Fast Spherical Centroidal Voronoi Mesh Generation: A Lloyd-preconditioned LBFGS Method in Parallel

Huanhuan Yang\textsuperscript{a}, Max Gunzburger\textsuperscript{b}, Lili Ju\textsuperscript{c}

\textsuperscript{a}Department of Mathematics, Shantou University, Shantou, Guangdong 515063, China
\textsuperscript{b}Department of Scientific Computing, Florida State University, Tallahassee, FL 32306, USA
\textsuperscript{c}Department of Mathematics, University of South Carolina, Columbia, SC 29208, USA

Abstract

Centroidal Voronoi tessellation (CVT)-based mesh generation is a very effective technique for creating high-quality Voronoi meshes and their dual Delaunay triangulations that often play a crucial role in applications, including ocean and atmospheric simulations using finite volume schemes. In the next generation climate models, the spacing scales change dramatically across the whole sphere and require ultra-high resolution and smooth transitions from coarse to fine grid regions. Thus fast and robust spherical CVT (SCVT) meshing algorithms become highly desirable. In this paper, we first propose a Lloyd-preconditioned limited-memory BFGS method for constructing SCVTs that is also applicable to the construction of CVTs of general domains. This method is then parallelized based on overlapping domain decomposition, enabling excellent scalability on distributed systems. Results of several computational experiments show that the new method could incur computational time costs one order of magnitude smaller compared with some existing methods for generating large-scale highly variable-resolution meshes, while also providing significantly improvements in mesh quality.

Keywords: centroidal Voronoi tessellation, Lloyd-preconditioned LBFGS, mesh generation, climate modeling, domain decomposition

1. Introduction

A centroidal Voronoi tessellation (CVT) \cite{5} of a given domain is a Voronoi tessellation whose generators coincide with the mass centroids of their corresponding Voronoi cells. Because of its ability to construct point sets that are locally quasi-uniformly distributed with respect to a given point-density function, CVT is a desirable technique for creating high-quality Voronoi and Delaunay meshes \cite{6, 7}. For example, such meshes are often crucial for global or regional climate modeling using finite volume schemes\cite{19, 20, 22, 24}.

CVTs can be constructed using probabilistic methods such as MacQueen’s method \cite{17} or probabilistic Lloyd methods \cite{14} that are elegant random sequential sampling methods that do not require the explicit construction of Voronoi tessellations. However, these probabilistic methods
feature very slow convergence so that more efficient deterministic methods for CVT construction are more often used. Typical deterministic CVT construction algorithms include Lloyd’s method [16, 5], the Lloyd-Newton method [3], and quasi-Newton methods such as limited-memory BFGS (LBFGS) method [15, 9].

In addition to the geometric definition given above, CVTs also have an analytical definition as being a critical point of an “energy” functional associated with a set of generators [5]. In particular, a local minimizer of a CVT energy functional determines a stable CVT. Lloyd’s method can be viewed as a linearly convergent gradient descent optimization method [5] with respect to the CVT energy function. The Lloyd-Newton method is an attempt to accelerate CVT construction by taking advantage of its quadratic rate of convergence. However, it is not often used for high-resolution CVT grid generation due to the burdensome computation of a Hessian inverse at each iteration.

Quasi-Newton methods are usually more feasible choices because they make use of efficient approximate Hessian matrices or their inverses, and generally have super-linear convergence. In [15] the LBFGS quasi-Newton method and a preconditioned version thereof are used to construct CVTs in planar regions or surfaces. The preconditioner used in [15] is the incomplete Cholesky factorization of a modified Hessian matrix; however, this approach is not guaranteed to be stable or effective [11, 9] and, in addition, it has been shown that such LBFGS methods preconditioned in this manner do not apparently improve the efficiency of LBFGS [15] in terms of computational time. A graph Laplacian preconditioner is used in [9] which reduces the computational time to 56.9% on average for constructing CVT meshes with certain non-uniform point-density functions. However, the graph Laplacian preconditioner can be troublesome for generating high-resolution meshes not only because of the limited efficiency due to having to repeatedly solve a large-scale linear system, but also because it is limited effectiveness in reducing the CVT energy.

In recent years, the need for improving the efficiency of the next generation of the computational climate models has driven the need for the more efficient construction of large-scale meshes on the sphere, meshes that feature high-quality and highly variable-resolution with smooth transitions. In this paper, we propose a fast and effective quasi-Newton method for the construction of spherical CVTs (SCVT) but which is also applicable to the construction of CVTs in more general domains. One of our key contributions is the development of a Lloyd-preconditioned LBFGS algorithm; specifically, the Lloyd step is iteratively executed and used as approximations of varying initial Hessian inverses. For quasi-uniform meshes, our Lloyd-preconditioned LBFGS scheme performs similarly to LBFGS. However, for highly variable-resolution meshes, it can dramatically speed up LBFGS while also providing significantly improved mesh quality. Also differing from all previous efforts using LBFGS for computing CVTs, we develop a parallel implementation of our method on distributed systems by using an overlapping domain decomposition approach [12]. Our parallel algorithm features well-balanced loading of mesh points and has excellent performance with respect to strong scaling efficiency. Parallel test results show almost linear speedup when the number of mesh points is of the order of millions.

The remainder of the paper is organized as follows. In Section 2, we review the definitions and basic properties of CVTs and their extension to surfaces/manifolds and also review some popular CVT construction algorithms. Then, in Section 3 we propose the Lloyd-preconditioned LBFGS method for computing SCVTs and its parallelization on distributed systems. Several numerical experiments and comparisons are presented in Section 4 to demonstrate the performance of our method in both serial and parallel environments. Some concluding remarks are made in Section 5.
2. Review of centroidal Voronoi tessellation and its generalizations

In this section, we first introduce some concepts and properties of centroidal Voronoi tessellation and then review some existing algorithms for their construction.

2.1. Definitions and properties of CVTs

Given a bounded domain $\Omega \subset \mathbb{R}^N$ and a set of points \( \{z_i\}_{i=1}^K \subset \Omega \), the Voronoi region \( V_i \) corresponding to the generator (or site) \( z_i \) is defined as

\[
V_i = \{ x \in \Omega : \| x - z_i \| < \| x - z_j \| \text{ for } j = 1, \ldots, K \text{ and } j \neq i \},
\]

where \( \| \cdot \| \) denotes the Euclidean norm in \( \mathbb{R}^N \). A Voronoi tessellation or Voronoi diagram \( \mathcal{V} = \{V_i\}_{i=1}^K \) forms a special partition of \( \Omega \) in the sense that \( V_i \cap V_j = \emptyset \) for \( i \neq j \) and \( \Omega = \bigcup_i V_i \). Voronoi cells are convex polytopes except possibly for those whose boundary intersect with the boundary of \( \Omega \). The dual graph of a Voronoi tessellation corresponds to the Delaunay triangulation of the set of Voronoi generators.

Given a point-density function \( \rho(x) > 0 \) defined over \( \Omega \), the mass center \( z_i^* \) of a Voronoi cell \( V_i \) is defined by

\[
z_i^* = \frac{\int_{V_i} y \rho(y) dy}{\int_{V_i} \rho(y) dy}, \quad i = 1, \ldots, K.
\]

In general, \( z_i \neq z_i^* \). In the special case that \( z_i = z_i^* \) for \( i = 1, \ldots, K \), i.e., each generator coincides with the mass centroid of the corresponding Voronoi cell, the Voronoi tessellation is referred to as a **centroidal Voronoi tessellation** (CVT) [5].

Alternately and equivalently to the just given geometric characterization of CVTs, one can define them through a variational approach. Letting \( Z = \{z_i\}_{i=1}^K \) denote an arbitrary ordered sequence of points in \( \Omega \) and \( \mathcal{V} = \{V_i\}_{i=1}^K \) an arbitrary tessellation of \( \Omega \), we define the “energy” functional

\[
\mathcal{F}(Z, \mathcal{V}) = \sum_{i=1}^K \int_{V_i} |y - z_i|^2 \rho(y) dy.
\]

A CVT is a critical point of this functional [5] in the sense that among all possible sets of \( K \) points \( Z \) in \( \Omega \) and among all possible tessellations \( \mathcal{V} \) of \( \Omega \) into \( K \) nonoverlapping subregions, this functional is rendered stationary if and only if \( \{Z, \mathcal{V}\} \) form a CVT. Furthermore, local minimizers \( \{Z, \mathcal{V}\} \) of \( \mathcal{F} \) determine stable CVTs. Stable CVTs are more desirable in practical applications than unstable ones which correspond to saddle points of \( \mathcal{F} \).

Next, consider the tessellation of a compact surface or manifold \( S \subset \mathbb{R}^N \). Given a set of points \( \{z_i\}_{i=1}^K \subset S \), we can define the corresponding Voronoi tessellation of \( S \) similarly to (1) as

\[
V_i^c = \{ x \in S : \| x - z_i \| < \| x - z_j \| \text{ for } j = 1, \ldots, K, j \neq i \},
\]

For each Voronoi cell \( V_i^c \), we use a generalized definition for the center of mass to ensure that it lies on the surface \( S \). Specifically, we refer to \( z_i^c \) as the **constrained mass centroid** [7] of \( V_i^c \) on \( S \)

\[
z_i^c = \arg \min_{z \in S} \int_{V_i^c} |y - z|^2 \rho(y) d\sigma(y),
\]

3
where the point-density function $\rho(x) > 0$ is defined on the surface $S$ and $d\sigma(\cdot)$ denotes the surface area element. Note that in general the classical mass centroid $z^*_i$ defined by (2) does not lie on $S$. However, as pointed out in [7], there is a connection between the constrained mass centroid $z^c_i$ and the classical mass centroid $z^*_i$, namely, if $z^c_i \in S - \partial S$, then $z^*_i - z^c_i$ is normal to the surface $S$ at $z^c_i$, i.e., $z^c_i$ is the projection of $z^*_i$ onto $S$ along the normal direction to $S$ at $z^c_i$. This feature is especially useful if $S$ is a sphere because on the one hand, the construction of a constrained mass centroid using its definition (4) is considerably more difficult compared to the construction of the classical mass centroid and, on the other hand, for $S$ a sphere, the former is merely the intersection of the radial line going through the latter with the sphere.

The tessellation of $S$ defined by (3) is referred as a constrained centroidal Voronoi tessellation (CCVT) if $z_i = z^c_i$ for $i = 1, \ldots, K$, i.e., the generator associated with each Voronoi cell $V^c_i$ coincides with the constrained mass centroid of that cell. It is also proved in [7] that a stable CCVT can also be defined as a local minimizer of the CCVT energy function

$$F^c(Z, V) = \sum_{i=1}^{K} \int_{V^c_i(Z)} \|y - z_i\|^2 \rho(y) d\sigma(y).$$

It is worth emphasizing that although the generators of CCVTs are constrained to lie on a surface/manifold, the distance metric used is still the Euclidean distance rather than the geodesic distance.

When $S$ is a sphere in $\mathbb{R}^3$ or a subset of a sphere, as is the case in ocean modeling, a CCVT is referred to as spherical CVT (SCVT).

### 2.2. Some existing deterministic CVT construction algorithms

#### 2.2.1. Lloyd’s method

An elegant deterministic method for computing CVTs/CCVTs is Lloyd’s method which is a natural byproduct of the geometric characterization of CVTs. Starting from an initial placement of $K$ points in $\Omega$, a Lloyd iteration determines the Voronoi diagram of $\Omega$ associated with those points, then moves each point to the mass centroid of its Voronoi cell. This pair of steps is repeated until a sufficient small decrease in the point movement is met. Lloyd’s method can be viewed as a fixed-point iteration for the Lloyd map

$$T : Z \rightarrow \left\{ \left( \frac{\int_{V_i(Z)} y \rho(y) dy}{\int_{V_i(Z)} \rho(y) dy} \right) \right\}_{i=1}^{K},$$

where $\{V_i(Z)\}_{i=1}^{K}$ denotes the Voronoi tessellation associated with points $\{z_i\}_{i=1}^{K}$. Convergence analyses for some typical cases are given in [5, 4].

Based on the variational characterization of a CVT as a critical point of the CVT energy function $F(Z, V)$, a CVT can also be constructed by derivative-based optimization methods because the energy function $F$ is $C^2$ smooth for a convex domain [15]. Recognizing that in Lloyd’s method $V = V(Z)$, i.e., the tessellation $V$ is the Voronoi tessellation corresponding to $Z$, we have that $F = F(Z, V(Z))$, i.e., we can view $F$ as a function of $Z$ only. Then the gradient of $F(Z, V(Z))$ is given by

$$\frac{dF}{dz_i} = \frac{\partial F}{\partial z_i}(Z, V(Z)) = 2z_i \int_{V_i} \rho(y) dy - 2 \int_{V_i} y \rho(y) dy,$$
where we use the fact $\frac{\partial F}{\partial V} = 0$ at $(Z, V(Z))$ due to the property of Voronoi regions [10, 5]. Denoting the mass $\int_{V_i} \rho(y)dy$ by $m_i$, we have

$$m_i^{-1} \frac{dF}{dz_i} = 2z_i - 2T_i(Z).$$

Thus, the Lloyd iteration $Z^{(n+1)} = T(Z^{(n)})$ can be rewritten as

$$Z^{(n+1)} = Z^{(n)} - (2M^{(n)})^{-1} \nabla F(Z^{(n)}),$$

(5)

with the mass matrix $M^{(n)}$ being a diagonal matrix diag($m_i$) associated with the Voronoi tessellation $V(Z^{(n)})$. In the context of optimization, Lloyd’s method is a special case of a gradient descent method. Although it has only a linear rate of convergence, the Lloyd iteration always decreases the energy function $F$ until a local minimizer is obtained without the need of line searches for step-size control.

2.2.2. Newton and quasi-Newton methods

An attempt to accelerate CVT construction is the Lloyd-Newton method [3]. Although the Newton iteration has quadratic rate of convergence, it is usually not feasible for large scale CVT grid generation due to the burden of storing and computation of Hessian inverses. For this reason, quasi-Newton methods are better choices because they use approximations of the Hessian matrices or their inverses and have super-linear rates of convergence. Specifically, given an approximation $B^{(n)}$ of the Hessian matrix $H(Z^{(n)})$ of $F(Z^{(n)})$, a quasi-Newton iteration operates as follows: for \( n = 0, 1, 2, \ldots \)

1: solve $B^{(n)}q = -\nabla F(Z^{(n)})$;

2: update $Z^{(n+1)} = Z^{(n)} + \alpha^{(n)}q$.

Here $q$ is a search direction along which an optimal step size $\alpha^{(n)}$ is determined by a line search algorithm to satisfy the Wolfe conditions [13]. The choice $B^{(n)} = H(Z^{(n)})$ results in the classical Newton method.

One of the most popular quasi-Newton methods is the BFGS method [18]. In the standard BFGS approach, at the $n$-th step, the inverse of Hessian matrix is iteratively updated by a symmetric matrix $H^{(n+1)}$ being close to the last Hessian inverse approximation and satisfying the secant equation

$$\tilde{H}^{(n+1)}y_n = s_n,$$

where $s_n = Z^{(n+1)} - Z^{(n)}$, $y_n = \nabla F(Z^{(n+1)}) - \nabla F(Z^{(n)})$.

Such derivation forces the quadratic forms of function $F$ around $Z^{(n)}$ and $Z^{(n+1)}$ match gradients respectively at these two steps. The BFGS updating formula in the product form can be given explicitly as

$$\tilde{H}^{(n)} = (S_{n-1}^T \cdots S_{n-m(n)}^T)\tilde{H}^{(0)}(S_{n-m(n)} \cdots S_{n-1})$$

$$+ (S_{n-1}^T \cdots S_{n-m(n)+1}^T)\rho_{n-m(n)}S_{n-m(n)}^T S_{n-m(n)}(S_{n-m(n)+1} \cdots S_{n-1})$$

$$+ (S_{n-1}^T \cdots S_{n-m(n)+2}^T)\rho_{n-m(n)+1}S_{n-m(n)+1}^T S_{n-m(n)+1}(S_{n-m(n)+2} \cdots S_{n-1})$$

$$+ \cdots + \rho_{n-1}S_{n-1}^T.$$
where  \( \rho_i = 1/(y_i^Ts_i), \quad S_i = I - \rho_is_is_i^T \).

In the classic BFGS updating scheme, \( m(n) = n \) is taken and the initial approximation \( \tilde{H}^{(0)} \) is often set to be a constant multiple of the identity matrix.

In the limited-memory BFGS (LBFGS), one needs to specify \( M \), the number of BFGS corrections needed to be stored. That is, only the recent \( M \) pairs of \( \{s_i, y_i\} \) are kept. The BFGS approximation is modified by setting \( m(n) = \min\{n, M\} \) and allowing the initial approximation \( \tilde{H}^{(0)} \) to vary from iteration to iteration. For instance, \( \tilde{H}^{(0)} \) is typically replaced by

\[
\gamma_n = \frac{y_n^Ts_{n-1}}{y_{n-1}^Ty_n-1}
\]

is close to the reciprocal of an eigenvalue of \( H(Z^{(n)}) \). In practical computation, the product \( \tilde{H}^{(n)}\nabla F(Z^{(n)}) \), rather than the matrix \( \tilde{H}^{(n)} \), is recursively updated. Specifically, the LBFGS updating at step \( n \) is done as follows:

1. Initialize \( q \leftarrow -\nabla F(Z^{(n)}) \);
2. \( \text{for } i = n - 1, \cdots, n - m(n) \text{ do } \) % backward loop
   \( \alpha_i \leftarrow \rho_is_i^Tq; \)
   \( q \leftarrow q - \alpha_i y_i; \)
3. \( \text{end for} \)
4. \( q \leftarrow \tilde{H}^{(0)}_n q; \)
5. \( \text{for } i = n - m(n), \cdots, n - 1 \text{ do } \) % forward loop
   \( q \leftarrow q + s_i(\alpha_i - \rho_i y_i^T q); \)
6. \( \text{end for} \)
7. Update \( Z^{(n + 1)} = Z^{(n)} + \alpha^{(n)} q. \) % line search

3. Lloyd-preconditioned LBFGS in parallel

In this section, we will first develop a Lloyd preconditioner for the LBFGS method for SCVT construction so that its performance can be further improved, after which we propose an efficient parallelization algorithm for the method on distributed systems based on overlapping domain decomposition, which is important in terms of both computational and storage efficiency.

3.1. Lloyd-preconditioned LBFGS

The idea of a preconditioned LBFGS method (P-LBFGS) proposed in [13] is to choose a matrix \( B^{(n)} \) very alike to the Hessian \( H(Z^{(n)}) \) and periodically replace line 4 in the LBFGS iteration by

\[
q \leftarrow (B^{(n)})^{-1} q.
\] (6)

It requires that \( B^{(n)} \) should be inexpensively achievable, and most importantly, the above equation can be easily solved. If the matrix \( B^{(n)} \) can capture significant features of the Hessian, then the convergence of LBFGS can be hastened.

It has been shown that the CVT energy function has second-order smoothness for convex domains [15] and that the Hessian matrix can be computed exactly; see [10, 15]. Denote by \( J_i \) the
set of node indices of the Voronoi cells adjacent to \( V_i \) (the Voronoi cell corresponding to generator \( z_i \)). Let the coordinates of the point \( z_i \) (resp. \( y \)) be denoted by \( z_{ik} \) (resp. \( y_k \)), \( k = 1, \ldots, N \). Then, the second-order derivatives of the CVT energy are given by the following explicit formulas: for \( k, l = 1, \ldots, N, \)

\[
\begin{align*}
\frac{\partial^2 F}{\partial z_{ik}} &= 2m_i - \sum_{j \in J_i} \int_{V_i \cap V_j} \frac{2}{\|z_j - z_i\|} (z_{ik} - y_K)^2 \rho(y) dy, \\
\frac{\partial^2 F}{\partial z_{ik} \partial z_{il}} &= -\sum_{j \in J_i} \int_{V_i \cap V_j} \frac{2}{\|z_I - z_i\|} (z_{ik} - y_K) (z_{il} - y_I) \rho(y) dy, \quad k \neq l, \\
\frac{\partial^2 F}{\partial z_{ij} \partial z_{jl}} &= \int_{V_i \cap V_j} \frac{2}{\|z_I - z_i\|} (z_{ik} - y_K) (z_{il} - y_I) \rho(y) dy, \quad j \in J_i, \\
\frac{\partial^2 F}{\partial z_{ij} \partial z_{jl}} &= 0, \quad j \neq i, j \notin J_i.
\end{align*}
\]

In [15], a modified Cholesky factorization of the Hessian matrix \( H(Z^{(n)}) \) is applied to obtain the positive-definite matrix \( B^{(n)} \), and perform (6) every \( \bar{N} \) iterative steps, where \( \bar{N} \) is a user-defined parameter. The numerical experiments in [15] show that LBFGS and P-LBFGS yield better performance compared to Lloyd’s method and Newton’s method. However, the preconditioned version of LBFGS used in [15] only improves LBFGS a little, and these two perform very similarly even in the case of non-uniform point-density functions.

Another approach at developing a P-LBFGS method is to use a graph Laplacian preconditioner (graph Laplacian P-LBFGS), proposed in [9]. They consider CVT construction in a polygonal domain \( \Omega \) in two dimensions. Let \( T_{ij} \) denote the triangle inside \( V_i \) containing the point \( z_i \) and the edge \( V_i \cap V_j \). If \( V_i \) has an edge \( e_{ib} \) in \( \partial \Omega \), denote the triangle formed by \( e_{ib} \) and \( z_i \) as \( T_{ib} \). Then, the graph Laplacian matrix \( A = (a_{ij}) \) is given by

\[
\begin{align*}
a_{ii} &= \sum_{j \in J_i} \int_{T_{ij} \cup T_{ji}} \rho(y) dy + 2 \sum_{e_{ib}} \int_{T_{ib}} \rho(y) dy, \\
a_{ij} &= -\int_{T_{ij} \cup T_{ji}} \rho(y) dy, \quad j \in J_i, \\
a_{ij} &= 0, \quad j \neq i, j \notin J_i.
\end{align*}
\]

According to the numerical tests in [9], for a constant point-density function, the graph Laplacian P-LBFGS has similar efficiency compared to the classical LBFGS algorithm, but for non-uniform point-density functions, the graph Laplacian preconditioner can reduce the computational time by 56.9% on average. However, the use of graph Laplacian preconditioners to speed up LBFGS could be limited for ocean grids due to some difficulties: first, solving the Laplacian linear system on large-scale meshes at each iteration is computationally expensive; second, the Laplacian matrix \( A \) is singular for closed surfaces such as the surface of the sphere and thus special care is needed in choosing an appropriate solution for the preconditioning process. Furthermore, the graph Laplacian preconditioner sometimes does not work well in significantly reducing the CVT energy, as we reported in Section 4.2.

In this paper, we propose a new preconditioner (initial Hessian inverse, line 4 in the LBFGS iteration), inspired by Lloyd’s iteration, as

\[
\tilde{H}^{(0)}_n = \left(2M^{(n)}\right)^{-1} = \text{diag} \left( 2 \int_{V_i(Z^{(n)})} \rho(y) dy \right)^{-1}.
\]
We call this the Lloyd-preconditioned LBFGS (Lloyd P-LBFGS) method because this approach is equivalent to taking the initial BFGS direction to be the Lloyd’s updating direction (see (5)) and the initial line-search step size as 1.

There are several reasons behind this idea. First of all, because the Lloyd map $T$ is continuous, the Lloyd iteration converges globally to a critical point of $F$ if the iterations stay in a compact set $[4]$. Although the compactness of the iteration has not been rigorously justified in the literature, it seems to be intuitively true and matches practical experiences. This nice property of Lloyd’s method is not shared by Newton or quasi-Newton methods. Second, the initial Hessian inverse given by (8) is very simple while still capturing a lot of Hessian matrix information. Note that $2M^{(n)}$ is the main diagonal part of the Hessian matrix as shown in (7). A connection of (8) to the graph Laplacian matrix is more obvious: it is an approximation of the diagonal components of the Laplacian matrix $A$ because

$$
\int_{V_i} = \sum_{j \in J_i} \int_{T_{ij}} + \int_{T_{ih}}
$$

and we can treat $\int_{T_{ij} \cup T_{ji}}$ approximately as $2 \int_{T_{ij}}$. Lastly, the diagonal matrix $M^{(n)}$ itself contains significant information about the density distribution.

To apply LBFGS for minimizing the energy function $F^c$ in the case of CCVT, in [15] a “pseudo” derivative is taken by projecting the gradient of the CVT energy onto the tangent plane of the compact surface $S$. That is, they proposed to use the tangential derivative

$$
\frac{\partial F^c}{\partial z_i} \bigg|_S = \frac{\partial F^c}{\partial z_i} - \left( \frac{\partial F^c}{\partial z_i} \cdot N(z_i) \right) N(z_i)
$$

in the minimization process, where $N(z_i)$ is the normal to the surface $S$ at $z_i$. The updated set of generators needs to be projected back onto the surface at each iteration. Following this approach, we can derive the Lloyd preconditioner for the case of the unit sphere analogous to that for planar domains. Note that

$$
\frac{\partial F^c}{\partial z_i} = 2z_i \int_{V_i^c} \rho(y) d\sigma - 2 \int_{V_i^c} y \rho(y) d\sigma.
$$

Setting $c_i = \int_{V_i^c} y \rho(y) d\sigma$, the tangential derivative can be written as

$$
\frac{\partial F^c}{\partial z_i} \bigg|_S = 2(c_i^T z_i) z_i - 2c_i
$$

so that we have

$$
c_i = (c_i^T z_i) d_i, \quad \text{with} \ d_i = z_i - (2c_i^T z_i)^{-1} \frac{\partial F^c}{\partial z_i} \bigg|_S.
$$

The updating by $d_i$ is then a Lloyd iteration because the projection of $d_i$ onto the unit surface coincides with that of $c_i$. As such, we can take the Lloyd preconditioner in $\mathbb{R}^{K \times K}$ as

$$
\tilde{H}_n^{(0)} = \text{diag} \left( 2(c_i^{(n)})^T z_i^{(n)} \right)^{-1}.
$$
3.2. Parallel implementation

Our parallel implementation of the Lloyd-preconditioned LBFGS method for SCVT mesh generation is based on overlapping domain decomposition.

3.2.1. Parallelization of Delaunay triangulations on sphere

As is true for all other deterministic methods for CVT/SCVT construction, the Lloyd-preconditioned LBFGS algorithm involves explicit integrations over Voronoi cells. The Voronoi cells can be determined from the dual Delaunay triangulation. In this subsection we present a parallel algorithm for computing Delaunay triangulations and spherical Delaunay triangulations first studied in [11, 12].

Planar Delaunay triangulations. A Delaunay triangulation for a set of points \( \{z_i\}_{i=1}^K \) in a plane is a triangulation \( T \) such that no point in \( \{z_i\}_{i=1}^K \) is strictly inside the circumcircle of any triangle in \( T \). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid extremely acute angles of the triangles. The uniqueness of Delaunay triangulation for \( \{z_i\}_{i=1}^K \) is guaranteed if the set of points is in general position (no set of four points lie on a circle whose interior does not contain points from \( \{z_i\}_{i=1}^K \)).

The parallelization of planar Delaunay triangulation algorithms has been widely studied in the past decades; see, e.g., [2, 8]. Popular algorithms commonly split the global point set into equally distributed subsets, each of which can then be triangulated simultaneously in parallel. The resulting local triangulations are then stitched together to form a global triangulation. Such a merging step may need to modify significant portions of the local triangulations, thus it is the most significant step that affects time efficiency. In this work, we resort to an alternative merging step that needs no modifications of the local triangulations and thus performs well in parallel. The algorithm for planar Delaunay triangulation consists of the following three steps (see Figure 1-left).

1. Given a planar domain \( \Omega \) and a set of points \( \{z_i\}_{i=1}^K \) in \( \Omega \), we preform an overlapping covering of \( \Omega \) and record the overlapping connectivity or neighboring list. A simple example of the covering \( C \) can be a set of \( p \) disks denoted by \( \{D_l(p_l, r_l)\}_{l=1}^p \), where \( p_l \) and \( r_l \) are the disk center and radius respectively. The \( K \) points in \( \Omega \) are then distributed into \( p \) subregions, each of which is handled by a single processor. Because of the overlapping feature of the covering, some points may belong to multiple processors.

2. On the \( l \)-th processor, a local Delaunay triangulation \( \tilde{T}_l \) is carried out independently. To this end, one can employ a serial Delaunay algorithm such as the Delaunay triangulator in the Triangle software package [23].

3. On each processor, we only keep triangles whose circumcircles are completely contained inside the corresponding subregion. Denoting the new triangulation on the \( l \)-th processor by \( \hat{T}_l \), then the circumcircle of any triangle in \( \cup_l \hat{T}_l \) does not contain any points from \( \{z_i\}_{i=1}^K \). Each local triangulation \( \hat{T}_l \) is then exactly a portion of the global Delaunay triangulation due to the uniqueness of Delaunay triangulations.

Spherical Delaunay triangulation. We now consider spherical Delaunay triangulation on the unit sphere \( S \) in \( \mathbb{R}^3 \). In this case, triangles and Voronoi cells are replaced by spherical ones whose boundaries are geodesic arcs. The planar parallelization scheme is combined with stereographic projections.
A stereographic projection is a conformal mapping that projects a sphere onto a plane tangent to the sphere. Letting $t$ denote the contact point of the sphere and its tangent plane, the stereographic projection of a point $z$ on the unit sphere to the point $x$ on the tangent plane is given by (in Cartesian coordinates)

$$P : z \rightarrow x = sz + (s - 1)t,$$

where $s = \frac{2}{(t \cdot (z + t))}$.

The conformality of stereographic projections implies the preservation of circumcircles of triangles along with their interiors and hence the Delaunay criteria. Therefore, the Delaunay triangulation of a subregion of the sphere can be obtained by the stereographic mapping from the planar Delaunay tessellation on the corresponding tangent plane. The adaptation to spherical Delaunay triangulation from a planar one is described by the following three steps; see Figure 1.

1. We preform an overlapping covering of the unit sphere $S$ and record the overlapping connectivity or neighboring list. The covering $\mathcal{U}$ could simply be a set of geodesic disks $\{\mathcal{U}(t_i, r_l)\}_{l=1}^p$, where $t_i$ is the disk center and $r_l$ is the geodesic radius. Points $\{z_i\}_{i=1}^K$ on $S$ are then distributed into $p$ subregions with overlaps.

2. On the $l$-th processor, we project the point set $\{z_i\}_{i=1}^K \cap \mathcal{U}(t_l, r_l)$ onto the plane tangent to the sphere at $t_l$. We then proceed to construct a planar Delaunay triangulation $\mathcal{T}_l^p$. The local spherical Delaunay triangulation $\mathcal{T}_l^s$ is then produced by stereographic projection from $\mathcal{T}_l^p$.

3. In each subregion, say $\mathcal{U}(t_i, r_l)$, we only keep spherical triangles whose circumcircles are completely contained inside, so as to guarantee the Delaunay property with respect to the global point set $\{z_i\}_{i=1}^K$. In particular, for disk subregions, let $p_j^s$ and $r_j^s$ denote the center and radius of the circumcircle of the $j$-th spherical triangle in $\mathcal{U}(t_i, r_l)$; the triangle selection criteria is $\arccos(t_i \cdot p_j^s) + r_j^s \leq r_l$. Each local spherical triangulation is now exactly a portion of the global spherical Delaunay triangulation.
To obtain optimal load balancing and robustness, the target domain should be decomposed based on the given point-density function. A precomputed ultra-coarse CVT is then a natural choice for effecting the partition. The main challenge is now the determination of the sizes of the overlaps. The overlapping regions should be large enough to make sure no true Delaunay triangles are missing in the final triangulation. A simple overlapping decomposition for quasi-uniform meshes is to use disk regions centered at partition cell centers with radius being the maximum distance from the center to its adjacent centers. More challenges exist for variable resolution meshes. In this work, we take the Voronoi sort method, i.e., the union of the owned partition cell and its immediately adjacent partition cells defines the overlapping domain decomposition for parallel triangulation. Failure may occur when the global point set does not include enough points for the decomposition of choice. A practical suggestion on the number of grid points is at least 16 multiples of the number of partitions, or a number corresponding to at least two-level bisections from the partition CVT, as pointed out in [12].

3.2.2. Parallelization of Lloyd P-LBFGS

To parallelize the Lloyd P-LBFGS scheme, we first transform the global point vector $Z = (z_i)_{i=1}^K$ into non-overlapping local vectors, so that arithmetic operations like additions and dot products can be efficiently applied on the distributed vectors. This non-overlapping decomposition can be initially carried out according to the disjoint partition cells $C_1, \ldots, C_p$; see Figure 2-left. On each processor, the global IDs of points in the distributed vector are recorded and will be used for vector assembly after point updating. For any point, say $z_i$, if it is initially in partition cell $l$ by disjoint partition, we call cell $l$ its arithmetic position. As the iterations proceed, $z_i$ could move from one partition cell to another, and we call the current partition cell it belongs to as its geometric position. Whereas the geometric position could vary, we always fix the arithmetic position for convenience of computations inside LBFGS.

![Figure 2](image)

**Figure 2:** The global point vector is initially decomposed by non-overlapping partitions. We assume Voronoi points move at most into their neighboring partition cells; if a point moves further, reset LBFGS.

The evaluations of the energy function, the gradient, and the Lloyd preconditioner require integrations over Voronoi cells. All integrations are actually computed based on the dual Delaunay triangulation. Given a triangle $\triangle(z_a, z_b, z_c)$ in the Delaunay tessellation, with its circumcenter $c$ and three edge midpoints $z_{ab}, z_{bc}$, and $z_{ca}$, the triangle can be split into six sub-triangles, each made of an original triangle vertex, an edge midpoint, and the circumcenter. The sub-triangle $\triangle(z_a, z_{ab}, c)$, for instance, is part of the Voronoi cell whose generator is located at $z_a$. Should the
The orientation of sub-triangles be taken into account, an integration over a Voronoi cell is the sum of integration over such sub-triangles inside the cell.

The evaluation of the functional gradient and the Lloyd preconditioner described above are performed in parallel and stored according to the geometric positions of grid points. To be clear, we will call the partition cell associated with which current processor is handling as “my cell”, and the union with its overlapped layers as “my region”. After points are assigned into “my region”, local Delaunay triangulation is performed. We then integrate over “my cell” for energy function evaluation. In the meanwhile, at each point in “my cell”, we compute the function gradient and the Lloyd preconditioner. Non-blocking communication is then used to assemble their arithmetic vectors, whose distribution is the same as the non-overlapping partition of the initial global points.

It is worth noting that in our code there is no need for global gathering of local Delaunay triangulations or updated mesh points (Voronoi generators). The communications for updating mesh points are only among two-level neighboring partition cells, under the assumption that grid points move at most into their neighboring partition cells. Here the “two-level” approach is to ensure the updating of points in overlapped layers. The assumption is generally fulfilled in practice by consecutively performing optimization-bisection until the final level of optimization; this is detailed in next section. As such, the communications for assembling the gradient and preconditioner are only in one level, i.e., only among neighboring partition cells. When the assumption is not satisfied, a reset on P-LBFGS is executed (see the illustration in Figure 2, right), i.e., the arithmetic position for each mesh point is re-computed.

3.3. Description of the parallel algorithm

There are still several ingredients that need to be considered for parallel P-LBFGS for SCVT construction, such as what are the point-density function, the initialization, and the stopping criteria.

The point-density function $\rho(x)$ plays a crucial role in how the converged mesh points are distributed. For the numerical tests in this paper, we use analytic point-density functions that feature significant physically relevant information drawn from climate modeling. A point-density function of course also can be formulated from a given initial mesh, or even from a local mesh of a subregion of the sphere.

The convergence of iterative methods is often highly sensitive to the initial configuration. Exploring a good initialization for SCVT computation has been an interesting topic in the field. To the best of our knowledge, a Monte Carlo initialization based on the point-density function is often suitable for coarse mesh generation. For high-resolution meshes, however, a consecutive optimization-bisection procedure is almost mandatory to obtain an initial guess that is good enough to save computational cost.

The stopping criteria for P-LBFGS here are based on the maximum number of iterations, the norm of gradient of the energy function, and the step size of mesh point movement.

In summary, the parallel version of the Lloyd P-LBFGS algorithm for SCVT construction consists of the following steps.

1: Build the partition information from a coarse CVT;
2: Set $n \leftarrow 0$;
3: Decompose $Z^{(n)}$ into disjoint partition cells accordingly; store global IDs of points;
4: while stopping criteria not satisfied do
5: Preprocess:
   assign points from $Z^{(n)}$ into “my region” (2-level neighboring comm.);
go to Step 3 if a point in $\mathbf{Z}^{(n)}$ moves out of its initial neighboring partition cells for points in “my region”, build local Delaunay triangulation;
6: For points in “my cell”, compute $\mathcal{F}(\mathbf{Z}^{(n)})$, $\nabla \mathcal{F}(\mathbf{Z}^{(n)})$, $\mathbf{H}_n^{(0)}$;
7: Sum $\mathcal{F}(\mathbf{Z}^{(n)})$; assemble $\nabla \math(\mathbf{Z}^{(n)})$, $\mathbf{H}_n^{(0)}$ to arithmetic positions (1-level neighboring comm. )
8: Compute search direction $\mathbf{q}$ by P-LBFGS;
9: Update $\mathbf{Z}^{(n+1)} = \mathbf{Z}^{(n)} + \alpha^{(n)} \mathbf{q}$ by line search:
occasionally perform Step $3$ on $\mathbf{Z}^{(n)} + s \mathbf{q}$ and evaluate $\mathcal{F}$, for some step size $s$.
10: $n \leftarrow n + 1$
11: end while
12: Gather all updated points;
13: Build local Delaunay triangulation and then gather all triangles (non-repeatedly).

4. Performance tests

Our code for the Lloyd-preconditioned LBFGS solver for SCVT construction is written in C++ and MPI; the Delaunay triangulation is handled by the “Triangle” package, the LBFGS routine is from the serial HLBFGS package by Yang Liu (in which the line-search routine is converted from the original LBFGS Fortran code by Jorge Nocedal). We set the maximum number of function evaluations for the line search to ten and store at most seven BFGS corrections in memory ($M = 7$), as suggested in [15]. The SCVT grid generator in this study is under active development and the code is available at https://github.com/hyang52/scvt_HLBFGS.

The quality of the Voronoi cell $V$ of the generated meshes is quantified by $\text{cellQ}$

$$\text{cellQ}(V) = \frac{\min \text{ cell edge}}{\max \text{ cell edge}}$$

and the triangle quality $\text{triQ}$, which is defined on an triangle $\triangle$ with edge lengths $a, b, c$ by

$$\text{triQ}(\triangle) = \frac{(a + b - c)(b + c - a)(c + a - b)}{abc}. \quad (9)$$

Three different point-density functions defined on the unit sphere are used in our tests to represent some typical cases. For each point-density function, we use the max-min ratio of the point-density to control the max-min ratio of mesh size because it is conjectured that for any two Voronoi cells $V_i$ and $V_j$ from a SCVT, their mesh spacings are related to the point-density function as

$$\frac{h_i}{h_j} \approx \left( \frac{\rho(z_j)}{\rho(z_i)} \right)^{1/4}.$$ 

The first point-density function we used is given by

$$\rho(x, y, z) = (1 - \gamma)z^4 + \gamma$$

with $\gamma = (1/3)^4$, so that the mesh size on the equator is three times of that on the poles. This size distribution is typically used in some simulations of ocean modeling. We will call it the X3 point-density; see the SCVT mesh in the left column of Figure 3.

The other two point-density functions are defined as

$$\rho(z_i) = \frac{1}{2(1 - \gamma)} \left[ \tanh \left( \frac{\beta - d(z_i)}{\alpha} \right) + 1 \right] + \gamma,$$
where \( d(z_i) \) is a distance function that can be simply the geodesic distance from \( z_i \) to a pre-specified fixed point \( z_c \) on the sphere. This definition results in a relatively large value of \( \rho \) within a distance \( \beta \) from the point \( z_c \). The point-density transitions from a large value to small value across a geodesic distance of \( \alpha \). To have a variety of tested densities, we define \( d(z_i) \) in different ways for the second and third point-density functions. Specifically, for the second point-density function, we define

\[
d(z_i) = \left( \frac{|z_i - z_{ic}|_S^2}{w_{lon}^2} + \frac{|z_i - z_{ci}|_S^2}{w_{lat}^2} \right)^{1/2},
\]

where \( z_{ic} \) is a point whose latitude and longitude coincide with \( z_i \) and \( z_c \), respectively, and vice versa for \( z_{ci} \). The norm \( |\cdot|_S \) denotes geodesic distance. We choose the widths \( w_{lon} \) and \( w_{lat} \) to be 0.3 and 1.2 with \( z_c = (1.0, 0.0, 0.0) \), and set \( \beta = \pi/6 \) and \( \alpha = 0.3 \). The minimum point-density is set to be \( \gamma = (1/16)^4 \), so we refer to the defined function as the X16 point-density; see the SCVT grid in the middle column of Figure 3.

For the third point-density, we take a function used in [19] for a multi-resolution modeling with the shallow water equations. The distance is now \( d(z_i) = |z_i - z_c|_S \) with \( z_c = (0.0, -0.866, 0.5) \), the parameters are taken as \( \beta = \pi/6 \) and \( \alpha = 0.15 \). Here, we set the minimum point-density \( \gamma = (1/64)^4 \) and refer to the function as the X64 point-density; see the SCVT grid in the right column of Figure 3. There are two reasons of using this X64 point-density. On the one hand, it corresponds to a highly variable multi-resolution mesh that is needed in global ocean and coastal system modeling for which the mesh size may range from a \( O(10^4) \) km global scale to \( O(10^{-1}) \) km local scale. On the other hand, it can be used in global spherical mesh generation for the purpose for limited-area climate modeling, if one culls the coarse portion from the global one.

Figure 3: The SCVT meshes with 2,562 generators created by Lloyd P-LBFGS with three point-density functions: X3, X16, X64 (from left to right).

4.1. Comparison of Lloyd, LBFGS and Lloyd P-LBFGS in serial

Lloyd’s method is the best-known algorithm for SCVT construction and LBFGS is a sound way to speed up the Lloyd iteration, but we believe that the Lloyd P-LBFGS method will further speed up LBFGS, especially for highly variable multi-resolution meshes. To compare the performance of these three methods, we execute a sequence of tests for different scenarios for serial implementations. In these tests, we only work on coarse meshes with 2562 and 10242 SCVT generators and all the runs
are effected on a PC with a 2.70GHz Intel i7-3740QM CPU. The mesh points are initialized by Monte Carlo sampling based on the corresponding point-density functions. To compare both the efficiency and the mesh quality, we set the maximum number of iterations to be 2,000 and the tolerance on the size of the point movement to be $5 \times 10^{-4}$. Except for that, the quasi-Newton iterations will stop when $\|\nabla F^n\|/F^n \leq 5 \times 10^{-4}$ or $|F^n - F^{n-1}|/F^{n-1} < 10^{-7}$. The tolerance $10^{-7}$ on the control of the energy decrease is to avoid wandering. For the integration over a triangle, we use a 4-point quadrature rule in the X3 and X16 cases, and a 9-point quadrature in the X64 case which includes extremely coarse cells. Table 1 provides detailed information of the three tested methods with 2,562 generators. Because the SCVT energy functional has many local minimizers, different optimization methods with even the same initial guess can still lead to different minimizers. For this reason, the final gradient norm $\|\nabla F\|$ plays a role in judging the quality of final SCVT mesh.

| Density | Method       | # iter. | # $F$ eval. | Time (sec.) | Final $F$ | Final $\|\nabla F\|$ |
|---------|--------------|---------|-------------|-------------|-----------|-----------------|
| X3      | Lloyd        | 1235    | —           | 113.869     | 1.32665e-03 | 2.6344e-05 |
|         | LBFGS        | 204     | 207         | 19.965      | 1.32598e-03 | 5.4127e-07 |
|         | Lloyd P-LBFGS| 170     | 171         | 16.702      | 1.32537e-03 | 1.9154e-06 |
| X16     | Lloyd        | 2000    | —           | 255.361     | 3.91468e-05 | 8.0502e-07 |
|         | LBFGS        | 787     | 813         | 111.521     | 3.90003e-05 | 9.5421e-08 |
|         | Lloyd P-LBFGS| 309     | 334         | 45.230      | 3.89749e-05 | 5.6632e-08 |
| X64     | Lloyd        | 1977    | —           | 338.103     | 1.04809e-04 | 2.1095e-06 |
|         | LBFGS        | 1201    | 1290        | 236.211     | 1.08224e-04 | 1.6599e-06 |
|         | Lloyd P-LBFGS| 257     | 261         | 47.443      | 1.04699e-04 | 8.6221e-08 |

Table 1 shows that for all three point-density functions, Lloyd P-LBFGS is much more efficient than LBFGS and Lloyd (1.18 to 5.02 times faster than LBFGS and 5.66 to 7.19 times faster than Lloyd). We also observed that the relative efficiency of Lloyd P-LBFGS becomes even better in the case with 10,242 generators, as illustrated in Table 2. Among the three methods Lloyd P-LBFGS always returns the smallest value on the energy function and nearly so for the gradient norm, which usually mean better mesh quality. The speed-up of LBFGS by the Lloyd preconditioner is more obvious as we increase the mesh size ratio; this is also shown by the plots of the iteration histories in Figure 4 which provides the numerical results in terms of the energy functional and the gradient norm with respect to the number of iterations and the computation time.

SCVT meshes with 2,562 generators generated by Lloyd P-LBFGS are plotted in Figure 3 from which we can see that the mesh points are very regularly distributed and that the mesh transitions from coarse to fine regions are very smooth. The quality of SCVT meshes can also be characterized by the triangle quality measure ranging from 0 to 1; see the definition in (9). In Figure 5, we plot the distribution of the triangle quality of the SCVT meshes with 10,242 generators produced by the three methods; we here only plot the case of X3 and X16 with 10,242 generators because the X64 case has some very triangular cells that are too coarse to be properly visualized by the Paraview software. In this figure, the lighter the color, the better the quality. The results show that in general both LBFGS and Lloyd P-LBFGS produce better SCVT meshes in term of quality than Lloyd and for the X16 case, the Lloyd P-LBFGS clearly provides meshes with better quality than other two
Table 2: Comparison of the Lloyd, LBFGS, and Lloyd P-LBFGS methods in serial with 10,242 generators.

| Density | Method       | # iter. | # $\mathcal{F}$ eval. | Time (sec.) | Final $\mathcal{F}$ | Final $\|\nabla \mathcal{F}\|$ |
|---------|--------------|---------|------------------------|-------------|---------------------|-----------------------------|
| X3      | Lloyd        | 2000    | —                      | 851.965     | 3.31424e-04         | 3.3334e-06                  |
|         | LBFGS        | 347     | 354                    | 145.716     | 3.31306e-04         | 1.6461e-07                  |
|         | Lloyd P-LBFGS| 315     | 316                    | 130.262     | 3.31325e-04         | 1.5729e-07                  |
| X16     | Lloyd        | 2000    | —                      | 1138.490    | 9.93638e-06         | 1.1110e-07                  |
|         | LBFGS        | 1608    | 1621                   | 1089.000    | 9.76267e-06         | 2.8646e-08                  |
|         | Lloyd P-LBFGS| 533     | 534                    | 335.890     | 9.75698e-06         | 4.8184e-09                  |
| X64     | Lloyd        | 2000    | —                      | 1406.740    | 2.65790e-05         | 3.6220e-07                  |
|         | LBFGS        | 2000    | 2024                   | 1504.600    | 2.73630e-05         | 1.1220e-07                  |
|         | Lloyd P-LBFGS| 405     | 417                    | 320.680     | 2.61190e-05         | 1.7556e-08                  |

4.2. Comparison of Lloyd and graph Laplacian preconditioners in P-LBFGS in serial

We next make a direct performance comparison of the Lloyd preconditioner with the graph Laplacian preconditioner in the P-LBFGS scheme. The graph Laplacian is computed with a Cholesky decomposition solver as done in [9]. Because the Laplacian matrix is singular, the numerical solution is unique up to a constant; depending on which graph Laplacian preconditioner is used, the corresponding P-LBFGS method could perform very differently. Some solutions of the Laplacian system may lead to a direction with a poor length for line search or even a non-decent direction. As a result, we will handle the singularity of the Laplacian operators by a diagonal perturbation. Two approaches are tested. One is the diagonal shifting by adding scaled identity: $A \leftarrow A + 10^{-6}I$. We call this approach Laplacian(a6). The other one is to multiply the diagonal entries by a constant factor: $A_{ii} \leftarrow A_{ii} \cdot (1 + 10^{-2})$. This method is denoted by Laplacian(m2). We apply the graph Laplacian preconditioner in each iteration so that the numerical results can represent, in principle, the best performance of Laplacian preconditioner.

It is worth mentioning that the point-density function in [9, Example 4] has min-max ratio $e^{-80}$, thus the mesh size ratio is as huge as $e^{20}$; similarly in [9, Example 5], the mesh size ratio is $e^{2.5} \approx 12.18$. Our testing densities X16 and X64 are in a category comparable to [9]. In the following tests, the parameter configuration is exactly the same as in Section 4.1. We investigate, in different scenarios, not only the necessary number of iterations for convergence but also the final mesh quality which is an even higher priority in many applications [19, 21].

Detailed test results are provided in Table 3 and Figure 6. We see that Laplacian(m2) has the overall fastest convergence rate if we only focus on the number of iterations without mentioning the computational time. Laplacian(a6) requires fewer iteration than the Lloyd preconditioner in the X3 test case, whereas not significantly fewer in the X16 case, and, in the X64 test case, it requires more iterations. Thus, we take Laplacian(m2) as a better representative for the graph Laplacian preconditioners proposed in [9]. The graph Laplacian preconditioner can dramatically reduce the necessary number of iterations in P-LBFGS; this is their main advantage as it was proposed. A natural concern is the computational cost incurred by having to solve discrete Laplacian systems. We observe that with respect to computational time, the Lloyd preconditioner costs much less time.
Figure 4: Plots of the performance of the Lloyd, LBFGS, and Lloyd P-LBFGS methods in serial with 2,562 generators corresponding to Table 1. From top to bottom: \( \mathcal{F} \) vs. computational time, \( \|\nabla \mathcal{F}\| \) vs. computational time, \( \|\nabla \mathcal{F}\| \) vs. # of iterations; from left to right: X3, X16, X64 densities.

in all test cases. Another aspect which we care even more about is mesh quality. As one can see from Table 3 and Figure 6, Laplacian(m2) always returns larger values than Lloyd P-LBFGS for the final energy and the gradient norm. We also find that in this set of tests Laplacian(m2) actually always stops based on the criterion of \( |\mathcal{F}^n - \mathcal{F}^{n-1}|/\mathcal{F}^{n-1} \) rather than on the gradient criterion. It seems the final energies only have little differences, but the mesh qualities are significantly different, as illustrated by the plot of cell quality in the third row of Figure 6. We conclude that Lloyd preconditioner outperforms the graph Laplacian(m2) at providing a SCVT grid with “optimal” quality and computational times.

4.3. Parallel scalability of the Lloyd P-LBFGS

Now we study the scaling performance of the parallel P-LBFGS method. All the parallel tests were run on a high performance computing (HPC) cluster with 16 Intel Xeon “model E5-2670” cores per node.

We first present in Figure 7 the speedup of parallelization with varying number of generators (163,842, 655,362, 2,621,442) and different point-density functions. The speedup is measured
Figure 5: Distribution of triangle qualities of the produced SCVT meshes with 10,242 generators. From top to bottom: X3, X16 densities; from left to right: Lloyd, LBFGS, Lloyd P-LBFGS.

Table 3: Comparison of the Lloyd and graph Laplacian preconditioners for P-LBFGS with 2,562 generators.

| Density | Method            | # iter. | # $\mathcal{F}$ eval. | Time(sec) | Final $\mathcal{F}$ | Final $\|\nabla \mathcal{F}\|$ |
|---------|-------------------|---------|------------------------|-----------|----------------------|--------------------------|
| X3      | Laplacian(a6) P-LBFGS | 73      | 93                     | 29.503    | 1.32712e-03          | 7.2298e-06               |
|         | Laplacian(m2) P-LBFGS | 104     | 106                    | 32.805    | 1.32664e-03          | 2.5014e-06               |
|         | Lloyd P-LBFGS      | 170     | 171                    | 16.701    | 1.13253e-03          | 1.9154e-06               |
| X16     | Laplacian(a6) P-LBFGS | 255     | 262                    | 90.697    | 3.89986e-05          | 6.0327e-08               |
|         | Laplacian(m2) P-LBFGS | 197     | 205                    | 70.725    | 3.90087e-05          | 4.4906e-07               |
|         | Lloyd P-LBFGS      | 309     | 334                    | 45.229    | 3.89749e-05          | 5.6632e-08               |
| X64     | Laplacian(a6) P-LBFGS | 329     | 463                    | 179.691   | 1.04747e-04          | 1.8575e-07               |
|         | Laplacian(m2) P-LBFGS | 172     | 202                    | 78.739    | 1.04720e-04          | 1.5299e-07               |
|         | Lloyd P-LBFGS      | 257     | 261                    | 47.443    | 1.04699e-04          | 8.6221e-08               |

using the cost per iteration. It is observed that in generating relatively small meshes, for instance 163,842 generators, communication seems to dominate the overall execution time. That is why the
scability ends up being sub-linear. As the number of generators increases, the speedup gets better and better. When the number of grid points is in million scale, the parallel solver features almost linear scaling up to 256 processors.

We next show the scaling results towards generating the SCVTs with 655,362 generators. Initialized from a convergent CVT mesh of 2,562 generators, our code consecutively bisects and optimizes the mesh until a converged SCVT with 655,362 generators is obtained. The iteration history is recorded for the 655,362 generator mesh and also the 40,962 generator mesh generated along the consecutive bisection procedure. To measure the parallel efficiency, we launch the program with different number of processors ranging from 1 to 128. Figure 8 shows the log-plot of the running time $t$ against the iteration number $n$. As they are related by $t = a \cdot n + c$, when $n$ is relatively large we can approximate $\log t \approx \log a + \log n$. Thus, the distance between the right ends of adjacent curves reflects the gain on time efficiency per iteration when the number of processors is doubled. Excellent scalability is confirmed because those distances are fairly evenly distributed. The plots of $F$ and $\|\nabla F\|$ vs. the running time for the 40,962 points mesh are shown in Figure 9.
Figure 7: Speedups of the parallel Lloyd P-LBFGS under different number of generators. From left to right: X3, X16, X64 densities.

Figure 8: Plots of running time vs. # of iterations of the parallel Lloyd P-LBFGS with the X3 point-density. Left: 40,962 generators; right: 655,362 generators.

Finally, in Figure 10, we plot the distribution of triangle qualities of the SCVT meshes with 655,362 generators created by the parallel Lloyd P-LBFGS method with 128 processors. The total running times are respectively 1296.4 seconds for X3 point-density, 1939.6 seconds for X16 point-density, and 1914.9 seconds for X64 point-density.

5. Summary

A Lloyd-preconditioned LBFGS method and its parallel implementation are proposed and numerically investigated for SCVT computation. Whereas LBFGS often speeds up the CVT computation as compared to the classical Lloyd method, it may lose “optimal efficiency” in generating highly variable multi-resolution meshes. The Lloyd-preconditioned LBFGS method can further speeds up the CVT computation and overcome this difficulty in large scale SCVT computation.

For quasi-uniform meshes, the Lloyd-preconditioned LBFGS method performs similarly as LBFGS, which is also the case when using the graph Laplacian preconditioner. For typical multi-resolution
Figure 9: Strong scaling efficiency check of the parallel Lloyd P-LBFGS with 40,962 generators using X3 point-density: plot of $F$-vs. running time.

Figure 10: Distribution of triangle qualities of the SCVT meshes with 655,362 generators created by the parallel Lloyd P-LBFGS using 128 processors. From left to right: X3, X16, X64 densities.

meshes, however, the Lloyd-preconditioned LBFGS method significantly improves the performance of LBFGS, while also producing better mesh quality. Because the method enables efficient SCVT generation in highly variable multi-resolution, it is also useful for limited-area climate modeling by culling coarse grids. Note that it is often challenging to maintain regularity of meshes at boundaries by direct local mesh generation.

The parallel Lloyd-preconditioned LBFGS algorithm features well-balanced loading of grid points and performs well in the parallel scalability tests. It enables more convenient (faster mesh generation) and more stable (better mesh quality) global climate modeling, especially for meshes with
ultra high-resolution. The proposed efficient CVT grid generator will thus play a fundamental role in next generations of climate modeling, where the spacing scale could be as small as 100 meters. Our interests will also include more accurate numerical simulations in ocean and atmosphere modeling based on SCVTs.

References

[1] M. Benzi, M. Tuma, A comparative study of sparse approximate inverse preconditioners, Applied Numerical Mathematics 30 (1999) 305–340.

[2] P. Cignoni, C. Montani, R. Scopigno, Dewall: A fast divide and conquer Delaunay triangulation algorithm in ed, Computer-Aided Design 30 (1998) 333–341.

[3] Q. Du, M. Emelianenko, Acceleration schemes for computing centroidal Voronoi tessellations, Numer. Linear Algebra Appl. 13 (2006) 173–192.

[4] Q. Du, M. Emelianenko, L. Ju, Convergence of the lloyd algorithm for computing centroidal Voronoi tessellations, SIAM journal on numerical analysis 44 (2006) 102–119.

[5] Q. Du, V. Faber, M. Gunzburger, Centroidal Voronoi tessellations: Applications and algorithms, SIAM review 41 (1999) 637–676.

[6] Q. Du, M. Gunzburger, Grid generation and optimization based on centroidal Voronoi tessellations, Applied Mathematics and Computation 133 (2002) 591–607.

[7] Q. Du, M. Gunzburger, L. Ju, Constrained centroidal Voronoi tessellations for surfaces, SIAM Journal on Scientific Computing 24 (2003) 1488–1506.

[8] P. Dymond, J. Zhou, X. Deng, A 2-D parallel convex hull algorithm with optimal communication phases, Parallel Computing 27 (2001) 243–255.

[9] J. Hateley, H. Wei, L. Chen, Fast methods for computing centroidal Voronoi tessellations, Journal of Scientific Computing 63 (2015) 185–212.

[10] M. Iri, K. Murota, T. Ohya, A fast Voronoi-diagram algorithm with applications to geographical optimization problems, in: System Modelling and Optimization, Springer, 1984, pp. 273–288.

[11] D. Jacobsen, Parallel Grid Generation and Multi-Resolution Methods for Climate Modeling Applications, Ph.D. thesis, Florida State University, 2011.

[12] D. Jacobsen, M. Gunzburger, T. Ringler, J. Burkardt, J. Peterson, Parallel algorithms for planar and spherical Delaunay construction with an application to centroidal Voronoi tessellations, Geoscientific Model Development 6 (2013) 1353–1365.

[13] L. Jiang, R. Byrd, E. Eskow, R. Schnabel, A preconditioned L-BFGS algorithm with application to molecular energy minimization, Technical Report, DTIC Document, 2004.
[14] L. Ju, Q. Du, M. Gunzburger, Probabilistic methods for centroidal Voronoi tessellations and their parallel implementations, Parallel Computing 28 (2002) 1477–1500.

[15] Y. Liu, W. Wang, B. Lévy, F. Sun, D. Yan, L. Lu, C. Yang, On centroidal Voronoi tessellation: Energy smoothness and fast computation, ACM Transactions on Graphics 28 (2009) 101.

[16] S. Lloyd, Least squares quantization in pcm, IEEE transactions on information theory 28 (1982) 129–137.

[17] J. MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, Oakland, CA, USA, pp. 281–297.

[18] J. Nocedal, S.J. Wright, Numerical Optimization, 2nd ed., Springer, New York, 2006.

[19] T. Ringler, D. Jacobsen, M. Gunzburger, L. Ju, M. Duda, W. Skamarock, Exploring a multiresolution modeling approach within the shallow-water equations, Monthly Weather Review 139 (2011) 3348–3368.

[20] T. Ringler, M. Petersen, R. Higdon, D. Jacobsen, P. Jones, M. Maltrud, A multi-resolution approach to global ocean modeling, Ocean Modelling 69 (2013) 211–232.

[21] K. Sakaguchi, L. Leung, C. Zhao, Q. Yang, J. Lu, S. Hagos, S. Rauscher, L. Dong, T. Ringler, P. Lauritzen, Exploring a multiresolution approach using AMIP simulations, Journal of Climate 28 (2015) 5549–5574.

[22] K. Sakaguchi, J. Lu, L. Leung, C. Zhao, Y. Li, S. Hagos, Sources and pathways of the upscale effects on the southern hemisphere jet in MPAS-CAM4 variable-resolution simulations, Journal of Advances in Modeling Earth Systems 8 (2016) 1786–1805.

[23] J. Shewchuk, Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, in: Applied Computational Geometry Towards Geometric Engineering, Springer Berlin Heidelberg, 1996, pp. 203–222.

[24] C. Zhao, L. Leung, S. Park, S. Hagos, J. Lu, K. Sakaguchi, J. Yoon, B. Harrop, W. Skamarock, M. Duda, Exploring the impacts of physics and resolution on aqua-planet simulations from a nonhydrostatic global variable-resolution modeling framework, Journal of Advances in Modeling Earth Systems 8 (2016) 1751–1768.