Practical and Efficient Point Insertion Scheduling Method for Parallel Guaranteed Quality Delaunay Refinement*

Andrey N. Chernikov
Computer Science Department
College of William and Mary
Williamsburg, VA 23185
ancher@cs.wm.edu

Nikos P. Chrisochoides
Computer Science Department
College of William and Mary
Williamsburg, VA 23185
nikos@cs.wm.edu

ABSTRACT

We describe a parallel scheduler, for guaranteed quality parallel mesh generation and refinement methods. We prove a sufficient condition for the new points to be independent, which permits the concurrent insertion of more than two points without destroying the conformity and Delaunay properties of the mesh. The scheduling technique we present is much more efficient than existing coloring methods and thus it is suitable for practical use. The condition for concurrent point insertion is based on the comparison of the distance between the candidate points against the upper bound on triangle circumradius in the mesh. Our experimental data show that the scheduler introduces a small overhead (in the order of 1–2% of the total execution time), it requires local and structured communication compared to irregular, variable and unpredictable communication of the other existing practical parallel guaranteed quality mesh generation and refinement method. Finally, on a cluster of more than 100 workstations using a simple (block) decomposition our data show that we can generate about 900 million elements in less than 300 seconds.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques—*Concurrent Programming*; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*; I.3.5 [**Computing Methodologies**]: Computer Graphics—*Computational Geometry and Object Modeling*

General Terms

Algorithms, Performance, Design, Theory

*This work was supported by NSF grants: CCR-0049086, ACI-0085969, EIA-9972853, EIA-0203974, and ACI-0312980

ICS'04, June 26–July 1, 2004, Saint-Malo, France.

Keywords

Mesh generation, Delaunay triangulation, Parallel Scientific Computing, COTS Software

1. INTRODUCTION

The prevention of catastrophic failures in health-care, transportation and defense depends on our ability to understand complex and dynamic phenomena like crack propagation in fracture mechanics [9]. Parallel mesh generation is a crucial building block for simulating such phenomena on parallel computing platforms.

However, it takes about ten years to develop the basic theoretical and software infrastructure for *sequential industrial strength mesh generation libraries.* Moreover, improvements in terms of quality, speed, and functionality are open ended and permanent. Thus, in order to be effective, the implementation of parallel mesh generation software should leverage the ever evolving and fully functional sequential mesh generation libraries.

In this paper, we present a new provable scheduling algorithm for parallel point insertion technique using off-the-shelf sequential mesh generation software [30]. Specifically, we focus on parallel circumcenter point insertion for parallel guaranteed quality Delaunay mesh generation [31, 10].

Parallel mesh generation procedures decompose the original mesh generation problem into N smaller subproblems which are meshed concurrently using P ($\ll N$) processors. The subproblems can be formulated to be either tightly or partially coupled or even decoupled. The coupling of the subproblems determines the intensity of the communication and the degree of dependency (or synchronization) between the subproblems. The scheduling method we present here is partially coupled (i.e., loosely synchronous with bulk communication) and balances trade-offs between domain decomposition and communication.

The new point insertion scheduling method focuses on three important challenges we need to address in order to use commercial off-the-shelf (COTS) software for parallel guaranteed quality mesh generation:

- *stability* of the parallel mesher (i.e., retain the same high quality of elements that could be generated by the state-of-the-art sequential meshing codes for the same geometry);
- *100% code re-use* (i.e., leverage the continuously evolving and fully functional off-the-shelf sequential meshers);

- *simple decompositions* of the original domain (i.e., avoid reducing the parallel mesh generation problem into another equally hard problem [23]).

The only practical parallel guaranteed quality mesh generation method until now was presented in [26], for polyhedral domains. The algorithm in [26] maintains the stability of the mesher by simultaneously partitioning and refining the interface surfaces and volume of the subdomains [13]—a refinement due to a point insertion might extend across subproblem (or subdomain) boundaries (or interfaces). The extension of a cavity beyond subdomain interfaces is a source of irregular and intensive communication with variable and unpredictable patterns. Although the method in [26] can tolerate up to 90% of the communication—by concurrently refining other regions of the subdomain while it waits for remote data to arrive—its scalability is of the order of $\mathcal{O}(\log P)$, P is the number of processors. Unfortunately, the concurrent refinement can lead to a non-conforming mesh and/or a non-Delaunay mesh [26]. These non-conformities are resolved at the cost of setbacks which require algorithm/code re-structuring [12].

Throughout this paper we call the method in [26] Optimistic Delaunay (OD) method because it is based on speculative (optimistic) execution model [20] for tolerating high communication latencies. The OD method satisfies the stability and simplicity of domain decomposition, but it does not address the code re-use requirement. The lack of code re-use not only makes this approach labor intensive and very hard to maintain, but it affects its performance (i.e., scalability) due to low single processor performance because of its inability to leverage the highly optimized sequential codes like Triangle [30] for 2D geometries and TetMesh-GHS3D [2] and Gridgen [1] for 3D domains. In this paper, we extend the work in [26, 12, 13] in two ways. First, we develop a theory which guarantees mathematically that at each step of the mesh refinement the points we choose to insert concurrently will be independent, i.e., after they are inserted, the mesh will still be conformal and Delaunay. This theory leads to an efficient and practical parallel Delaunay refinement scheduling algorithm. Second, the use of our theory allows to eliminate the restructuring of serial mesh generation codes and, hence, permits to employ the COTS-based approach to parallel Delaunay mesh generation using existing sequential Delaunay [31] and modified Delaunay [1, 2] meshers.

In order to address the code re-use requirement and eliminate the communication and synchronization costs, in [23] the authors present a Parallel Delaunay Domain Decoupling (PD^3) method. The PD^3 is based on the idea of decoupling the individual subdomains (subproblems) so that they can be meshed independently with zero communication/synchronization and by re-using existing sequential mesh generation codes. However, the construction of "proper" decompositions that can decouple the mesh is an equally challenging problem, since its solution is based on Medial Axis [7, 19, 34, 3] which is very expensive and difficult to construct (even to approximate) for complex 3-dimensional geometries. Other, similar to PD^3, attempts to parallelize Delaunay triangulations [6] and implement parallel Delaunay mesh generation [17] were presented in the past. However, the method in [6] can not be used for parallel meshing; recent attempts to extend it appeared in [21], but with zero code re-use. On the other hand, the method in [17] can not always guarantee the decoupling of the subproblems, in these cases it uses re-partitioning and re-meshing.

All the methods above maintain the stability at some high cost which prevents us from having industrial strength parallel mesh generation codes today. Is it possible to have a practical and working parallel 3D mesh generation codes (today) if we relax the stability requirement? To answer this question we developed a 3D parallel advancing front technique (PAFT) method [4] similar to one presented in [28] which might generate "good" quality elements. We observed that even if we relax the stability criterion the problem of parallel mesh generation does not become easier. Our experience with the PAFT even for simple 3D geometries suggests that generic domain decompositions obtained from state-of-the-art partitioning libraries can not always be meshed by state-of-the-art sequential AFT meshers like Jmesh [27][1]. On the other hand, simple block domain decompositions can create, for the subdomains, artificial and arbitrarily small features between the external boundary of the domain and the interfaces of the subdomains that deteriorate the quality of the elements. The scheduler we present here does not have this problem, because the block domain decomposition is used only for data distribution rather than for a geometric domain decomposition.

Independently of the above efforts, Löhner and Cebral [24] developed a parallel advancing front scheme, which is stable, has some code re-use, and addresses the domain decomposition problem. They implement a "shift and regrid" technique, where pre-computed shift distances (they depend on the sequential meshing method[2]) are used. These distances work well in the case of advancing front meshing, where there is a clear distinction between triangulated and empty areas. However, Delaunay refinement, in addition to maintaining a mesh which at all times covers the entire domain, also requires that all triangle circumcenters be empty of mesh points.

Also, De Cougny, Shephard, and Ozturan [14] use an underlying octree to aid in parallel mesh generation. After the generation of the octree and template meshing of the interior octants, their algorithm connects a given surface triangulation to the interior octants using face removal. The face removal procedure eliminates problems due to the small distance between the interior quadrants and boundary faces, by defining "an entity too close to the boundary triangulation" [14] and "using the distance of about one-half the octant edge length as the minimum works well" [14]. We explore a somewhat similar approach in the context of Delaunay refinement and derive precise distances that are necessary between the interiors and boundaries of concurrently refined subdomains.

The difference between the proposed scheduling method and the last two methods [14, 24] is that it does not rely on re-partitioning. The re-partitioning has to take place in a "guided" way which is not yet completely understood. The key idea of a "guided" re-partitioning method is to gen-

[1]This is because the decomposition libraries [29] are designed to solve the domain decomposition (DD) problem for field solvers [11]. The DD problem from parallel mesh generation has different requirements [23].

[2]In the case of an advance front method based on octree they use $\min(0.5s_{min}, 2.0d_{min})$, where s_{min} is the minimum box size in which elements are to be generated, and d_{min} is the minimum element size in the active front.

erate new subdomains whose interfaces are in the interior of the subdomains of the previous partition and thus the old interfaces are placed in the interior of the new subdomains and therefore the regions around them can be meshed without further communication. Of course, the "guided" re-partitioning method itself requires communication. Instead, the proposed method uses a simple block decomposition once, at the beginning of the parallel mesh generation. Like the re-partitioning methods in [14, 24], we have to perform some communication between refinement phases, but this communication happens before the re-meshing of a block of data (subblock), thus off-the-shelf mesh generation codes can be used unchanged. Unlike the De Cougny et. al. and Löhner et. al. methods, we do not rely on equidistribution for load balancing, but on overdecomposition and a parallel runtime system we developed for dynamic load balancing [4, 5].

Finally, Edelsbrunner and Guoy [16] studied the possibility of parallel insertion of independent points. They define the prestar of point x as the difference between the closure of the set of tetrahedrons whose circumspheres enclose x and the closure of the set of remaining tetrahedra. They define the points x and y as independent if the closures of their prestars are disjoint. We start with proving a similar condition of point independence. The difference between the independence condition in [16] 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, 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 we prove a lemma, which allows to use only the distance between the points to check whether they are independent. The minimum separation distance argument in [16] 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 distributed and shared memory implementations. In [16], a shared memory algorithm based on finding the maximal independent set of points is used.

2. PARALLEL DELAUNAY REFINEMENT

2.1 Notation

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*[3] *triangulation* [18] 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 Ω.
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.

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

We will denote point number i as p_i and the triangle with vertices p_i, p_j, and p_k as $\triangle p_i p_j p_k$. We will use letters a, b, and c to refer to triangle's side lengths ($a \le b \le c$) and letters A, B, and C to represent triangle's angle measures ($A \le B \le C$). Let $\bigcirc(\triangle p_i p_j p_k)$ represent the circumcircle of $\triangle p_i p_j p_k$, $\odot(\triangle p_i p_j p_k)$ — its circumcenter, and $r(\triangle p_i p_j p_k)$ — its circumradius. Also, let $\bar{r}$, $\bar{\rho}$, and $\bar{\Delta}$ be upper bounds on circumradius, circumradius-to-shortest edge ratio, and area of triangles, respectively.

2.2 Sequential Delaunay Refinement

The applications that use Delaunay meshes usually impose two constraints on the quality of 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 [25] and Shewchuk [31], in two dimensions the circumradius-to-shortest edge ratio ρ of a triangle can be expressed in terms of its minimal angle A as

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

As the circumradius-to-shortest edge ratio is the metric that is naturally optimized by the Delaunay refinement [25], we will consider $\bar{\Delta}$ and $\bar{\rho}$ 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. For example (Table 1), if we run Jonathan Shewchuk's Triangle [30] on the contour of the letter "A", which is supplied with the Triangle package, it takes 0.643 sec. to create a mesh of 130 405 triangles respecting the 20° minimal angle bound (corresponding to $\bar{\rho} \approx 1.41$) and $\bar{\Delta} = 10^{-6}$. However, for the same model and angle bound, it takes 24.641 sec. to create 5 232 435 triangles if we set $\bar{\Delta} = 2.5 \cdot 10^{-8}$. Although asymptotically $\mathcal{O}(n \log n)$ time is required to triangulate n points ("Delaunay" phase), while refining an existing mesh ("Quality" phase) is on average linear in the number of triangles produced, for fairly large final meshes the latter time significantly dominates.

In this paper, we focus on parallelizing the Delaunay refinement ("Quality") stage, which is usually the most memory- and computation-expensive. The general idea of the Delaunay refinement is to insert points in the circumcenters of 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 [22], which uses edge flips, and the Bowyer/Watson algorithm [8, 33], 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 [31], thus, we will use only the Bowyer/Watson algorithm for our analysis. The following definition, adapted from [18], plays a major role in studying the Bowyer/Watson algorithm:

Definition 2. Let the *cavity* $\mathcal{C}_{\mathcal{M}}(p_i)$ *of point* p_i *with respect to mesh* $\mathcal{M}$ be the set of triangles in $\mathcal{M}$, whose circumcircles include p_i:

$$\mathcal{C}_{\mathcal{M}}(p_i) = \{\triangle p_k p_l p_m \in \mathcal{M} \mid p_i \in \bigcirc(\triangle p_k p_l p_m)\}.$$

Table 1: Running Jonathan Shewchuk's Triangle [30] sequentially on the portrayal of the letter "A", which is supplied with the Triangle package. $\bar{\Delta}$ is the upper bound on triangle area, which already holds for the given input (output) data. In all experiments, the bound on the circumradius-to-shortest edge ratio corresponds to the default Triangle bound of 20° on the smallest angle.

Input				Output				Time, sec.		
Triangles	Points	Segments	$\bar{\Delta}$	Triangles	Points	Segments	$\bar{\Delta}$	Delaunay	Reconstr	Quality
0	29	29	n/a	130 405	66 647	2 889	$1 \cdot 10^{-6}$	0.000	n/a	0.643
0	29	29	n/a	5 232 435	2 625 249	18 063	$2.5 \cdot 10^{-8}$	0.000	n/a	24.641
0	29	29	n/a	1 307 048	658 062	9 076	$1 \cdot 10^{-7}$	0.000	n/a	6.253
1 307 048	658 062	9 076	$1 \cdot 10^{-7}$	5 195 468	2 606 427	17 386	$2.5 \cdot 10^{-8}$	n/a	42.367	19.988
0	658 062	9 076	$1 \cdot 10^{-7}$	5 194 174	2 605 776	17 378	$2.5 \cdot 10^{-8}$	7.038	n/a	19.395

We will write simply $\mathcal{C}(p_i)$ when $\mathcal{M}$ is clear from the context. Also, we will denote $\mathcal{B}_{\mathcal{M}}(p_i)$ to be the set of edges which belong to only one triangle in $\mathcal{C}_{\mathcal{M}}(p_i)$, i.e., external edges.

In the absence of external boundaries, the sequential Delaunay refinement algorithm, based on the Bowyer/Watson point insertion procedure [8, 33], maintains a Delaunay mesh $\mathcal{M}$, and at every iteration performs the following steps:

1. Select a triangle from the queue of unsatisfactory triangles.
2. Compute the circumcenter p_i of this triangle.
3. Find $\mathcal{C}_{\mathcal{M}}(p_i)$ and $\mathcal{B}_{\mathcal{M}}(p_i)$.
4. Delete all triangles in $\mathcal{C}_{\mathcal{M}}(p_i)$ from $\mathcal{M}$.
5. Add triangles obtained by connecting p_i with every edge in $\mathcal{B}_{\mathcal{M}}(p_i)$ to $\mathcal{M}$.

This procedure maintains the following invariant:

Loop Invariant 1 (Delaunay Triangulation). *The condition that $\mathcal{M}$ is (i) conformal and (ii) Delaunay holds both before and after the insertion of a point.*

2.3 Theoretical Framework

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

2.3.1 Maintaining the Delaunay Triangulation Loop Invariant

We expect our algorithm to produce a conformal mesh, i.e., it should not generate triangles whose edges intersect in points other than the vertices of the mesh, or allow untriangulated polygonal regions to emerge inside the domain. 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 1.

However, even if the cavities of two points do not have common triangles, the Delaunay triangulation loop invariant can still be violated due to the creation of non-Delaunay triangles, i.e., triangles whose circumcircles are not empty of mesh vertices. This can happen if two cavities have common edges (see Figure 2). The following Lemma provides a sufficient condition for the simultaneous retriangulation of two cavities to yield a conformal Delaunay mesh.

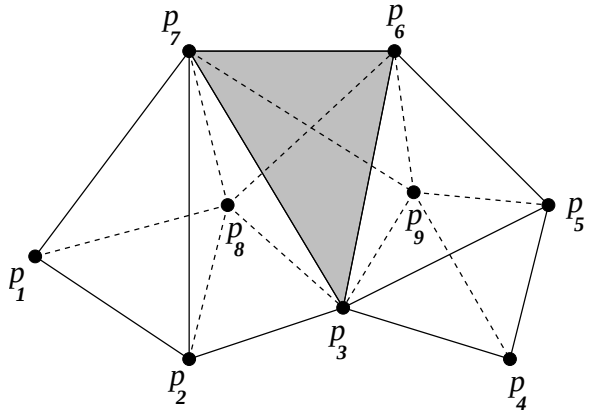

Figure 1: If $\triangle p_3 p_6 p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$, then concurrent insertion of p_8 and p_9 results in a non-conformal mesh. Solid lines represent the edges of the initial triangulation, and dashed lines — edges created by the insertion of p_8 and p_9.

Lemma 1. *If $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ have no common triangles and do not share any triangle edges, then 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 [15, 18], which states that if the empty sphere 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 [31]. Hence, the mesh will be globally Delaunay. □

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

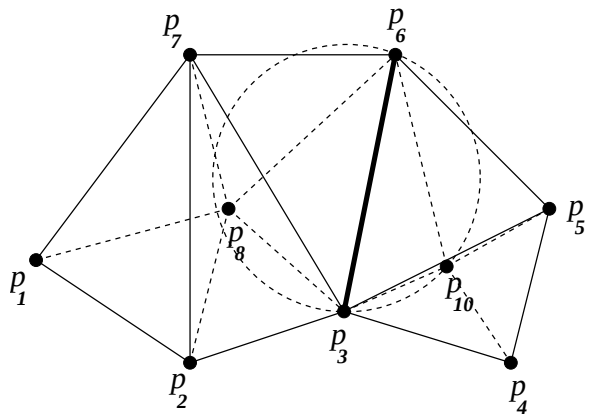

Figure 2: If edge p_3p_6 is shared by $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ and $\mathcal{C}(p_{10}) = \{\triangle p_3p_5p_6, \triangle p_3p_4p_5\}$, then the new triangle $\triangle p_3p_{10}p_6$ can have point p_8 inside its circumcircle, thus, violating the Delaunay property.

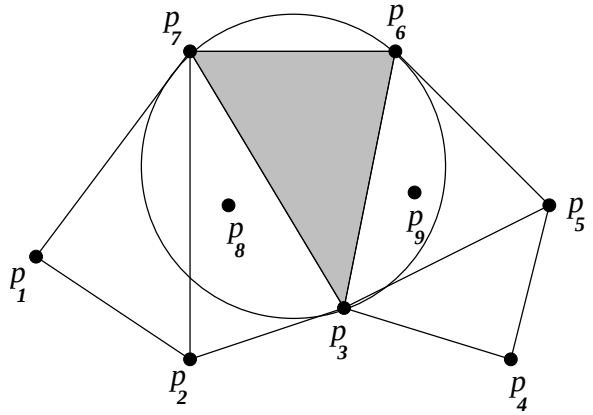

Figure 3: If $\triangle p_3p_6p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$, then $\|p_8 - p_9\| < 2r(\triangle p_3p_6p_7) < 2\bar{r}$.

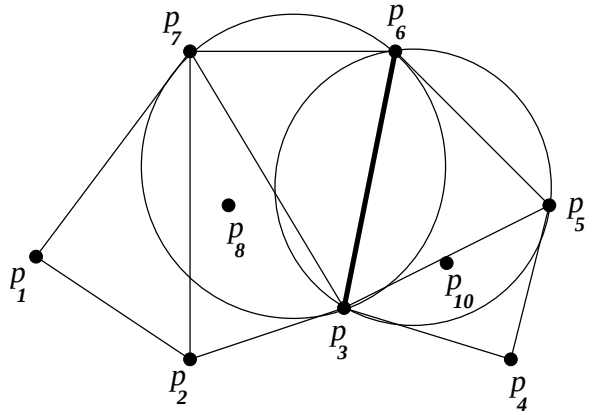

Figure 4: If edge p_3p_6 is shared by $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ and $\mathcal{C}(p_{10}) = \{\triangle p_3p_5p_6, \triangle p_3p_4p_5\}$, then $\|p_8 - p_{10}\| < 2r(\triangle p_3p_6p_7) + 2r(\triangle p_3p_5p_6) < 4\bar{r}$.

LEMMA 2. *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 $\mathcal{C}(p_i)$ and $\mathcal{C}(p_j)$ do not intersect and have no shared edges.*

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 $\triangle p_kp_lp_m \in \mathcal{C}(p_i) \cap \mathcal{C}(p_j)$ (see Figure 3). By the definition of cavity, $p_i \in \bigcirc(\triangle p_kp_lp_m)$ and $p_j \in \bigcirc(\triangle p_kp_lp_m)$, 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 4). 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.

□

Remark 1. The bound $4\bar{r}$ can be somewhat decreased by further analysis based on the way $\bigcirc(\triangle p_kp_lp_m)$ and $\bigcirc(\triangle p_kp_lp_n)$ overlap in case (ii). However, the practical value of the additional analysis is questionable.

Combined with Lemma 1, Lemma 2 allows to check in constant time whether two points are independent, and, hence, can be inserted concurrently.

2.3.2 The Circumradius Upper Bound Invariant

For Lemma 2 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 Boyer/Watson algorithm, either sequentially or in parallel, does not violate Loop Invariant 2.

Let the *reflection of circle $\bigcirc(\triangle p_kp_lp_m)$ about edge p_kp_l* be the circle $\bigcirc'_{p_kp_l}(\triangle p_kp_lp_m)$ that has the same radius, whose circumference passes through 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 5.

LEMMA 3. *For any point p_i inside the region $\bigcirc(\triangle p_kp_lp_m) \setminus \bigcirc'_{p_kp_l}(\triangle p_kp_lp_m)$, $r(\triangle p_kp_lp_i) < r(\triangle p_kp_lp_m)$.*

PROOF. Consider Figure 5. Let o'' be the center of circle $\bigcirc(\triangle p_kp_lp_i)$, where p_i is any point inside the shaded region $\bigcirc(\triangle p_kp_lp_m) \setminus \bigcirc'_{p_kp_l}(\triangle p_kp_lp_m)$. o'' has to lie on the straight line through o and o'. Let v be the point of intersection of this straight line with edge p_kp_l (i.e. the midpoint of p_kp_l), and u — with the circumference of $\bigcirc(\triangle p_kp_lp_i)$ inside the shaded region. Let $x = \|u - v\|$. We will express the radius of $\bigcirc(\triangle p_kp_lp_i)$ as the function of x: $r(\triangle p_kp_lp_i) = f(x)$.

If the length of the edge p_kp_l is a, then by considering the right triangle $\triangle p_kvo''$ and noting that $\|p_k - v\| = a/2$, $\|v - o''\| = x - f(x)$, and $\|o'' - p_k\| = f(x)$, we have:

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

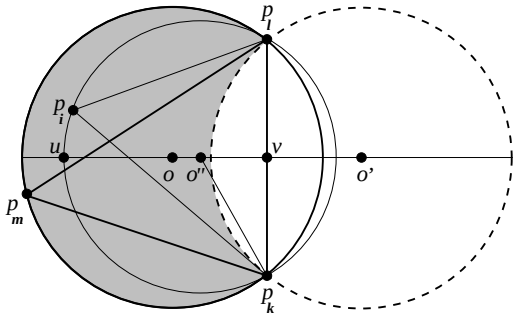

Figure 5: The solid circumference represents $\bigcirc(\triangle p_k p_l p_m)$ with center in point o, and the dashed circumference — $\bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$ with center in point o'. Point o'' is the center of the variable-radius circle, whose circumference passes through p_k and p_l. We prove that for any point p_i inside the shaded region, $r(\triangle p_k p_l p_i) < r(\triangle p_k p_l p_m)$.

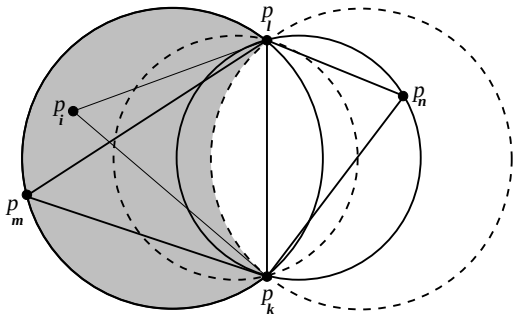

Figure 6: $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$, $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, and $r(\triangle p_k p_l p_m) > r(\triangle p_k p_n p_l)$.

where $0 < x < \infty$.

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

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

Because $f(x)$ is continuous and convex on $0 < x < \infty$ and it is equal to $r(\triangle p_k p_l p_m)$ when point u lies on the circumference of $\bigcirc(\triangle p_k p_l p_m)$ and on the circumference of $\bigcirc'_{p_k p_l}(\triangle p_k p_l p_m)$, it has values less then $r(\triangle p_k p_l p_m)$ if u lies anywhere between these two circumferences. □

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 $r(\triangle p_k p_l p_i) < \max(r(\triangle p_k p_l p_m), r(\triangle p_k p_n p_l))$.*

PROOF. There are two cases:

(i) $r(\triangle p_k p_l p_m) > r(\triangle p_k p_n p_l)$. See Figure 6. 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 in this case 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,

$$r(\triangle p_k p_l p_i) < r(\triangle p_k p_l p_m).$$

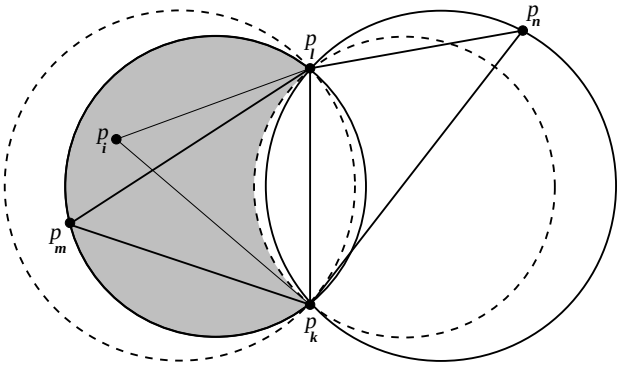

Figure 7: $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$, $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, and $r(\triangle p_k p_l p_m) \leq r(\triangle p_k p_n p_l)$.

(ii) $r(\triangle p_k p_l p_m) \leq r(\triangle p_k p_n p_l)$. See Figure 7. 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,

$$r(\triangle p_k p_l p_i) < r(\triangle p_k p_n p_l).$$

□

Since the Bowyer/Watson algorithm creates only triangles of the form $\triangle p_k p_l p_i$, where there exist points p_m and p_n such that $\triangle p_k p_l p_m \in \mathcal{C}(p_i)$ and $\triangle p_k p_n p_l \notin \mathcal{C}(p_i)$, by Lemma 4, every new triangle in the mesh will have circumradius which is less than the circumradius of some existing triangle, and, as a result, the maximal circumradius will not increase. Hence, once the upper bound $\bar{r}$ on triangle circumradius holds, it will be maintained throughout the execution of the Bowyer/Watson algorithm.

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

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

where $\bar{r}$ is computed based on the number of available processors and the total area of the domain (see subsection 2.3.4).

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 maximal area. In the rest of this subsection, we prove equation (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 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*

$$a^2 = 2(\cot B + \cot C)\Delta,$$

where Δ is the area of the triangle.

PROOF. Directly follows from the well known formulas

$$\cot A + \cot B + \cot C = \frac{a^2+b^2+c^2}{4\Delta}, \quad \cot A = \frac{b^2+c^2-a^2}{4\Delta}.$$

□

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

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

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

PROOF. Since A, B, C are positive and sum up to π, we can denote

$$\begin{aligned} f(B) &= \cot B + \cot C \\ &= \cot B + \cot(\pi - A - B) \\ &= \cot B - \cot(A + B), \end{aligned}$$

where

$$0 < A \le B \le \frac{\pi - A}{2} \le C \le \pi - 2A, \text{ and } A \le \frac{\pi}{3}. \qquad (3)$$

$f(B)$ has no points of local maxima which satisfy (3), thus, it can reach its maxima only in the ends of the interval $\left[A, \frac{\pi-A}{2}\right]$. Direct substitution yields

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

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,

$$f(B) \le \frac{1}{\sin A}.$$

□

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

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

$$r \le \rho\sqrt{2\frac{1}{\sin A}\Delta} = \rho\sqrt{\frac{2\Delta}{\sin A}}. \qquad (4)$$

By substituting $\sin A$ from (1) into (4), we have:

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

□

In other words, by preprocessing the mesh to the degree that the bounds $\bar{\rho}$ and $\bar{\Delta}$ hold, the bound

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

is forced to hold.

2.3.4 Coarse Grain Partitioning

One possible way to break the meshing problem of the entire domain into smaller independent subproblems is to partition the domain into subdomains in such a way that the circumcenters of triangles in each subdomain can be inserted concurrently with the circumcenters in any other subdomain. The partitioning and decoupling of 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.

As we have shown with Lemmas 1 and 2, if the distance between two circumcenters is greater or equal to $4\bar{r}$, these circumcenters can be inserted independently. It also obviously follows from the proof of Lemma 2 (Figure 3) that $\mathcal{C}(p_i)$ can only include triangles whose circumcenters are less than $2\bar{r}$ distance away from p_i. This fact naturally leads us to the following definition:

Definition 3. Let the *buffer zone $\mathcal{Z}_{\mathcal{M}}(p_i)$ of point p_i with respect to mesh $\mathcal{M}$* be the set of triangles in $\mathcal{M}$ whose circumcenters are within $2\bar{r}$ distance from p_i:

$$\mathcal{Z}_{\mathcal{M}}(p_i) = \{\triangle p_k p_l p_m \in \mathcal{M} \mid \|p_i - \odot(\triangle p_k p_l p_m))\| < 2\bar{r}\}.$$

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. Hence, if we impose a square grid $\mathcal{D} = \{d_{ij}\}$ with cell side equal to $2\bar{r}$ over the entire domain, and for each cell, d_{ij}, find the set of triangles with circumcenters inside this cell, then all cells pairwise separated by at least two cells in $\mathcal{D}$ can be refined concurrently. Upon the completion of this phase the set of cells that served as buffer cells, are selected for parallel refinement. The process continues until there are no "bad" triangles left. It takes four refinement phases to guarantee that there are no "bad" triangles left (see Figure 8).

3. PARALLEL IMPLEMENTATION

For the experimental evaluation of our method, we triangulated 1000 uniformly distributed random points selected inside a unit square to obtain some initial mesh, and refined it in parallel using Triangle [30] as a sequential mesher on each processor. 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 runtime 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 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 triangulation and to construct a fine triangulation from scratch. Contrary to our expectations, building a fine triangulation anew turned out to be much faster. For example (Table 1), in the case of letter "A", 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, provided that there are no four cocircular points, is unique [31], 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 side effect of this optimization is the decrease in the size of network traffic.

Although our theory allows to assign as few as nine cells of side length $2\bar{r}$ per processor, it turned out to be more efficient to increase the level of preprocessing with the purpose of allowing some overlap of the regions under refinement with the regions that have already been refined. If such overlap is not provided, the refinement in one region can create badly shaped triangles in its already refined neighboring regions, and vise-versa. Due to the high cost of data migration and synchronization on a distributed memory system, an iterative refinement process turned out to be prohibitively expensive. Thus, we adopted the refinement and communication schedule schematically depicted in Figure 8. We set the overlap between the refinement regions equal to $2\bar{r}$, which is the maximum distance to which the effects of a point insertion can propagate, and the refinement process is now guaranteed to improve all "big" and "badly" shaped triangles in four phases, alternated with communication. All processors are arranged in a logical two-dimensional grid, and each processor in position (i, j) communicates only with its neighbors in positions (s, t), such that $|i - s| \leq 1$ and $|j - t| \leq 1$.

In this paper, we do not address the question of dealing with potential processor work load imbalance, since it has been discussed elsewhere [4].

Finally, since we so far have not proved analogous properties of Delaunay refinement nearby the constrained edges, we have extended the initial triangulation of the unit square with an auxiliary layer of triangles into $4\bar{r}$ distance outward.

We have conducted our experiments on the Sciclone cluster computing system[4] 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 results are summarized in Table 2.

The stability of our method follows from the fact that all cells eventually get refined with Triangle in such a way that there are no zones with potentially bad elements left. Experimental results also show that the method exhibits very good scalability: if the number of processors is increased by more than 30 times, the total running time increases by less than 3%. Finally, we didn't have to write any mesh generation code, since it was sufficient to call Triangle as a library.

[4]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.

4. CONCLUSIONS

We introduced a point insertion scheduling method which with a simple mesh decomposition scheme is used for parallel guaranteed quality Delaunay mesh generation on both distributed and shared memory systems. We proved that the scheduling method leads to stable and scalable guaranteed quality parallel mesh generation codes on a cluster of workstations. Moreover, we demonstrated that the scheduler provides the ability to use fully functional and optimized sequential meshers on each processor, i.e., we proved that a COTS software alternative for parallel mesh generation is possible with simple domain decompositions at the cost of some communication.

Our future plans are to implement (using OpenMP) the same scheduler on shared memory machines, for broader use in game and health care industries. Also we plan to develop a parallel and out-of-core mesh generation codes by utilizing the absence of communication during subdomain refinement phases. Finally, we are extending the method to handle external and internal constrained segments/faces for both two- and three-dimensional domains.

5. REFERENCES

[1] Gridgen. http://www.pointwise.com/gridgen/index.shtml. Accessed on Apr. 21, 2004.

[2] TetMesh-GHS3D V3.1 The fast, reliable, high quality tetrahedral mesh generator and optimiser. White paper. http://www.simulog.fr/mesh/tetmesh3p1d-wp.pdf. Accessed on Feb. 27, 2004.

[3] C. Armstrong, D. Robinson, R. McKeag, T. Li, S. Bridgett, R. Donaghy, and C. McGleenan. Medials for meshing and more. In *Proceedings of 4th International Meshing Roundtable*, pages 277–288. Sandia National Laboratories, 1995.

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

[5] K. Barker and N. Chrisochoides. An evalaution of a framework for the dynamic load balancing of highly adaptive and irregular applications. In *Supercomputing Conference*. ACM, Nov. 2003.

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

[7] H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of speech and Visual Form*, pages 362–380. MIT Press, 1967.

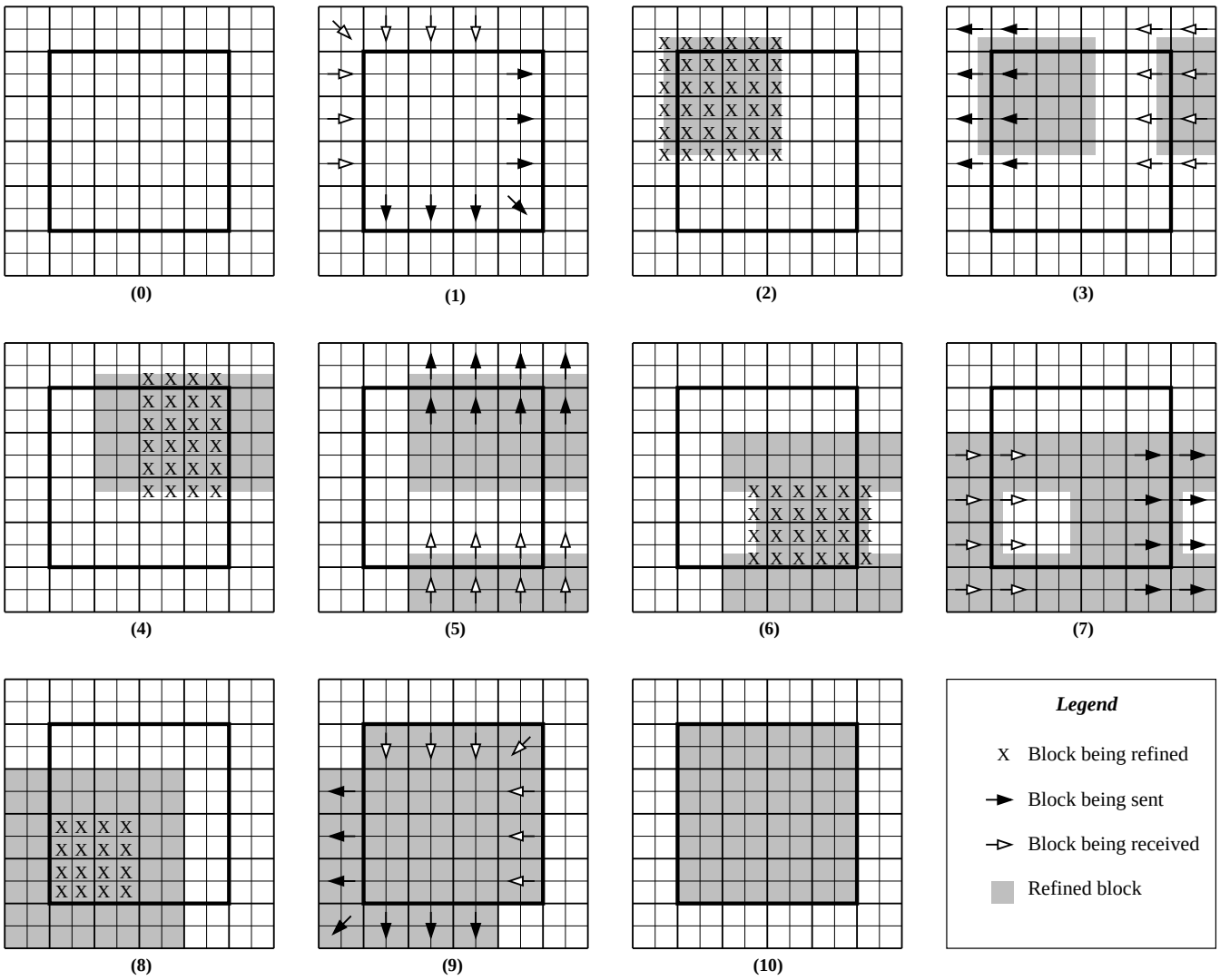

Figure 8: A schedule of the parallel Delaunay refinement and communication on distributed memory. Each phase (0)–(10) depicts the actions performed by a single processor. The smallest cells have side length $2\bar{r}$, they are the atomic units of refinement. Since the refinement may leave fractions 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.

Table 2: Scaled Speedup Evaluation

Number of Processors	Total Time, sec.	Number of Elements, Millions	Some Timings for a Single Processor				Area Bound $\times 10^{-8}$
			Prerefinement with Triangle	Total Refinement Time with Triangle	Communication and Synchronization	Filling in and Merging Cells	
4	293.7	23.8	0.24	218.7	32.7	39.9	5.00
9	294.7	58.8	0.44	220.0	35.4	40.3	2.22
16	295.4	109.3	0.71	218.8	37.1	40.0	1.25
25	296.8	175.4	1.05	219.1	39.1	40.8	0.80
36	293.4	255.0	1.43	218.6	36.1	40.0	0.56
49	294.5	352.6	1.88	217.7	37.5	40.7	0.41
64	300.1	470.7	2.49	221.0	40.7	41.5	0.31
81	296.2	587.8	3.04	216.6	46.2	41.1	0.25
100	300.3	738.9	3.69	218.4	46.5	41.5	0.20
121	293.7	873.5	4.42	210.3	53.6	40.9	0.17

[8] A. Bowyer. Computing Dirichlet tesselations. *Computer Journal*, 24:162–166, 1981.

[9] B. Carter, C.-S. Chen, L. P. Chew, N. Chrisochoides, G. R. Gao, G. Heber, A. R. Ingraffea, R. Krause, C. Myers, D. Nave, K. Pingali, P. Stodghill, S. Vavasis, and P. A. Wawrzynek. Parallel FEM simulation of crack propagation—challenges, status, and perspectives. *Lecture Notes in Computer Science*, 1800:443–449, 2000.

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

[11] N. Chrisochoides, E. Houstis, and J. Rice. Mapping algorithms and software environment for data parallel PDE iterative solvers. *Journal of Parallel and Distributed Computing*, 21(1):75–95, 1994.

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

[13] N. P. Chrisochoides. A new approach to parallel mesh generation and partitioning problems. *Computational Science, Mathematics and Software*, pages 335–359, 2002.

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

[15] B. N. Delaunay. Sur la sphere vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Mataematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[16] H. Edelsbrunner and D. Guoy. Sink-insertion for mesh improvement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pages 115–123. ACM Press, 2001.

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

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

[19] H. N. Gürsoy and N. M. Patrikalakis. An automatic coarse and fine surface mesh generation scheme based on medial axis transform: Part I algorithms. *Engineering With Computers*, 8:121–137, 1992.

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

[21] C. Kadow and N. 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.

[22] C. L. Lawson. Software for C^1 surface interpolation. *Mathematical Software*, III:161–194, 1977.

[23] L. Linardakis and N. Chrisochoides. Parallel domain decoupling Delaunay method. *SIAM Journal on Scientific Computing*, Submitted Dec. 2003.

[24] R. Löhner and J. R. Cebral. Parallel advancing front grid generation. In *Proceedings of the Eighth International Meshing Roundtable*, pages 67–74, 1999.

[25] G. L. Miller, D. Talmor, S.-H. Teng, and N. 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*, pages 683–692. ACM Press, May 1995.

[26] D. Nave, N. Chrisochoides, and L. P. Chew. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 135–144, 2002.

[27] J. C. Neto, P. Wawrzynek, M. Carvalho, L. Martha, and A. Ingraffea. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. *Engineering with Computers*, 17:75–91, 2001.

[28] R. Said, N. Weatherill, K. Morgan, and N. Verhoeven. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering*, (177):109–125, 1999.

[29] K. Schloegel, G. Karypis, and V. Kumar. A unified algorithm for load-balancing adpative scientific simulations. Technical Report TR 00-033, Department of Computer Science and Engineering, University of Minnesota, http://www-users.cs.umn.edu/ karypis/publications/partitioning.html, May 2000.

[30] J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Proceedings of the First workshop on Applied Computational Geometry*, pages 123–133, Philadelphia, PA, 1996.

[31] J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, 1997.

[32] J. R. Shewchuk. Lecture notes on Delaunay mesh generation. 1999.

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

[34] F.-E. Wolter. Cut locus and medial axis in global shape interrogation and represenation. Technical report, MIT, Department of Ocean Engeneering, Design Laboratory, 1993.