Efficient Delaunay Tessellation through K-D Tree Decomposition

Dmitriy Morozov
Lawrence Berkeley National Laboratory
1 Cyclotron Rd.
Berkeley CA 94720 USA
dmitriy@mrzv.org

Tom Peterka
Argonne National Laboratory
9700 S. Cass Ave.
Argonne IL 60439 USA
tpeterka@mcs.anl.gov

***Abstract*—Delaunay tessellations are fundamental data structures in computational geometry. They are important in data analysis, where they can represent the geometry of a point set or approximate its density. The algorithms for computing these tessellations at scale perform poorly when the input data is unbalanced. We investigate the use of k-d trees to evenly distribute points among processes and compare two strategies for picking split points between domain regions. Because resulting point distributions no longer satisfy the assumptions of existing parallel Delaunay algorithms, we develop a new parallel algorithm that adapts to its input and prove its correctness. We evaluate the new algorithm using two late-stage cosmology datasets. The new running times are up to 50 times faster using k-d tree compared with regular grid decomposition. Moreover, in the unbalanced data sets, decomposing the domain into a k-d tree is up to five times faster than decomposing it into a regular grid.**

I. Introduction

Delaunay and Voronoi tessellations are fundamental geometric structures for converting a point set into a mesh. Point data are common in simulations, for example, in N-body cosmology, atomistic molecular dynamics, particle-in-cell plasma physics, and finite-element models; and in sensor measurements such as those from weather reporting stations, census data, or LIDAR point clouds. For simplicity, in this paper we only talk about Delaunay tessellations, but everything we say immediately applies to Voronoi tessellations by duality.

Although expensive in both time and space, converting a point set into a mesh yields many useful properties. The mesh consists of cell elements, which fill the space. Given a field defined on the points, it is straightforward to extend it anywhere on the tessellation by interpolating the values in the interiors of the cells—not possible directly in the points' original discrete form. A Delaunay tessellation has the added property that the mesh automatically adapts to the distribution of the points: cells are smaller where points are closer together and larger where points are farther apart. This property can be used for accurate density estimation [1], [2], [3] or for deriving geometric statistics about cells (volume, surface area, connectivity) that further inform features such as clusters and boundaries in the data [4]. Tessellations can even be incorporated into the computation of N-body calculations, for example alternating between particle and mesh representation in hydrodynamics simulations [5].

The data sizes in the above applications—which can exceed one trillion points—and the time and space requirements to compute and store a mesh representation of those points, demand a parallel tessellation algorithm designed for distributed-memory high-performance computing (HPC). The parallel efficiency of such algorithms depends primarily on load balance. A perfectly balanced point distribution with the same number of points per processor can achieve near perfect speedup, while the worst possible distribution with all points on one processor is essentially serial. The other effect of a balanced point distribution is memory scalability. The ideal memory usage of an algorithm is inversely proportional to the number of processors, but actual memory usage depends on the load balance. A skewed data distribution can easily cause an algorithm to exceed available memory, and simply using more processors cannot solve the problem.

We cannot control the nature of the input data, but we can control its distribution to processors. Data distribution involves two steps: the global domain is first decomposed into 3D regions (blocks), and then the blocks are assigned to processors. One approach is to decompose the domain into a regular grid of blocks of uniform size. However, when a point set is highly clustered, such a regular decomposition yields blocks with widely varying numbers of points. This imbalance can be compensated by assigning different numbers of blocks to processors, but the granularity of this correction is only as fine as the size of one block. The advantage of the regular grid decomposition is that parallel algorithms that rely on it are easier to implement because the connectivity of a block with its neighbors is simple and consistent.

An alternative approach is to subdivide the global domain into unequal size blocks whose extents contain an approximately equal number of points, and then to assign equal number of blocks to processors. Unlike in the regular grid decomposition, the placement of the block boundaries—and hence, the load balance—can be as accurate as desired, but parallel algorithms are more difficult to develop because the connectivity of blocks with their neighbors is more complicated.

In this paper, we implement a distributed k-d tree to decompose a 3D domain into irregular size blocks by recursively splitting alternating dimensions of the domain with approximately equal number of input points in each child block. We evaluate two different methods for computing the approximate median of the parent block's points in order to determine where to split the children: a histogram of the distribution of point coordinates along the current dimension and a set of random samples of points. The accuracy of the approximation can be

controlled by the number of histogram bins or the number of samples.

Next, we develop a new parallel Delaunay tessellation algorithm based on the k-d tree decomposition. The assumptions of prior parallel Delaunay algorithms designed for regular grid decompositions no longer hold for the k-d tree because blocks and their neighbors are no longer connected in the same way, and Delaunay tetrahedra can now easily extend beyond the original neighborhood of a block. We prove the correctness of our parallel algorithm; that is, we show that the distributed tessellation it returns is globally Delaunay. We evaluate the performance of our algorithms by comparing the run time, scalability, and load balance of our tessellation algorithm with previously published results based on a regular grid decomposition. We break down the performance into the time to build the k-d tree and the time to compute the Delaunay tessellation and compare each of those components to building and using a regular block decomposition.

Section II summarizes related literature. Section III reviews an existing distributed Delaunay tessellation algorithm and explains its main assumption on the point distribution. Section IV covers the construction of the k-d tree—decomposing the domain into blocks and exchanging the input points. To do so, we develop a multi-round swap-reduce exchange algorithm that sorts points into half-spaces in each round according to an approximate median. The median can be determined in one of two ways: histograms and random samples. Once the k-d tree is built, a new adaptive neighborhood algorithm in Section V computes the parallel tessellation without any restrictions on neighborhood connectivity. In Section VI, we test the scalability and performance of both building the k-d tree decomposition and computing Delaunay tessellations of cosmological dark matter particles at scales up to 4096^3 points and 128Ki MPI processes on two supercomputing platforms.

To summarize, the contributions of this paper are:

- a distributed k-d tree decomposition using interleaved swap- and merge-reductions with configurable methods for computing the approximate median split coordinate;
- an adaptive neighborhood parallel Delaunay algorithm with no restrictions on neighborhood connectivity and a proof of its correctness;
- an evaluation of the new algorithm, including comparison of its performance with an existing method.

II. Related Work

Trees are often used as search structures to accelerate query processing. The original k-d tree was invented by Bentley in 1975 [6] to accelerate multidimensional queries of k-dimensional records. A binary tree where the children at level $j+1$ are split from the parent at level j along its median in the dimension $k \mod j$, k-d trees differ from regular spatial subdivision hierarchies (such as octrees in 3D) in two respects. For high-dimensional data, such as feature vectors in machine learning [7], the width of the tree is controlled because the tree is binary; it does not grow wider with increasing dimensionality. The other property is more useful for our purposes: k-d trees are load-balanced by definition. Figure 1 illustrates a k-d tree on a planar point set.

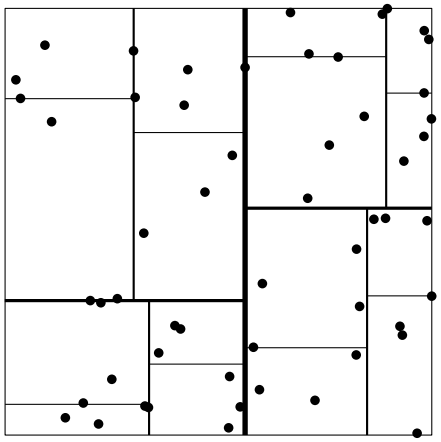

Fig. 1. Four levels of a k-d tree decomposition of a point set.

When the data are spatial, for example 3D points, variants of the k-d tree can be used for load-balancing spatial search. This is the case for point or cell location in computer graphics. For example, Garth and Joy [8] presented a celltree bounding volume hierarchy similar to a k-d tree for searching for the cell in a 3D mesh that contains a given point. Like a k-d tree, celltrees featured a recursive splitting of the domain in alternating dimensions; however, the split planes between two children were allowed to overlap. Andrysco and Tricoche implemented octrees and k-d trees for the same purpose using sparse matrices to conserve memory [9]. Spatial k-d and octrees are also used to compute force interactions in N-body simulations using the Barnes–Hut algorithm [10] and Fast Multipole Method [11].

Parallel k-d trees have appeared recently in the literature. Aly et al. [12] implemented a distributed k-d tree over 2048 machines in a MapReduce cluster and used it to execute parallel queries on an image database. Di Fatta and Pettinger [7] used a distributed k-d tree for static and dynamic load balancing a k-means clustering algorithm in a distributed-memory architecture. Their implementation in Java was tested on a 128-node cluster. Zhang et al. implemented several distributed trees, k-d trees included, in up to 20,000 processors in a peer-to-peer system [13]. Patwary et al. [14] used k-d tree decomposition to implement a parallel DBSCAN algorithm.

Distributed trees other than k-d were also published. Colbrook and Smythe devised a balanced search tree called $2^{P-2} - 2^P$ [15] for distributed-memory parallel machines, and Colbrook et al. [16] introduced a variation of a distributed B-tree for processes connected in a ring topology. Tu et al. [17] used the same parallel octree and regular grid decomposition for parallel volume rendering as was used to compute a seismic finite-element simulation. A different approach was taken by Larkins et al. [18], who developed Global Trees as a general-purpose interface to creating and accessing distributed trees in distributed-memory systems. Built on top of the ARMCI [19] one-sided communication library, their work enabled the programmer to have a logical global view of the distributed tree. Dynamic load balancing was provided by the Scioto library [20].

Serial algorithms for 3D Delaunay tessellations rely on randomized incremental construction. We refer the reader

to works by Clarkson and Shor [21], Guibas et al. [22], Edelsbrunner and Shah [23], and Barber et al. [24] for details. As a practical matter, several libraries exist for computing Delaunay tessellations in serial. In theory, our parallel algorithm is independent of the underlying serial engine. In practice, we include build options for using either Qhull[1], which implements the algorithm of Barber et al. [24], or CGAL[2] (Computational Geometry Algorithms Library) [25], which implements a numerically robust and efficient version of a serial Delaunay algorithm. Our experiments in this paper use CGAL.

Parallel computational geometry algorithms have been researched as well. Rycroft [26] published a parallel Voronoi algorithm for shared memory. In 2D, Miller and Stout studied parallel algorithms for 2D convex hulls [27]; Jeong [28] derived a parallel 2D Voronoi diagram algorithm, and Blelloch et al. [29] generated a 2D Delaunay triangulation on 8 processes. In 3D, Dehne et al. [30] computed a parallel distributed-memory 3D convex hull, and Cignoni et al. [31] presented a parallel 3D Delaunay tessellation for 64 processes. The closest work to ours is Peterka et. al [32] who, based on a 3D regular domain decomposition, demonstrated parallel strong scaling efficiency of up to 90% for 2048^3 points uniformly distributed over 128Ki MPI processes. The efficiency dropped, however, to as low as 14% when dark matter particles from a late-stage cosmological simulation were used as input because of their unbalanced distribution.

III. Background: Parallel Delaunay algorithm

A Delaunay tessellation of a set of points is the collection of tetrahedra, with vertices in the point set, whose circumspheres contain no points in their interior. We are interested in computing a distributed representation of the Delaunay tessellation. We assume that all the points are partitioned between a set of blocks (these initial points are *local* to the blocks), and each block may replicate some points from other blocks. The collection of Delaunay tessellations of the points within blocks is a distributed representation of the full tessellation if the set of tetrahedra incident on any local point in any block is exactly equal to the set of tetrahedra incident on that point in the full tessellation.

Peterka et al. [32] introduce Algorithm 1 to compute the distributed Delaunay tessellation. The algorithm ensures that

Algorithm 1 Parallel Delaunay algorithm

```
1: decompose domain into regular blocks
2: sort points into the blocks
3: for each block b do                                  ▷ in parallel
4:     compute the Delaunay tessellation of the points
5:     for each circumsphere S of a Delaunay tetrahedron do
6:         for each neighbor block b' of the block b do
7:             if S intersects b' then
8:                 enqueue vertices of S to b'
9:     enqueue points on the convex hull to all neighbors
10: exchange points
11: for each block b do
12:     insert the newly received points into the tessellation
```

[1] qhull.org
[2] cgal.org

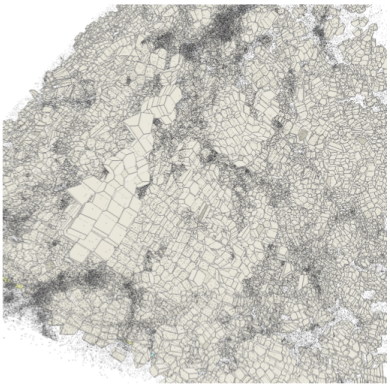

Fig. 2. Dark matter particles in cosmology simulations cluster into halos and voids whose densities can vary by six orders of magnitude in late time steps.

the circumspheres of the tetrahedra incident on local points are empty. It simplifies the naive approach that would send the local Delaunay circumspheres to the neighbors they intersect and wait to receive back the points that fall into those circumspheres. Instead, it takes advantage of a symmetry in the tessellations. Delaunay edges are undirected: if a point p needs to be connected to a point q by a Delaunay edge, then q needs to be connected to p. This symmetry allows the above algorithm to exchange points directly, skipping the circumsphere exchange.

As Peterka et al. [32] show, the result of the above algorithm is a distributed Delaunay tessellation. This result is guaranteed to be correct under the following assumption.

Neighborhood assumption: no edge in the global Delaunay tessellation leaves the neighborhood of a block containing one of its vertices.

The assumption requires that for every point in the local block, its Delaunay neighbors must be contained either in the block or inside one of its neighbors. In the case of the regular decomposition with periodic boundary conditions, the assumption is satisfied in all the data sets we encounter in applications: Delaunay neighbors are contained in one of the 26 neighboring blocks.

IV. K-d Tree Decomposition

Although the simplicity of regular domain decomposition is appealing—both the decomposition and the above tessellation algorithm are easy to implement—it suffers from load imbalance if the data is not distributed evenly throughout the domain. This situation is common in practice, for example, in late-stage cosmological simulations, where dark matter clumps into dense clusters called halos and vacates regions called voids (Figure 2). In this case, the number of points that fall inside a block of a regular decomposition varies by orders of magnitude based on the number of clusters in the block. The situation is exacerbated with increasing number of smaller blocks.

To cope with this problem, we can decompose the domain into a k-d tree. A data structure commonly used for proximity queries, a k-d tree is built from a point set by finding the median of the points along the first coordinate, splitting the data into two halves based on the median, and recursively repeating the procedure on the two halves, cycling through the coordinates used for splitting. By construction, the result is a

balanced tree, where every block at a given level has the same number of points.

We decompose the domain in parallel into a prescribed number of blocks (assumed to be a power of two). Our blocks are 3D rectangular boxes containing a subset of the global input points. In keeping with the block-parallel programming model, the following discussion is presented in terms of blocks rather than processes. It allows us to ignore the distinction of whether a block is processed by an MPI rank or a thread. Readers unfamiliar with this point of view may assume that an MPI process has only one block and may substitute "process" or "processor" for "block" and "MPI rank" for "block global ID (gid)" in most instances below without changing the algorithm.

We initially assume that any block may contain points anywhere in the domain. The computation interleaves two types of rounds: median selection and point exchange. We come back to median selection later. For point exchange, the blocks are organized into groups based on what part of the domain they may contain. Initially, there is a single group—all blocks may contain points anywhere in the domain. Once the first median is picked, each block whose global ID (gid) has the least significant bit set to 0 sends its points above the median threshold to its partner, the block with the same gid except for the least significant bit set to 1. Vice versa, the latter block sends to the former those points that lie below the median threshold. Once all blocks complete this pairwise exchange, we divide them into two groups: the first consists of those gids with the least significant bit set to 0 (they contain the lower half of the domain); the second has the bit set to 1 (upper half of the domain). In subsequent iterations, the communication happens only within the groups (with splitting decisions made based on the second, third, and so on significant bits). Algorithm 2 summarizes the procedure.

It's computationally expensive to pick the median exactly, so we implement two approximation schemes: building a histogram of the coordinate distribution and random sampling. In the former case, each block computes a histogram of the coordinates of its local points, the histograms are reduced to a designated root block within the group (the histograms are added in pairs along the way); the root picks the median from the histogram and broadcasts it to the rest of the blocks in the group. In the latter case, each block picks a random sample of its local points, the samples are combined via a reduction to the root of the group, the root picks the median of the combined sample and broadcasts it to the rest of the blocks.

Links. The Delaunay tessellation algorithm needs to exchange information between neighboring blocks. It is, therefore, necessary for each block to keep track of which blocks it intersects. We build up such neighbor links simultaneously with the decomposition. Initially, each block has either an empty set of links, if the domain is Euclidean, or it sets the links to point to itself in all six directions, if we are working with periodic boundary conditions. In the i-th iteration of the k-d tree construction, after a median split is selected for a group of blocks, each block sends to each one of its neighbors (which at later rounds fall into different groups) the median split value selected in its group. Once it gets the split values from its neighbors, the block has enough information to update the links. It prunes those blocks that it no longer intersects. It updates the i-th least significant bit in the gid of every neighbor to either 0 or 1 (or possibly both, adding an extra block to the link), depending on its own and the neighbor's split configuration. Finally, it adds its partner block (with the same gid, except for the inverted i-th bit) to its link.

Algorithm 2 K-d tree decomposition algorithm

for each block **do** ▷ in parallel
  $points \leftarrow$ input points in the block
  $dim \leftarrow 0$
  $group \leftarrow [0, b-1]$ ▷ b = number of blocks
  $gid \leftarrow$ global id of the block
  for round $i \in [0, \log_2 b)$ **do**
    $m \leftarrow$ select-median$(points, group, dim)$
    $i\text{-}bit \leftarrow 1 \ll i$
    $partner \leftarrow gid$ XOR $i\text{-}bit$
    $points \leftarrow$ swap-points$(partner, points, m)$
    $group \leftarrow \{x \in group \mid x$ AND $i\text{-}bit = gid$ AND $i\text{-}bit\}$
    exchange medians with neighbors
    update links ▷ details omitted for verbosity
    $dim \leftarrow dim + 1 \mod 3$

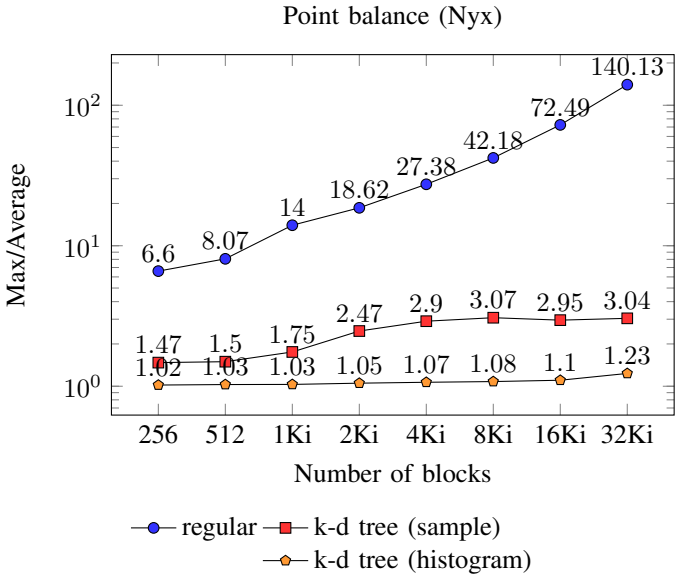

Fig. 3. Point balance for regular vs k-d tree domain decomposition of 2048^3 points from a late-stage cosmological simulation.

Balance. Figure 3 illustrates the ratio of the maximum to the average number of points in any block for varying number of blocks in the decomposition, for both median approximation schemes (we use 1024 histogram bins and 1024 samples per block). We also show this ratio for the regular decomposition. The dataset consists of 2048^3 dark matter particles from a late-stage cosmological simulation. These particles are the input points for our algorithm. As Figure 4 and the figures in Section VI illustrate, the improved load balance is not only better for the Delaunay tessellation algorithm, but it also improves the time for the point distribution itself.

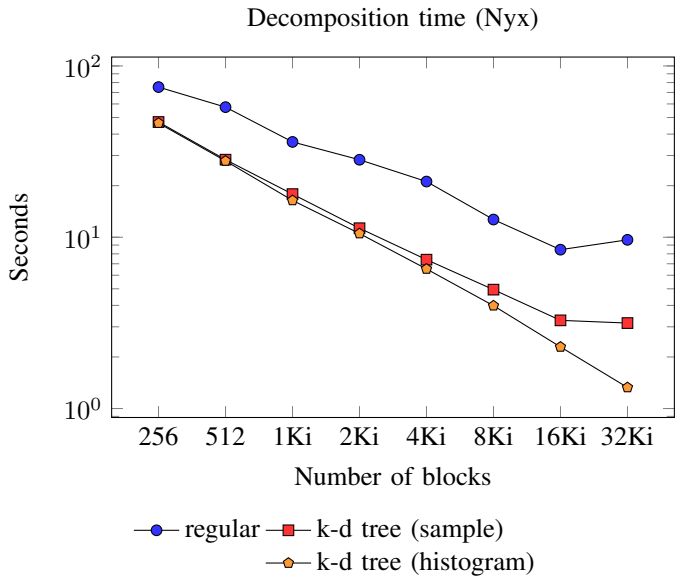

Fig. 4. Decomposition time for regular vs k-d tree domain decomposition of 2048^3 points from a late-stage cosmological simulation.

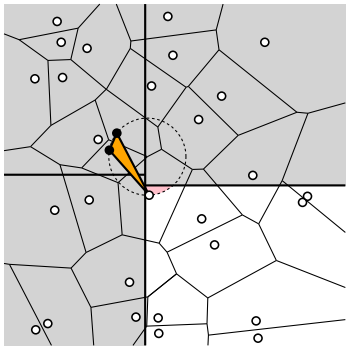

Fig. 5. Edges of the highlighted triangle leave the 1-neighborhood of the top-left block in the k-d tree decomposition of the points. So does its circumsphere.

V. Adaptive Neighborhood Algorithm

The load balance of the k-d tree decomposition comes at a price. Blocks in such a decomposition are not evenly spaced, and a block may be very close to another block that it doesn't intersect. This means that the *neighborhood assumption*, stated in Section III, can be violated. Figure 5 illustrates an example of such a violation. The neighborhood of the top-left block is highlighted in gray; it does not include the bottom-right block. On the other hand, the triangle highlighted in the figure contains points from both blocks. If the two don't communicate with each other, there is no way to construct the correct (distributed) Delaunay tessellation.

Algorithm. To work around this problem, we adaptively grow the blocks' neighborhoods. The algorithm proceeds in rounds. In each round, as in Algorithm 1, each block b iterates over all the circumspheres of its current Delaunay tessellation. If a circumsphere intersects a neighbor block b', then we enqueue to b' those points on the boundary of the circumsphere that are local to block b, i.e., they were assigned to it in the initial decomposition. (As a technical detail, the facets of the convex hull are treated as infinite tetrahedra, whose circumspheres degenerate into half-spaces.) In addition to the points, each block enqueues information about all of its original neighbors to every other neighbor.

Algorithm 3 Adaptive neighborhood algorithm

```
1:  decompose domain using a k-d tree
2:  for each block b do                                   ▷ in parallel
3:      compute the Delaunay tessellation of the local points
4:      original-nbrs ← initial block neighbors
5:      nbrs ← original-nbrs
6:      while not done do
7:          for each circumsphere S do
8:              for each neighbor b' ∈ nbrs do
9:                  if S intersects b' then
10:                     enqueue vertices of S (local to b) to b'
11:         enqueue original-nbrs to nbrs
12:         all-queued = all-add(number of points enqueued)
13:         if all-queued = 0 then
14:             break
15:         exchange points
16:         dequeue points, insert into the tessellation
17:         dequeue new-neighbors, add to the neighborhood
18:         nbrs ← new-neighbors
```

How do we decide when to stop? As we prove below, it suffices to stop once no block enqueues any points, which happens when none of the circumspheres that contain local points on their boundaries intersect any of the neighbor blocks added in the prior round. That's precisely the stopping condition we use in Algorithm 3, which summarizes the above description.

Correctness. As Peterka et al. [32] show, the distributed Delaunay tessellation is correct once the circumsphere of every tetrahedron that contains any of the block's local points is contained entirely in the neighborhood of the block. Accordingly, to prove the correctness of Algorithm 3, we need to prove correctness of our stopping condition. To do so, we consider the dual graph of blocks: its vertices are the blocks; two vertices are connected by an edge if and only if the two blocks intersect; see Figure 6. We define the k-neighborhood of a block to be the set of blocks at distance at most k in the dual graph. We verify that if a circumsphere does not intersect any blocks in the k-neighborhood of a given block, then it does not intersect any block in the k'-neighborhood, for $k' > k$. This claim would be trivial in the regular decomposition, where the block neighborhoods are convex, but it's not immediate in the k-d tree decomposition. As Figure 6 illustrates, a block in the 3-neighborhood may be closer to the given block than a block in the 2-neighborhood.

Fortunately, because the circumspheres themselves are convex, the stated claim is true in general; it's the contrapositive of the following lemma.

Lemma 1. *A sphere intersects $k+1$-neighborhood of a block only if it intersects the k-neighborhood of the block.*

Proof: Suppose a sphere intersects both the original block

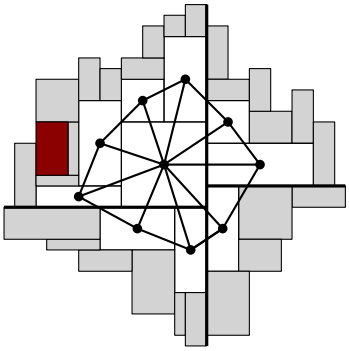

Fig. 6. Dual graph of the 1-neighborhood. 2-neighborhood in gray. Box from the 3-neighborhood shown in red; a sphere may intersect it even if it doesn't intersect the box to its left in the 2-neighborhood.

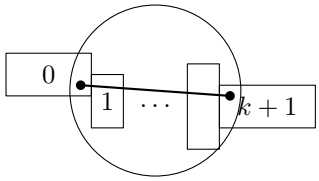

Fig. 7. Boxes intersected by a line segment within a circumsphere.

and a block at distance $k+1$ away from it. Consider two points, one in each block, and the line segment connecting them, which by convexity falls inside the sphere; see Figure 7. Let $0 = i_0, i_1, \ldots, i_n = k+1$ be the graph distances from the original block to the blocks intersected by the line segment, in order of intersection. The adjacent integers in this sequence differ by at most one because the blocks are neighbors in the dual graph. We claim that k appears in the sequence. Let A consist of all the indices in the sequence of values greater than k, $A = \{j \in \{0, \ldots, n\} \mid i_j > k\}$. A is clearly not empty, since $i_n = k+1$, and it does not contain all the indices since $i_0 = 0$ (we assume k to be positive). Let $m = \min A$. $i_m = k+1$, since if $i_m > k+1$, then $i_{m-1} > k$, and $m-1 \in A$, contradicting the definition of m. It follows that $i_{m-1} = k$, proving the claim. ■

Remark 2. *The proof does not rely on the structure of k-d tree decompositions. The claim is true in the general setting.*

Because in each round we test circumspheres against only the new neighbors, received in the previous round, we need to verify that as we add more points to the Delaunay tessellation, the circumspheres with local points on the boundary cannot start intersecting a neighbor that we have previously dismissed. This is true because, as the following lemma shows, a union of circumspheres around a point can only shrink as we add new points. Let S_σ denote a circumsphere of the Delaunay tetrahedron σ, and let $S_p = \bigcup_{\sigma \mid p \in \sigma} S_\sigma$ denote the union of circumspheres that contain p on their boundary. (For points on the convex hull, we treat the half-spaces supported by the convex hull facets as infinite circumspheres.)

Lemma 3. *If S_p is the union of circumspheres of tetrahedra in the Delaunay tessellation $\mathrm{Del}\, P$ of a point set P, and S'_p is the same union in the Delaunay tessellation $\mathrm{Del}\, P'$ of point set $P' = P \cup \{q\}$, then $S_p \supseteq S'_p$.*

Proof: Suppose the statement is false. Then there is a point $x \in S'_p$ such that $x \notin S_p$. Then there is a tetrahedron $\sigma \in \mathrm{Del}\, P'$ such that $x \in S_\sigma$ (with $p \in \sigma$). If q is not in σ, then $\sigma \in \mathrm{Del}\, P$, and we have a contradiction. Consider Delaunay triangle $\sigma - q$; it contains point p. Let σ_1 and σ_2 be two Delaunay tetrahedra in $\mathrm{Del}\, P$ that intersect in this triangle. Any sphere that intersects this triangle and no other point in P lies in the union of the two circumspheres, $S_{\sigma_1} \cup S_{\sigma_2}$. But this means that so does S_σ, which contradicts the assumption that $x \notin S_p$. ■

VI. Evaluation

The experiments in this section were performed on *Edison*, a Cray XC30 at the National Energy Research Scientific Computing Center (NERSC), and on *Mira*, an IBM Blue Gene/Q at the Argonne Leadership Computing Facility (ALCF). Edison is a 2.57-petaflop machine with 5576 nodes, each node with 24 cores (Intel Ivy Bridge 2.4 GHz) and 64 GB RAM. Mira is a 10-petaflop system with 49,152 nodes, each node with 16 cores (PowerPC A2 1.6 GHz) and 16 GB RAM. We use CGAL for the serial computation of Delaunay tessellations. All experiments use periodic boundary conditions and one block per MPI process (i.e., the number of blocks equals the number of MPI processes). All our source code is publicly available,[3] under BSD license.

A. Nyx Data

We used Edison to tessellate three outputs of cosmological simulations performed by Nyx, an N-body and gas dynamics code [33] written in C++ and Fortran 90, based on the BoxLib framework[4] for structured-grid adaptive mesh methods. Nyx follows the evolution of dark matter particles gravitationally coupled to a gas using a combination of multi-level particle-mesh and shock-capturing Eulerian methods. High dynamic range is achieved by applying adaptive mesh refinement to both gas dynamics and gravity calculations. The multi-grid method is used to solve the Poisson equation for self-gravity. The mesh structure used to update fluid quantities also evolves the particles via the particle-mesh method.

The three outputs, a subset of the simulations described by Lukić et al. [34], were all saved at the same redshift, $z = 2$, which corresponds to a late stage in the simulation, where the matter is already clumped together. The three differ in the number of particles (our input points), 1024^3, 2048^3, 4096^3, and in the box size, 20, 40, 80 h^{-1}Mpc, respectively. Figure 3 illustrates the load balance in the 2048^3 dataset, which we define as the ratio of maximum number of points in any block to the average, for the regular and k-d tree decompositions. For the former, the ratio varies from 6.6 at 256 blocks to 225 for 64Ki blocks. For the latter, using histogram median approximation scheme, which we use in all the experiments in this section, the load balance stays almost constant, going from 1.02 for 256 blocks up to 1.23 for 64Ki blocks.

Figure 8 shows the overall computation time: both domain decomposition (which includes point distribution) and the multi-round Delaunay tessellation. As is evident from the figure, the better load balance is responsible both for a significantly faster

[3] github.com/diatomic/tess2
[4] ccse.lbl.gov/BoxLib

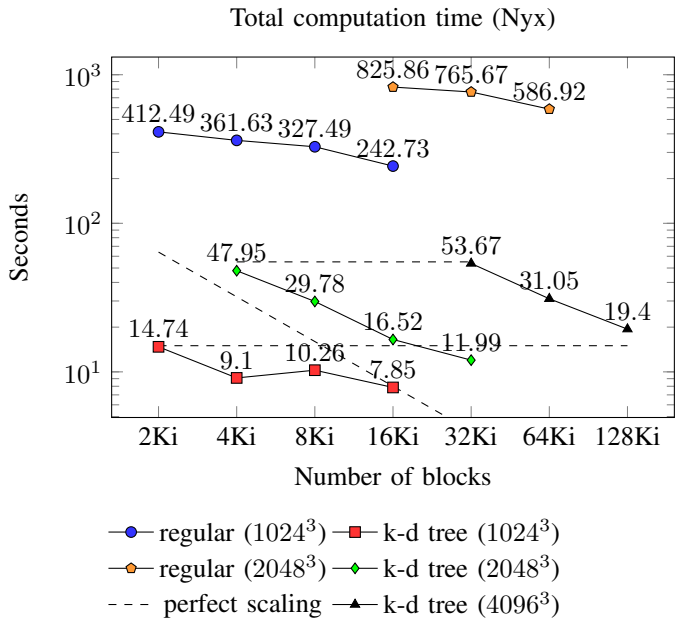

Fig. 8. Strong and weak scaling on a late snapshot of a Nyx simulation: total computation time includes point exchange and multi-round computation of the Delaunay tessellation.

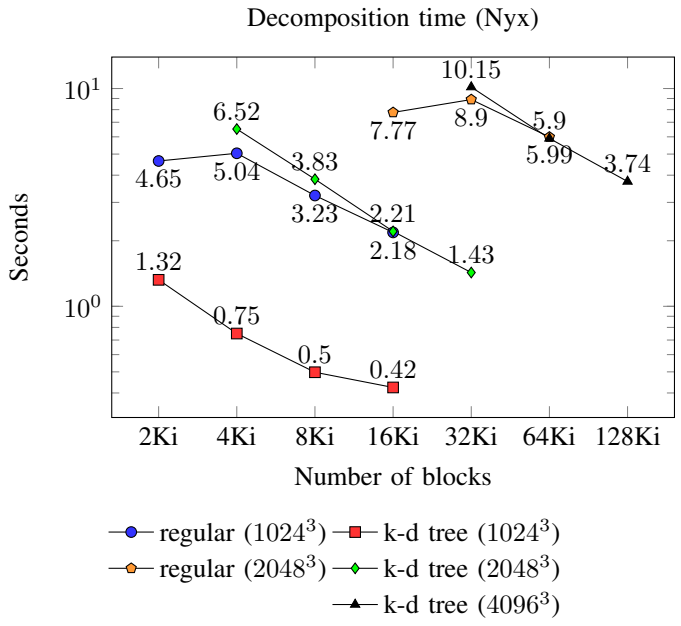

Fig. 9. Strong and weak scaling on a late snapshot of a Nyx simulation: point exchange time for k-d tree vs regular decomposition.

overall computation time (up to a factor of 50) and for much better scaling. At the higher end, on the 4096^3 data set, the strong scaling efficiency of the computation built on the k-d tree decomposition is 69%, going from 53.67 seconds on 32Ki to 19.4 seconds on 128Ki blocks.

The running time of the adaptive algorithm is sensitive to the number of rounds it takes to compute the correct tessellation. In the k-d tree experiments all runs take three rounds, except the 8Ki and 16Ki block runs on 1024^3 points, which take four rounds each. The extra round is what explains the slowdown evident in Figure 8 when going from 4Ki to 8Ki blocks on the 1024^3 point dataset.

Figure 9 breaks out the time it takes to exchange points between blocks. Here, despite a more complicated computation involved in the k-d tree construction (because of the extra rounds to pick an approximate median), the better point balance is responsible for roughly five times faster distribution time.

Figure 10 illustrates the memory used by the tessellation code, as measured at the end of the execution by the high-water mark, accessible through Linux's `/proc` interface. The advantage of the balanced point distribution is evident again. In case of the k-d tree decomposition, the per-block memory never exceeds 2GiB (which, conveniently, is below the 2.66 GiB per-core limit on Edison) and scales down with the increasing number of blocks. In case of the regular decomposition, for 2048^3 points, the memory use spikes to 29GiB. We do not evaluate the 4096^3 data set with regular decomposition simply because the computation ran out of memory.

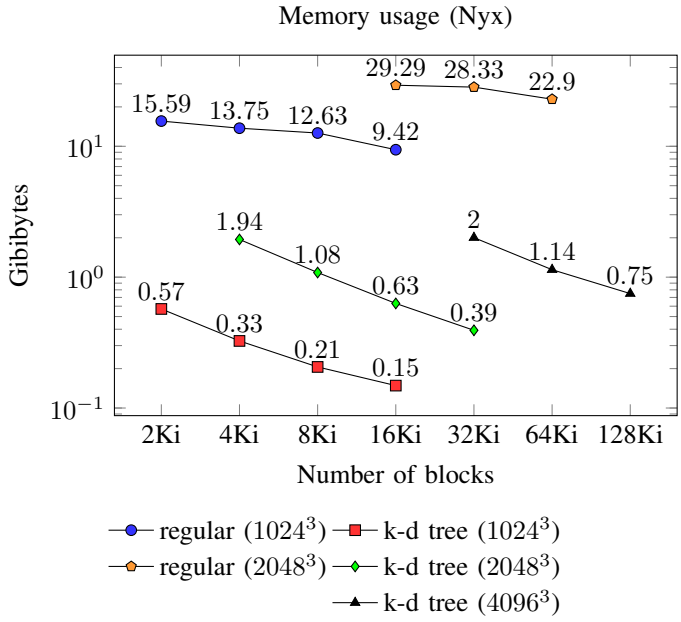

Fig. 10. Strong and weak scaling on a late snapshot of a Nyx simulation: overall memory usage.

B. HACC Data

Dark matter is thought to account for over 80% of the universe and is the basis for large-scale structure in the universe such as the formation of galaxies. The HACC code framework [35] simulates the nonlinear time evolution of the universe to high precision by using dark matter tracer particles.

HACC, or Hardware/Hybrid Accelerated Cosmology Code, is a simulation framework for computing cosmological models on various supercomputing architectures such as GPUs and many-core CPUs. It solves the Vlasov–Poisson equation to evolve the phase-space distribution function for particles in an expanding universe. The solution method is based on N-body

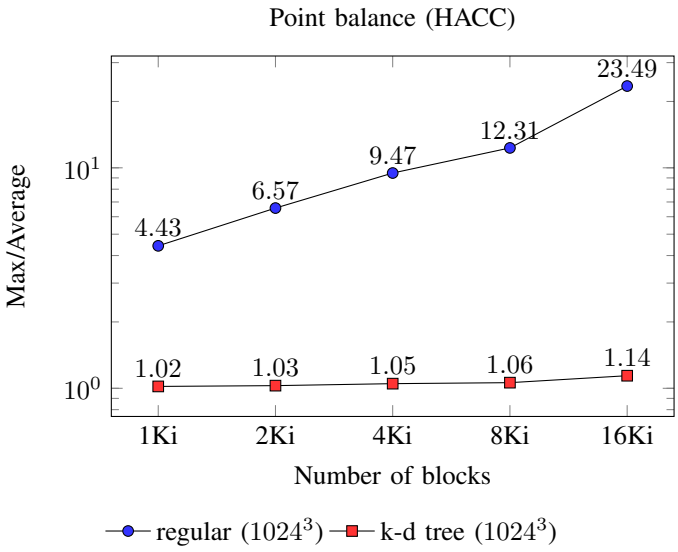

Fig. 11. Points balance for regular vs k-d tree domain decomposition of a 1024^3 points from a late snapshot of a HACC simulation.

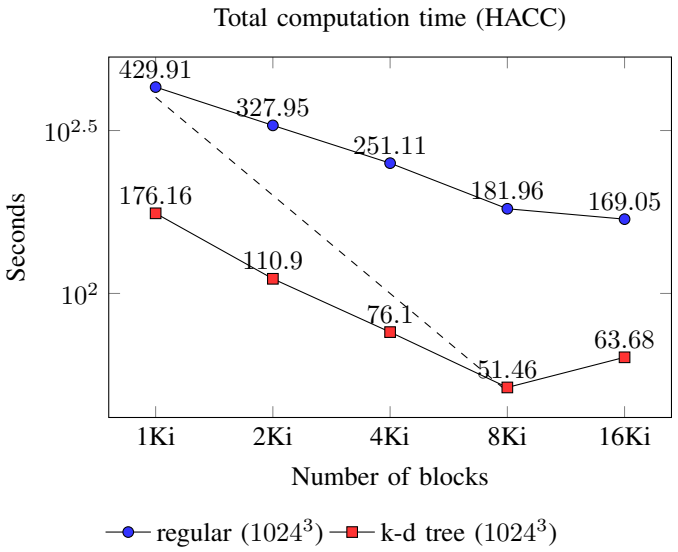

Fig. 12. Strong scaling on a late snapshot of a HACC simulation: total computation time includes point exchange and multi-round computation of the Delaunay tessellation.

techniques with three kinds of force solvers: direct particle summation, tree/multipole expansion, and spectral particle-mesh. HACC was one of the first scientific simulations to attain a sustained performance of greater than 10 PFlops on any architecture [36].

Large simulation sizes are required in order to compute even a fraction of the observable sky. Hundreds of billions to trillions of particles are required to track galaxy-scale mass distributions correctly [37]. The resulting spatial dynamic range of particle density is $10^6 : 1$. HACC can easily generate data sizes over 100 terabytes to several petabytes in a single run.

We used Mira to tessellate the particle positions from the final time step (step 499 at redshift $z = 0$) of a simulation run of 1024^3 particles conducted in 2014. The original simulation ran using 512 blocks. To perform a strong scaling study, we distributed the original particles into the desired number of blocks—from 1Ki to 16Ki—decomposed into either a regular or k-d tree decomposition.

Figure 11 illustrates the load balance, which as in the Nyx case is defined as the ratio of the maximum number of points in any block to the average, for the regular and k-d tree decompositions using the histogram median splitting method. The ratio varies from 4.4 at 1Ki blocks to 23.5 for 16Ki blocks for the regular decomposition, while the ratio remains nearly constant, between 1.02 and 1.14, for the k-d tree.

Figure 12 shows the overall computation time: both domain decomposition (which includes point distribution) and the multi-round Delaunay tessellation. As in the Nyx experiment, the better load balance is responsible both for a faster overall computation time (up to a factor of 3.5) and for better scaling. Between 1Ki and 8Ki blocks, the tessellation using the regular grid scales at 30% strong scaling efficiency while the efficiency using the k-d tree is 43%.

The upturn in the k-d tree curve from 8Ki to 16Ki blocks is because the number of rounds needed to complete the adaptive algorithm increased by one. From 1Ki through 8 Ki blocks, the k-d tree algorithm ran in three rounds, but four rounds were required for 16 Ki blocks. The regular decomposition tessellation completed in 2 rounds for all block counts.

To better understand why the improvement using the k-d tree over the regular decomposition with the HACC data was less than with the Nyx data, we break down the total time further. Figure 13 shows the decomposition time (building the regular or k-d tree decomposition and distributing the points into it). The k-d tree decomposition is nearly two times faster than the regular one. Compared with the Nyx experiment, the point balance in the HACC data (23 : 1) is closer than the point balance in the Nyx data (72 : 1), which explains why the improvement in decomposition time is smaller for the HACC data (k-d tree is 1.8 times faster than regular decomposition) than for Nyx (k-d tree is 3.5 times faster than regular decomposition). The scalability of the HACC decomposition is relatively flat for both the regular and k-d tree decompositions. Looking back at the Nyx decomposition in Figure 9, the curves are also roughly parallel, but more steeply inclined than the HACC data. Based on the absolute times, the HACC points were more expensive to redistribute on Mira, both into a regular and k-d tree decomposition, than the Nyx points on Edison. We plan to investigate further how much of this difference is caused by the data sets and how much is due to the different machine architectures.

Figure 14 shows the time to compute the Delaunay tessellation after the decomposition has been completed. Similar to Figure 12, the strong scaling efficiency for just the tessellation computation is 32% for the regular decomposition versus 54% for the k-d tree. As in Figure 12, the upturn in the k-d tree curve is caused by the extra round required at 16Ki blocks.

C. Data Balance

To understand the regime where we expect the regular decomposition to perform better than the k-d tree decompo-

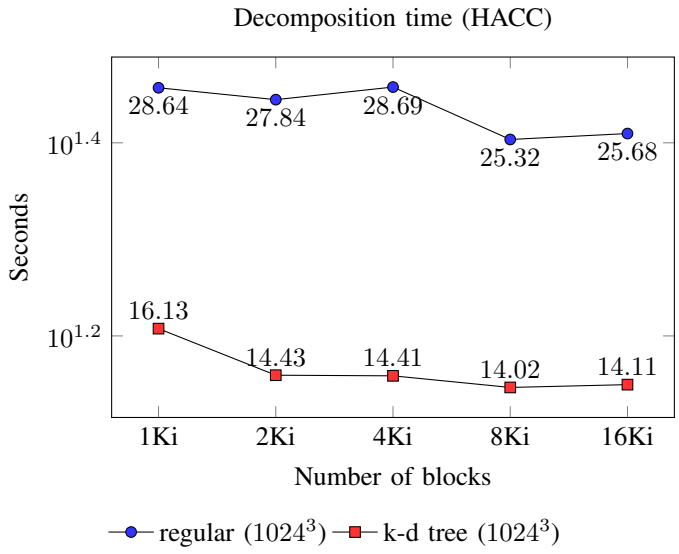

Fig. 13. Strong scaling on a late snapshot of a HACC simulation: point exchange time for k-d tree vs regular decomposition.

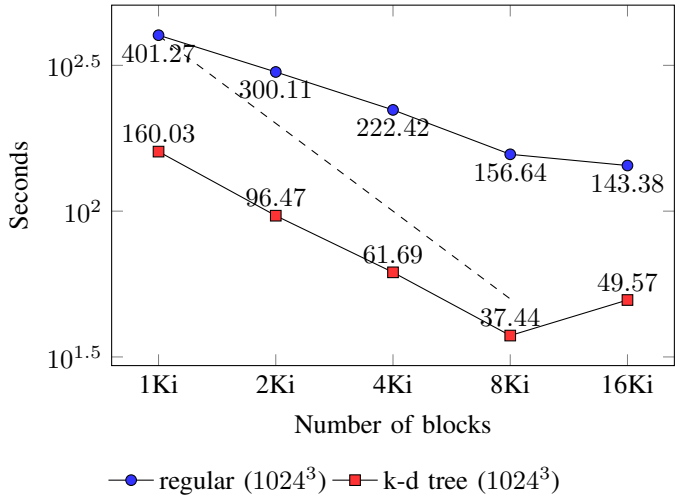

Fig. 14. Strong scaling on a late snapshot of a HACC simulation: Delaunay tessellation time for k-d tree vs regular decomposition.

sition, we compare the two algorithms on balanced data — specifically, the initial conditions of the Nyx simulation used in Section VI-A. Figure 15 shows scaling of both regular and k-d tree decomposition versions of the algorithm. Its message is clear: in the balanced case, the two algorithms scale equally well, but the overhead of the multiple rounds of the adaptive neighborhood algorithm makes it almost twice as slow.

We view this slowdown as a reasonable trade-off: even if the adaptive neighborhood algorithm is twice slower at the beginning of the simulation, it becomes 50 times faster towards the end (and, moreover, allows us to process snapshots whose Delaunay tessellations exhaust the memory of the regular decomposition).

To understand the relationship between data balance and

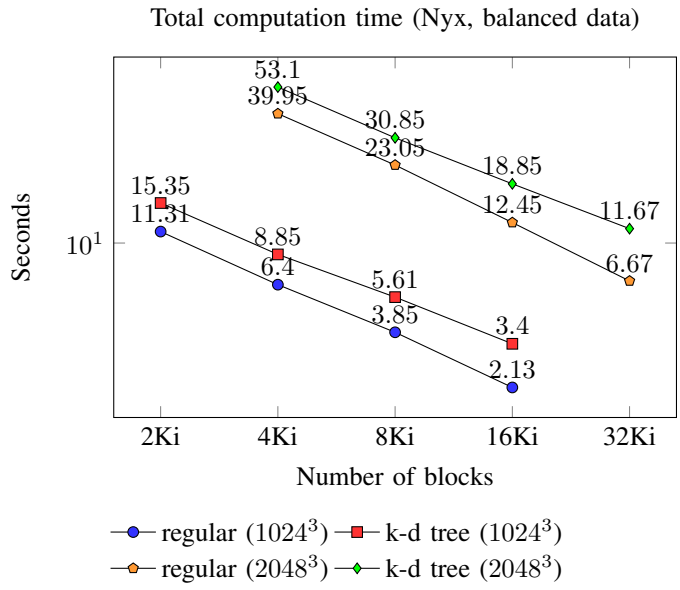

Fig. 15. Strong and weak scaling on initial conditions of a Nyx simulation: total computation time includes point exchange and multi-round computation of the Delaunay tessellation.

Data	Round 1	Round 2	Round 3	Round 4
IC	10.673	6.773	3.888	5.200
205	10.465	4.270	3.490	—
893	11.137	5.317	4.750	—
1839	11.604	7.582	6.798	—

TABLE I. INDIVIDUAL ROUND TIMES FOR 8Ki BLOCKS.

computation time better, we take roughly every 100-th time step of the Nyx simulation and plot its balance (with respect to the regular decomposition) versus the running times of the two algorithms. Figure 16 presents the results. Already by the 200-th time step the k-d tree version of the algorithm performs better. Even more significant is the further progress: the algorithm operating on the regularly decomposed data slows down linearly as a function of particle imbalance, while the adaptive algorithm balances the data using the k-d tree decomposition and requires roughly constant amount of time.

The speed-up of the algorithm operating on a k-d tree decomposition, apparent in Figure 16, is due to the decreasing number of rounds. Tables I and II give the individual round times. To process the initial conditions data (IC), the adaptive neighborhood algorithm takes four rounds. By the 205-th time step (the third data point on each curve in Figure 16), when the running time stabilizes, there are only three rounds. Their times remain stable until the last data point on the k-d tree curves in the figure (893-rd time step). The individual round times increase only slightly by the late stage snapshot (1839-th time step) used in Section VI-A.

Data	Round 1	Round 2	Round 3	Round 4
IC	5.362	4.237	2.218	3.031
205	5.194	2.504	1.937	—
893	5.564	3.115	2.766	—
1839	5.918	4.275	4.241	—

TABLE II. INDIVIDUAL ROUND TIMES FOR 16Ki BLOCKS.

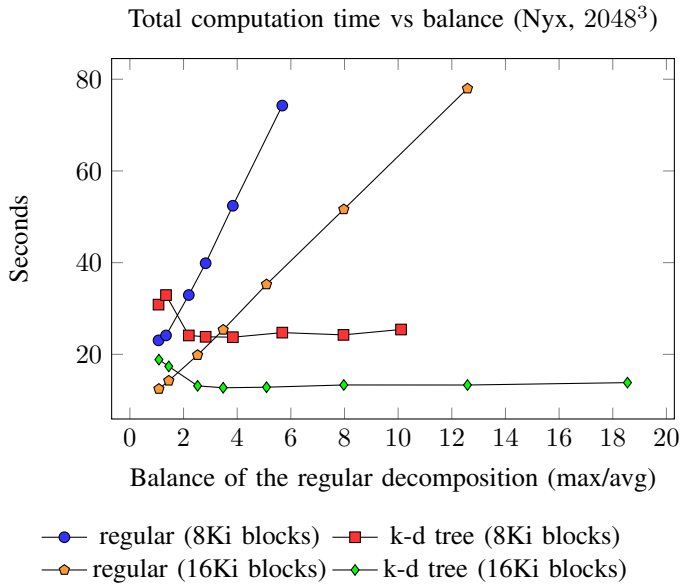

Fig. 16. Computation time on regular and k-d tree decompositions as functions of particle balance. The points missing from the figure for the regular decomposition, where the corresponding k-d tree points are present, indicate that the computation ran out of memory.

VII. Conclusion

We have presented a parallel algorithm for computing distributed Delaunay tessellations. The algorithm adapts to the data in two way: first, it sorts the points into a distributed k-d tree to balance their distribution between the processes; second, it grows the neighborhood of each block to guarantee that enough information is exchanged for the resulting distributed Delaunay tessellation to be correct. We have evaluated this algorithm on densely clustered data from late-stage cosmological simulations. Its performance and scalability are significantly better than that of the existing algorithms.

In future work, we intend to investigate the behavior of our algorithm on more types of data (including plasma physics data and CAD finite-element data, which exhibit qualitatively different distributions of points than cosmology). We also plan to try a different neighborhood expansion scheme: growing neighborhoods exponentially, rather than linearly, as done in this work. Finally, we intend to investigate how to couple our new algorithm with density estimation on a uniform grid—the challenge here stems from the need to redistribute information across different domain decompositions: k-d tree to a regular grid decomposition.

Acknowledgements

We would like to thank Zarija Lukić for providing the Nyx data, Salman Habib and Katrin Heitmann for the HACC data. We are grateful to Janine Bennett for shepherding this paper. This work was supported by Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contracts DE-AC02-06CH11357 and DE-AC02-05CH11231. Work is also supported by DOE with agreement No. DE-FC02-06ER25777.

REFERENCES

[1] W. Schaap and R. van de Weygaert, "Continuous Fields and Discrete Samples: Reconstruction Through Delaunay Tessellations," *Astronomy and Astrophysics*, vol. 363, pp. L29–L32, 2000.

[2] W. E. Schaap, *DTFE: The Delaunay Tesselation Field Estimator*, University of Groningen, The Netherlands, 2007, Ph.D. Dissertation.

[3] T. Peterka, H. Croubois, N. Li, E. Rangel, and F. Cappello, "Self-Adaptive Density Estimation of Particle Data," *SIAM Journal on Scientific Computing SISC Special Edition on CSE'15: Software and Big Data*, 2016.

[4] S. Shandarin, S. Habib, and K. Heitmann, "Cosmic Web, Multistream Flows, and Tessellations," *Physical Review D*, vol. 85, p. 083005, Apr 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevD.85.083005

[5] V. Springel, "Hydrodynamic Simulations on a Moving Voronoi Mesh," *ArXiv e-prints*, Sept. 2011.

[6] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[7] G. Di Fatta and D. Pettinger, "Dynamic Load Balancing in Parallel KD-Tree K-Means," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 2478–2485.

[8] C. Garth and K. I. Joy, "Fast, Memory-Efficient Cell Location in Unstructured Grids for Visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 6, pp. 1541–1550, 2010.

[9] N. Andrysco and X. Tricoche, "Matrix Trees," in *Computer Graphics Forum*, vol. 29, no. 3. Wiley Online Library, 2010, pp. 963–972.

[10] J. Barnes and P. Hut, "A Hierarchical O(N log N) Force-Calculation Algorithm," *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[11] J. Carrier, L. Greengard, and V. Rokhlin, "A Fast Adaptive Multipole Algorithm for Particle Simulations," *SIAM journal on scientific and statistical computing*, vol. 9, no. 4, pp. 669–686, 1988.

[12] M. Aly, M. Munich, and P. Perona, "Distributed Kd-Trees for Retrieval from Very Large Image Collections," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2011.

[13] C. Zhang, A. Krishnamurthy, and R. Y. Wang, "Brushwood: Distributed Trees in Peer-to-Peer Systems," in *Peer-to-Peer Systems IV*. Springer, 2005, pp. 47–57.

[14] M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukić, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey, "Bd-cats: Big data clustering at trillion particle scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 6:1–6:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807616

[15] A. Colbrook and C. Smythe, "Efficient Implementations of Search Trees on Parallel Distributed Memory Architectures," *Computers and Digital Techniques, IEE Proceedings E*, vol. 137, no. 5, pp. 394–400, 1990.

[16] A. Colbrook, E. A. Brewer, C. N. Dellarocas, and W. E. Weihl, "Algorithms for Search Trees on Message Passing Architectures," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 97–108, 1996.

[17] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'hallaron, "From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 91.

[18] D. B. Larkins, J. Dinan, S. Krishnamoorthy, S. Parthasarathy, A. Rountev, and P. Sadayappan, "Global Trees: A Framework for Linked Data Structures on Distributed Memory Parallel Systems," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 57.

[19] J. Nieplocha and B. Carpenter, "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems," in *Parallel and Distributed Processing*. Springer, 1999, pp. 533–546.

[20] J. Dinan, S. Krishnamoorthy, D. B. Larkins, J. Nieplocha, and P. Sadayappan, "Scioto: A Framework for Global-View Task Parallelism," in *Parallel Processing, 2008. ICPP'08. 37th International Conference on*. IEEE, 2008, pp. 586–593.

[21] K. L. Clarkson and P. W. Shor, "Applications of Random Sampling in Computational Geometry," *Discrete and Computational Geometry*, vol. 4, pp. 387–421, 1989.

[22] L. J. Guibas, D. E. Knuth, and M. Sharir, "Randomized Incremental Construction of Delaunay and Voronoi Diagrams," *Algorithmica*, vol. 7, pp. 381–413, 1992.

[23] H. Edelsbrunner and N. R. Shah, "Incremental Topological Flipping Works for Regular Triangulations," *Algorithmica*, vol. 15, pp. 223–241, 1996.

[24] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull Algorithm for Convex Hulls," *ACM Trans. Math. Softw.*, vol. 22, pp. 469–483, Dec. 1996. [Online]. Available: http://doi.acm.org/10.1145/235815.235821

[25] A. Fabri and S. Pion, "CGAL: the Computational Geometry Algorithms Library," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '09. New York, NY, USA: ACM, 2009, pp. 538–539. [Online]. Available: http://doi.acm.org/10.1145/1653771.1653865

[26] C. Rycroft, "Voro++: A Three-dimensional Voronoi Cell Library in C++," Tech. Rep., 2009, http://www.osti.gov/energycitations/servlets/purl/946741-A8FxbI/946741.pdf.

[27] R. Miller and Q. F. Stout, "Efficient Parallel Convex Hull Algorithms," *IEEE Trans. Comput.*, vol. 37, no. 12, pp. 1605–1618, Dec. 1988.

[28] C.-S. Jeong, "An Improved Parallel Algorithm for Constructing Voronoi Diagram on a Mesh-Connected Computer," *Parallel Computing*, vol. 17, no. 4, pp. 505–514, 1991.

[29] G. Blelloch, J. C. Hardwick, G. L. Miller, and D. Talmor, "Design and Implementation of a Practical Parallel Delaunay Algorithm," *ALGORITHMICA Special Issue on Coarse Grained Parallel Algorithms*, vol. 24, pp. 243–269, August 1999.

[30] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Khokhar, "A Randomized Parallel 3D Convex Hull Algorithm for Coarse Grained Multicomputers," in *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '95. New York, NY, USA: ACM, 1995, pp. 27–33. [Online]. Available: http://doi.acm.org/10.1145/215399.215410

[31] P. Cignoni, C. Montani, R. Perego, and R. Scopigno, "Parallel 3d Delaunay Triangulation," in *Computer Graphics Forum*, vol. 12, no. 3. Wiley Online Library, 1993, pp. 129–142.

[32] T. Peterka, D. Morozov, and C. Phillips, "High-Performance Computation of Distributed-Memory Parallel 3D Voronoi and Delaunay Tessellation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 997–1007.

[33] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel, "Nyx: A Massively Parallel AMR Code for Computational Cosmology," *The Astrophysical Journal*, vol. 765, pp. 39–52, 2013.

[34] Z. Lukić, C. W. Stark, P. Nugent, M. White, A. A. Meiksin, and A. Almgren, "The lyman α-forest in optically thin hydrodynamical simulations," *Monthly Notices of the Royal Astronomical Society*, vol. 446, no. 4, pp. 3697–3724, 2015. [Online]. Available: http://mnras.oxfordjournals.org/content/446/4/3697.abstract

[35] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Luki, S. Sehrish, and W. keng Liao, "HACC: Simulating Sky Surveys on State-of-the-Art Supercomputing Architectures," *New Astronomy*, vol. 42, pp. 49 – 65, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138410761500069X

[36] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, "HACC: Extreme Scaling and Performance Across Diverse Architectures," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 6:1–6:10.

[37] K. Heitmann, M. White, C. Wagner, S. Habib, and D. Higdon, "The Coyote Universe. I. Precision Determination of the Nonlinear Matter Power Spectrum," *Astrophysical Journal*, vol. 715, pp. 104–121, May 2010.