Geosci. Model Dev., 6, 1353–1365, 2013
www.geosci-model-dev.net/6/1353/2013/
doi:10.5194/gmd-6-1353-2013

Geoscientific Model Development
Open Access

Parallel algorithms for planar and spherical Delaunay construction with an application to centroidal Voronoi tessellations

D. W. Jacobsen[1,2], **M. Gunzburger**[1], **T. Ringler**[2], **J. Burkardt**[1], **and J. Peterson**[1]

[1]Department of Scientific Computing, Florida State University, Tallahassee, FL 32306, USA
[2]Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Correspondence to: D. W. Jacobsen (jacobsen.douglas@gmail.com)

Received: 5 February 2013 – Published in Geosci. Model Dev. Discuss.: 22 February 2013
Revised: 18 June 2013 – Accepted: 2 July 2013 – Published: 30 August 2013

Abstract. A new algorithm, featuring overlapping domain decompositions, for the parallel construction of Delaunay and Voronoi tessellations is developed. Overlapping allows for the seamless stitching of the partial pieces of the global Delaunay tessellations constructed by individual processors. The algorithm is then modified, by the addition of stereographic projections, to handle the parallel construction of spherical Delaunay and Voronoi tessellations. The algorithms are then embedded into algorithms for the parallel construction of planar and spherical centroidal Voronoi tessellations that require multiple constructions of Delaunay tessellations. This combination of overlapping domain decompositions with stereographic projections provides a unique algorithm for the construction of spherical meshes that can be used in climate simulations. Computational tests are used to demonstrate the efficiency and scalability of the algorithms for spherical Delaunay and centroidal Voronoi tessellations. Compared to serial versions of the algorithm and to STRIPACK-based approaches, the new parallel algorithm results in speedups for the construction of spherical centroidal Voronoi tessellations and spherical Delaunay triangulations.

1 Introduction

Voronoi diagrams and their dual Delaunay tessellations have become, in many settings, natural choices for spatial gridding due to their ability to handle arbitrary boundaries and refinement well. Such grids are used in a wide range of applications and can, in principle, be created for almost any geometry in two and higher dimensions. The recent trend towards the exascale in most aspects of high-performance computing further demands fast algorithms for the generation of high-resolution spatial meshes that are also of high quality, e.g. featuring variable resolution with smooth transition regions; otherwise, the meshing part can dominate the rest of the discretization and solution processes. However, creating such meshes can be time consuming, especially for high-quality, high-resolution meshing. Attempts to speed up the generation of Delaunay tessellations via parallel divide-and-conquer algorithms were made in, e.g. Cignoni et al. (1998); Chernikov and Chrisochoides (2008), however these approaches are restricted to two-dimensional planar surfaces. With the current need for high, variable-resolution grid generators in mind, we first develop a new algorithm that makes use of a novel approach to domain decomposition for the fast, parallel generation of Delaunay and Voronoi grids in Euclidean domains.

Centroidal Voronoi tessellations provide one approach for high-quality Delaunay and Voronoi grid generation (Du et al., 1999, 2003b, 2006b; Du and Gunzburger, 2002; Nguyen et al., 2008). The efficient construction of such grids involves an iterative process (Du et al., 1999) that calls for the determination of multiple Delaunay tessellations; we show how our new parallel Delaunay algorithm is especially useful in this context.

Climate modelling is a specific field which has recently begun adopting Voronoi tessellations as well as triangular meshes for the spatial discretization of partial differential equations (Pain et al., 2005; Weller et al., 2009; Ju et al., 2008, 2011; Ringler et al., 2011). As this is a special interest of ours, we also develop a new parallel algorithm for the generation of Voronoi and Delaunay tessellations for the entire sphere or some subregion of interest. The algorithm uses stereographic projections, similar to Lambrechts et al.

(2008), to transform these tasks into planar Delaunay constructions for which we apply the new parallel algorithm we have developed for that purpose. Spherical centroidal Voronoi tessellation (SCVT) based grids are especially desirable in climate modelling because they not only provide for precise grid refinement, but also feature smooth transition regions (Ringler et al., 2011, 2013). We show how such grids can be generated using the new parallel algorithm for spherical Delaunay tessellations.

The paper is organized in the following fashion. In Sect. 2.1, we present our new parallel algorithm for the construction of such tessellations. In Sect. 2.2, we provide a brief discussion of centroidal Voronoi tessellations (CVTs) followed by showing how the new parallel Delaunay tessellation algorithm can be incorporated into an efficient method for the construction of CVTs. In Sect. 3.1, we provide a short review of stereographic projections followed, in Sect. 3.2, by a presentation of the new parallel algorithm for the construction of spherical Delaunay and Voronoi tessellations. In Sect. 3.3, we consider the parallel construction of spherical CVTs. In Sect. 4, the results of numerical experiments and demonstrations of the new algorithms are given; for the sake of brevity, we only consider the algorithms for grid generation on the sphere. Finally, in Sect. 5, concluding remarks are provided.

2 Parallel Delaunay and Voronoi tessellation construction in $\mathbb{R}^k$

In this section, we present a new method for the construction of Delaunay and Voronoi tessellations. The algorithms are first presented in $\mathbb{R}^2$ and later extended to $\mathbb{R}^3$.

2.1 Parallel algorithm for Delaunay tessellation construction

The construction of planar Delaunay triangulations in parallel has been of interest for several years; see, e.g. Amato and Preparata (1993); Cignoni et al. (1998); Zhou et al. (2001); Batista et al. (2010). Typically, such algorithms divide the point set up into several smaller subsets, each of which can then be triangulated independently from the others. The resulting triangulations need to be stitched together to form a global triangulation. This stitching, or merge step, is typically computed serially because one may need to modify significant portions of the individual triangulations. The merge step is the main difference between the different parallel algorithms. Here, we provide a new alternative merge step that, because no modifications of the individual triangulations are needed, can be performed in parallel.

We are given a set of points $P = \{\boldsymbol{x}_j\}_{j=1}^n$ in a given domain $\Omega \subset \mathbb{R}^k$. We begin by covering Ω by a set $S = \{S_k(\boldsymbol{c}_k, r_k)\}_{k=1}^N$ of N overlapping spheres S_k, each of which is defined by a centre point $\boldsymbol{c}_k$ and radius r_k. For each sphere S_k, a connectivity or neighbour list is defined which consists of the indices of all the spheres $S_i \in S$, $i \neq k$, that overlap with S_k. From the given point set P, we then create N smaller point sets P_k, $k = 1, \ldots, N$, each of which consists of the points in P which are in the sphere S_k. Due to the overlap of the spheres S_k, a point in P may belong to multiple points sets P_k. This defines what we are referring to as our sorting method. The presented sort method is the simplest to understand, but provides issues with load balancing on variable resolution meshes. For that reason, we use an alternate sort method in practice which is described Sect. 2.1.1.

The next step is to construct the N Delaunay tessellations T_k, $k = 1, \ldots, N$, of the N point sets P_k, $k = 1, \ldots, N$. For this purpose, one can use any Delaunay tessellation method at one's disposal; for example, in the plane, one can use the Delaunay triangulator that is part of the Triangle software of Shewchuk (1996). A note should be made, that these triangulation software packages only aid in the computation of the triangulation. For example, they can not be used to determine which points should be triangulated. These Delaunay tessellations, of course, can be constructed completely in parallel after assigning one point set P_k to each of N processors.

At this point, we are almost but not quite ready to merge the N local Delaunay tessellations into a single global one. Before doing so, we deal with the fact that although, by construction, the tessellation T_k of the point set P_k is a Delaunay tessellation of P_k, there are very likely to be simplices in the tessellation T_k that may not be globally Delaunay, i.e. that may not be Delaunay with respect to points in P that are not in P_k. This follows because the points in any T_k are unaware of the triangles and points outside of S_k so that there may be points in P that are not in P_k that lie within the circumsphere of a simplex in T_k. In fact, only simplices whose circumspheres are completely contained inside of the ball S_k are guaranteed to be globally Delaunay; those points satisfy the criteria

$$||\boldsymbol{c}_k - \widehat{\boldsymbol{c}}_i|| + \widehat{r}_i < r_k, \tag{1}$$

where $\widehat{\boldsymbol{c}}_i$ and $\widehat{r}_i$ are the centre and radius of the circumsphere of the i-th triangle in the local Delaunay tessellation T_k.

So, at this point, for each $k = 1, \ldots, N$, we construct the triangulation $\widehat{T}_k$ by discarding all simplices in the local Delaunay tessellation T_k whose circumspheres are not completely contained in S_k, i.e. all simplices that do not satisfy Eq. (1). Figure 1 shows one of the local Delaunay triangulations T_k and the triangulation $\widehat{T}_k$ after the deletion of triangles that are not guaranteed to satisfy the Delaunay property globally.

The final step is to merge the N modified tessellations $\widehat{T}_k$, $k = 1, \ldots, N$, that have been constructed completely in parallel into a single global tessellation. The key observation here is that *each regional tessellation $\widehat{T}_k$ is now exactly a portion of the global Delaunay tessellation* because if two local tessellations $\widehat{T}_i$ and $\widehat{T}_k$ overlap, they must coincide wherever they overlap. This follows from the uniqueness property

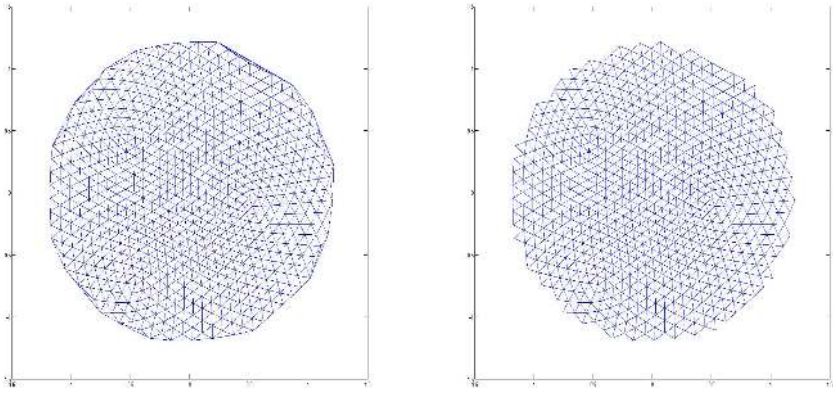


Fig. 1. A local Delaunay triangulation T_k (left) and the triangulation $\widehat{T}_k$ after the deletion of triangles that are not guaranteed to satisfy the Delaunay property globally.

of the Delaunay tessellations. Thus, *the union of the local Delaunay tessellations is the global Delaunay tessellation*; by using an overlapping domain decomposition, the stitching of the regional Delaunay tessellations into a global one is transparent. Of course, some bookkeeping chores have to be done such as rewriting simplex information in terms of global point indices and counting overlapping simplices only once.

In summary, the algorithm for the construction of a Delaunay tessellation in parallel consists of the following steps:

- define overlapping balls S_k, $k = 1, \ldots, N$, that cover the given region Ω;
- sort the given point set P into the subsets P_k, $k = 1, \ldots, N$, each containing points in P that are in S_k;
- for $k = 1, \ldots, N$, construct in parallel the N Delaunay tessellations T_k of the points sets P_k;
- for $k = 1, \ldots, N$, construct in parallel the tessellation $\widehat{T}_k$ by removing from T_k all simplices that are not guaranteed to satisfy the circumsphere property;
- construct the Delaunay tessellation of P as the union of the N modified Delaunay tessellations $\{\widehat{T}_k\}_{k=1}^N$.

Once a Delaunay tessellation is determined, it is an easy matter, at least for domains in $\mathbb{R}^2$ and $\mathbb{R}^3$, to construct the dual Voronoi tessellation; see, e.g. Okabe et al. (2000).

2.1.1 Methods for decomposing global point set

In Sect. 2.1, an initial method was presented for distributing points between regions. We refer to this method as a distance sort. The distance between a point and a region centre is computed, and if the distance is less than the region's radius the point is considered for triangulation. The key difficulty for this sort method is determining a radius which provides enough overlap, without including significantly more points than are required. This is an even greater challenge when considering variable resolution meshes and regions that straddle both fine and coarse regions.

In order to properly sort the points, the overlap of the regions need to be large enough to satisfy the following three requirements:

- all points contained in a region's Voronoi cell are not vertices of any non-Delaunay triangles;
- the resulting triangulation includes all Delaunay triangles for which owned points are vertices of;
- the point set defines all triangles whose circumcircles fall in the region's Voronoi cell.

If all three of these are satisfied, the point set can be used in any of the algorithms described in this paper.

Whereas we have yet to describe a method for the choice of region radius, let us first begin by describing the two extremes. First, consider a radius that encompasses the entire domain. This obviously provides enough overlap to guarantee the three conditions are satisfied. Second, consider a radius that is equivalent to the edges of the region's Voronoi cell, and no larger. In this case, the overlap is obviously insufficient to satisfy the three conditions. This can be seen if we assume at least one non-Delaunay triangle exists at the boundary of the Voronoi cell, in which case both criteria 1 and 2 are not met.

In practice, choosing the radii $\{r_k\}_{k=1}^N$ to be equal to the maximum distance from the centre $\boldsymbol{c}_k$ of its ball S_k to the centre of all its adjacent balls allows enough overlap in quasi-uniform cases. However, this method still fails to provide enough overlap in certain cases. All of the failures occur when the global point set does not include enough points for the decomposition of choice. Before discussing the target number of points for a given decomposition, an alternative sorting method is introduced.

In an attempt to provide better load balancing in general for arbitrary density functions, we also developed an

alternative sorting method. We refer to this sorting method as the Voronoi sort. In this case, the method for sorting points makes use of the Voronoi property that all points contained within a Voronoi cell are closer to that cell's generator than any other generator in the set. It is important to note that the coarse Voronoi cells used for domain decomposition here are not the same as Voronoi cells related to the Delaunay triangulation we are attempting to construct. To begin, the point set is divided into the Voronoi cells defined by the region's centre, and its neighbour's region centres. The union of the owned region's point set and its immediately adjacent region's point sets defines the final set of points for triangulation.

As with the distance sort, the Voronoi sort meets all of the criteria for use in the described algorithms. However, it can also still fail in certain cases. Both sort methods can fail if the target point set is too small for the target decomposition. In this case, one or more regions do not include enough points to correctly compute a triangulation through any method. In practice, these algorithms fail if the decomposition includes more than two regions N, and the point set contains less than $16N$ points, or two bisections. For example, a 12 processor decomposition cannot be used with a target of 42 points, but a 12 processor decomposition can be used with a target of 162 points.

In general, the decomposition should be created using the target density function to provide optimal load balancing, and to improve robustness of the algorithm. Results for these sort methods are presented in Sect. 4.2.2.

2.2 Application to the construction of centroidal Voronoi and Delaunay tessellations

Given a Voronoi tessellation $V = \{V_j\}_{j=1}^n$ of a bounded region $\Omega \subset \mathbb{R}^k$ corresponding to the set of generators $P = \{\boldsymbol{x}_j\}_{j=1}^n$ and given a nonnegative function $\rho(\boldsymbol{x})$ defined over Ω, referred to as the *point-density function*, we can define the *centre of mass* or *centroid* $\boldsymbol{x}_j^*$ of each Voronoi region as

$$\boldsymbol{x}_j^* = \frac{\int_{V_j} \boldsymbol{x}\rho(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}{\int_{V_j} \rho(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}, \qquad j = 1, \ldots, n. \tag{2}$$

In general, $\boldsymbol{x}_j \neq \boldsymbol{x}_j^*$, i.e. the generators of the Voronoi cells do not coincide with the centres of mass of those cells. The special case for which $\boldsymbol{x}_j = \boldsymbol{x}_j^*$ for all $j = 1, \ldots, n$, i.e. for which all the generators coincide with the centres of mass of their Voronoi regions, is referred to as a *centroidal Voronoi tessellation* (CVT).

Given Ω, n, and $\rho(\boldsymbol{x})$, a CVT of Ω must be constructed. The simplest means for doing so is Lloyd's method (Lloyd, 1982) in which, starting with an initial set of n distinct generators, one constructs the corresponding Voronoi tessellation, then determines the centroid of each of the Voronoi cells, and then moves each generator to the centroid of its Voronoi cell. These steps are repeated until satisfactory convergence is achieved, e.g. until the movement of generators falls below a prescribed tolerance. The convergence properties of Lloyd's method are rigorously studied in Du et al. (2006a).

The point-density function ρ plays a crucial role in how the converged generators are distributed and the relative sizes of the corresponding Voronoi cells. If we arbitrarily select two Voronoi cells V_i and V_j from a CVT, their grid spacing and density are related as

$$\frac{h_i}{h_j} \approx \left(\frac{\rho(\boldsymbol{x}_j)}{\rho(\boldsymbol{x}_i)}\right)^{\frac{1}{d+2}}, \tag{3}$$

where h_i denotes a measure of the linear dimension, e.g. the diameter, of the cell V_i and $\boldsymbol{x}_i$ denotes a point, e.g. the generator, in V_i. Thus, the point-density function can be used to produce nonuniform grids in either a passive or adaptive manner by prescribing a ρ or by connecting ρ to an error indicator, respectively. Although the relation Eq. (3) is at the present time a conjecture, its validity has been demonstrated through many numerical studies; see, e.g. Du et al. (1999); Ringler et al. (2011). CVTs and their dual Delaunay tessellations have been successfully used for the generation of high-quality nonuniform grids; see, e.g. Du and Gunzburger (2002); Du et al. (2003b, 2006b); Ju et al. (2011); Nguyen et al. (2008).

As defined above, every iteration of Lloyd's method requires the construction of the Voronoi tessellation of the current set of generators followed by the determination of the centroids of the Voronoi cells. However, the construction of the Voronoi tessellation can be avoided until after the iteration has converged and instead, the iterative process can be carried out with only Delaunay tessellation constructions. Thus, the first task within every iteration of Lloyd's method is to construct the Delaunay tessellation of the current set of generators. The parallel algorithm for the generation of Delaunay tessellations given in Sect. 2.1 is thus especially useful in reducing the costs of the multiple tessellations needed in CVT construction.

After the Delaunay tessellation of the current generators is computed, every Voronoi cell centre of mass must be computed by integration, so its generator can be replaced by the centre of mass. Superficially, it seems that we cannot avoid constructing the Voronoi tessellation to do this. However, it is easy to see that one does not actually need the Voronoi tessellation and instead one can determine the needed centroids from the Delaunay tessellation. For simplicity, we restrict this discussion to tessellations in $\mathbb{R}^2$. Each triangle in a Delaunay triangulation contributes to the integration over three different Voronoi cells. The triangle is split into three kites, each made up of two edge midpoints, the triangle circumcentre, and a vertex of the triangle. Each kite is part of the Voronoi cell whose generator is located at the triangle vertex associated with the kite. Integrating over each kite and updating a portion of the integrals in the centroid formula Eq. (2) allows one to only use the Delaunay triangulation to

determine the centroids of the Voronoi cells. Thus, because both steps within each Lloyd iteration can be performed using only the Delaunay triangulation, determining the generators of a CVT does not require construction of any Voronoi tessellations nor does it require any mesh connectivity information. If one is interested in the CVT, one need only construct a single Voronoi tessellation after the Lloyd iteration has converged; if one is interested in the Delaunay tessellation corresponding to the CVT, then no Voronoi construction is needed.

To make this algorithm parallel, one can compute the Voronoi cell centroids using the local Delaunay tessellations and not on the stitched-together global one. However, because the local Delaunay tessellations overlap, one has to ensure that each generator is only updated by one region, i.e. by only one processor. This can be done using one of a variety of domain decomposition methods. We use a coarse Voronoi diagram corresponding to the region centres $\boldsymbol{c}_k$. Each processor only updates the generators that are inside of its coarse Voronoi cell. Because Voronoi cells are non-overlapping, each generator will only get updated by one processor. After all the generators are updated, each coarse region needs to transfer its newly updated points only to its adjacent regions and not to all active processors. This limits each processor's communications to roughly six sends and receives, regardless of the total number of processors used.

We use two metrics to *check for the convergence* of the Lloyd iteration, namely the ℓ_2 and ℓ_∞ norms of generator movement given by

$$\ell_2\text{norm} = \left(\frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{x}_j^{\text{old}} - \boldsymbol{x}_j^{\text{new}})^2\right)^{1/2} \qquad \text{and}$$

$$\ell_\infty\text{norm} = \max_{j=1,\ldots,n}(|\boldsymbol{x}_j^{\text{old}} - \boldsymbol{x}_j^{\text{new}}|),$$

respectively, are compared with a given tolerance; here, old and new refer to the previous and current Lloyd iterates. If either norm falls below the tolerance, the iterative process is deemed to have converged. The ℓ_∞ norm is more strict, but both norms follow similar convergence paths when plotted against the iteration number.

A variety of point sets can be used to *initiate* Lloyd's method for CVT construction. The obvious one is Monte Carlo points (Metropolis and Ulam, 1949). These can either be sampled uniformly over Ω or sampled according to the point-density function $\rho(\boldsymbol{x})$. The latter approach usually reduces the number of iterations required for convergence. One can instead use a bisection method to build fine grids from a coarse grid (Heikes and Randall, 1995). To create a bisection grid, a coarse CVT is constructed using as few points as possible. After this coarse grid is converged, in $\mathbb{R}^2$, one would add the midpoint of every Voronoi cell edge or Delaunay triangle edge to the set of points. This causes the overall grid spacing to be reduced by roughly a factor of two in every cell so that the refined point set is roughly four times larger.

In summary, the algorithm for the construction of a CVT in parallel consists of the following steps; the steps in Roman font are the same as in the algorithm of Sect. 2.1 whereas the italicized steps, as well as the do loop, are additions for CVT construction:

- define overlapping balls that cover the given region;
- **while** not converged **do**
  - sort the current point set;
  - construct in parallel the local Delaunay tessellations;
  - remove from the Delaunay tessellations simplices which are not guaranteed to satisfy the circumsphere property;
  - *in parallel, determine the centroids of the Voronoi cells by integrating over simplices*;
  - *move each generator to the corresponding cell centroid*;
  - *test the convergence criteria*;
  - *communicate new generator positions to neighbouring balls*;
- **end**
- construct the Delaunay tessellation corresponding to the CVT as the union of the modified local Delaunay tessellations.

3 Parallel Delaunay and Voronoi tessellation construction on the sphere

Replacing the planar constructs of Delaunay triangulations and Voronoi tessellations with their analogous components on the sphere, i.e. on the surface of a ball in $\mathbb{R}^3$, creates the spherical versions of Delaunay triangulations and Voronoi tessellations. The Euclidean distance metric is replaced by the geodesic distance, i.e. the shortest of the two lengths along the great circle joining two points on the sphere. Triangles and Voronoi cells are replaced with spherical triangles and spherical Voronoi cells whose boundaries are geodesic arcs. To develop a parallel Delaunay and Voronoi tessellation construction algorithm on the sphere, we could try to parallelize the STRIPACK algorithm of Renka (1997). Here, however, we take a new approach that is based on a combination of stereographic projections (which we briefly discuss in Sect. 3.1) and the algorithm of Sect. 2.1 applied to planar domains. The serial version of the new Delaunay tessellation construction algorithm is therefore novel as well.

3.1 Stereographic projections

Stereographic projections are special mappings between the surface of a sphere and a plane tangent to the sphere. Not only are stereographic projections conformal mappings, meaning that angles are preserved, but they also preserve circularity, meaning that circles on the sphere are mapped to circles on the plane. Stereographic projections also map the interior of these circles to the interior of the mapped circles (Bowers et al., 1998; Saalfeld, 1999). For our purposes, the importance of circularity preservation is that it implies that stereographic projections preserve the Delaunay circumcircle property. This follows because triangle circumcircles (along with their interiors) are preserved. Therefore, Delaunay triangulations on the sphere are mapped to Delaunay triangulations on the plane and conversely. As a result, stereographic projections can be used to construct a Delaunay triangulation of a portion of the sphere by first constructing a Delaunay triangulation in the plane, which is a simpler and well-studied task.

Without loss of generality, we assume that we are given the unit sphere in $\mathbb{R}^3$ centred at the origin. We are also given a plane tangent to the sphere at the point $\boldsymbol{t}$. The focus point $\boldsymbol{f}$ is defined as the reflection of $\boldsymbol{t}$ about the centre of the sphere, i.e. $\boldsymbol{f}$ is the antipode of $\boldsymbol{t}$. Let $\boldsymbol{p}$ denote a point on the unit sphere. The stereographic projection of $\boldsymbol{p}$ onto the plane tangent at $\boldsymbol{t}$ is the point $\boldsymbol{q}$ on the plane defined by

$$\boldsymbol{q} = s\boldsymbol{p} + (1-s)\boldsymbol{f}, \qquad \text{where} \qquad s = 2\frac{1}{\boldsymbol{f}\cdot(\boldsymbol{f}-\boldsymbol{p})}. \quad (4)$$

For our purposes, it is more convenient to define the projection relative to $\boldsymbol{t}$ rather than $\boldsymbol{f}$. The simple substitution of $\boldsymbol{f} = -\boldsymbol{t}$ into Eq. (4) results in

$$\boldsymbol{q} = s\boldsymbol{p} + (s-1)\boldsymbol{t}, \qquad \text{where} \qquad s = 2\frac{1}{\boldsymbol{t}\cdot(\boldsymbol{p}+\boldsymbol{t})}. \quad (5)$$

The definitions Eqs. (4) and (5) can also be used to define stereographic projections in $\mathbb{R}^k$ for $k > 3$.

3.2 Parallel algorithm for spherical Delaunay triangulation construction

A spherical Delaunay triangulation (SDT) is the Delaunay triangulation of a point set P defined on the surface of the sphere. We adapt the algorithm developed in Sect. 2.1 for domains in $\mathbb{R}^k$ to develop a parallel algorithm for the construction of SDTs. The most important adaptation is to use stereographic projections so that the actual construction of Delaunay triangulations is done on planar domains.

We are now given a set of points $P = \{\boldsymbol{x}_j\}_{j=1}^n$ on a subset Ω of the sphere; Ω may be the whole sphere. We begin by covering Ω by a set $U = \{U_k(\boldsymbol{t}_k, r_k)\}_{k=1}^N$ of N overlapping "umbrellas" or spherical caps U_k, each of which is defined by a centre point $\boldsymbol{t}_k$ and geodesic radius r_k. See Fig. 2a. For each spherical cap U_k, a connectivity (or neighbours) list is defined that consists of the indices of all the spherical caps $U_i \in U, i \neq k$, that overlap with U_k. From the given point set P, we then create N smaller point sets $P_k, k = 1, \ldots, N$, each of which consists of the points in P which are in the spherical cap U_k. Due to the overlap of the spherical caps U_k, a point in P may end up belonging to multiple points sets P_k.

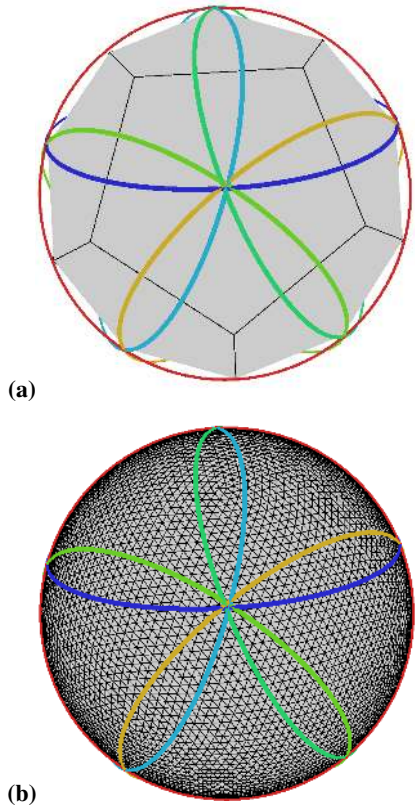


Fig. 2. (a) An overlapping subdivision of the sphere into $N = 12$ spherical caps. The cap centres $\boldsymbol{t}_k$ are the projections onto the sphere of the centroids of the 12 pentagons of the inscribed regular icosahedron. Each coloured ring represents the edge of cap of radius r_k. **(b)** A 10 242 generator Delaunay triangulation by the parallel algorithm.

At this point in Sect. 2.1, we assigned each of the point subsets P_k to a processor and constructed N Delaunay tessellations in parallel. Before doing so in the spherical case, we project each of the point sets P_k onto the plane tangent at the corresponding point $\boldsymbol{t}_k$. This additional step allows each processor to construct a planar Delaunay triangulation instead of spherical one. Specifically, for each k, we construct the planar point set $\widetilde{P}_k = \mathcal{S}(P_k; \boldsymbol{t}_k)$, where $\mathcal{S}(P_k; \boldsymbol{t}_k)$ denotes the stereographic projection of the points in P_k onto the plane tangent to the sphere at the point $\boldsymbol{t}_k$. We then assign each of the point sets $\widetilde{P}_k$ to a different processor and have each processor construct the planar Delaunay triangulation of its point set $\widetilde{P}_k$. Because stereographic projections preserve circularity and therefore preserve the Delaunay circumcircle property, it is then just a simple matter of keeping

track of triangle vertex and edge indices to define the spherical Delaunay triangulations of the point sets P_k.

As in Sect. 2.1, we now proceed to remove spherical triangles that are not guaranteed to be Delaunay with respect to the global point set P. Instead of Eq. (1), in the spherical case, triangles must satisfy

$$\cos^{-1}||\boldsymbol{t}_k - \widehat{\boldsymbol{c}}_i|| + \widehat{r}_i < r_k,$$

for that guarantee to be in effect, where $\widehat{\boldsymbol{t}}_i$ and $\widehat{r}_i$ are the centre and radius, respectively, of the circumsphere of the i-th triangle in the local Delaunay tessellation of P_k. Once the possibly unwanted triangles are removed, we can, as in Sect. 2.1, transparently stitch together the modified Delaunay triangulations into a global one.

In summary, the algorithm for the construction of a spherical Delaunay tessellation in parallel consists of the following steps, where the italicized steps are the ones added to the algorithm of Sect. 2.1:

- define overlapping spherical caps U_k, $k = 1, \ldots, N$, that cover the given region Ω on the sphere;
- sort the given point set P into the subsets P_k, $k = 1, \ldots, N$, each containing the points in P that are in U_k;
- *for $k = 1, \ldots, N$, construct in parallel the point set $\widetilde{P}_k$ by stereographically projecting the points in P_k onto the plane tangent to the sphere at the point $\boldsymbol{t}_k$*;
- for $k = 1, \ldots, N$, construct in parallel the planar Delaunay triangulation $\widetilde{T}_k$ of the points set $\widetilde{P}_k$;
- *for $k = 1, \ldots, N$, construct in parallel the spherical Delaunay triangulation T_k by mapping the planar Delaunay triangulation $\widetilde{T}_k$ onto the sphere*;
- for $k = 1, \ldots, N$, construct in parallel the spherical triangulation $\widehat{T}_k$ by removing from T_k all simplices that are not guaranteed to satisfy the circumsphere property;
- construct the Delaunay tessellation of P as the union of the N modified spherical Delaunay tessellations $\{\widehat{T}_k\}_{k=1}^N$.

For an illustration of a spherical Delaunay triangulation determined by this algorithm see Fig. 2b.

Because of the singularity in Eqs. (4) or (5) for $\boldsymbol{p} = \boldsymbol{f}$, the serial version of this algorithm, i.e. if $N = 1$, can run into trouble whenever the antipode $\boldsymbol{f}$ of the tangency point $\boldsymbol{t}$ is in the spherical domain Ω. Of course, this is always the case if Ω is the whole sphere. In such cases, the number of subdomains used cannot be less than two, even if one is running the above algorithm in serial mode.

In setting the extent of overlap, i.e. the radii of the spherical caps, as the maximum geodesic distance from the a cap centre $\boldsymbol{t}_j$ to neighbouring cap centres $\boldsymbol{t}_i$, the geodesic distance is given by $\cos^{-1}(\boldsymbol{t}_j \cdot \boldsymbol{t}_i)$.

Table 1. Description of generator count labels, along with approximate grid resolution on an earth sized sphere.

Generator Label	Generator Count	Approx. Resolution
coarse	40 962	120 km
medium	163 842	60 km
fine	2 621 442	15 km

3.3 Application to the construction of spherical centroidal Voronoi tessellations (SCVTs)

The parallel CVT construction algorithm of Sect. 2.2 is easily adapted for the construction of centroidal Voronoi tessellations on the sphere (SCVTs). Obviously, one uses the spherical version of the Delaunay triangulation algorithm as given in Sect. 3.2 instead of the version of Sect. 2.1. One also has to deal with the fact that if one computes the centroid of a spherical Voronoi cell using Eq. (2), then that centroid does not lie on the sphere; a different definition of a centroid has to be used (Du et al., 2003a). Fortunately, it is shown in Du et al. (2003a) that the correct centroid can be determined by using Eq. (2), which yields a point inside the sphere, and then projecting that point onto the sphere (actually, this was shown to be true for general surfaces) so that the correct spherical centroid is simply the intersection of the radial line going through the point determined by Eq. (2) and the sphere.

4 Results

All results are created using a high performance computing (HPC) cluster with 24 AMD Opteron "model 6176" cores per node and 64 GB of RAM per node.

Results are presented for three different generator counts, and are presented with the labels "coarse", "medium", and "fine". Table 1 can be used to refer these labels to actual counts as well as approximate resolution on an earth sized sphere.

We use STRIPACK (Renka, 1997), a serial Fortran 77 Association of Computing Machinery (ACM) Transactions on Mathematical Software (TOMS) algorithm that constructs Delaunay triangulations on a sphere, as a baseline for comparison with our approach. It is currently one of the few well-known spherical triangulation libraries available.

The algorithm described in this paper has been implemented as a software packaged referred to as MPI-SCVT, and is freely available following Jacobsen and Womeldorff (2013). It utilizes the Triangle software (Shewchuk, 1996) to perform planar triangulations of specified point sets in the tangent planes of each partition of the domain.

4.1 Delaunay triangulations of the full sphere

Table 2 compares MPI-SCVT with STRIPACK for the construction of spherical Delaunay triangulations. The results given compare the cost to compute a single triangulation of a medium resolution global grid. The results show, for the computation of a full spherical triangulation MPI-SCVT is roughly a factor of 7 slower than STRIPACK in serial mode, and a factor of 12 faster when using 96 processors.

4.2 Initial generator placement and sorting heuristics for SCVT construction

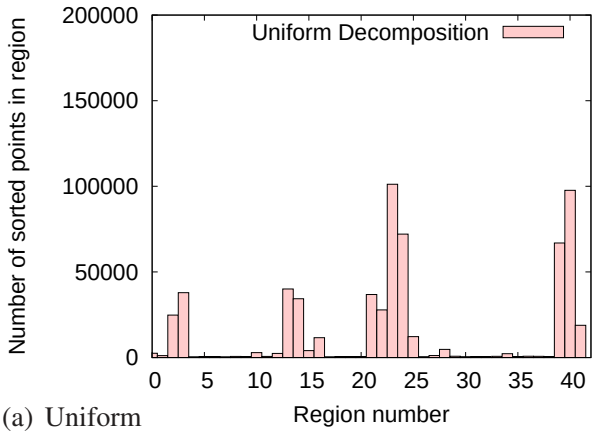
We now examine the effects of initial generator placement and sorting heuristics with regards to SCVT generation. The results of this section are intended to guide decisions about which initial condition or sorting method we use later to generate SCVTs.

4.2.1 Initial generator placement

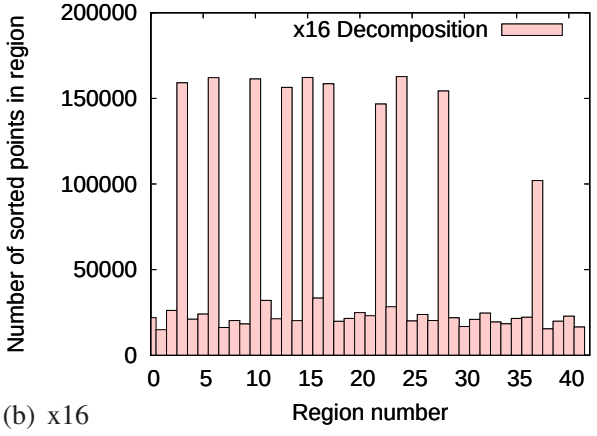
Because most climate models are shifting towards global high-resolution simulations, we begin by exploring the creation of a fine resolution grid. We compare SCVT grids constructing starting with uniform Monte Carlo and bisection initial generator placement. The times for these grids to converge to a tolerance 10^{-6} in the ℓ_2 norm are presented. The 10^{-6} threshold is the strictest convergence level that the Monte Carlo grid can attain in a reasonable amount of time, and is therefore chosen as the convergence threshold for this study. However, the bisection grid can converge well beyond this threshold in a similar amount of time. Table 3 shows timing results for the parallel algorithm for the two different options for initial generator placement. Although the time spent converging a mesh with Monte Carlo initial placements is highly dependent on the initial point set, it is clear from this table that bisection initial conditions provide a significant speedup in the overall cost to generate an SCVT grid. Based on the results presented in Table 3 and unless otherwise specified, only bisection initial generator placements are used in subsequent experiments. Bisection initial conditions are not specific to this algorithm, and can be used to accelerate any SCVT grid generation method.

4.2.2 Sorting of points for assignment to processors

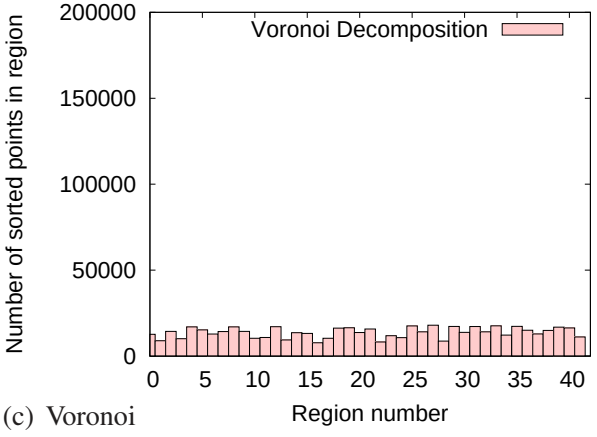
As described in Sect. 2.1.1, the MPI-SCVT code utilizes two methods for determining a region's point set. Whereas the dot product sort method is faster, it provides poor load balancing in variable resolution meshes, thus causing the iteration cost to be greater. Timings using both approaches are given in Table 4. Figure 3 shows the number of points that each processor has to triangulate on a per iteration basis. Timings presented are for a medium resolution grid, 42 regions, 42 processors, and are averages over 3000 iterations. Three sets of timings are given. Timings labeled Uniform and $\times 8$ both utilize the distance sort, but differ in the decomposition used. Uniform uses a quasi-uniform 42 region decomposition, whereas $\times 8$ uses a variable resolution decomposition with sixteen to one ration in maximum and minimum grid sizes. Timings labeled Voronoi use the same decomposition as $\times 8$, but use the Voronoi sort method rather than the distance sort method.



(a) Uniform



(b) x16



(c) Voronoi

Fig. 3. Number of points each processor has to triangulate. **(a)**, **(b)**, and **(c)** use the sorting approaches corresponding to the three rows of Table 4. All plots were created with the same medium resolution grid.

Table 2. Comparison of STRIPACK with serial and parallel versions of MPI-SCVT for computation of a full spherical triangulation. Computed using the same initial conditions for a medium resolution grid.

Algorithm	Processors	Regions	Time (ms)	Speedup (STRIPACK/MPI-SCVT)
STRIPACK	1	1	563.87	Baseline
MPI-SCVT	1	2	3 724.08	0.15
MPI-SCVT	2	2	2 206.06	0.26
MPI-SCVT	96	96	45.75	12.33

Table 3. Timing results for MPI-SCVT with bisection and Monte Carlo initial generator placements and the speedup of bisection relative to Monte Carlo. Final mesh is fine resolution.

Timed Portion	Bisection (B)	Monte Carlo (MC)	Speedup $\frac{\text{MC}}{\text{B}}$
Total Time (ms)	112 890	358 003 000	3171.25
Triangulation Time (ms)	10 887	48 034 600	4412.11
Integration Time (ms)	30 893	66 374 400	2148.52
Communication Time (ms)	70 878	110 958 000	1565.47

Table 4. Timings based on sorting approach used. Uniform uses a coarse quasi-uniform SCVT to define region centres and their associated radii and sorts using maximum distance between centres and neighbouring centres. ×8 uses a coarse SCVT with a 8 to 1 ratio in grid sizes to effect the same type of sorting. Voronoi uses the same ×8 coarse SCVT grid to define regions centres and sorts using the neighbouring Voronoi cell-based sort.

Sorting Approach	Decomposition Grid	Costs Of Different Algorithm Steps			Cost Per Iteration	Speedup
		Triangulation	Integration	Communication		
Distance	Uniform	14.53	37.62	1314.22	1396.53	Baseline
Distance	×8	35.84	92.91	865.32	995.04	1.40
Voronoi	×8	16.71	86.24	177.55	305.85	4.56

Timings presented in Table 4 are taken relative to processor number 0, and as can be seen in Fig. 3a, processor 0 has a very small load which causes the majority of its iteration time being spent waiting for the processors with large loads to catch up; this idling time is included in the Communication column of the table.

Table 4 and Fig. 3 show there is a significant advantage to the Voronoi based sort method in that it not only speeds up the overall cost per iteration, but also provides more balanced loads across the processors.

4.3 SCVT generation

We now provide both quasi-uniform and variable resolution SCVT generation results. The major contributor to the differences in computational performance arises as a result of load balancing differences. Results in this section make use of the density function

$$\rho(\boldsymbol{x}_i) = \frac{1}{2(1-\gamma)}\left[\tanh\left(\frac{\beta - |\boldsymbol{x}_c - \boldsymbol{x}_i|}{\alpha}\right) + 1\right] + \gamma \qquad (6)$$

which is visualized in Fig. 4, where $\boldsymbol{x}_i$ is constrained to lie on the surface of the unit sphere. This function results in relatively large value of ρ within a distance β of the point $\boldsymbol{x}_c$, where β is measured in radians and $\boldsymbol{x}_c$ is also constrained to lie on the surface of the sphere. The function transitions to relatively small values of ρ across a radian distance of α. The distance between $\boldsymbol{x}_c$ and $\boldsymbol{x}_i$ is computed as $|\boldsymbol{x}_c - \boldsymbol{x}_i| = \cos^{-1}(\boldsymbol{x}_c \cdot \boldsymbol{x}_i)$ with a range from 0 to π.

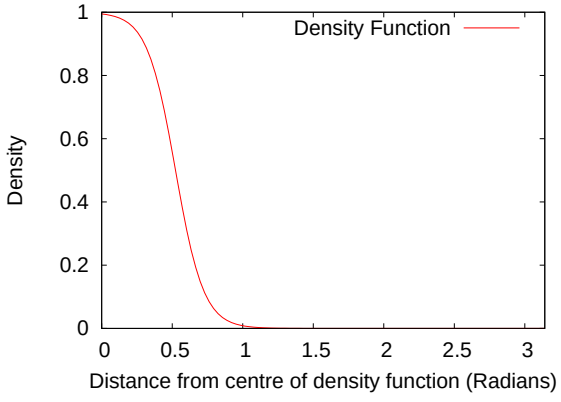
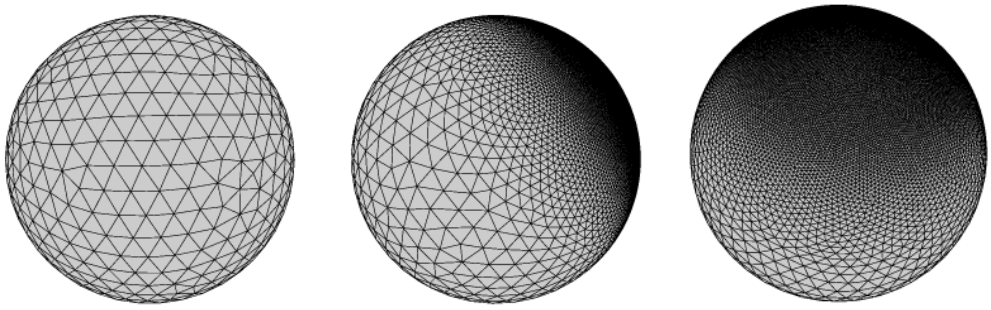


Fig. 4. Density function that creates a grid with resolutions that differ by a factor of 8 between the coarse and the fine regions. The maximum value of the density function is 1 whereas the minimum value is $(1/8)^4$.

Figure 5 shows an example grid created using this density function, with $\boldsymbol{x}_c$ set to be $\lambda_c = 3\pi/2$, $\phi_c = \pi/6$, where λ denotes longitude and ϕ latitude, $\gamma = (1/8)^4$, $\beta = \pi/6$, and



(a) Coarse region (b) Transition region (c) Fine region

Fig. 5. Different views of the same variable resolution grid created using the density function Eq. (6). **(a)** shows the coarse region of the grid, **(b)** shows the transition region, and **(c)** shows the fine region.

Table 5. Comparisons of SCVT generators using STRIPACK, serial MPI-SCVT, and parallel MPI-SCVT. Cost per triangulation and iteration are presented. Speedup is compared using the cost per iteration. Computed using the same initial conditions for a medium resolution grid. Variable resolution results are presented for MPI-SCVT only.

Algorithm	Procs	Regions	Triangulation Time (ms)	Iteration Time (ms)	Speedup Per Iteration (STRIPACK/MPI-SCVT)
STRIPACK	1	1	563.871	3 009.74	Baseline
MPI-SCVT $\times 1$	1	2	3 480.43	9 485.78	0.32
MPI-SCVT $\times 1$	2	2	2 114.38	5 390.82	0.56
MPI-SCVT $\times 1$	96	96	14.23	207.26	14.52
MPI-SCVT $\times 8$	96	96	16.71	305.855	9.84

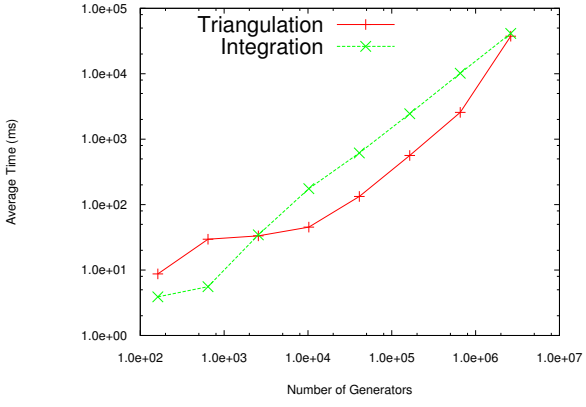


Fig. 6. Timings for STRIPACK-based SCVT construction for various generator counts. Red solid lines represent the time spent in STRIPACK computing a triangulation whereas green dashed lines represent the time spent integrating the Voronoi cells outside of STRIPACK in one iteration of Lloyd's algorithm.

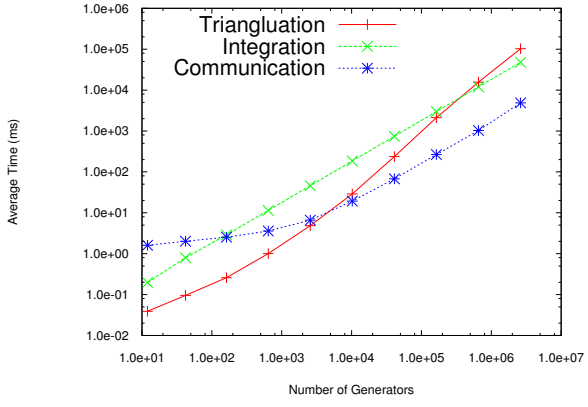


Fig. 7. Timings for various portions of MPI-SCVT using 2 processors and 2 regions. As the problem size increases the slope for both triangulation (red-solid) and integration (green-dashed) remain roughly constant.

$\alpha = 0.20$ with 10 242 generators. This set of parameters used in Eq. (6) is referred to as $\times 8$. The quasi-uniform version is referred to as $\times 1$.

In Sect. 4.1, we showed that MPI-SCVT performs comparably to STRIPACK when computing a single full triangulation. However, computing a full triangulation is only part of the story. For SCVT generation, a triangulation needs to be computed at every iteration of Lloyd's algorithm as described in Sect. 2.2. When using STRIPACK, the full triangulation needs to be computed at every iteration, but with MPI-SCVT only each regional triangulation needs to be computed at each iteration. This means the merge step can be skipped resulting in significantly cheaper triangulations.

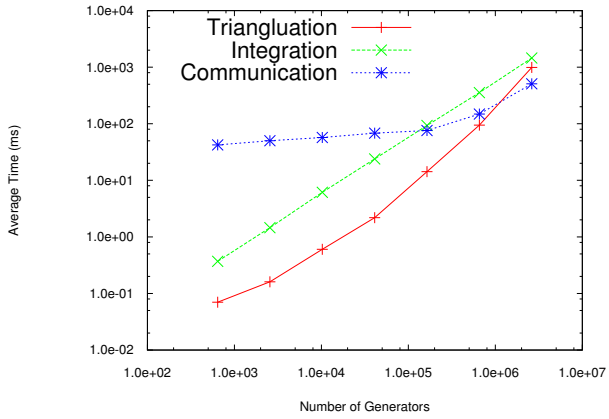


Fig. 8. Same information as in Fig. 7 but for 96 processors and 96 regions.

Figure 6 shows the performance of a STRIPACK-based SCVT construction as the number of generators is increased through bisection as mentioned in Sect. 2.2. Values are averages over 2000 iterations. The green dashed line represents the portion of the code that computes the centroids of the Voronoi regions whereas the red solid line represent the portion of the code that computes the Delaunay triangulation.

Table 5 compares STRIPACK with the triangulation routine in MPI-SCVT that is called on every iteration. The results presented relative to MPI-SCVT are averages over 2000 iterations.

As a comparison with Fig. 6, in Figs. 7 and 8 we present timings made for MPI-SCVT for two and 96 regions and processors, respectively, as the problem size, i.e. the number of generators, increases. A minimum of two processors are used because the stereographic projection used in MPI-SCVT has a singularity at the focus point.

Whereas Fig. 7 shows performance similar to that of STRIPACK (see Fig. 6), Fig. 8 shows roughly two orders of magnitude faster performance relative to STRIPACK. As mentioned previously, this is only the case when creating SCVTs as the full triangulation is no longer required when computing a SCVT in parallel.

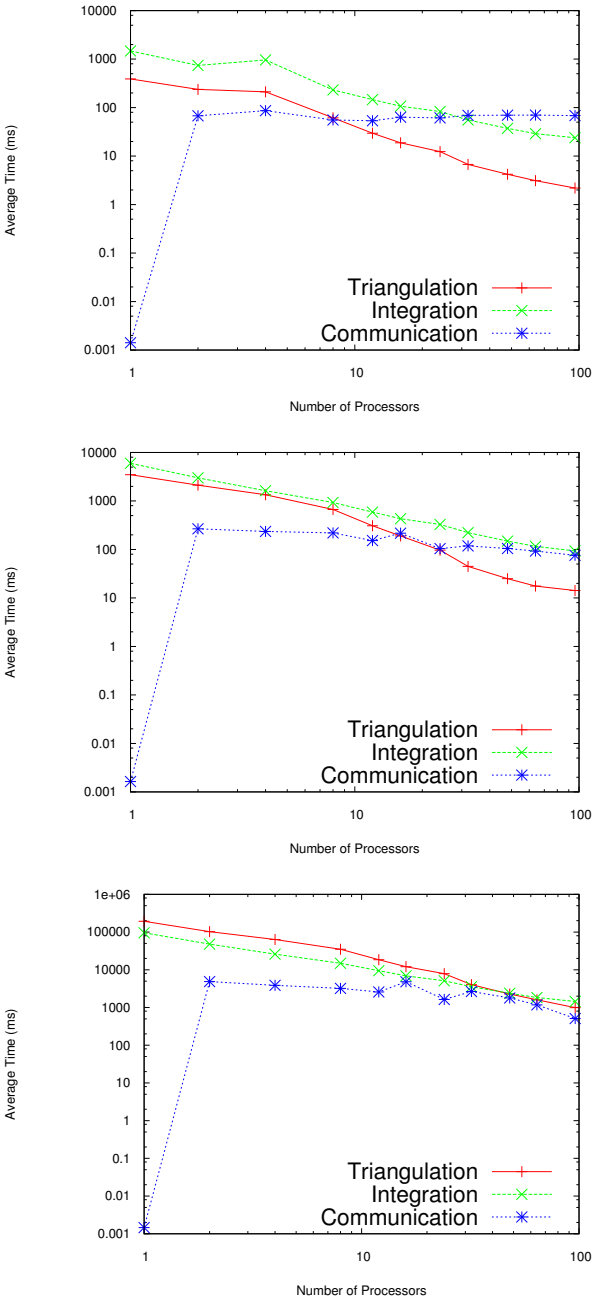


Fig. 9. Timing results for MPI-SCVT vs. number of processors for three different problem sizes. Red solid-lines represent the cost of computing a triangulation, whereas green-dashed lines represent the cost of integrating all Voronoi cells, and blue-dotted lines represent the cost of communicating each region's updated point set to its neighbours.

4.4 General algorithm performance

This section is intended to showcase some general performance results of MPI-SCVT. Figure 9a–c show the timings for a coarse resolution grid, a medium resolution grid, and a fine resolution grid, respectively.

To assess the overall performance of the MPI-SCVT algorithm, scalability results are presented in Fig. 10. Figure 10a shows that this algorithm can easily under-saturate processors; when this happens, communication ends up dominating the overall runtime for the algorithm which is seen in Fig. 9a; as a result, scalability ends up being sub-linear. As the number of generators increases (as seen in Fig. 10b, c), the limit for being under-saturated is higher. Currently in the algorithm, communications are done asynchronously using non-blocking sends and receives. Also, overall communications are reduced by only communicating with a region's

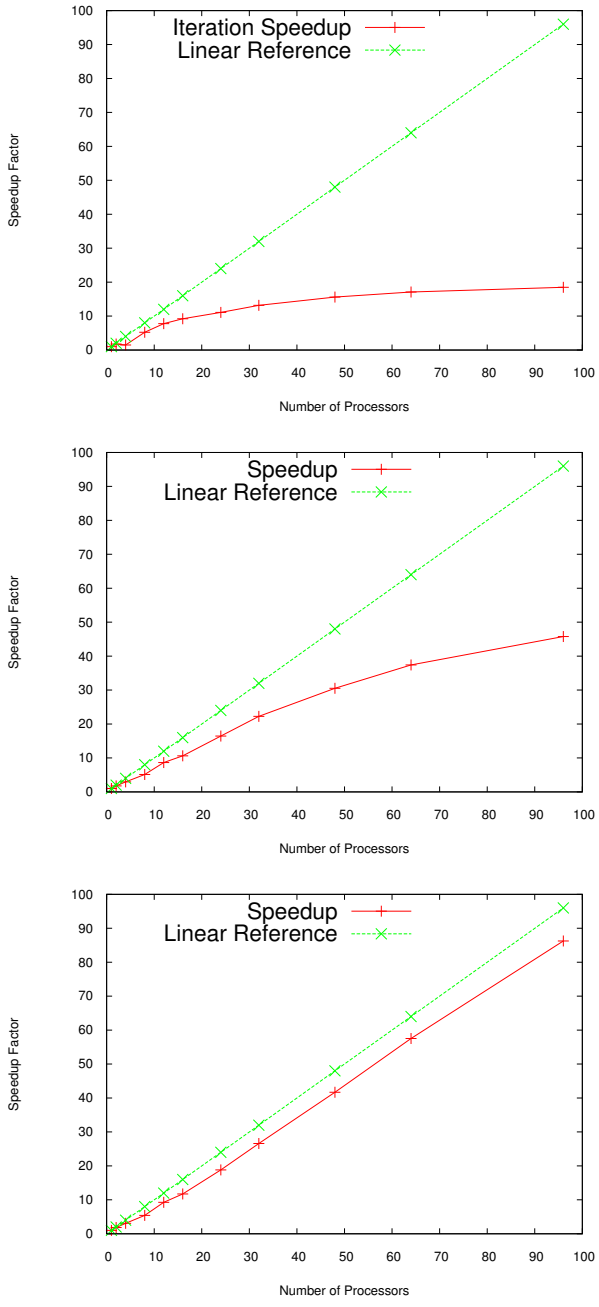


Fig. 10. Scalability results based on number of generators. Green is a linear reference whereas red is the speedup computed using the parallel version of MPI-SCVT compared to a serial version.

neighbours. This is possible because points can only move within a region radius on any two subsequent iterations, and because of this can only move into another region which is overlapping the current region. More efficiency gains could be realized by improving communication and integration algorithms. This addition of shared memory parallelization to the integration routines might improve overall performance, as the integration routines are embarrassingly parallel. Modification of the data structures used could also enable the generation of ultra-high resolution meshes (100M generators). In principle, because all of the computation is local this algorithm should scale linearly very well up to hundreds if not thousands of processors.

5 Summary

A novel technique for the parallel construction of Delaunay triangulations is presented, utilizing unique domain decomposition techniques combined with stereographic projections. This parallel algorithm can be applied to the generation of planar and spherical centroidal Voronoi tessellations. Results were presented for the generation of spherical centroidal Voronoi tessellations, with comparisons to STRIPACK, a well-known algorithm for the computation of spherical Delaunay triangulations. The algorithm presented in the paper (MPI-SCVT) shows slower performance than STRIPACK when computing a single triangulation in serial and faster performance when using roughly 100 processors. When paired with a SCVT generator, the algorithm shows additional speed up relative to a STRIPACK based SCVT generator.

Acknowledgements. We would like to thank Geoff Womeldorff and Michael Duda for many useful discussions. The work of Doug Jacobsen, Max Gunzburger, John Burkardt, and Janet Peterson was supported by the US Department of Energy under grants number DE-SC0002624 and DE-FG02-07ER64432.

Edited by: R. Redler

References

Amato, N. and Preparata, F.: An NC parallel 3D convex hull algorithm, in: Proceedings of the ninth annual symposium on Computational geometry – SCG '93, 289–297, May 1993.

Batista, V., Millman, D., Pion, S., and Singler, J.: Parallel geometric algorithms for multi-core computers, Comp. Geom.-Theor. Appl., 43, 663–677, 2010.

Bowers, P., Diets, W., and Keeling, S.: Fast algorithms for generating Delaunay interpolation elements for domain decomposition, available at: http://www.math.fsu.edu/~aluffi/archive/paper77.ps.gz, (last access: May 2011), 1998.

Chernikov, A. and Chrisochoides, N.: Algorithm 872: parallel 2D constrained Delaunay mesh generation, ACM T. Math. Software, 34, 6:1–6:20, 2008.

Cignoni, P., Montani, C., and Scopigno, R.: DeWall: a fast divide and conquer Delaunay triangulation algorithm in E-d, Comput. Aided Design, 30, 333–341, 1998.

Du, Q. and Gunzburger, M.: Grid generation and optimization based on centroidal Voronoi tessellationss, Appl. Math. Comput., 133, 591–607, 2002.

Du, Q., Faber, V., and Gunzburger, M.: Centroidal Voronoi tessellations: applications and algorithms, SIAM Rev., 41, 637–676, 1999.

Du, Q., Ju, L., and Gunzburger, M.: Constrained centroidal Voronoi tessellations for surfaces, SIAM J. Sci. Comput., 24, 1488–1506, 2003a.

Du, Q., Ju, L., and Gunzburger, M.: Voronoi-based finite volume methods, optimal Voronoi meshes, and PDEs on the sphere, Comput. Method. Appl. M., 192, 3933–3957, 2003b.

Du, Q., Emelianenko, M., and Ju, L.: Convergence of the Lloyd Algorithm for computing centroidal Voronoi tessellations, SIAM J. Numer. Anal., 44, 102–119, 2006a.

Du, Q., Ju, L., and Gunzburger, M.: Adaptive finite element methods for elliptic PDE's based on conforming centroidal Voronoi Delaunay triangulations, SIAM J. Sci. Comput., 28, 2023–2053, 2006b.

Heikes, R. and Randall, D.: Numerical integration of the shallow-water equations on a twisted icosahedral grid. Part I: Basic design and results of tests, Mon. Weather Rev., 123, 1862–1880, 1995.

Jacobsen, D. and Womeldorff, G.: MPI-SCVT Source Code: https://github.com/douglasjacobsen/MPI-SCVT 2013.

Ju, L., Ringler, T., and Gunzburger, M.: A multi-resolution method for climate system modeling: application of spherical centroidal Voronoi tessellations, Ocean Dynam., 58, 475–498, 2008.

Ju, L., Ringler, T., and Gunzburger, M.: Voronoi tessellations and their application to climate and global modeling, Ocean Dynam., 58, 313–342, 2011.

Lambrechts, J., Comblen, R., Legat, V., Geuzaine, C., and Remacle, J.-F.: Multiscale mesh generation on the sphere. Ocean Dynam., 58, 461–473. (2008). doi:10.1007/s10236-008-0148-3

Lloyd, S.: Least squares quantization in PCM, IEEE T. Inform. Theory, 28, 129–137, 1982.

Metropolis, N. and Ulam, S.: The Monte Carlo method, J. Am. Stat. Assoc., 44, 335–341, 1949.

Nguyen, H., Burkardt, J., Gunzburger, M., Ju, L., and Saka, Y.: Constrained CVT meshes and a comparison of triangular mesh generators, Ocean Dynam., 42, 1–19, 2008.

Okabe, A., Boots, B., Sugihara, K., and Chiu, S.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, John Wiley, Baffins Lane, Chichester, West Sussex, PO19 1UD, England, 2000.

Pain, C. C., Piggot, M. D., Goddard, A. J. H., Fang, F., Gorman, G. J., Marshall, D. P., Eaton, M. D., Power, P. W., and de Oliveira, C. R. E.: Three-dimensional unstructured mesh ocean modelling, Ocean Model., 10, 5–33, 2005.

Renka, R.: Algorithm 772: STRIPACK: Delaunay triangulation and Voronoi diagram on the surface of a sphere, ACM T. Math. Software, 23, 416–434, 1997.

Ringler, T., Petersen, M., Higdon, R. L., Jacobsen, D., Jones, P. W., and Maltrud, M.: A Multi-Resolution Approach to Global Ocean Modeling, Ocean Modelling, accepted, doi:10.1016/j.ocemod.2013.04.010, 2013.

Ringler, T. D., Jacobsen, D., Gunzburger, M., Ju, L., Duda, M., and Skamarock, W.: Exploring a Multiresolution Modeling Approach within the Shallow-Water Equations, Mon. Weather Rev., 139, 3348–3368, doi:10.1175/MWR-D-10-05049.1., 2011.

Saalfeld, A.: Delaunay triangulations and stereographic projections, Cartogr. Geogr. Inform., 26, 289–296, 1999.

Shewchuk, J.: Triangle: engineering a 2D quality mesh generator and Delaunay triangulator, Applied Computational Geometry: Towards Geometric Engineering, 1148, 203–222, 1996.

Weller, H., Weller, H., and Fournier, A.: Voronoi, Delaunay, and block-structured mesh tefinement for solution of the shallow-water equations on the sphere, Mon. Weather Rev., 137, 4208–4224, 2009.

Zhou, J., Deng, X., and Dymond, P.: A 2-D parallel convex hull algorithm with optimal communication phases, in: Proceedings 11th International Parallel Processing Symposium, April 1997, University of Geneva, Geneva, Switzerland, 596–602, 2001.