



LAWRENCE  
LIVERMORE  
NATIONAL  
LABORATORY

# A New Parallel Algorithm for Constructing Voronoi Tessellations from Distributed Input Data

D. P. Starinshak, J. M. Owen, J. N. Johnson

November 27, 2013

Computer Physics Communications

## **Disclaimer**

---

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

# A New Parallel Algorithm for Constructing Voronoi Tessellations from Distributed Input Data

D. P. Starinshak <sup>\*a</sup>, J. M. Owen<sup>a</sup>, and J. N. Johnson<sup>b</sup>

<sup>a</sup>*Lawrence Livermore National Laboratory, AX Division, M/S L-38, P.O. Box 808, Livermore, CA 94550*

<sup>b</sup>*Lawrence Berkeley National Laboratory, Earth Sciences Division, 1 Cyclotron Road M/S 74R316C, Berkeley, CA 94720*

## Abstract

We present a unique parallel algorithm for generating consistent Voronoi diagrams from distributed input data for the purposes of simulation and visualization. The algorithm acts as a parallel interface to any serial Delaunay or Voronoi tessellation method by computing processor communication structures. The result is a generalized methodology for adding distributed capabilities to serial tessellation packages. Results from several two-dimensional tests are presented, including strong and weak scaling of its current implementation.

## 1 Introduction

This article outlines a new algorithm for computing Voronoi and Voronoi-like tessellations in a robust and efficient manner from both shared and distributed input data. The design principle of the algorithm is unique in that any arbitrary serial method for computing Voronoi or Delaunay tessellations may be employed. The algorithm outlines a procedure for generating communication structures around serial tessellation packages to transform distributed input points into Voronoi tessellation data that is consistent across processor boundaries.

The introduction is subdivided into a discussion of Voronoi tessellations and terminology (1.1), a description of the computation framework for constructing Voronoi tessellations in shared and distributed-memory environments (1.2), and a summary of key application areas for distributed Voronoi construction (1.3). Section 2 outlines the parallel algorithm, including background information (2.1), a detailed statement of the algorithm (2.2), and remarks on efficiency improvements (2.3). Finally, Section 3 demonstrates results from a variety of two-dimension unit tests, including strong and weak scaling of the new algorithm.

---

<sup>\*</sup>Corresponding author. Email address: starinshak1@llnl.gov

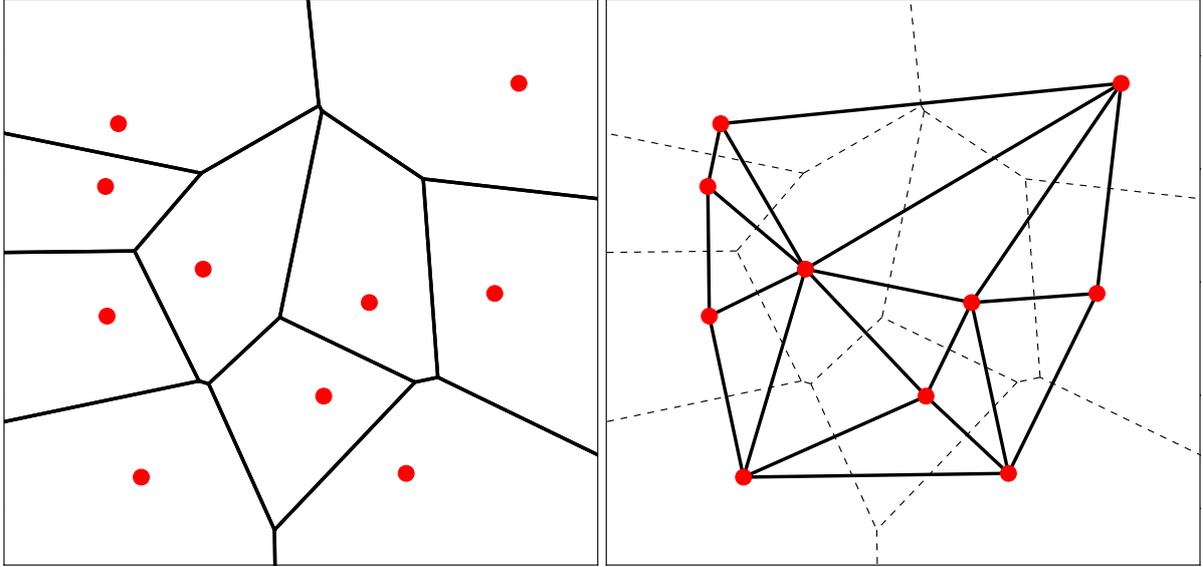


Figure 1: 2D Voronoi tessellation with generators (left) and related Delaunay dual tessellation.

## 1.1 Voronoi Tessellation

Let  $\Omega \in \mathbb{R}^N$  define a closed region of space. A *tessellation* of  $\Omega$  defines a set of disjoint *cells* whose union covers  $\Omega$ . Let  $d(\cdot, \cdot)$  be a distance norm on  $\mathbb{R}^N$ , and let  $\{\mathbf{g}_i\}_{i=1}^G$  be a collection of points inside  $\Omega$ . The *Voronoi tessellation* of  $\Omega$  is given by the set of *Voronoi cells*  $\{V_i\}_{i=1}^G$  with  $i$ -th cell corresponding to point  $\mathbf{g}_i$  and satisfying

$$V_i = \{ \mathbf{x} \in \Omega \mid d(\mathbf{x}, \mathbf{g}_i) < d(\mathbf{x}, \mathbf{g}_j) \quad \forall j \neq i \}. \quad (1)$$

The points  $\{\mathbf{g}_i\}_{i=1}^G$  are termed the *generators* of the Voronoi tessellation, and we denote their set  $\mathcal{G}$ .

The dual tessellation associated with the Voronoi is known as the *Delaunay tessellation*. Figure 1 illustrates an example Voronoi tessellation in two dimensions under Euclidean distance norm  $d$  as well as its associated Delaunay dual. Voronoi cells are given by arbitrary polygons, and Delaunay cells are triangles. (In 3D, cells are respectively polyhedra and tetrahedra.) The mathematical relationship between Voronoi and Delaunay tessellations is a rich and interesting subject but outside the focus of this article. For our purposes, we simply observe that any Delaunay construction algorithm may be used to generate a Voronoi tessellation by computing the functional dual.

In this article, we limit discussion to two and three dimensions and consider only the Euclidean distance norm. Results and explanatory visuals are limited to two dimensions. The terms *Voronoi tessellation*, *Voronoi diagram*, and *Voronoi grid* are used interchangeably throughout this article. The details of Voronoi construction are based largely on geometric considerations. To that end, discussions oftentimes refer to the geometric *elements* or data structures of a tessellation: namely, cells are composed of faces, edges, and nodes/vertices, and two neighboring cells in a tessellation share at least one of such elements.

*Remark:* Generalized Voronoi tessellations based on non-point generators may also be

formulated, where distances in (1) are computed with respect to generator surfaces  $\{\Gamma_i\}_{i=1}^G$  having dimension smaller than the global domain  $\Omega$  [19].

## 1.2 Problem Statement

We seek an algorithm for constructing Voronoi diagrams based on *distributed* input generators. By “distributed” we mean that the global set of generators is subdivided and stored on separate processors. Each processor sees only its local subset of generators, and no generator is stored on more than one processor. We assume no memory is shared between processors. Furthermore, the parallel distribution of generators may be non-disjoint and no underlying geometric structure is assumed.

The geometric interpretation of (1) is that Voronoi cell  $V_i$  consists of all points in  $\Omega$  that are closer to its generator  $\mathbf{g}_i$  than to any other generator. The difficulty of distributed-memory construction of this definition is apparent: the boundary of any local generator’s Voronoi cell depends on the locations of all neighboring generators, whether their positions are stored on the local processor or not. Parallel communication between processors is therefore required to construct the consistent global Voronoi topology. Such communication should strive to be efficient, minimizing both the size and frequency of messages (i.e. generator positions) passed between processors.

The design principle of our parallel algorithm is to provide a generalized parallel interface to any serial or shared-memory construction algorithm. We establish parallel communication in order to determine a minimal but complete set of generator positions from neighboring domains such that each domain can construct the consistent subset of the global Voronoi diagram locally.

The number of serial construction algorithms for Voronoi and Delaunay diagrams are too numerous to mention here. (See [19] for an excellent summary.) Optimally-efficient implementations of many algorithms exist in the literature as do robust implementations in many popular two- and three-dimensional tessellation libraries (for instance, [23, 24, 26, 30]). We broadly divide such algorithms into two categories: direct and dual-based algorithms. Direct algorithms construct the Voronoi directly from input generators. Popular methods include incremental inclusion, divide-and-conquer, and plane-sweep methods such as Fortune’s algorithm [8]. Dual-based algorithms compute the Delaunay diagram directly from input generators, then construct the Voronoi as its geometric dual.

At present, no direct algorithm for parallel Voronoi diagram construction has been formulated specifically for distributed-memory platforms. So-called “stitching algorithms” perform distributed construction by piecing together local tessellations computed on separate processors [11, 31]. However, such algorithms rely on a geometric decomposition of a stored global set of generators—a memory-intensive process for many scientific applications [17, 28, 18]. Numerous algorithms for direct Voronoi construction on shared-memory devices have been formulated, for instance [5].

By contrast, several parallel construction algorithms for Delaunay diagrams have been formulated for both shared-memory [2] and distributed-memory [3, 4, 15, 28] environments. Computing the Voronoi dual from a distributed Delaunay diagram is also discussed in [28].

*Remark:* Many serial Voronoi construction algorithms generalize to higher dimensions, non-Euclidean distance metrics, and non-point generators [19]. We note that the pro-

posed parallel algorithm depends only on the resulting tessellations that such methods return, not on their underlying machinery. Thus our algorithm offers a means to potentially parallelize the construction of Voronoi tessellations having greater generality than those considered in this paper.

### 1.3 Applications

Applications for Voronoi tessellations as a mesh structure are numerous. As one of only a handful of automatic, on-the-fly methods for tessellating a domain, the Voronoi is a convenient mesh structure for mesh-free numerical methods or finite-volume-based methods having a dynamic mesh topology [17, 20, 21, 28]. Voronoi meshes are also widely used in Geographic Information Systems [13, 14] and subsurface flow calculations [18]. One particularly nice feature of the Voronoi for Lagrangian moving-mesh methods [17] is that, under continuous motion of the generating points, dynamic evolution of the Voronoi cells is smooth with respect to their volume topology. By way of comparison, dynamic Delaunay triangulation in general changes discontinuously (and globally so) as triangles flip edges.

Voronoi diagram generation is also robust to changes in length scale, making it an attractive data structure for physical models that span several spatial scales, such as astrophysical hydrodynamics, geophysical flows, and fluid dynamics [17, 20, 21, 28]. As an intuitive means of decomposing domains, Voronoi tessellations are a useful tool for visualizing discrete, spatially-distributed data and dynamically allocating resources in parallel molecular dynamics simulations [12, 22].

Applications for centroidal Voronoi diagrams are particularly numerous. The generators of such tessellations coincide with the geometric centroids of their associated Voronoi cells (see Figure 2). Centroidal tessellations satisfy a number of important optimality conditions, with applications ranging from image processing and data compression to data interpolation methods, point quadrature, and minimal-error finite differences. Consider [6] for an excellent overview of these ideas.

Uses for Voronoi diagrams as finite element meshes are explored in [7, 27]. Such meshes require high amounts of grid regularity to ensure numerical accuracy; the authors explore methods for efficiently optimizing Voronoi mesh geometry to that end. Similarly, the authors of [32] propose using boundary-restricted Voronoi diagrams to iteratively improve mesh isotropy. We note that iterative optimization schemes oftentimes require repeated construction of Voronoi diagrams to achieve convergence. Such schemes benefit greatly from parallel construction methods as a means to improve overall efficiency.

## 2 Parallel Algorithm

This section describes the parallel algorithm for distributed Voronoi grid generation. The design principle of the algorithm is unique in that an arbitrary serial Voronoi or Delaunay construction method may be employed in its implementation.

### 2.1 Background and Terminology

We consider a distributed generator set: each processor sees only a local subset of the global generator set. The global set is never stored on any single processor. The parallel

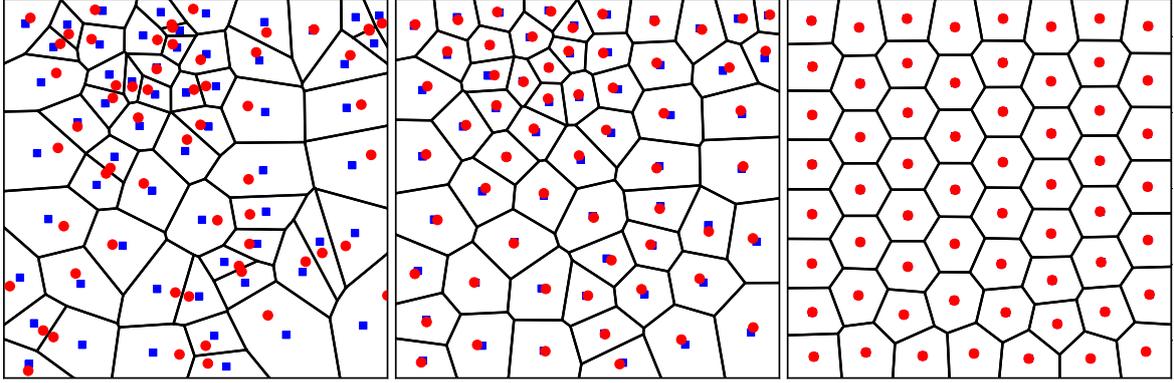


Figure 2: From left to right, a Voronoi tessellation after 0, 4, and 100 iterations of Lloyd’s algorithm [16]. Generators (red circles) move towards cell centroids (blue squares) with each iteration. The tessellation slowly converges to a centroidal Voronoi diagram.

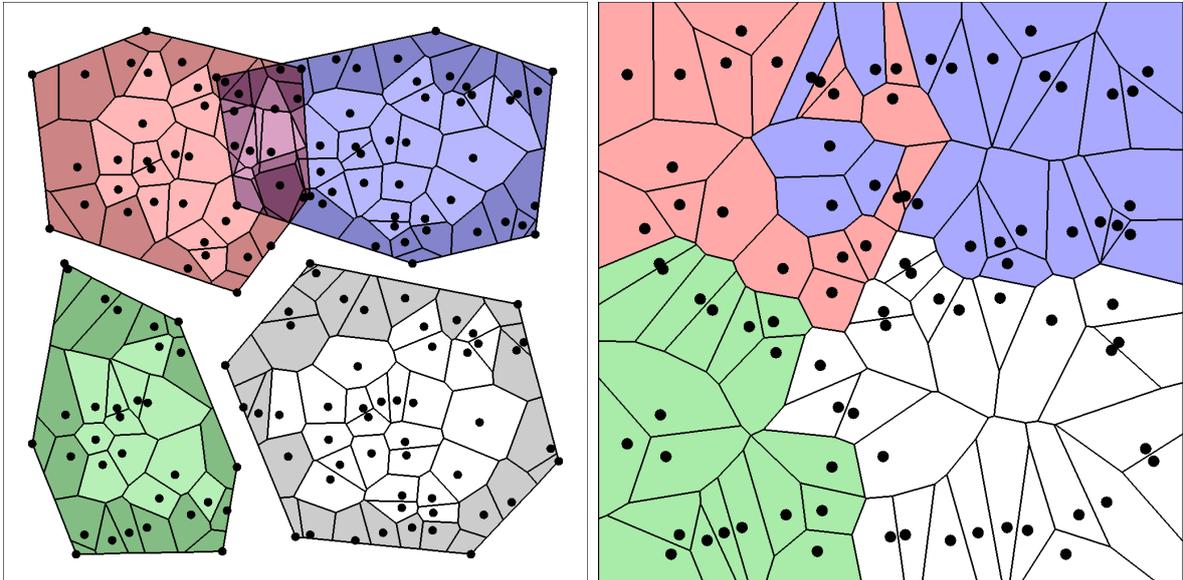


Figure 3: Diagrams generated to establish communication: local diagram on each processor, with visible generator cells shaded darker (left); and the visible mesh, identical on every processor, generated using the set of all visible generators to check adjacency (right). The top-right and bottom-left domains do not communicate.

algorithm establishes communication between spatially-distributed (and not necessarily disjoint) domains of points. We say that two processors  $P$  and  $Q$  *communicate* if, in the global Voronoi diagram, there exists at least one generator stored on  $P$  and one generator stored on  $Q$  whose Voronoi cells neighbor one another (that is, share a Voronoi face, edge, or node). Using generator positions communicated from neighboring domains, a consistent subset of the full Voronoi diagram is computed locally.

We begin by defining terms. Let  $\mathcal{G}$  be the global generator set and  $\mathcal{G}_P$  the subset stored on processor  $P$ . Let  $\mathcal{C}_P$  be the convex hull of point set  $\mathcal{G}_P$ . We define the *local diagram* on processor  $P$  by the Voronoi diagram constructed from  $\mathcal{G}_P$  and bounded inside its convex hull (see Figure 3, left).

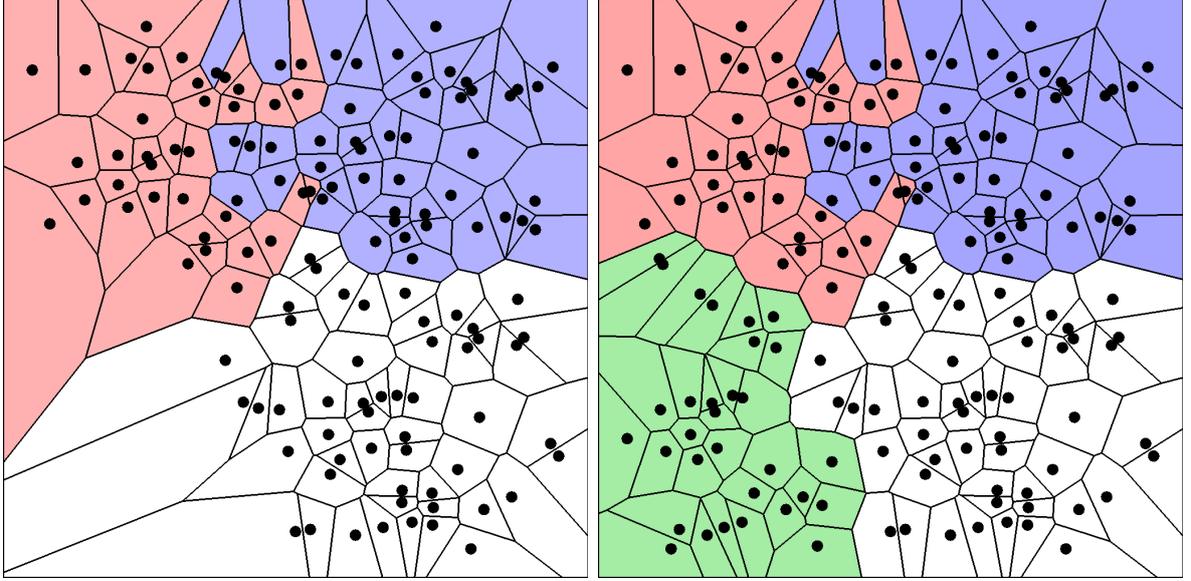


Figure 4: Diagrams generated post-communication: diagram constructed on the top-right processor using points from neighboring domains (left); cells from nonlocal generators will be deleted. The final diagram on each domain overlaid (right).

To establish communication, we define the set of *visible generators* on each processor  $P$ , denoted  $\mathcal{V}_P$ . A generator is *visible* if its Voronoi cell in the local diagram has at least one exterior element (face, edge, node). The visible sets for four different domains is illustrated in Figure 3 by shaded cells. Note that  $\mathcal{V}_P$  always contains the vertices of convex hull  $\mathcal{C}_P$ ; however, the set of hull vertices alone are not sufficient to establish communication. (See Section 2.3, Remark 1.) In general, for  $\mathcal{O}(N^2)$  local generators in 2D, the visible set is of size approximately  $\mathcal{O}(N)$  points.

The visible set is used to determine communication. In particular, if two domains are disjoint (i.e. their local convex hulls are non-intersecting) but neighbor one another, adjacent cells on separate domains correspond to visible generators.

The argument for this is straightforward. Suppose processors  $P$  and  $Q$  communicate such that the Voronoi cell of generator  $\mathbf{g}_P$  on  $P$  neighbors the Voronoi cell of  $\mathbf{g}_Q$  on  $Q$  in the global diagram. Further, assume  $\mathcal{C}_P$  and  $\mathcal{C}_Q$  are disjoint. (We return to the non-disjoint case later.) Finally, suppose neither  $\mathbf{g}_P$  nor  $\mathbf{g}_Q$  are visible. This implies that all of their local Voronoi elements lie completely inside their respective processors' convex hulls.

Let  $\mathbf{x}$  be a point on the element shared by  $\mathbf{g}_P$  and  $\mathbf{g}_Q$ . We show that the existence of  $\mathbf{x}$  contradicts the two generators not being visible:

- $\mathbf{x}$  cannot be inside the *local* Voronoi cell of  $\mathbf{g}_P$  or on its boundary. Otherwise,  $\mathbf{g}_Q$  would also be inside  $\mathcal{C}_P$ , contradicting the fact that the convex hulls are disjoint.
- $\mathbf{x}$  cannot be outside the *local* Voronoi cell of  $\mathbf{g}_P$ . This would make  $\mathbf{x}$  closer to some other generator in  $\mathcal{G}_P$  than to  $\mathbf{g}_P$ , thus violating the Voronoi property (1) locally.

The same reasoning applies to  $\mathbf{g}_Q$ . If  $\mathbf{x}$  can be neither inside nor outside either of two generators' local cells, then  $\mathbf{g}_P$  and  $\mathbf{g}_Q$  cannot share a Voronoi element in the global diagram. We arrive at a contradiction unless at least one of  $\mathbf{g}_P$  and  $\mathbf{g}_Q$  is visible.

## 2.2 Algorithm Statement

Each processor first computes its local Voronoi diagram and determines its set of visible generators. This can be accomplished by finding all local Voronoi cells intersected by the planar segments composing the local convex hull boundary.

Once computed, each processor then sends its visible set to every other processor. Although the potential exists for this to be a large number of point, we note that it is a dimensional reduction from the full local set: only points on the bounding surface of the processor are shared, not those on its interior volume. For large numbers of processors, this step may lead to a computational bottleneck for the algorithm. (See Section 2.3, Remark 4.)

At this point, processor  $P$  has access to the visible set from all other processors as well as their convex hulls, by the definition of  $\mathcal{V}$ .  $P$  establishes communication by performing two tests locally:

- (i) **Hull Intersection:** If convex hulls  $\mathcal{C}_P$  and  $\mathcal{C}_Q$  intersect, then  $P$  and  $Q$  *likely* communicate.
- (ii) **Adjacency:** A new Voronoi diagram is constructed on each domain using the full set of shared visible generators. We refer to this as the *visible diagram* and note that it is guaranteed to be identical on every domain (see Figure 3, right). If a cell whose generator belongs to  $P$  is adjacent to a cell whose generator belongs to  $Q$  on the visible diagram, then  $P$  and  $Q$  communicate.

Any processor  $Q$  satisfying either (i) or (ii) are added to  $P$ 's list of neighboring processors.

Each processor now maintains a consistent list of its neighbors. A second and final communication stage is performed: each processor shares its full generator set with each of its neighbors. A final Voronoi diagram is then constructed using the set of local generators plus the generators from each of the neighbors (see Figure 4, left). Voronoi faces, edges, and nodes are accumulated, and the neighbor list is revised (if necessary) based on the existence of shared elements. Cells not belonging to the local domain are deleted, leaving behind a local Voronoi diagram whose topology is consistent across processor boundaries (see Figure 4, right). We summarize the algorithm below.

### Parallel Algorithm:

#### Stage 1: Establish Communication

1. Construct *local Voronoi diagram* using local generator set, bounded by convex hull.
2. Compute set of *visible generators* and share with all other processors. Note this stage includes communicating the convex hulls.
3. Construct *visible Voronoi diagram* using the set of all visible generators. (Diagram is identical on each processor.)
4. Evaluate tests to establish domain neighbor list:
  - (i) Hull intersection: does another domain's convex hull intersect the local hull?
  - (ii) Cell adjacency: does a Voronoi cell from another domain neighbor any local cell in the visible diagram?

5. Send local generator set to neighbors.

## Stage 2: Compute Final Diagram

1. Construct Voronoi diagram from local generators plus neighboring domain generators.
2. Identify shared faces, edges, and nodes and revise neighbor list, if necessary.
3. Delete nonlocal cells from diagram.

*Remark:* Note that hull intersection does not imply communication in general. It is possible for the test to return a “false positive” for non-convex decompositions of the global generator set. (For instance, consider concentric rings of generators, with each ring assigned to a separate domain.) For the sake of robustness in the algorithm, we allow positive hull intersection to imply communication, at the potential cost of more communication than strictly necessary. This is only an issue for pathological domain decompositions.

## 2.3 Remarks on Efficiency

*Remark 1:* The set of visible points on a processor may be large, especially when generators are clustered close to the convex hull. This can severely impact parallel efficiency for large numbers of processors. It is reasonable to ask whether we really need to share all visible points. Can we get by sending only the convex hull vertices? Unfortunately the answer is no, as illustrated in Figure 5. The bottom of three disjoint domains has a visible point near the boundary of its convex hull (left). The top and bottom domains share a common edge on the global diagram (center). However, if only convex hull points are shared when computing the visible diagram (right), the middle domain can “shield” the top and bottom domains from seeing one another in the adjacency test. In general, an arbitrary number of cells on the visible diagram may shield two neighboring processors from seeing one another.

*Remark 2:* Sharing the full set of local generators with all neighboring processors is sufficient to construct a consistent Voronoi topology, but it is not necessary. Using a smaller subset would decrease message size and improve parallel efficiency, especially for large numbers of processors. It is reasonable to ask whether the visible set itself constitutes a sufficient subset of points. Unfortunately, the answer is again no. The case of neighboring processors having intersecting convex hulls is clear. Non-visible generators on each processor can have neighboring Voronoi cells in the global diagram. The case of disjoint neighboring processors is less obvious. Consider Figure 6: a point can be on the interior of its processor’s local convex hull (i.e. not visible) but be close enough to a point on a neighboring processor (left) to influence its Voronoi cell on the global diagram. If the bottom processor shares only its visible set (center), then the neighboring processor will compute an incorrect Voronoi topology locally. Sharing the full set (right) results in a globally-consistent topology.

It may be possible for each processor to compute a sufficient subset of generators to share with neighbors by computing a second visible diagram using the full set of local generators and the visible sets of all its confirmed neighbors. This hypothesis is not yet

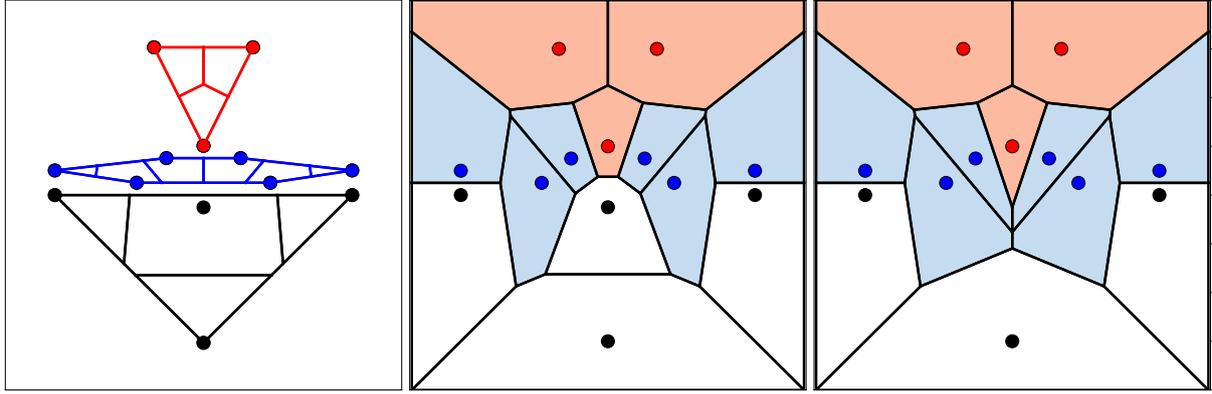


Figure 5: Local diagrams on each convex hull, colored by processor (left) and the final diagram (center); the diagram composed only of convex hull vertices, and excluding the visible set, (right) does not capture communication between the top and bottom processors.

tested; we regard it as part of ongoing efficiency improvements for the parallel algorithm.

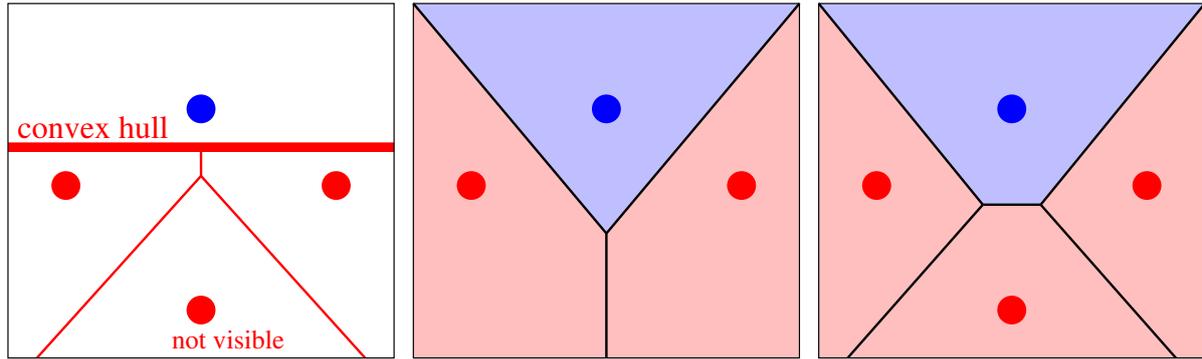


Figure 6: Zoom in of the convex hull boundary of local Voronoi tessellation and a single generator from a neighboring processor (left). The Voronoi diagram computed on the neighboring processor is incorrect if the first processor only shares its visible set (center) compared to sharing its full set (right).

*Remark 3:* The hull intersection test may be narrowed. For processors  $P$  and  $Q$ , a better test would check if any point in the visibility set of  $P$  is inside  $\mathcal{C}_Q$  and vice versa. This potentially reduces the number of “false positives” in the communication list and cuts back on the total number of send/receive operations.

*Remark 4:* Sharing the set of visible points with *every* processor is not mandatory for the algorithm to succeed. Preprocessing may be done to approximate a domain’s communication list, for instance by quantizing processor locations using a global octree. Such a strategy would improve efficiency for massively-parallel computations and speed up the construction of the visible diagram on each processor.

### 3 Results

The following set of results were obtained using Polytope, an open-source library for Voronoi grid generation [34]. An implementation of the parallel algorithm exists within Polytope.

In addition, many of the generated tessellations conform to a given input boundary (from simple boxes to complicated, non-convex figures). In each case, an unbounded Voronoi diagram is computed using generators lying inside the input boundary. Voronoi cells are then clipped by the boundary geometry, in the spirit of [32, 33], returning a boundary-conformal tessellation. Note that in the case of non-convex boundaries, the tessellation may no longer be strictly Voronoi along those boundaries.

#### Example Boundaries

Figure 7 presents a collection of complicated 2D example boundaries. In most cases, 2000 random generator positions are distributed across 36 processors. Color changes indicate domain boundaries. The parallel algorithm respects non-convex regions, and Voronoi topology is consistent across domain boundaries.

#### Centroidal Relaxation

Centroidal Voronoi diagrams possess a number of attractive properties for applications in image processing, discrete statistics, and finite element quadrature [6]. Further, dynamic simulations computed on centroidal Voronoi meshes benefit from approximately-equal angles and aspect ratios within cells [17].

Parallel construction significantly improve the efficiency of iterative optimization schemes which attempt to relax generator positions to cell-centroid locations. Capabilities are demonstrated on a toroidal boundary, with 4000 generators distributed over 20 processors. The diagram is relaxed using 1000 iterations of Lloyd’s algorithm [16]. Figure 8 gives the log of the condition number for each cell before and after relaxation as a relative measure of grid optimization [9]. Cell shapes approach regular hexagons, and the condition number approaches a constant value everywhere.

#### Moving Generators: Taylor-Green Vortex

The following test demonstrates the robustness of the parallel algorithm.  $50 \times 50$  generators initially on a unit lattice are decomposed disjointly onto 16 domains. The generators are then moved smoothly following a prescribed background velocity from the Taylor-Green vortex [29]. This test highlights the robustness of the parallel communication algorithm since each update of the generator positions can lead to changes in processor communication. Voronoi meshes at times 0, 0.5, 2, and 4 are given in Figure 9.

Note that while the original domain decomposition at time 0 (upper-left panel of Figure 9) allows our algorithm to successfully reduce the number of processors communicating with one and other, by the end state (in the lower-right panel) essentially all domains are in communication. While we lose the efficiency of parallel Voronoi generation as the problem proceeds, it is a nice demonstration of how the algorithm remains robust as we transition from a reasonable domain decomposition to a pathological one.

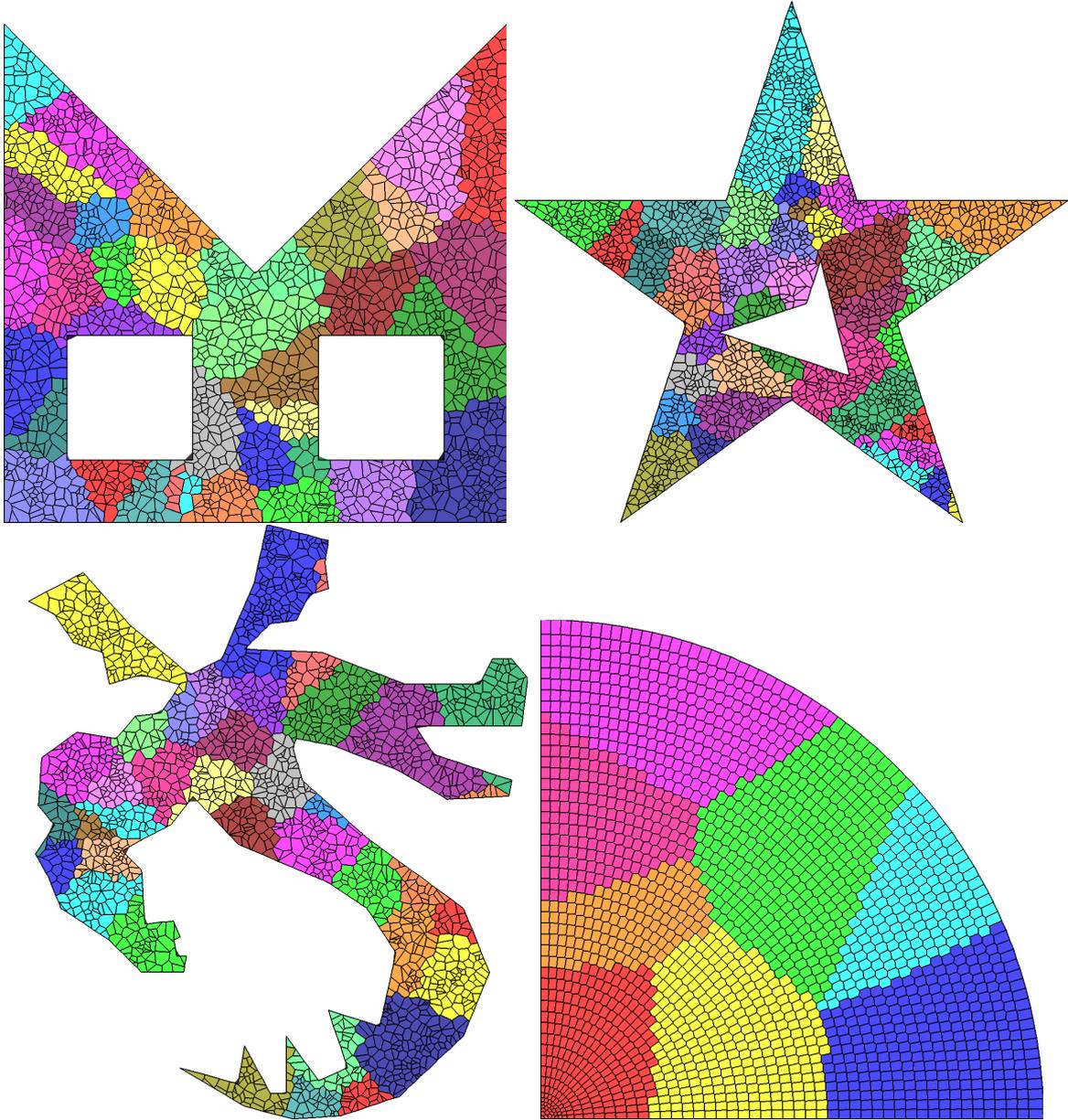


Figure 7: Distributed diagram computed on a collection of complicated boundaries. Colors indicate different computational domains.

## Cosmological Dataset

The following test presents a particularly challenging situation for Voronoi grid generation. The input consists of more than 2 million points distributed according to an underlying spatial structure but with point densities varying over 5 orders of magnitude. The generators in this example correspond to point masses and are taken from a meshless  $N$ -body simulation of cosmological-scale structure growth.

Figure 10 gives the resulting Voronoi diagram, zoomed into smaller and smaller length scales. Cells are colored by the log of the area, with values spanning  $10^{-12}$  to  $10^{-2}$ . Problems of this size benefit greatly from parallel construction. In this example, the full point set was decomposed onto 64 processors; it took 89 seconds to construct the

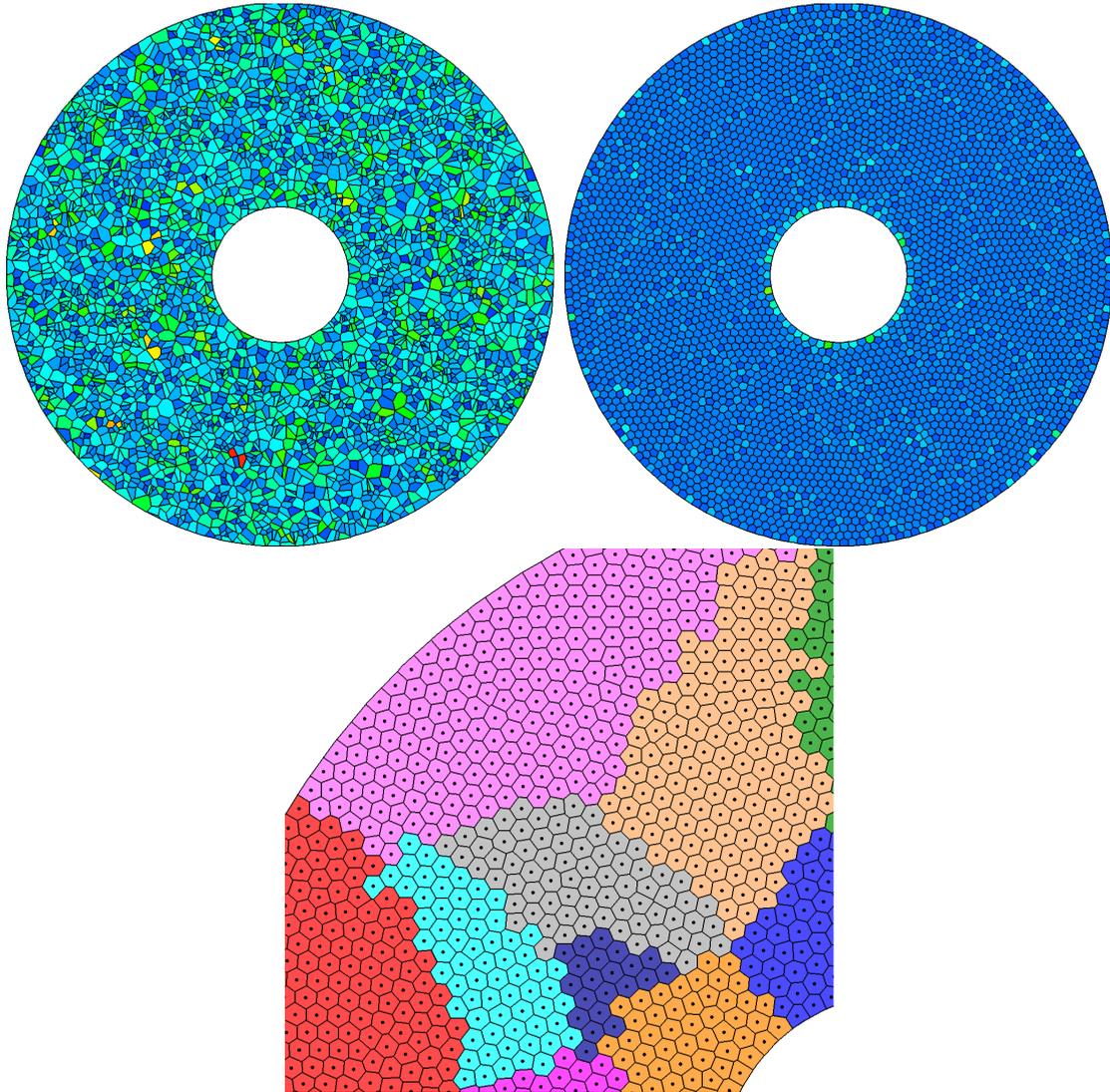


Figure 8: Centroidal relaxation performed on 4000 input generators distributed over 20 processors. Log condition number plotted for the initial mesh and after 1000 iterations of Lloyd's algorithm (top). Zoom in of the relaxed mesh showing domain decomposition (bottom).

tessellation.

## Scaling Study

We present parallel scaling results for the problem of meshing a unit square using  $G$  input generators on  $N$  processors. Three domain decompositions are considered and illustrated in Figure 11.

*Optimal*: generators and domains are laid out on a Cartesian lattice, with equal numbers of points per processor.

*Unbalanced*: points are randomly-distributed and assigned to processors to give spatially-disjoint domains (as in Figure 7). Work is not balanced between domains; generators per processor can differ by more than a factor of two.

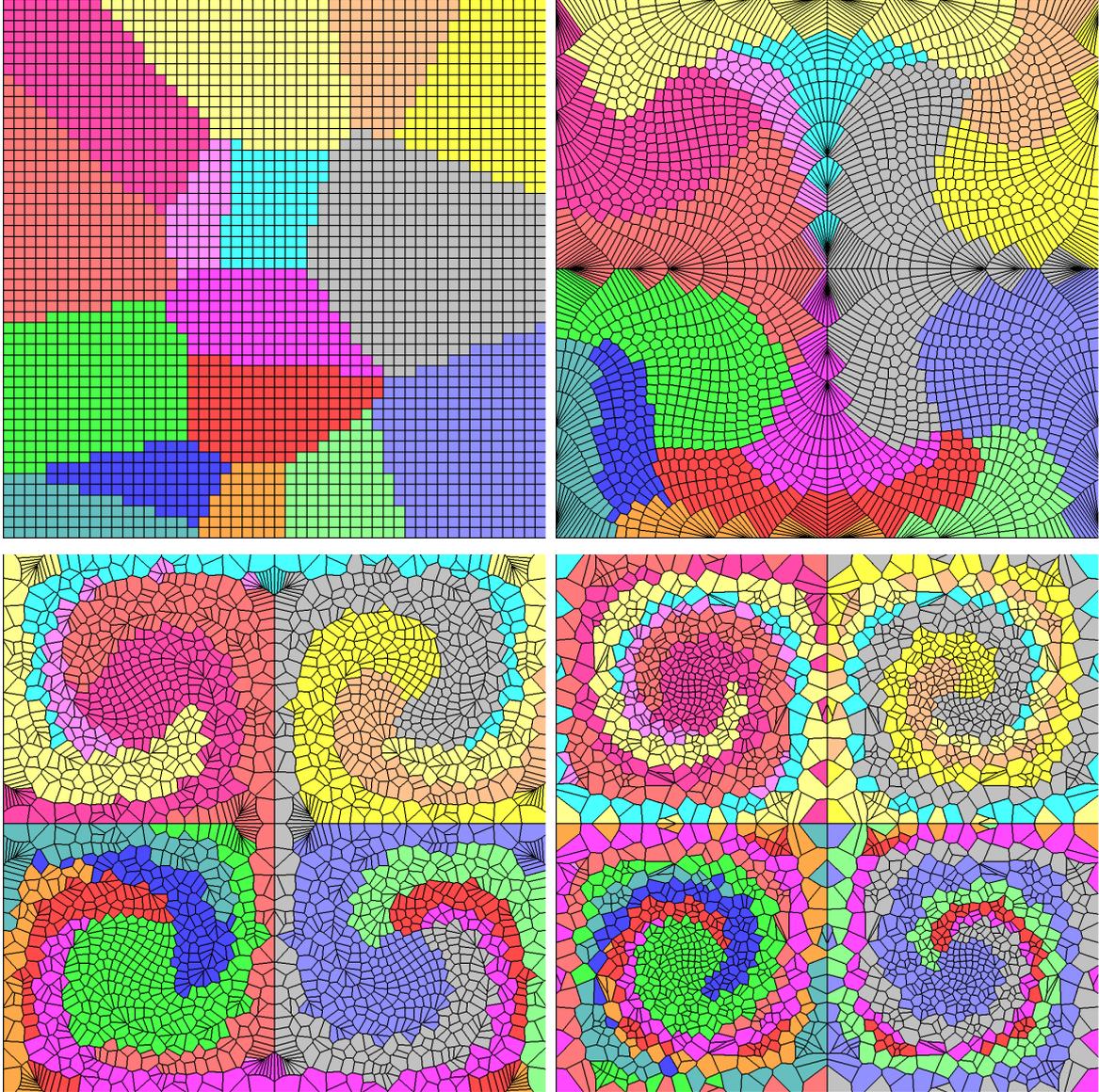


Figure 9:  $50 \times 50$  generators distributed over 16 domains. Generator positions advect according to a Taylor-Green vortex velocity field. Voronoi meshes at times 0, 0.5, 2, and 4. Colors indicate domain index.

*Random:* points are randomly-distributed and assigned randomly to processors. This is the worst-case scenario for our algorithm, as all processors must communicate with one another to form a valid topology.

Results from two different serial Voronoi tessellators are presented: a dual method using the Delaunay tessellator Triangle [24] and a direct method using the Boost.Polygon Voronoi library utilizing Fortune’s optimal sweep algorithm [8, 30]. As expected, the serial overhead is much smaller for the latter algorithm.

### Strong Scaling:

Strong scaling results are presented in Figure 12.  $G = 640,000$  generators are assigned to  $N$  processors such that per-processor workload decreases and communication increases in  $N$ . For the unbalanced decomposition, the number of generators per processor is not

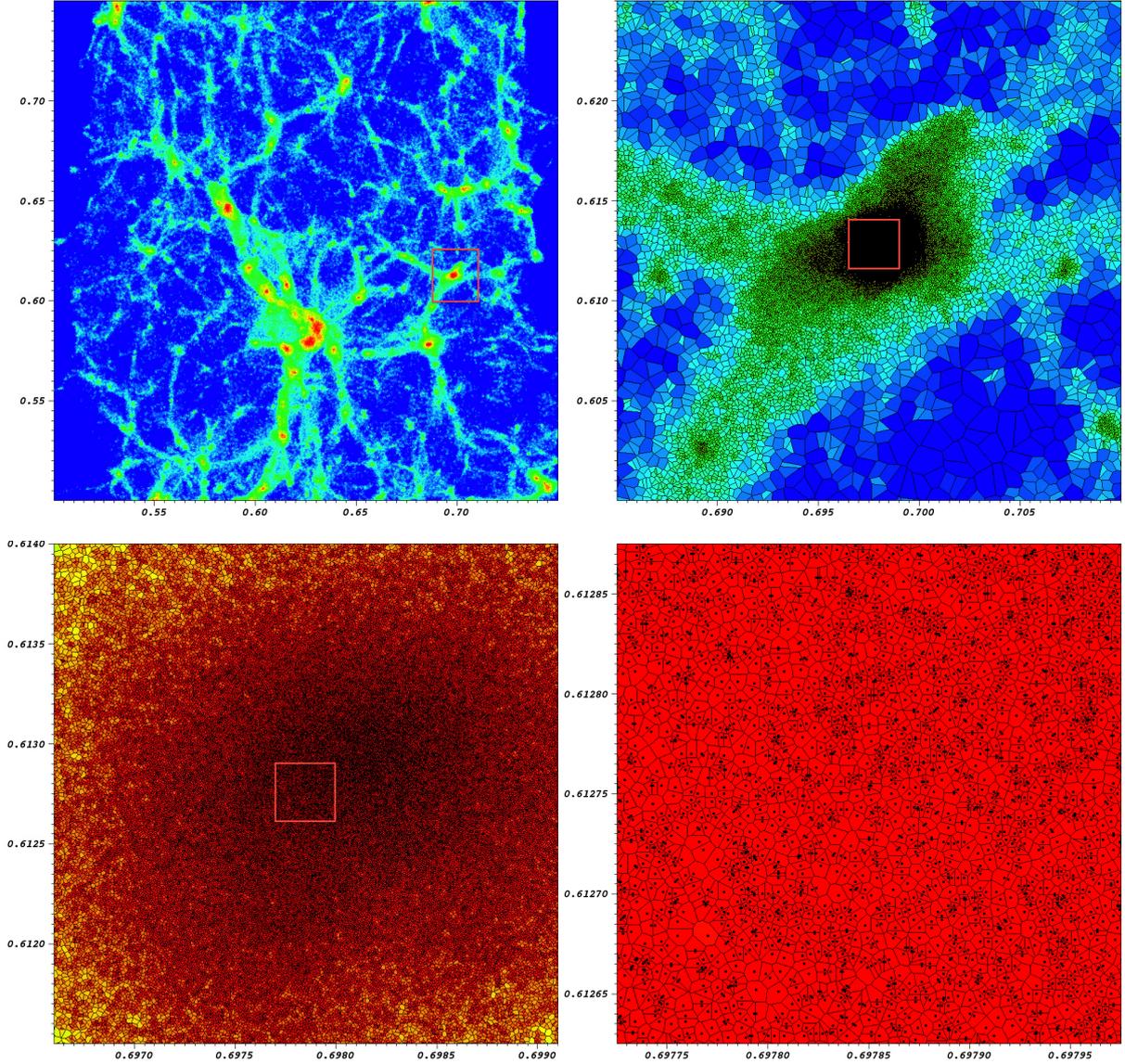


Figure 10: Progressive zoom-ins on a tessellation of a sample  $N$ -body data set. The original  $N$ -body calculation (provided by Jens Villumsen) is 3D; we have extracted a slice in  $z$  and flattened it to 2D. The upper-left panel shows the full tessellation (colored by  $\log(A_{\text{cell}})$ ), while the upper-right, lower-left, and lower-right are progressively zoomed-in views of a high density knot.

equal.

The various decompositions perform as expected. The random case leads to no parallel speedup: all domains communicate, causing processors to compute identical meshes as if in serial. For the other decompositions, total time decreases as per-processor workload decreases, with the optimal case outperforming the unbalanced case. The trend flattens out for larger numbers of processors indicating a larger and larger communication cost. Exploring this trend for processes numbering greater than 256 is a source of future work.

### Weak Scaling:

Weak scaling results are presented in Figure 13.  $G$  scales with the number of processors

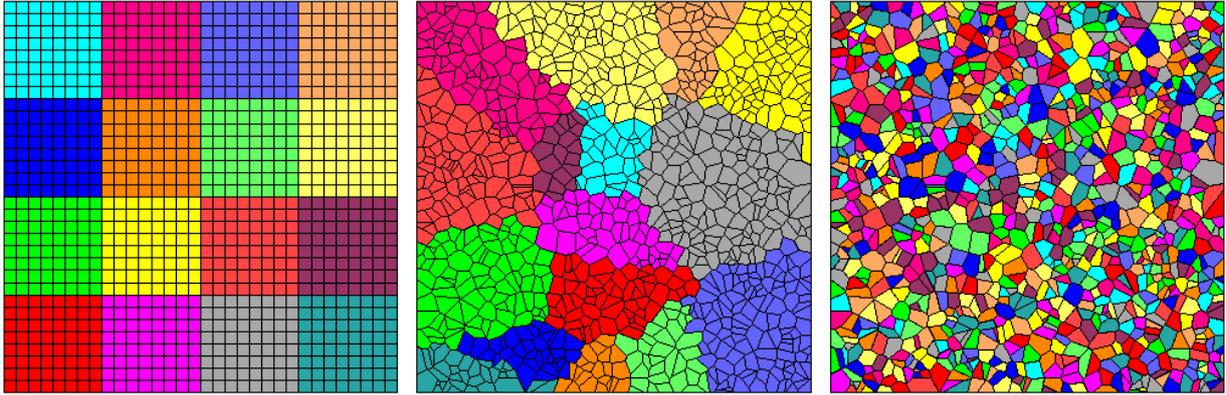


Figure 11: Decompositions of 1024 points onto 16 processors to test different scaling regimes: optimal (left), unbalanced (center), and random (right).

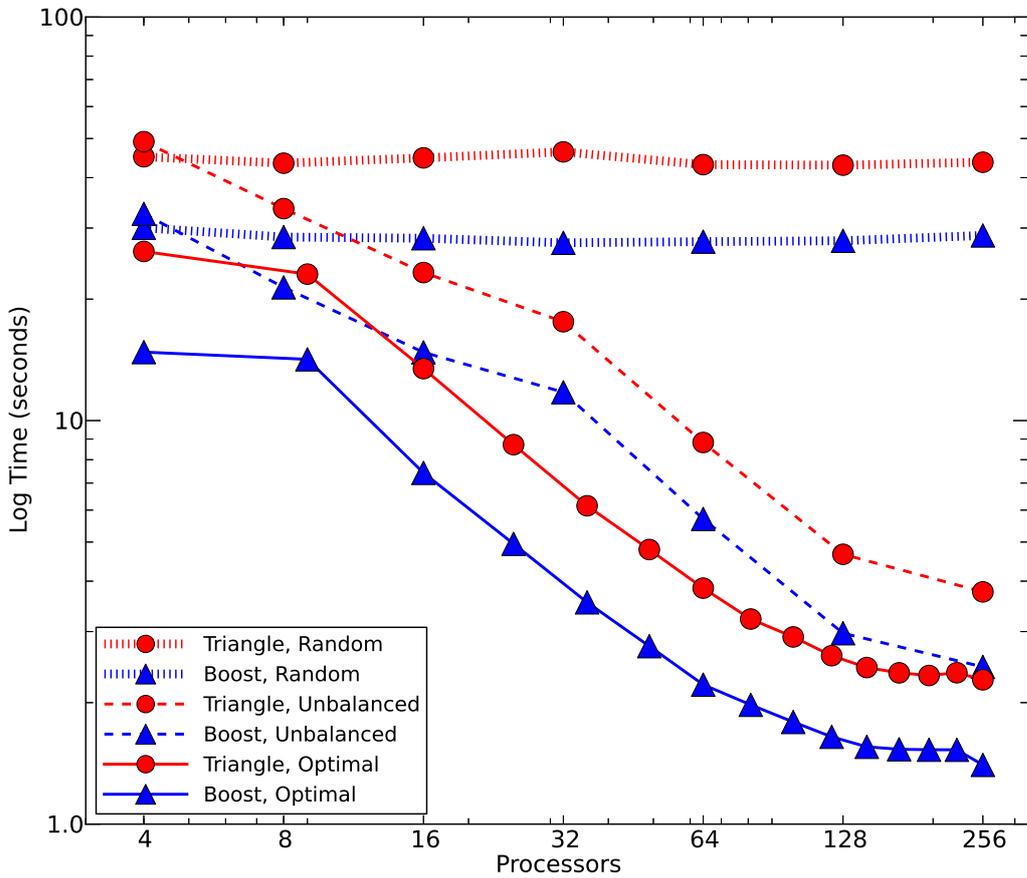


Figure 12: Parallel strong scaling of 512,000 generators on  $N$  processors. Generators uniformly distributed in a unit box. Comparison of reasonable and random domain decompositions.

such that each domain is assigned 2500 generators (exactly 2500 for random and optimal decompositions and 2500 on average for the unbalanced case). The scaling trend, normalized by generators per processor, was found to converge for generators numbering greater than 2500.

The random decomposition again exhibits serial scaling, as total calculation time is linear in the number of processors. The optimal case outperforms the unbalanced case, as expected. However, both trends are monotonically increasing in process count, indicating calculation costs are dominated by communication.

We would like to point out that the weak scaling results, while certainly not optimal, are not necessarily algorithmic in nature. Current implementation of the parallel algorithm has focused on robustness and generality. Work has not been done to profile the parallel algorithm or optimize its implementation at the time data was taken. Overall, the algorithm demonstrates the ability to compute consistent Voronoi grid topologies on large numbers of processors without failure. Optimization of its implementation is ongoing work; initial ideas for improving parallel scaling are provided in the closing remarks of Section 2.

## Conclusions

We present a novel procedure for constructing Voronoi diagrams from distributed input data. The algorithm is unique in that it computes a parallel communication layer for any arbitrary serial method for Delaunay or Voronoi grid generation. We have demonstrated the robustness of the algorithm on a variety of challenging two-dimensional geometries and input data. We have also provided initial scaling results which appear promising. Future work involves optimization of the algorithm as well as expansion of its testing into three dimensions.

The results presented in this paper were obtained using Polytope, an open-source library for the construction and storage of Voronoi diagrams as unstructured mesh data. Interested readers are encouraged to download the software and verify our results.

## Acknowledgments

Our thanks to Jens Villumsen for providing the  $N$ -body simulation dataset. Thanks also to Misha Shashkov for many productive discussions on Voronoi mesh generation. This work was performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

## References

- [1] F. Aurenhammer, Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure, *ACM Computing Surveys*, 23(3), pp. 345-405, 1991.
- [2] D. Blandford, G. Blelloch, C. Kadow, Engineering a Compact Parallel Delaunay Algorithm in 3D, *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, p. 292, 2006.

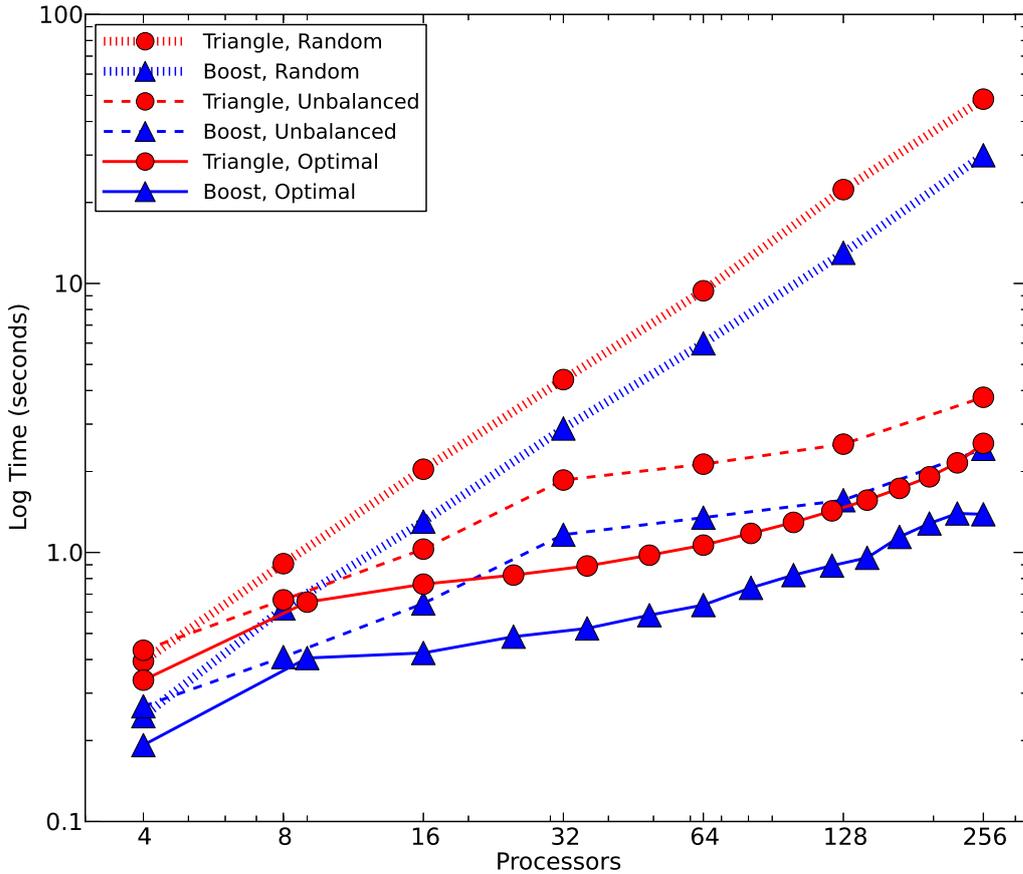


Figure 13: Parallel weak scaling for approximately 2000 generators per processor. Generators uniformly distributed in a unit box and decomposed into domains so that domain hulls are non-overlapping.

- [3] G. Blelloch, J. Hardwick, G. Miller, D. Talmor, Design and Implementation of a Practical Parallel Delaunay Algorithm, *Algorithmica*, 24, pp. 243-269, 1999.
- [4] N. Chrisochoides, D. Nave, Parallel Delaunay Mesh Generation Kernel, *Int. J. Numer. Meth. Engng.*, 58, pp. 161-176, 2003.
- [5] R. Cole, M. Goodrich, C. Dunlaing, A Nearly Optimal Deterministic Parallel Voronoi Diagram Algorithm, *Algorithmica*, 16, pp. 569-617, 1996.
- [6] Q. Du, V. Faber, M. Gunzburger, Centroidal Voronoi Tessellations: Applications and Algorithms, *SIAM Review*, 41(4), pp. 637-676, 1999.
- [7] Q. Du, D. Wang, L. Zhu, On Mesh Geometry and Stiffness Matrix Conditioning for General Finite Element Spaces, *SIAM J. on Numer. Anal.*, 47(2), pp. 1421-1444, 2009.
- [8] S. Fortune, A Sweepline Algorithm for Voronoi Diagrams, *Algorithmica*, 2, pp. 153-174, 1987.

- [9] R. Garimella, M. Shashkov, Polygonal Surface Mesh Optimization, *Engineering with Computers*, 20, pp. 265-272, 2004.
- [10] B. Gehrels, B. Lalande, M. Loskot, A. Wulkiewicz, *Boost.Geometry Library* (Version 1.0) , Available at <http://www.boost.org> (Accessed 7 June 2012).
- [11] D. Jacobsen, M. Gunzburger, T. Ringler, J. Burkardt, J. Peterson, Parallel Algorithms for Planar and Spherical Delaunay Construction with an Application to Centroidal Voronoi Tessellations, *Geosci. Model Dev. Discuss.*, 6, pp. 1427-1466, 2013.
- [12] R. Koradi, M. Billeter, P. Guntert, Point-Centered Domain Decomposition for Parallel Molecular Dynamics Simulation, *Comp. Phys. Comm.*, 124, pp. 139-147, 2000.
- [13] H. Ledoux and C. M. Gold, Modelling Three-Dimensional Geoscientific Fields with the Voronoi Diagram and its Dual, *Int. J. Geographical. Inf. Sci.*, 22, pp. 547-574, 2008.
- [14] H. Ledoux, Computing the 3D Voronoi Diagram Robustly: An Easy Explanation, *Voronoi Diagrams in Science and Engineering 2007 Conference Proceedings*, 2007.
- [15] S. Lee, C.-I. Park, C.-M. Park, An Improved Parallel Algorithm for Delaunay Triangulation on Distributed Memory Parallel Computers, *Parallel Processing Letters*, 11, 341, 2001.
- [16] S. Lloyd, Least Square Quantization in PCM, *IEEE Trans. Inform. Theory*, 28, pp. 129-137, 1982.
- [17] R. Loubere, P.-H. Maire, M. Shashkov, J. Breil, S. Galera, ReALE: A Reconnection-Based Arbitrary-Lagrangian-Eulerian Method, *Journal of Computational Physics*, 229, pp. 4724-4761, 2010.
- [18] R. Merland and B. Levy and G. Caumon, Building PEBI Grids Conforming to 3D Geological Features Using Centroidal Voronoi Tessellations, *IAMG 2011 Conference Proceedings*, 2011.
- [19] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, John Wiley and Sons, Chichester, West Sussex, England, 2000.
- [20] J. M. Owen, Augmenting Meshless Methods Using the Voronoi Tessellation, *SPHERIC Newsletter*, 13, 2011.
- [21] J. M. Owen, Applications of the Voronoi Tessellation for Mesh-Free Methods, *Multimat* [Presentation], 2011.
- [22] O. Pearce, T. Gamblin, B. de Supinski, M. Schulz, N. Amato, Quantifying the Effectiveness of Load Balance Algorithms. *Int. Conf. on Supercomputing*, pp. 185-194, 2012.
- [23] C. Rycroft, Voro++: A Three-Dimensional Voronoi Cell Library in C++, *Chaos*, 19, 041111, 2009.

- [24] J. Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, *Applied Computational Geometry*, 1148, pp. 203-222, 1996.
- [25] J. Shewchuk, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates, *Discrete and Computational Geometry*, 18(3), pp. 305-363, 1997.
- [26] H. Si (2011), *Tetgen* (Version 1.4.3) . Available at <http://tetgen.org> (Accessed 28 January 2012).
- [27] D. Sieger, P. Alliez, M. Botsch, Optimizing Voronoi Diagrams for Polygonal Finite Element Computations, *Proceedings of the 19th International Meshing Roundtable*, pp. 335-350, 2010.
- [28] V. Springel, E Pur Si Muove: Galilean-Invariant Cosmological Hydrodynamical Simulations on a Moving Mesh, *Mon. Not. of the R. Astron. Soc.*, in press, 2009.
- [29] G. Taylor and A. Green, Mechanism of the Production of Small Eddies from Large Ones, *Proc. Roy. Soc. A*, 158, pp. 499-521, 1937.
- [30] A. Sydoruk (2010-2012), *Boost.Polygon Voronoi Library* (Version 1.0) . Available at <http://www.boost.org> (Accessed 25 March 2013).
- [31] J. Wang, C. Cui, Y. Rui, L. Cheng, Y. Pu, W. Wu, Z. Yuan, A Parallel Algorithm for Constructing Voronoi Diagrams Based on Point-Set Adaptive Grouping, *Concurrency Computat.: Pract. Exper.*, 2013.
- [32] D.-M. Yan, B. Levy, F. Sun, W. Wang, Isotropic Remeshing with Fast and Exact Computation of Restricted Voronoi Diagram, *Computer Graphics Forum*, 28(5), pp. 1445-1454, 2009.
- [33] D.-M. Yan, W. Wang, B. Levy, Y. Liu, Efficient Computation of 3D Clipped Voronoi Diagram, *GMP 2010 Conference Proceedings*, 2010.
- [34] J. Johnson, J. M. Owen, D. Starinshak (2013), *Polytope* (Version 0.5.17). Available at <https://bitbucket.org/jjphatt/polytope>.