes waiting for their completion of the current refinement task.*

| $P$ | Scaled workload | | | Fixed workload, 14.6M elements, $\bar{\Delta} = 2 \times 10^{-2}$, total time, sec. |
|---|---|---|---|---|
| | Total Time, sec. | Number of Elements, Millions | Area Bound $\bar{\Delta}$, $\times 10^{-2}$ | |
| 4 | 179.1 | 14.6 | 2.00 | 179.1 |
| 9 | 204.8 | 32.8 | 0.88 | 105.6 |
| 16 | 225.7 | 58.3 | 0.50 | 80.5 |
| 25 | 239.8 | 91.1 | 0.32 | 54.2 |
| 36 | 255.3 | 131.2 | 0.22 | 40.1 |
| 49 | 253.7 | 178.6 | 0.16 | 30.2 |
| 64 | 273.6 | 233.3 | 0.13 | 21.9 |
| 81 | 258.6 | 295.3 | 0.10 | 17.1 |
| 100 | 232.2 | 364.6 | 0.08 | 15.0 |
| 121 | 212.7 | 441.1 | 0.06 | 14.8 |

Table 4.3

*Parallel Delaunay refinement for a mesh of the pipe model.* $\bar{\rho} = 1.41$.

in a B-W kernel showed that current SMTs are not capable to achieve performance improvement, while the experiments on a simulated SMT indicate that with modest hardware support it is possible to exploit fine-grain parallelism opportunities. Fourth, our future work includes the extension of the framework to the three-dimensional domains. We also plan to improve the performance of our parallel implementation, for more irregular geometries, by addressing the work-load balancing problem using our runtime system [2] and improving the memory management for a single processor. Finally, we are currently working on the extension of the theoretical framework for the construction of non-uniform (graded) meshes [7].

**6. Acknowledgments.** We thank Andriy Kot for the shared-memory implementation of the algorithm.

We appreciate the very detailed comments from the anonymous reviewers which helped us improve the manuscript.

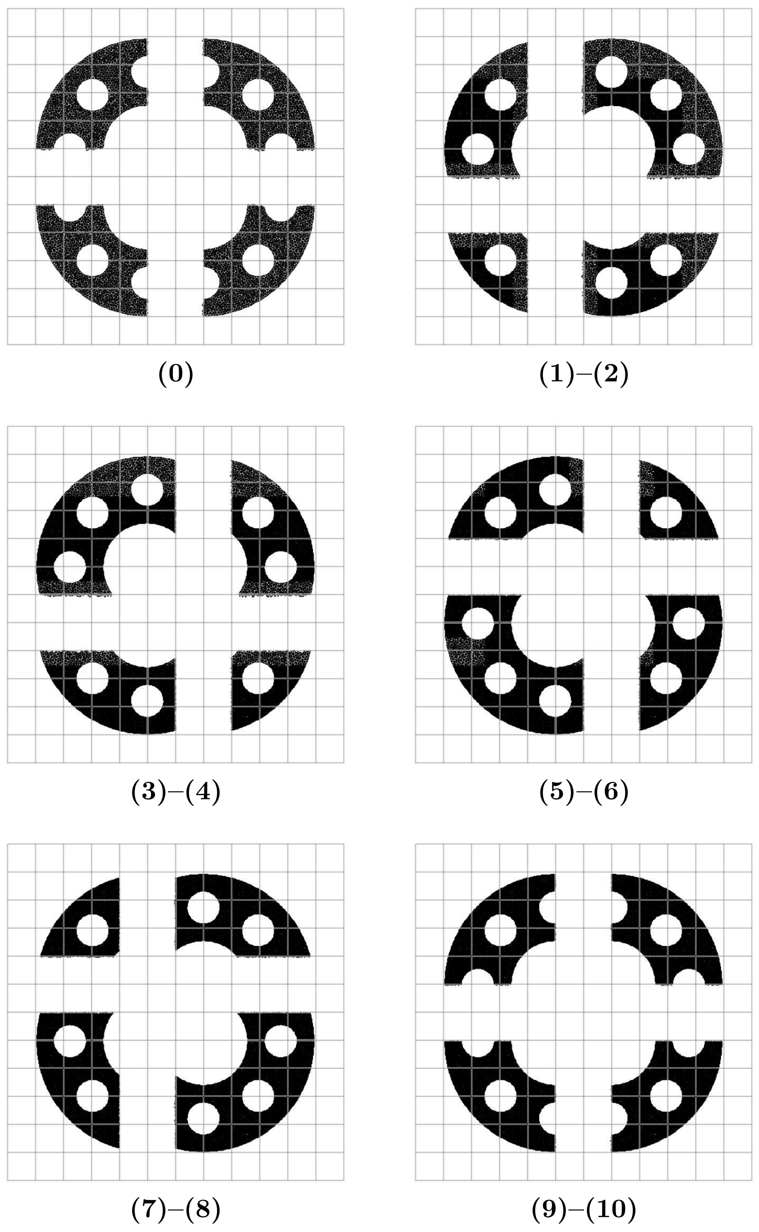

Fig. 4.2. *Snapshots of a mesh distributed among four processors. Each picture corresponds to the specified phases schematically shown in Figure 4.3. Dark areas are the parts of the mesh that have already been refined. Grey areas are those that have not been refined. Each processor starts with an initial coarse mesh and ends up with a refined mesh covering the same area.*

| Number of processors | Time, sec. MPI | Time, sec. OpenMP | Number of Elements, Millions |
|---|---|---|---|
| 4 | 220.3 | 214.1 | 14.6 |

Table 4.4
*Pipe cross-section, distributed (MPI) and shared (OpenMP) memory implementations.*

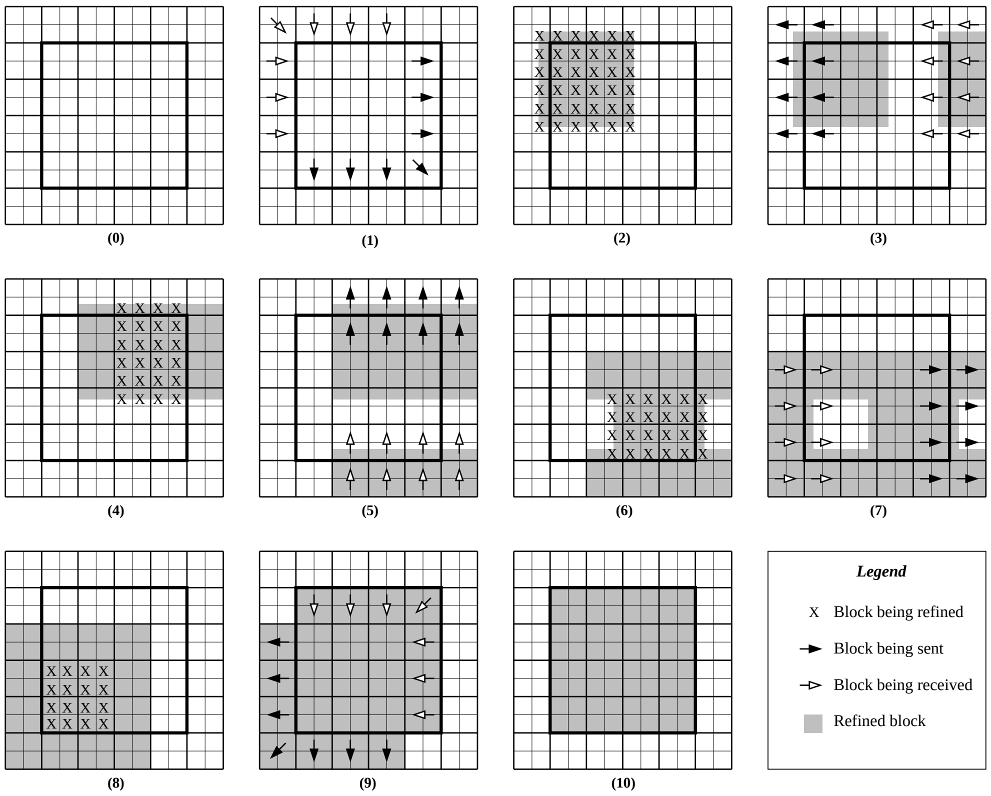

FIG. 4.3. *A schedule of the parallel Delaunay refinement and communication on distributed memory. Each phase (0)–(10) depicts the actions performed by a single processor. The smallest cells have side length $2\bar{r}$, they are the atomic units of refinement. Since the refinement may leave fractions of big triangles, whose circumcenters are outside the small cell, the refined cells sometimes are not shaded completely. The bigger cells, which consist of 4 small cells, are the atomic units of data migration. The thick lines outline the subdomain assigned to the given processor. A processor also holds parts of its neighbors' meshes. Arrows represent the direction of data migration.*

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.

## REFERENCES

[1] Christos D. Antonopoulos, Xiaoning Ding, Andrey N. Chernikov, Filip Blagojevic, Dimitris S. Nikolopoulos, and Nikos P. Chrisochoides, *Multigrain parallel Delaunay mesh generation: Challenges and opportunities for multithreaded architectures*, in Proceedings of the 19th annual international conference on Supercomputing, ACM Press, 2005, pp. 367–376.

[2] 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 (2004), pp. 183–192.

[3] Guntram Berti, *Gral - the grid algorithms library.*, in International Conference on Computational Science (3), Peter M. A. Sloot, Chih Jeng Kenneth Tan, Jack Dongarra, and Alfons G. Hoekstra, eds., vol. 2331 of Lecture Notes in Computer Science, Springer, 2002,

pp. 745–754.

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

[5] Adrian Bowyer, *Computing Dirichlet tesselations*, Computer Journal, 24 (1981), pp. 162–166.

[6] 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, ACM Press, 2004, pp. 48–57.

[7] ———, *Parallel 2D graded guaranteed quality Delaunay mesh refinement*, in Proceedings of the 14th International Meshing Roundtable, Springer, Sept. 2005, pp. 505–517.

[8] 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, Northwestern University, Evanston, IL, 1997, pp. 89–96.

[9] P.L. Chew, N. Chrisochoides, K. Pingali, and S. Stodghil, *Meshing using independent sets*. Unpublished manuscript, 1999.

[10] Nikos Chrisochoides and Démian Nave, *Parallel Delaunay mesh generation kernel*, Int. J. Numer. Meth. Engng., 58 (2003), pp. 161–176.

[11] Nikos P. Chrisochoides, *A survey of parallel mesh generation methods*, Tech. Report BrownSC-2005-09, Brown University, 2005. 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.

[12] Hugues L. de Cougny, Mark S. Shephard, and Can Ozturan, *Parallel three-dimensional mesh generation*, Computing Systems in Engineering, 5 (1994), pp. 311–323.

[13] Boris N. Delaunay, *Sur la sphere vide*, Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Mataematicheskii i Estestvennyka Nauk, 7 (1934), pp. 793–800.

[14] Suchuan Dong, Didier Lucor, and George Em Karniadakis, *Flow past a stationary and moving cylinder: DNS at Re=10,000*, in 2004 Users Group Conference (DOD_UGC'04), 2004, pp. 88–95.

[15] Herbert Edelsbrunner and Damrong Guoy, *Sink-insertion for mesh improvement*, in Proceedings of the Seventeenth Annual Symposium on Computational Geometry, ACM Press, 2001, pp. 115–123.

[16] Paul-Louis George and Houman Borouchaki, *Delaunay Triangulation and Meshing. Application to Finite Elements*, HERMES, 1998.

[17] Gerhard Globisch, *On an automatically parallel generation technique for tetrahedral meshes*, Parallel Computing, 21 (1995), pp. 1979–1995.

[18] ———, *Parmesh – a parallel mesh generator*, Parallel Computing, 21 (1995), pp. 509–524.

[19] Y. Ito, A. M. Shih, A. K. Erukala, B. K. Soni, A. N. Chernikov, N. P. Chrisochoides, and K. Nakahashi, *Generation of unstructured meshes in parallel using an advancing front method*, in 9th International Conference on Numerical Grid Generation in Computational Field Simulations, 2005.

[20] D. A. Jefferson, *Virtual time*, in ACM Transactions on Programming Languages and Systems, vol. 7, July 1985, pp. 404–425.

[21] Clemens Kadow, *Adaptive dynamic projection-based partitioning for parallel Delaunay mesh generation algorithms*, in SIAM Workshop on Combinatorial Scientific Computing, Feb. 2004.

[22] 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.

[23] G.E. Karniadakis and S.A. Orszag, *Nodes, modes, and flow codes*, Physics Today, 46 (1993), pp. 34–42.

[24] Andriy Kot, Andrey N. Chernikov, and Nikos P. Chrisochoides, *Out-of-core parallel Delaunay mesh generation*, in 17th IMACS World Congress Scientific Computation, Applied Mathematics and Simulation, 2005. Paper T1-R-00-0710.

[25] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[26] Charles L. Lawson, *Software for $C^1$ surface interpolation*, Mathematical Software, III (1977), pp. 161–194.

[27] Leonidas Linardakis and Nikos Chrisochoides, *Parallel domain decoupling Delaunay method*, SIAM Journal on Scientific Computing, (2004). In print.

[28] Reinald Löhner and Juan R. Cebral, *Parallel advancing front grid generation*, in Proceedings of the Eighth International Meshing Roundtable, 1999, pp. 67–74.

[29] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, and Noel Walkington, *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, May 1995, pp. 683–692.

[30] Démian Nave, Nikos Chrisochoides, and L. Paul Chew, *Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains*, Computational Geometry: Theory and Applications, 28 (2004), pp. 191–215.

[31] Jim Ruppert, *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms, 18(3) (1995), pp. 548–585.

[32] Jonathan Richard Shewchuk, *Lecture notes on Delaunay mesh generation.* http://www.cs.berkeley.edu/∼jrs/meshpapers/delnotes.ps.gz, accessed November 2005.

[33] ———, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, in Applied Computational Geometry: Towards Geometric Engineering, Ming C. Lin and Dinesh Manocha, eds., vol. 1148 of Lecture Notes in Computer Science, Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry.

[34] ———, *Delaunay Refinement Mesh Generation*, PhD thesis, Carnegie Mellon University, 1997.

[35] ———, *Delaunay refinement algorithms for triangular mesh generation*, Computational Geometry: Theory and Applications, 22 (2002), pp. 21–74.

[36] Daniel A. Spielman, Shang-Hua Teng, and Alper Üngör, *Parallel Delaunay refinement: Algorithms and analyses*, in Proceedings of the Eleventh International Meshing Roundtable, 2001, pp. 205–217.

[37] ———, *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, 2004, pp. 267–268.

[38] Shang-Hua Teng. Personal Communication. February 2004.

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

[40] Roger Webster, *Convexity*, Oxford Science Publications, 1994. Page 200.