Lecture 10

Partitioning and Load Balancing

Handling a large mesh or a linear system on a supercomputer or on a workstation cluster usually requires that the data for the problem be somehow partitioned and distributed among the processors. The quality of the partition affects the speed of solution: a good partition divides the work up evenly and requires as little communication as possible. Unstructured meshes may approximate irregular physical problems with fewer mesh elements, their use increases the difficulty of programming and requires new algorithms for partitioning and mapping data and computations onto parallel machines. Partitioning is also important for VLSI circuit layout and parallel circuit simulation.

10.1 Motivation from the Parallel Sparse Matrix Vector Multiplication

Multiplying a sparse matrix by a vector is a basic step of most commonly used iterative methods (conjugate gradient, for example). We want to find a way of efficiently parallelizing this operation.

Say that processor i holds the value of v_i. To update this value, processor need to compute a weighted sum of values at itself and all of its neighbors. This means it has to first receiving values from processors holding values of its neighbors and then computing the sum. Viewing this in graph terms, this corresponds to communicating with a node's nearest neighbors.

We therefore need to break up the vector (and implicitly matrix and graph) so that:

- We balance the computational load at each processor. This is directly related to the number of non-zero entries in its matrix block.
- We minimize the communication overhead. How many other values does a processor have to receive? This equals the number of these values that are held at other processors.

We must come up with a proper division to reduce overhead. This corresponds to dividing up the graph of the matrix among the processors so that there are very few crossing edges. First assume that we have 2 processors, and we wish to partition the graph for parallel processing. As an easy example, take a simplistic cut such as cutting the 2D regular grid of size n in half through the middle. Let's define the cut size as the number of edges whose endpoints are in different groups. A good cut is one with a small cut size. In our example, the cut size would be $\sqrt{n}$. Assuming that the cost of each communication is 10 times more than an local arithmetic operation. Then the total parallel cost of perform matrix-vector product on the grid is $(4n)/2 + 10\sqrt{n} = 2n + 10\sqrt{n}$. In general, for p processors, to need to partition the graph into p subgraphs.

Various methods are used to break up a graph into parts such that the number of crossing edges is minimized. Here we'll examine the case where p = 2 processors, and we want to partition the graph into 2 halves.

10.2 Separators

The following definitions will be of use in the discussion that follows. Let $G = (V, E)$ be an undirected graph.

- A **bisection** of G is a division of V into V_1 and V_2 such that $|V_1| = |V_2|$. (If $|V|$ is odd, then the cardinalities differ by at most 1). The **cost** of the bisection (V_1, V_2) is the number of edges connecting V_1 with V_2.
- If $\beta \in [1/2, 1)$, a β-**bisection** is a division of V such that $V_{1,2} \leq \beta|V|$.
- An **edge separator** of G is a set of edges that if removed, would break G into 2 pieces with no edges connecting them.
- A p-**way partition** is a division of V into p pieces V_1,V_2,...,V_p where the sizes of the various pieces differ by at most 1. The cost of this partition is the total number of edges crossing the pieces.
- A **vertex separator** is a set C of vertices that break G into 3 pieces A, B, and C where no edges connect A and B. We also add the requirement that A and B should be of roughly equal size.

Usually, we use edge partitioning for parallel sparse matrix-vector product and vertex partitioning for the ordering in direct sparse factorization.

10.3 Spectral Partitioning – One way to slice a problem in half

The three steps to illustrate the solution of that problem are:

A. Electrical Networks (for motiving the Laplace matrix of a graph)

B. Laplacian of a Graph

C. Partitioning (that uses the spectral information)

10.3.1 Electrical Networks

As an example we choose a certain configuration of resistors (1 ohm), which are combined as shown in fig. 10.1. A battery is connected to the nodes 3 and 4. It provides the voltage V_B. To obtain the current, we have to calculate the effective resistance of the configuration

$$I = \frac{V_B}{\text{eff. res.}}. \tag{10.1}$$

This so called *Graph* is by definition a *collection of nodes and edges.*

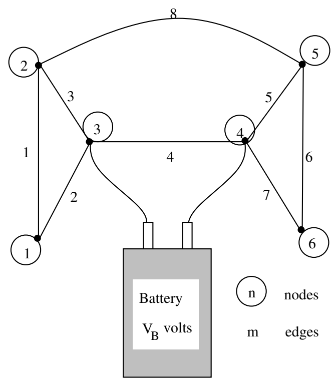


Figure 10.1: Resistor network with nodes and edges

10.3.2 Laplacian of a Graph

Kirchoff's Law tells us

$$\begin{pmatrix} 2 & -1 & -1 & & & \\ -1 & 3 & -1 & & -1 & \\ -1 & -1 & 3 & -1 & & \\ & & -1 & 3 & -1 & -1 \\ & -1 & & -1 & 3 & -1 \\ & & & -1 & -1 & 2 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \tag{10.2}$$

The matrix contains the information of how the resistors are connected, it is called the *Laplacian of the Graph*

$$\nabla_G^2 = n \times n \quad \text{matrix}, \quad \begin{array}{l} \text{G} \ldots \text{graph} \\ \text{n} \ldots \text{nodes} \\ \text{m} \ldots \text{edges} \end{array} \tag{10.3}$$

$$\left(\nabla_G^2\right)_{ii} = \text{degree of node } i \tag{10.4}$$

$$\left(\nabla_G^2\right)_{ij} = \begin{cases} 0 & \text{if } \nexists \text{ edge between node } i \text{ and node } j \\ -1 & \text{if } \exists \text{ edge between node } i \text{ and node } j \end{cases} \tag{10.5}$$

Yet there is another way to obtain the Laplacian. First we have to set up the so called *Node–edge Incidence matrix*, which is a $m \times n$ matrix and its elements are either 1, -1 or 0 depending on whether the edge (row of the matrix) connects the node (column of matrix) or not. We find

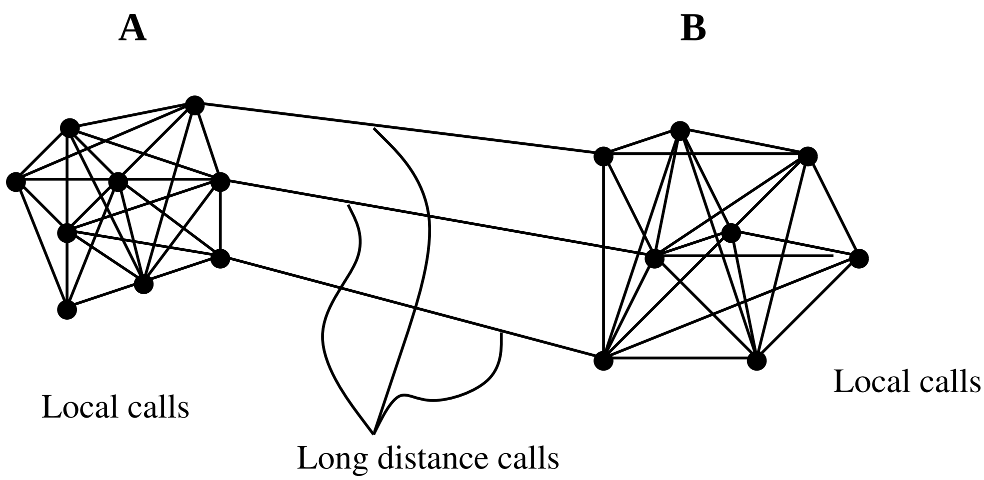


Figure 10.2: Partitioning of a telephone net as an example for a graph comparable in cost

$$
\mathbf{M}_G = \begin{array}{c} \text{nodes} \\ \text{edges} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{pmatrix}
 & & & & & \\
1 & -1 & & & & \\
1 & & -1 & & & \\
 & 1 & -1 & & & \\
 & & 1 & -1 & & \\
 & & & 1 & -1 & \\
 & & & & 1 & -1 \\
 & & & 1 & & -1 \\
 & 1 & & & -1 &
\end{pmatrix} \end{array} \quad (10.6)
$$

$$\mathbf{M}_G^T \mathbf{M}_G = \nabla_G^2 \quad (10.7)$$

10.3.3 Spectral Partitioning

Partitioning means that we would like to break the problem into 2 parts A and B, whereas the number of connections between both parts are to be as small as possible (because they are the most expensive ones), see fig. 10.2. Now we define a vector $\mathbf{x} \in \mathcal{R}^n$ having the values $\mathbf{x}_i = \pm 1$. $+1$ stands for *belongs to part A*, -1 means *belongs to part B*. With use of some vector calculus we can show the following identity

$$\sum_{i=1}^{n} (\mathbf{M}_G \mathbf{x})_i^2 = (\mathbf{M}_G \mathbf{x})^T (\mathbf{M}_G \mathbf{x}) = \mathbf{x}^T \mathbf{M}_G^T \mathbf{M}_G \mathbf{x} = \mathbf{x}^T \nabla_G^2 \mathbf{x} \quad (10.8)$$

$$\mathbf{x}^T \nabla_G^2 \mathbf{x} = 4 \times (\#\text{ edges between A and B}).$$

In order to find the vector $\mathbf{x}$ with the least connections between part A and B, we want to solve the minimization problem:

$$\left. \begin{array}{r} \text{Min. } \mathbf{x}^T \nabla_G^2 \mathbf{x} \\ x_i = \pm 1 \\ \sum x_i = 0 \end{array} \right\} \geq \left\{ \begin{array}{r} \text{Min. } \mathbf{x}^T \nabla_G^2 \mathbf{x} \\ x_i \in \mathcal{R}^n \\ \sum x_i^2 = n \\ \sum x_i = 0 \end{array} \right. \quad (10.9)$$

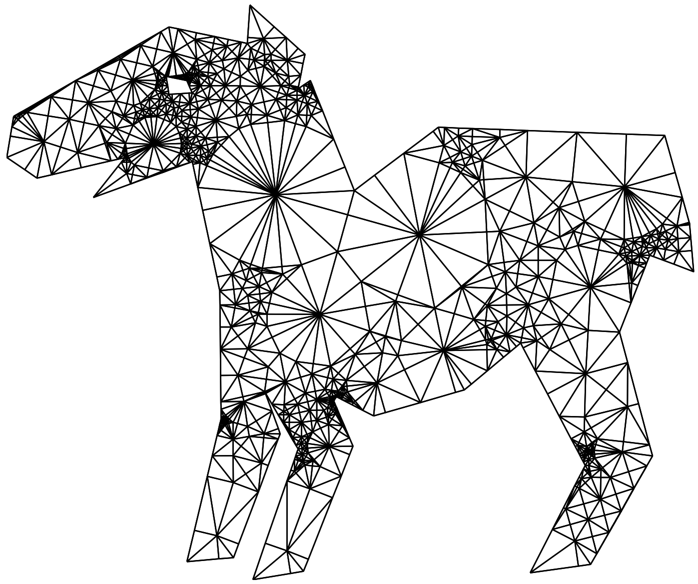


Figure 10.3: Tapir (Bern-Mitchell-Ruppert)

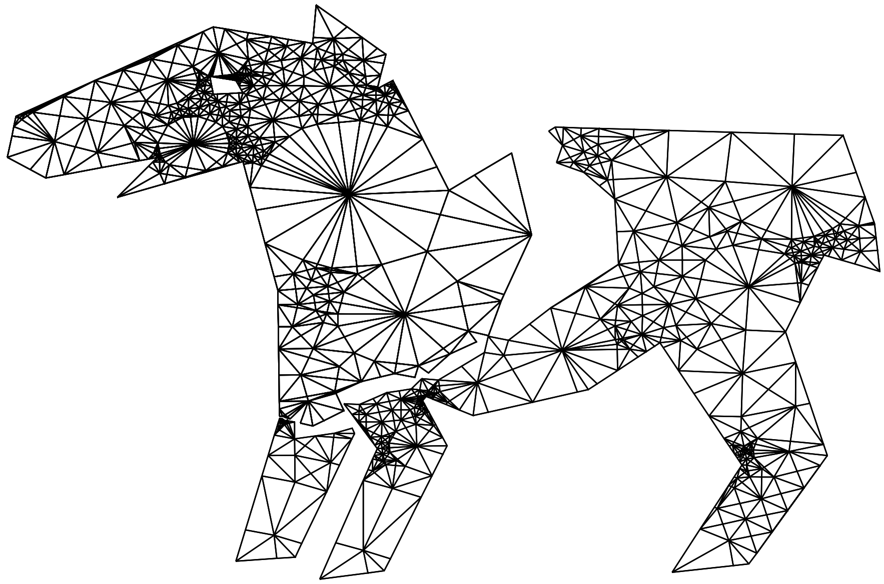
Finding an optimal ± 1 solution is intractable (NP-hard) in general. In practice, we relax the integer condition and allow the solution to be real. We do require the norm square of the solution, like the integer case, equal to n (see RHS of eqn. (10.9)). The relaxation enlarges the solution space, hence its optimal solution is no more than that of its integer counterpart. As an heuristic, we solve the relaxed version of the problem and "round" the solution to give an ± 1 solution.

The solution of the problem on the RHS of eqn. (10.9) gives us the second smallest eigenvalue of the Laplacian ∇_G^2 and the corresponding eigenvector, which is in fact the vector $\mathbf{x}$ with the least connections in the relaxation sense. To obtain a partition of the graph, we can divide its node set according to the vector x. We can even control the ratio of the partition. For example to obtain a bisection, i.e., a partition of each size, we can find the median of x and put those nodes whose x values is smaller than the median on one side and the remaining on the other side. We can also use the similar idea to divide the graph into two parts in which one is twice as big as the another.

Figure 10.3 shows a mesh together with its spectral partition. For those of you as ignorant as your professors, a tapir is defined by Webster as any of several large inoffensive chiefly nocturnal ungulates (family Tap iridae) of tropical America, Malaya, and Sumatra related to the horses and rhinoceroses. The tapir mesh is generated by a Matlab program written by Scott Mitchell based on a non-obtuse triangulation algorithm of Bern-Mitchell-Ruppert. The partition in the figure is generated by John Gilbert's spectral program in Matlab.

One has to notice that the above method is merely a heuristic. The algorithm does not come with any performance guarantee. In the worst case, the partition may be far from the optimal, partially due to the round-off effect and partially due the requirement of equal-sized partition. In fact the partition for the tapir graph is not quite optimal. Nevertheless, the spectral method, with the help of various improvements, works very well in practice. We will cover the partitioning problem more systematically later in the semester.

The most commonly used algorithm for finding the second eigenvalue and its eigenvector is the Lanczos algorithm though it makes far better sense to compute singular values rather than eigenvalues. Lanczos is an iterative algorithm that can converge quite quickly.



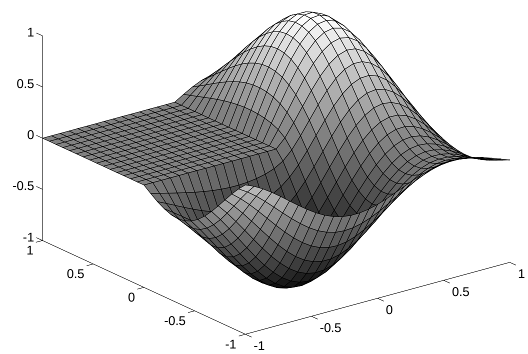


Figure 10.5: The 2nd eigenfunction of MathWork's logo

In general, we may need to divide a graph into more than one piece. The most commonly used approach is to recursively apply the partitioner that divides the graph into two pieces of roughly equal size. More systematic treatment of the partitioning problem will be covered in future lectures.

The success of the spectral method in practice has a physical interpretation. Suppose now we have a continuous domain in place of a graph. The Laplacian of the domain is the continuous counterpart of the Laplacian of a graph. The kth eigenfunction gives the kth mode of vibration and the second eigenfunction induce a cut (break) of the domain along the weakest part of the domain. (See figure10.5)

10.4 Geometric Methods

The spectral partitioning method only deals with the topology of the graph; the vertex locations are not considered. For many practical problems, the graph is derived from a mesh. The geometric method developed by Miller, Teng, Thurston, and Vavasis makes use of the vertex locations, and is guaranteed to partition well-shaped [1] d-dimensional unstructured mesh with n vertices using a

[1] For example, no grid angles can be too small/too large.

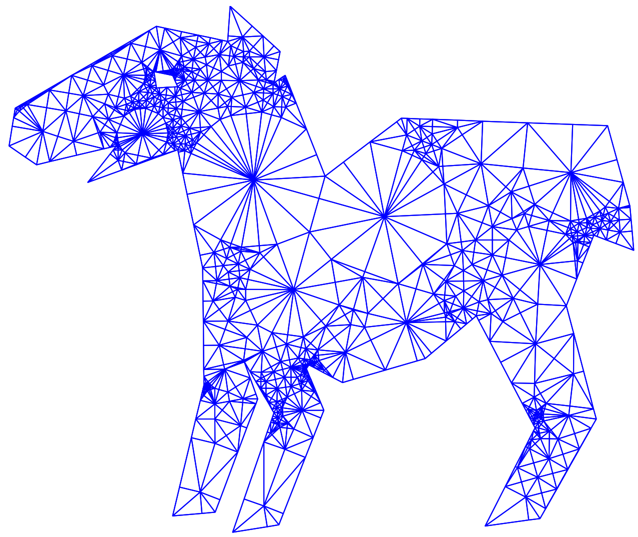


Figure 10.6: The input is a mesh with specified coordinates. Every triangle must be "well-shaped", which means that no angle can be too small. **Remarks:** The mesh is a subgraph of the intersection graph of a set of disks, one centered at each mesh point. Because the triangles are well-shaped, only a bounded number of disks can overlap any point, and the mesh is an "alpha-overlap graph". This implies that it has a good separator, which we proceed to find.

cut size $O(n^{1-1/d})$, where the cut size is the number of elements in the vertex separator. Note that this bound on the cut size is of the same order as for a regular grid of the same size. Such a bound does not hold for spectral method in general.

The geometric partitioning method has a great deal of theory behind it, but the implementation is relatively simple. For this reason, we will begin with an illustrative example before discussing the method and theoretical background in more depth. To motivate the software development aspect of this approach, we use the following figures (Figures 10.6 – 10.13) generated by a Matlab implementation (written by Gilbert and Teng) to outline the steps for dividing a well-shaped mesh into two pieces. The algorithm works on meshes in any dimension, but we'll stick to two dimensions for the purpose of visualization.

To recap, the geometric mesh partitioning algorithm can be summed up as follows (a more precise algorithm follows):

- Perform a stereographic projection to map the points in a d-dimensional plane to the surface of a $d+1$ dimensional sphere
- (Approximately) compute the centerpoint of the points on the sphere
- Perform a conformal map to move the centerpoint to the center of the sphere
- Find a plane through the center of the sphere that approximately divides the nodes equally; translate this plane to obtain a more even division
- Undo the conformal mapping and the stereographic mapping. This leaves a circle in the plane.

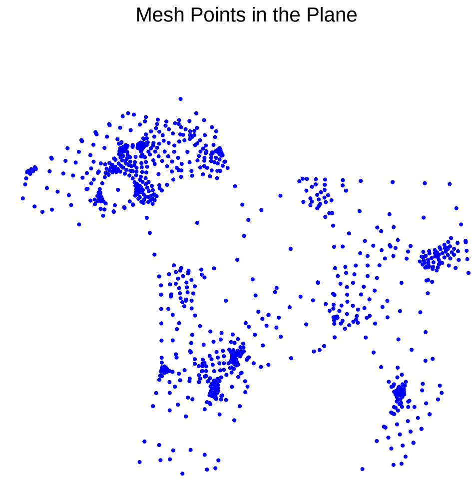


Figure 10.7: Let's redraw the mesh, omitting the edges for clarity.

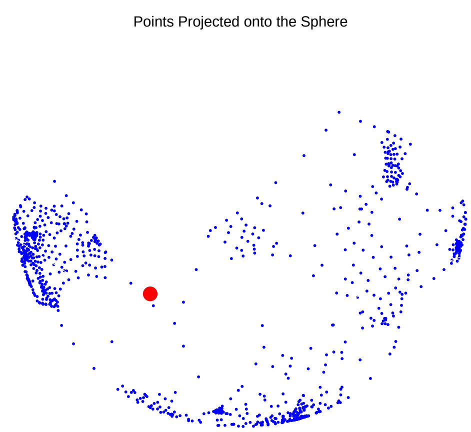


Figure 10.8: First we project the points stereographically from the plane onto the surface of a sphere (of one higher dimension than the mesh) tangent to the plane of the mesh. A stereographic projection is done by drawing a line between a point A on the plane and the north pole of the sphere, and mapping the point A to the intersection of the surface of the sphere and the line. Now we compute a "centerpoint" for the projected points in 3-space. A centerpoint is defined such that every plane through the centerpoint separates the input points into two roughly equal subsets. (Actually it's too expensive to compute a real centerpoint, so we use a fast, randomized heuristic to find a pretty good approximation.

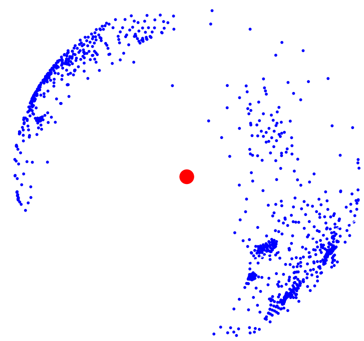


Figure 10.9: Next, we conformally map the points so that the centerpoint maps to the center of the sphere. This takes two steps: First we rotate the sphere about the origin (in 3-space) so that the centerpoint is on the z axis, and then we scale the points in the plane to move the centerpoint along the z axis to the origin (this can be thought of as mapping the sphere surface back to the plane, stretching the plane, then re-mapping the plane back to the surface of the sphere). The figures show the final result on the sphere and in the plane.

10.4.1 Geometric Graphs

This method applies to meshes in both two and three dimensions. It is based on the following important observation: graphs from large-scale problems in scientific computing are often defined geometrically. They are meshes of elements in a fixed dimension (typically two and three dimensions), that are *well shaped* in some sense, such as having elements of bounded aspect ratio or having elements with angles that are not too small. In other words, they are graphs embedded in two or three dimensions that come with natural geometric coordinates and with structures.

We now consider the types of graphs we expect to run these algorithms on. We don't expect to get a truly random graph. In fact, Erdös, Graham, and Szemerédi proved in the 1960s that with probability $= 1$, a random graph with cn edges does not have a two-way division with $o(n)$ crossing edges.

Structured graphs usually have divisions with $\sqrt{n}$ crossing edges. The following classes of graphs usually arise in applications such as finite element and finite difference methods (see Chapter **??**):

- **Regular Grids**: These arise, for example, from finite difference methods.
- **'Quad"-tree graphs and "Meshes"**: These arise, for example, from finite difference methods and hierarchical N-body simulation.
- **k-nearest neighbor graphs in d dimensions** Consider a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in $\mathbb{R}^d$. The vertex set for the graph is $\{1, 2, \ldots, n\}$ and the edge set is $\{(i, j) : p_j$ is one of the k-nearest neighbors of p_i or vice-versa$\}$. This is an important class of graphs for image processing.

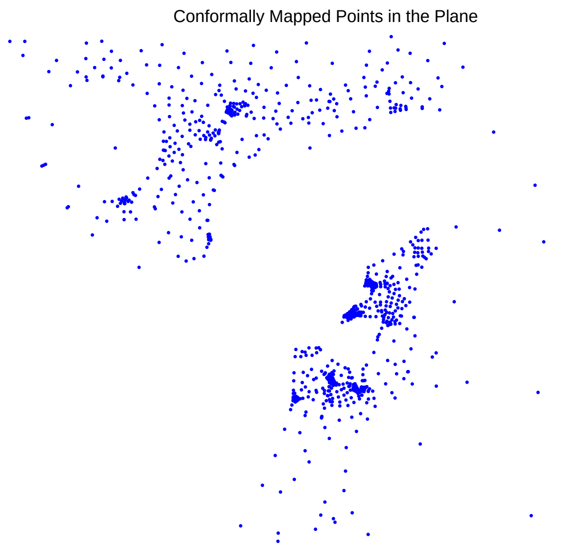


Figure 10.10: Because the approximate centerpoint is now at the origin, any plane through the origin should divide the points roughly evenly. Also, most planes only cut a small number of mesh edges ($O(\sqrt{(n)})$, to be precise). Thus we find a separator by choosing a plane through the origin, which induces a great circle on the sphere. In practice, several potential planes are considered, and the best one accepted. Because we only estimated the location of the centerpoint, we must shift the circle slightly (in the normal direction) to make the split exactly even. The second circle is the shifted version.

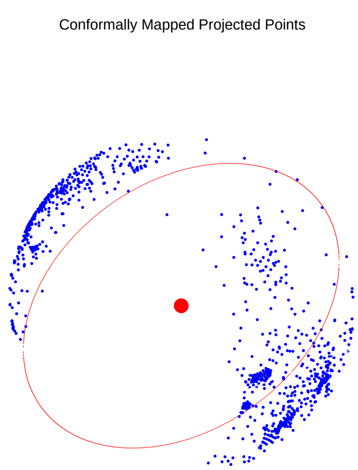


Figure 10.11: We now begin “undoing” the previous steps, to return to the plane. We first undo the conformal mapping, giving a (non-great) circle on the original sphere ...

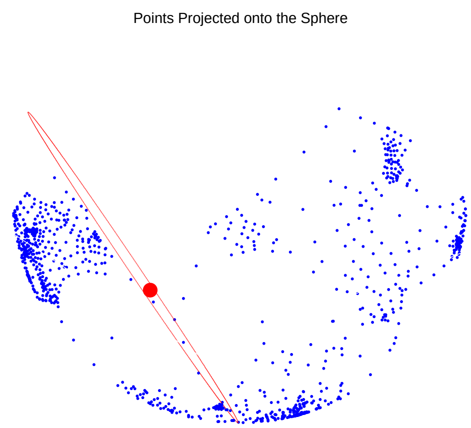


Figure 10.12: ... and then undo the stereographic projection, giving a circle in the original plane.

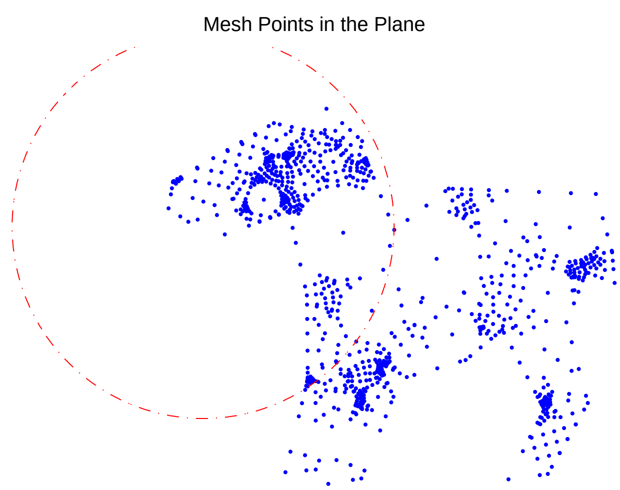


Figure 10.13: This partitions the mesh into two pieces with about n/2 points each, connected by at most $O(\sqrt(n))$ edges. These connecting edges are called an "edge separator". This algorithm can be used recursively if more divisions (ideally a power of 2) are desired.

- **Disk packing graphs**: If a set of non-overlapping disks is laid out in a plane, we can tell which disks touch. The nodes of a disk packing graph are the centers of the disks, and edges connect two nodes if their respective disks touch.
- **Planar graphs**: These are graphs that can be drawn in a plane without crossing edges. Note that disk packing graphs are planar, and in fact every planar graph is isomorphic to some disk-packing graph (Andreev and Thurston).

10.4.2 Geometric Partitioning: Algorithm and Geometric Modeling

The main ingredient of the geometric approach is a novel geometrical characterization of graphs embedded in a fixed dimension that have a small *separator*, which is a relatively small subset of vertices whose removal divides the rest of the graph into two pieces of approximately equal size. By taking advantage of the underlying geometric structure, partitioning can be performed efficiently.

Computational meshes are often composed of elements that are *well-shaped* in some sense, such as having bounded aspect ratio or having angles that are not too small or too large. Miller et al. define a class of so-called *overlap graphs* to model this kind of geometric constraint.

An overlap graph starts with a *neighborhood system*, which is a set of closed disks in d-dimensional Euclidean space and a parameter k that restricts how deeply they can intersect.

Definition 10.4.1 *A* k-ply neighborhood system in d dimensions *is a set* $\{D_1, \ldots, D_n\}$ *of closed disks in* $\mathbb{R}^d$*, such that no point in* $\mathbb{R}^d$ *is strictly interior to more than* k *of the disks.*

A neighborhood system and another parameter α define an overlap graph. There is a vertex for each disk. For $\alpha = 1$, an edge joins two vertices whose disks intersect. For $\alpha > 1$, an edge joins two vertices if expanding the smaller of their two disks by a factor of α would make them intersect.

Definition 10.4.2 *Let* $\alpha \geq 1$*, and let* $\{D_1, \ldots, D_n\}$ *be a* k*-ply neighborhood system. The* (α, k)-overlap graph *for the neighborhood system is the graph with vertex set* $\{1, \ldots, n\}$ *and edge set*

$$\{(i,j) | (D_i \cap (\alpha \cdot D_j) \neq \emptyset) \text{ and } ((\alpha \cdot D_i) \cap D_j \neq \emptyset)\}.$$

We make an overlap graph into a mesh in d-space by locating each vertex at the center of its disk.

Overlap graphs are good models of computational meshes because every mesh of bounded-aspect-ratio elements in two or three dimensions is contained in some overlap graph (for suitable choices of the parameters α and k). Also, every planar graph is an overlap graph. Therefore, any theorem about partitioning overlap graphs implies a theorem about partitioning meshes of bounded aspect ratio and planar graphs.

We now describe the geometric partitioning algorithm.

We start with two preliminary concepts. We let Π denote the *stereographic projection* mapping from $\mathbb{R}^d$ to S^d, where S^d is the unit d-sphere embedded in $\mathbb{R}^{d+1}$. Geometrically, this map may be defined as follows. Given $\boldsymbol{x} \in \mathbb{R}^d$, append '0' as the final coordinate yielding $\boldsymbol{x}' \in \mathbb{R}^{d+1}$. Then compute the intersection of S^d with the line in $\mathbb{R}^{d+1}$ passing through $\boldsymbol{x}'$ and $(0, 0, \ldots, 0, 1)^T$. This intersection point is $\Pi(\boldsymbol{x})$.

Algebraically, the mapping is defined as

$$\Pi(\boldsymbol{x}) = \begin{pmatrix} 2\boldsymbol{x}/\chi \\ 1 - 2/\chi \end{pmatrix}$$

where $\chi = \boldsymbol{x}^T\boldsymbol{x} + 1$. It is also simple to write down a formula for the inverse of Π. Let $\boldsymbol{u}$ be a point on S^d. Then

$$\Pi^{-1}(\boldsymbol{u}) = \frac{\bar{\boldsymbol{u}}}{1 - u_{d+1}}$$

where $\bar{\boldsymbol{u}}$ denotes the first d entries of $\boldsymbol{u}$ and u_{d+1} is the last entry. The stereographic mapping, besides being easy to compute, has a number of important properties proved below.

A second crucial concept for our algorithm is the notion of a *center point*. Given a finite subset $P \subset \mathbb{R}^d$ such that $|P| = n$, a *center point* of P is defined to be a point $\boldsymbol{x} \in \mathbb{R}^d$ such that if H is any open halfspace whose boundary contains $\boldsymbol{x}$, then

$$|P \cap H| \leq dn/(d+1). \tag{10.10}$$

It can be shown from Helly's theorem [25] that a center point always exists. Note that center points are quite different from centroids. For example, a center point (which, in the $d = 1$ case, is the same as a median) is largely insensitive to "outliers" in P. On the hand, a single distant outliers can cause the centroid of P to be displaced by an arbitrarily large distance.

Geometric Partitioning Algorithm

Let $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ be the input points in $\mathbb{R}^d$ that define the overlap graph.

1. Given $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$, compute $P' = \{\Pi(\boldsymbol{p}_1), \ldots, \Pi(\boldsymbol{p}_n)\}$ so that $P' \subset S^d$.

2. Compute a center point $\boldsymbol{z}$ of P'.

3. Compute an orthogonal $(d+1) \times (d+1)$ matrix Q such that $Q\boldsymbol{z} = \boldsymbol{z}'$ where

$$\boldsymbol{z}' = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \theta \end{pmatrix}$$

   such that θ is a scalar.

4. Define $P'' = QP'$ (i.e., apply Q to each point in P'). Note that $P'' \subset S^d$, and the center point of P'' is $\boldsymbol{z}'$.

5. Let D be the matrix $[(1-\theta)/(1+\theta)]^{1/2}I$, where I is the $d \times d$ identity matrix. Let $P''' = \Pi(D\Pi^{-1}(P''))$. Below we show that the origin is a center point of P'''.

6. Choose a random great circle S_0 on S^d.

7. Transform S_0 back to a sphere $S \subset \mathbb{R}^d$ by reversing all the transformations above, i.e., $S = \Pi^{-1}(Q^{-1}\Pi(D^{-1}\Pi^{-1}(S_0)))$.

8. From S compute a set of vertices of G that split the graph as in Theorem **??**. In particular, define C to be vertices embedded "near" S, define A be vertices of $G - C$ embedded outside S, and define B to be vertices of $G - C$ embedded inside S.

We can immediately make the following observation: because the origin is a center point of P''', and the points are split by choosing a plane through the origin, then we know that $|A| \leq (d+1)n/(d+2)$ and $|B| \leq (d+1)n/(d+2)$ regardless of the details of how C is chosen. (Notice that the constant factor is $(d+1)/(d+2)$ rather than $d/(d+1)$ because the point set P' lies in

$\mathbb{R}^{d+1}$ rather than $\mathbb{R}^d$.) Thus, one of the claims made in Theorem **??** will follow as soon as we have shown that the origin is indeed a center point of P''' at the end of this section.

We now provide additional details about the steps of the algorithm, and also its complexity analysis. We have already defined stereographic projection used Step 1. Step 1 requires $O(nd)$ operations.

Computing a true center point in Step 2 appears to a very expensive operation (involving a linear programming problem with n^d constraints) but by using random (geometric) sampling, an approximate center point can be found in random constant time (independent of n but exponential in d) [100, 48]. An *approximate* center point satisfies 10.10 except with $(d+1+\epsilon)n/(d+2)$ on the right-hand side, where $\epsilon > 0$ may be arbitrarily small. Alternatively, a deterministic linear-time sampling algorithm can be used in place of random sampling [65, 96], but one must again compute a center of the sample using linear programming in time exponential in d [67, 41].

In Step 3, the necessary orthogonal matrix may be represented as a single Householder reflection—see [43] for an explanation of how to pick an orthogonal matrix to zero out all but one entry in a vector. The number of floating point operations involved is $O(d)$ independent of n.

In Step 4 we do not actually need to compute P''; the set P'' is defined only for the purpose of analysis. Thus, Step 4 does not involve computation. Note that the $\boldsymbol{z}'$ is the center point of P'' after this transformation, because when a set of points is transformed by any orthogonal transformation, a center point moves according to the same transformation (more generally, center points are similarly moved under any affine transformation). This is proved below.

In Step 6 we choose a random great circle, which requires time $O(d)$. This is equivalent to choosing plane through the origin with a randomly selected orientation. (This step of the algorithm can be made deterministic; see [**?**].) Step 7 is also seen to require time $O(d)$.

Finally, there are two possible alternatives for carrying out Step 8. One alternative is that we are provided with the neighborhood system of the points (i.e., a list of n balls in $\mathbb{R}^d$) as part of the input. In this case Step 8 requires $O(nd)$ operations, and the test to determine which points belong in A, B or C is a simple geometric test involving S. Another possibility is that we are provided with the nodes of the graph and a list of edges. In this case we determine which nodes belong in A, B, or C based on scanning the adjacency list of each node, which requires time linear in the size of the graph.

Theorem 10.4.1 *If M is an unstructured mesh with bounded aspect ratio, then the graph of M is a subgraph of a bounded overlap graph of the neighborhood system where we have one ball for each vertex of M of radius equal to half of the distance to its nearest vertices. Clearly, this neighborhood system has ply equal to 1.*

Theorem 10.4.1 (Geometric Separators [67]) *Let G be an n-vertex (α, k)-overlap graph in d dimensions. Then the vertices of G can be partitioned into three sets A, B, and C, such that*

- *no edge joins A and B,*
- *A and B each have at most $(d+1)/(d+2)$ vertices,*
- *C has only $O(\alpha k^{1/d} n^{(d-1)/d})$ vertices.*

Such a partitioning can be computed by the geometric-partitioning-algorithm in randomized linear time sequentially, and in $O(n/p)$ parallel time when we use a parallel machine of p processors.

10.4.3 Other Graphs with small separators

The following classes of graphs all have small separators:

- Lines have edge-separators of size 1. Removing the middle edge is enough.
- Trees have a 1-vertex separator with $\beta = 2/3$ - the so-called centroid of the tree.
- Planar Graphs. A result of Lipton and Tarjan shows that a planar graph of bounded degree has a $\sqrt{8n}$ vertex separator with $\beta = 2/3$.
- d dimensional regular grids (those are used for basic finite difference method). As a folklore, they have a separator of size $n^{1-1/d}$ with beta $\beta = 1/2$.

10.4.4 Other Geometric Methods

Recursive coordinate bisection

The simplest form of geometric partitioning is recursive coordinate bisection (RCB) [91, 101]. In the RCB algorithm, the vertices of the graph are projected onto one of the coordinate axes, and the vertex set is partitioned around a hyperplane through the median of the projected coordinates. Each resulting subgraph is then partitioned along a different coordinate axis until the desired number of subgraphs is obtained.

Because of the simplicity of the algorithm, RCB is very quick and cheap, but the quality of the resultant separators can vary dramatically, depending on the embedding of the graph in $\mathbb{R}^d$. For example, consider a graph that is "+"-shaped. Clearly, the best (smallest) separator consists of the vertices lying along a diagonal cut through the center of the graph. RCB, however, will find the largest possible separators, in this case, planar cuts through the centers of the horizontal and vertical components of the graph.

Inertia-based slicing

Williams [101] noted that RCB had poor worst case performance, and suggested that it could be improved by slicing orthogonal to the principal axes of inertia, rather than orthogonal to the coordinate axes. Farhat and Lesoinne implemented and evaluated this heuristic for partitioning [33].

In three dimensions, let $v = (v_x, v_y, v_z)^t$ be the coordinates of vertex v in $\mathbb{R}^3$. Then the inertia matrix I of the vertices of a graph with respect to the origin is given by

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

where,

$$I_{xx} = \sum_{v \in V} v_y^2 + v_z^2, \quad I_{yy} = \sum_{v \in V} v_x^2 + v_z^2, \quad I_{zz} = \sum_{v \in V} v_x^2 + v_y^2$$

and, for $i, j \in \{x, y, z\}, i \neq j$,

$$I_{ij} = I_{ji} = -\sum_{v \in V} v_i v_j$$

The eigenvectors of the inertia matrix are the principal axes of the vertex distribution. The eigenvalues of the inertia matrix are the principal moments of inertia. Together, the principal axes and principal moments of inertia define the inertia ellipse; the axes of the ellipse are the principal axes of inertia, and the axis lengths are the square roots of the corresponding principal moments. Physically, the size of the principal moments reflect how the mass of the system is distributed with respect to the corresponding axis - the larger the principal moment, the more mass is concentrated at a distance from the axis.

Let I_1, I_2, and I_3 denote the principal axes of inertia corresponding to the principal moments $\alpha_1 \leq \alpha_2 \leq \alpha_3$. Farhat and Lesoinne projected the vertex coordinates onto I_1, the axis about which the mass of the system is most tightly clustered, and partitioned using a planar cut through the median. This method typically yielded a good initial separator, but did not perform as well recursively on their test mesh - a regularly structured "T"-shape.

Farhat and Lesoinne did not present any results on the theoretical properties of the inertia slicing method. In fact, there are pathological cases in which the inertia method can be shown to yield a very poor separator. Consider, for example, a "+"-shape in which the horizontal bar is very wide and sparse, while the vertical bar is relatively narrow but dense. I_1 will be parallel to the horizontal axis, but a cut perpendicular to this axis through the median will yield a very large separator. A diagonal cut will yield the smallest separator, but will not be generated with this method.

Gremban, Miller, and Teng show how to use moment of inertia to improve the geometric partitioning algorithm.

10.4.5 Partitioning Software

- **Chaco** written by Bruce Hendrickson and Rob Leland. To get code send email to bahendr@cs.sandia.gov (Bruce Hendrickson).

- Matlab Mesh Partitioning Toolbox: written by Gilbert and Teng. It includes both edge and vertex separators, recursive bipartition, nested dissection ordering, visualizations and demos, and some sample meshes. The complete toolbox is available by anonymous ftp from machine `ftp.parc.xerox.com` as file `/pub/gilbert/meshpart.uu`.

- **Spectral code**: Pothen, Simon, and Liou.

10.5 Load-Balancing N-body Simulation for Non-uniform Particles

The discussion of Chapter **??** was focused on particles that are more or less uniformly distributed. However, in practical simulations, particles are usually not uniformly distributed. Particles may be highly clustered in some regions and relatively scattered in some other regions. Thus, the hierarchical tree is adaptively generated, with smaller box for regions of clustered particles. The computation and communication pattern of a hierarchical method becomes more complex and often is not known explicitly in advance.

10.5.1 Hierarchical Methods of Non-uniformly Distributed Particles

In this chapter, we use the following notion of non-uniformity: We say a point set $P = \{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ in d dimensions is *μ-non-uniform* if the hierarchical tree generated for P has height $\log_{2^d}(n/m) + \mu$.

In other words, the ratio of the size of smallest leaf-box to the root-box is $1/2^{\log_{2^d}(n/m)+\mu}$. In practice, μ is less than 100.

The Barnes-Hut algorithm, as an algorithm, can be easily generalized to the non-uniform case. We describe a version of FMM for non-uniformly distributed particles. The method uses the box-box interaction. FMM tries to maximize the number of FLIPs among large boxes and also tries to FLIP between roughly equal sized boxes, a philosophy which can be described as: let parents do as much work as possible and then do the left-over work as much as possible before passing to the next generation. Let $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the set of child-boxes of the root-box of the hierarchical tree. FMM generates the set of all *interaction-pairs* of boxes by taking the union of Interaction-pair$(\boldsymbol{c}_i, \boldsymbol{c}_j)$ for all $1 \leq i < j \leq 2^d$, using the Interaction-Pair procedure defined below.
Procedure Interaction-Pair $(\boldsymbol{b}_1,\ \boldsymbol{b}_2)$

- If $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are β-well-separated, then $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ is an interaction-pair.
- Else, if both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are leaf-boxes, then particles in $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are near-field particles.
- Else, if both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are not leaf-boxes, without loss of generality, assuming that $\boldsymbol{b}_2$ is at least as large as $\boldsymbol{b}_1$ and letting $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the child-boxes of $\boldsymbol{b}_2$, then recursively decide interaction pair by calling: Interaction-Pair$(\boldsymbol{b}_1,\boldsymbol{c}_i)$ for all $1 \leq i \leq 2^d$.
- Else, if one of $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ is a leaf-box, without loss of generality, assuming that $\boldsymbol{b}_1$ is a leaf-box and letting $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the child-boxes of $\boldsymbol{b}_2$, then recursively decide interaction pairs by calling: Interaction-Pair$(\boldsymbol{b}_1,\boldsymbol{c}_i)$ for all $1 \leq i \leq 2^d$.

FMM for far-field calculation can then be defined as: for each interaction pair $(\boldsymbol{b}_1, \boldsymbol{b}_2)$, letting $\Phi_i^p()$ $(i = 1, 2)$ be the multipole-expansion of $\boldsymbol{b}_i$, flip $\Phi_1^p()$ to $\boldsymbol{b}_2$ and add to $\boldsymbol{b}_2$'s potential Taylor-expansion. Similarly, flip $\Phi_2^p()$ to $\boldsymbol{b}_1$ and add to $\boldsymbol{b}_1$'s potential Taylor-expansion. Then traverse down the hierarchical tree in a preordering, shift and add the potential Taylor-expansion of the parent box of a box to its own Taylor-expansion.

Note that FMM for uniformly distributed particles has a more direct description (see Chapter **??**).

10.5.2 The Communication Graph for N-Body Simulations

In order to efficiently implement an N-body method on a parallel machine, we need to understand its communication pattern, which can be described by a graph that characterizes the pattern of information exchange during the execution of the method. The communication graph is defined on basic computational elements of the method. The basic elements of hierarchical N-body methods are boxes and points, where points give the locations of particles and boxes are generated by the hierarchical method. Formally, the communication graph is an edge-weighted directed graph, where the edges describe the pattern of communication and the weight on an edge specifies the communication requirement along the edge.

A Refined FMM for Non-Uniform Distributions

For parallel implementation, it is desirable to have a communication graph that uses small edge-weights and has small in- and out-degrees. However, some boxes in the set of interaction-pairs defined in the last section may have large degree!

FMM described in the last subsection has a drawback which can be illustrated by the following 2D example. Suppose the root-box is divided into four child-boxes A, B, C, and D. Assume further

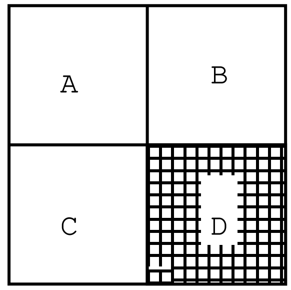


Figure 10.14: A non-uniform example

that boxes A, B and C contains less than m (< 100) particles, and most particles, say n of them, are uniformly distributed in D, see Figure 10.14. In FMM, we further recursively divide D by $\log_4(n/m)$ levels. Notice that A, B, and C are not well-separated from any box in D. Hence the FMM described in the previous subsection will declare all particles of D as near-field particles of A, B, and C (and vice versa). The drawback is two-folds: (1) From the computation viewpoint, we cannot take advantage of the hierarchical tree of D to evaluate potentials in A, B, and C. (2) From the communication viewpoint, boxes A, B, and C have a large in-degree in the sense that each particle in these boxes need to receive information from all n particles in D, making partitioning and load balancing harder. Notice that in BH most boxes of D are well-separated from particles in A, B, and C. Hence the well-separation condition is different in BH: *because BH uses the particle-box interaction, the well-separation condition is measured with respect to the size of the boxes in D. Thus most boxes are well-separated from particles in A, B, and C. In contrast, because FMM applies the FLIP operation, the well-separation condition must measure up against the size of the larger box. Hence no box in D is well-separated from A, B, and C.*

Our refined FMM circumvents this problem by incorporating the well-separation condition of BH into the Interaction-Pair procedure: if $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are not well-separated, and $\boldsymbol{b}_1$, the larger of the two, is a leaf-box, then we use a well-separation condition with respect to $\boldsymbol{b}_2$, instead of to $\boldsymbol{b}_1$, and apply the FLIP operation directly onto particles in the leaf-box $\boldsymbol{b}_1$ rather than $\boldsymbol{b}_1$ itself.

We will define this new well-separation condition shortly. First, we make the following observation about the Interaction-Pair procedure defined in the last subsection. We can prove, by a simple induction, the following fact: if $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are an interaction-pair and both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are not leaf-boxes, then $1/2 \leq size(\boldsymbol{b}_1)/size(\boldsymbol{b}_2) \leq 2$. This is precisely the condition that FMM would like to maintain. For uniformly distributed particles, such condition is always true between any interaction-pair (even if one of them is a leaf-box). However, for non-uniformly distributed particles, if $\boldsymbol{b}_1$, the larger box, is a leaf-box, then $\boldsymbol{b}_1$ could be much larger than $\boldsymbol{b}_2$.

The new *β-well-separation* condition, when $\boldsymbol{b}_1$ is a leaf-box, is then defined as: $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are *β-well-separated* if $\boldsymbol{b}_2$ is well-separated from all particles of $\boldsymbol{b}_1$ (as in BH). Notice, however, with the new condition, we can no longer FLIP the multipole expansion of $\boldsymbol{b}_1$ to a Taylor-expansion for $\boldsymbol{b}_2$. Because $\boldsymbol{b}_1$ has only a constant number of particles, we can directly evaluate the potential induced by these particles for $\boldsymbol{b}_2$. This new condition makes the FLIP operation of this special class of interaction-pairs *uni-directional*: We only FLIP $\boldsymbol{b}_2$ to $\boldsymbol{b}_1$.

We can describe the refined Interaction-Pair procedure using modified well-separation condition when one box is a leaf-box.

Procedure Refined Interaction-Pair ($\boldsymbol{b}_1$, $\boldsymbol{b}_2$)

- If $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are β-well-separated and $1/2 \leq size(\boldsymbol{b}_1)/size(\boldsymbol{b}_2) \leq 2$, then $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ is a bi-directional interaction-pair.

- Else, if the larger box, without loss of generality, $\boldsymbol{b}_1$, is a leaf-box, then the well-separation condition becomes: $\boldsymbol{b}_2$ is well-separated from all particles of $\boldsymbol{b}_1$. If this condition is true, then $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ is a uni-directional interaction-pair from $\boldsymbol{b}_2$ to $\boldsymbol{b}_1$.

- Else, if both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are leaf-boxes, then particles in $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are near-field particles.

- Else, if both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are not leaf-boxes, without loss of generality, assuming that $\boldsymbol{b}_2$ is at least as large as $\boldsymbol{b}_1$ and letting $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the child-boxes of $\boldsymbol{b}_2$, then recursively decide interaction-pairs by calling: Interaction-Pair($\boldsymbol{b}_1$,$\boldsymbol{c}_i$) for all $1 \leq i \leq 2^d$.

- Else, if one of $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ is a leaf-box, without loss of generality, assuming that $\boldsymbol{b}_1$ is a leaf box and letting $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the child-boxes of $\boldsymbol{b}_2$, then recursively decide interaction pairs by calling: Interaction-Pair($\boldsymbol{b}_1$,$\boldsymbol{c}_i$) for all $1 \leq i \leq 2^d$.

Let $\boldsymbol{c}_1, ..., \boldsymbol{c}_{2^d}$ be the set of child-boxes of the root-box of the hierarchical tree. Then the set of all interaction-pair can be generated as the union of Refined-Interaction-Pair($\boldsymbol{c}_i, \boldsymbol{c}_j$) for all $1 \leq i < j \leq 2^d$.

The refined FMM for far-field calculation can then be defined as: for each bi-directional interaction pair $(\boldsymbol{b}_1, \boldsymbol{b}_2)$, letting $\Phi_i^p()$ $(i = 1, 2)$ be the multipole expansion of $\boldsymbol{b}_i$, flip $\Phi_1^p()$ to $\boldsymbol{b}_2$ and add to $\boldsymbol{b}_2$'s potential Taylor-expansion. Similarly, flip $\Phi_2^p()$ to $\boldsymbol{b}_1$ and add to $\boldsymbol{b}_1$'s potential Taylor-expansion. Then traverse down the hierarchical tree in a preordering, shift and add the potential Taylor-expansion of the parent box of a box to its own Taylor-expansion. For each uni-directional interaction pair $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ from $\boldsymbol{b}_2$ to $\boldsymbol{b}_1$, letting $\Phi_2^p()$ be the multipole-expansion of $\boldsymbol{b}_2$, evaluate $\Phi_2^p()$ directly for each particle in $\boldsymbol{b}_2$ and add its potential.

Hierarchical Neighboring Graphs

Hierarchical methods (BH and FMM) explicitly use two graphs: the *hierarchical tree* which connects each box to its parent box and each particle to its leaf-box, and the *near-field graph* which connects each box with its near-field boxes. The hierarchical tree is generated and used in the first step to compute the multipole expansion induced by each box. We can use a bottom-up procedure to compute these multipole expansions: First compute the multipole expansions at leaf-boxes and then *SHIFT* the expansion to the parent boxes and then up the hierarchical tree until multipole-expansions for all boxes in the hierarchical tree are computed.

The near-field graph can also be generated by the Refined-Interaction-Pair procedure. In Section 10.5.3, we will formally define the near-field graph.

Fast-Multipole Graphs (FM)

The Fast-Multipole graph, FM^β, models the communication pattern of the refined FMM. It is a graph defined on the set of boxes and particles in the hierarchical tree. Two boxes $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are connected in FM^β iff (1) $\boldsymbol{b}_1$ is the parent box of $\boldsymbol{b}_2$, or vice versa, in the hierarchical tree; or (2) $(\boldsymbol{b}_1, \boldsymbol{b}_2)$ is an interaction-pair generated by Refined-Interaction-Pair defined in Section 10.5.2. The edge is bi-directional for a bi-directional interaction-pair and uni-directional for a uni-directional interaction-pair. Furthermore, each particle is connected with the box that contains the particle.

The following Lemma that will be useful in the next section.

Lemma 10.5.1 *The refined FMM flips the multipole expansion of $\boldsymbol{b}_2$ to $\boldsymbol{b}_1$ if and only if (1) $\boldsymbol{b}_2$ is well-separated from $\boldsymbol{b}_1$ and (2) neither the parent of $\boldsymbol{b}_2$ is well-separated from $\boldsymbol{b}_1$ nor $\boldsymbol{b}_2$ is well-separated from the parent of $\boldsymbol{b}_1$.*

It can be shown that both in- and out-degrees of FM^β are small.

Barnes-Hut Graphs (BH)

BH defines two classes of communication graphs: BH_S^β and BH_P^β. BH_S^β models the sequential communication pattern and BH_P^β is more suitable for parallel implementation. The letters S and P, in BH_S^β and BH_P^β, respectively, stand for "Sequential" and "Parallel".

We first define BH_S^β and show why parallel computing requires a different communication graph BH_P^β to reduce total communication cost.

The graph BH_S^β of a set of particles P contains two sets of vertices: P, the particles, and B, the set of boxes in the hierarchical tree. The edge set of the graph BH_S^β is defined by the communication pattern of the sequential BH. A particle $\boldsymbol{p}$ is connected with a box $\boldsymbol{b}$ if in BH, we need to evaluate $\boldsymbol{p}$ against $\boldsymbol{b}$ to compute the force or potential exerted on $\boldsymbol{p}$. So the edge is directed from $\boldsymbol{b}$ to $\boldsymbol{p}$. Notice that if $\boldsymbol{p}$ is connected with $\boldsymbol{b}$, then $\boldsymbol{b}$ must be well-separated from $\boldsymbol{p}$. Moreover, the parent of $\boldsymbol{b}$ is not well-separated from $\boldsymbol{p}$. Therefore, if $\boldsymbol{p}$ is connected with $\boldsymbol{b}$ in BH_S^β, then $\boldsymbol{p}$ is not connected to any box in the subtree of $\boldsymbol{b}$ nor to any ancestor of $\boldsymbol{b}$.

In addition, each box is connected directly with its parent box in the hierarchical tree and each point $\boldsymbol{p}$ is connected its leaf-box. Both types of edges are bi-directional.

Lemma 10.5.2 *Each particle is connected to at most $O(\log n + \mu)$ number of boxes. So the in-degree of BH_S^β is bounded by $O(\log n + \mu)$.*

Notice, however, BH_S^β is not suitable for parallel implementation. It has a large out-degree. This major drawback can be illustrated by the example of n uniformly distributed particles in two dimensions. Assume we have four processors. Then the "best" way to partition the problem is to divide the root-box into four boxes and map each box onto a processor. Notice that in the direct parallel implementation of BH, as modeled by BH_S^β, each particle needs to access the information of at least one boxes in each of the other processors. Because each processor has $n/4$ particles, the total communication overhead is $\Omega(n)$, which is very expensive.

The main problem with BH_S^β is that many particles from a processor need to access the information of the same box in some other processors (which contributes to the large out-degree). We show that a combination technique can be used to reduce the out-degree. The idea is to *combine* the "same" information from a box and send the information as one unit to another box on a processor that needs the information. We will show that this combination technique reduces the total communication cost to $O(\sqrt{n \log n})$ for the four processor example, and to $O(\sqrt{pn \log n})$ for p processors. Similarly, in three dimensions, the combination technique reduces the volume of messages from $\Omega(n \log n)$ to $O(p^{1/3} n^{2/3} (\log n)^{1/3})$.

We can define a graph BH_P^β to model the communication and computation pattern that uses this combination technique. Our definition of BH_P^β is inspired by the communication pattern of the refined FMM. It can be shown that the communication pattern of the refined FMM can be used to guide the message combination for the parallel implementation of the Barnes-Hut method!

The combination technique is based on the following observation: Suppose $\boldsymbol{p}$ is well-separated from $\boldsymbol{b}_1$ but not from the parent of $\boldsymbol{b}_1$. Let $\boldsymbol{b}$ be the largest box that contains $\boldsymbol{p}$ such that $\boldsymbol{b}$ is well-separated from $\boldsymbol{b}_1$, using the well-separation definition in Section 10.5.2. If $\boldsymbol{b}$ is not a leaf-box, then $(\boldsymbol{b}, \boldsymbol{b}_1)$ is a bi-directional interaction-pair in the refined FMM. If $\boldsymbol{b}$ is a leaf-box, then $(\boldsymbol{b}, \boldsymbol{b}_1)$ is a uni-directional interaction-pair from $\boldsymbol{b}_1$ to $\boldsymbol{b}$. Hence $(\boldsymbol{b}, \boldsymbol{b}_1)$ is an edge of FM^{β}. Then, any other particle $\boldsymbol{q}$ contained in $\boldsymbol{b}$ is well-separated from $\boldsymbol{b}_1$ as well. Hence we can combine the information from $\boldsymbol{b}_1$ to $\boldsymbol{p}$ and $\boldsymbol{q}$ and all other particles in $\boldsymbol{b}$ as follows: $\boldsymbol{b}_1$ sends its information (just one copy) to $\boldsymbol{b}$ and $\boldsymbol{b}$ forwards the information down the hierarchical tree, to both $\boldsymbol{p}$ and $\boldsymbol{q}$ and all other particles in $\boldsymbol{b}$. This combination-based-communication scheme defines a new communication graph BH_P^{β} for parallel BH: The nodes of the graph are the union of particles and boxes, i.e., $P \cup B(P)$. Each particle is connected to the leaf-box it belongs to. Two boxes are connected iff they are connected in the Fast-Multipole graph. However, to model the communication cost, we must introduce a weight on each edge along the hierarchical tree embedded in BH_P^{β}, to be equal to the number of data units needed to be sent along that edge.

Lemma 10.5.3 *The weight on each edge in BH_P^{β} is at most $O(\log n + \mu)$.*

It is worthwhile to point out the difference between the comparison and communication patterns in BH. In the sequential version of BH, if $\boldsymbol{p}$ is connected with $\boldsymbol{b}$, then we have to compare $\boldsymbol{p}$ against all ancestors of $\boldsymbol{b}$ in the computation. The procedure is to first compare $\boldsymbol{p}$ with the root of the hierarchical tree, and then recursively move the comparison down the tree: if the current box compared is not well-separated from $\boldsymbol{p}$, then we will compare $\boldsymbol{p}$ against all its child-boxes. However, in terms of force and potential calculation, we only evaluate a particle against the first box down a path that is well-separated from the particle. The graphs BH_S^{β} and BH_P^{β} capture the communication pattern, rather than the comparison pattern. The communication is more essential to force or potential calculation. The construction of the communication graph has been one of the bottlenecks in load balancing BH and FMM on a parallel machine.

10.5.3 Near-Field Graphs

The near-field graph is defined over all leaf-boxes. A leaf-box $\boldsymbol{b}_1$ is a *near-field neighbor* of a leaf-box $\boldsymbol{b}$ if $\boldsymbol{b}_1$ is not well-separated from some particles of $\boldsymbol{b}$. Thus, FMM and BH directly compute the potential at particles in $\boldsymbol{b}$ induced by particles of $\boldsymbol{b}_1$.

There are two basic cases: (1) if $size(\boldsymbol{b}_1) \leq size(\boldsymbol{b})$, then we call $\boldsymbol{b}_1$ a *geometric near-field neighbor* of $\boldsymbol{b}$. (2) if $size(\boldsymbol{b}_1) > size(\boldsymbol{b})$, then we call $\boldsymbol{b}_1$ a *hierarchical near-field neighbor* of $\boldsymbol{b}$. In the example of Section 10.5.2, A, B, C are hierarchical near-field neighbors of all leaf-boxes in D; while A, B, and C have some geometric near-field neighbors in D.

We introduce some notations. The *geometric in-degree* of a box $\boldsymbol{b}$ is the number of its geometric near-field neighbors. The *geometric out-degree* of a box $\boldsymbol{b}$ is the number of boxes to which b is the geometric near-field neighbors. The *hierarchical in-degree* of a box $\boldsymbol{b}$ is the number of its hierarchical near-field neighbors. We will define the *hierarchical out-degree* of a box shortly.

It can be shown that the geometric in-degree, geometric out-degree, and hierarchical in-degree are small. However, in the example of Section 10.5.2, A, B, and C are hierarchical near-field neighbors for all leaf-boxes in D. Hence the number of leaf-boxes to which a box is a hierarchical near-field neighbor could be very large. So the near-field graph defined above can have a very large out-degree.

We can use the combination technique to reduce the degree when a box $\boldsymbol{b}$ is a hierarchical near-field neighbor of a box $\boldsymbol{b}_1$. Let $\boldsymbol{b}_2$ be the ancestor of $\boldsymbol{b}_1$ of the same size as $\boldsymbol{b}$. Instead of $\boldsymbol{b}$ sending its information directly to $\boldsymbol{b}_1$, $\boldsymbol{b}$ sends it to $\boldsymbol{b}_2$ and $\boldsymbol{b}_2$ then forwards the information down the hierarchical tree. Notice that $\boldsymbol{b}$ and $\boldsymbol{b}_2$ are not well-separated. We will refer to this modified near-field graph as the near-field graph, denoted by NF^{β}. We also define the *hierarchical out-degree* of a box $\boldsymbol{b}$ to be the number of edges from $\boldsymbol{b}$ to the set of non-leaf-boxes constructed above. We can show that the *hierarchical out-degree* is also small.

To model the near-field communication, similar to our approach for BH, we introduce a weight on the edges of the hierarchical tree.

Lemma 10.5.4 *The weight on each edge in NF^{β} is at most $O(\log n + \mu)$.*

10.5.4 N-body Communication Graphs

By abusing notations, let $FM^{\beta} = FM^{\beta} \cup NF^{\beta}$ and $BH_P^{\beta} = BH_P^{\beta} \cup NF^{\beta}$. So the communication graph we defined simultaneously supports near-field and far-field communication, as well as communication up and down the hierarchical tree. Hence by partitioning and load balancing FM^{β} and BH_P^{β} , we automatically partition and balance the hierarchical tree, the near-field graph, and the far-field graph.

10.5.5 Geometric Modeling of N-body Graphs

Similar to well-shaped meshes, there is a geometric characterization of N-body communication graphs. Instead of using neighborhood systems, we use box-systems. A *box-system* in $\mathbb{R}^d$ is a set $B = \{B_1, \ldots, B_n\}$ of boxes. Let $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ be the centers of the boxes, respectively. For each integer k, the set B is a *k-ply box-system* if no point $\boldsymbol{p} \in \mathbb{R}^d$ is contained in more than k of $\mathrm{int}(B_1), \ldots, \mathrm{int}(B_n)$.

For example, the set of all leaf-boxes of a hierarchical tree forms a 1-ply box-system. The box-system is a variant of neighborhood system of Miller, Teng, Thurston, and Vavasis [67], where a neighborhood system is a collection of Euclidean balls in $\mathbb{R}^d$. We can show that box-systems can be used to model the communication graphs for parallel adaptive N-body simulation.

Given a box-system, it is possible to define the *overlap* graph associated with the system:

Definition 10.5.1 *Let $\alpha \geq 1$ be given, and let $\{B_1, \ldots, B_n\}$ be a k-ply box-system. The α-**overlap graph** for this box-system is the undirected graph with vertices $V = \{1, \ldots, n\}$ and edges*

$$E = \{(i,j) : B_i \cap (\alpha \cdot B_j) \neq \emptyset \text{ and } (\alpha \cdot B_i) \cap B_j \neq \emptyset\}.$$

The edge condition is equivalent to: $(i,j) \in E$ iff the α dilation of the smaller box touches the larger box.

As shown in [96], the partitioning algorithm and theorem of Miller *et al* can be extended to overlap graphs on box-systems.

Theorem 10.5.1 *Let G be an α-overlap graph over a k-ply box-system in $\mathbb{R}^d$, then G can be partitioned into two equal sized subgraphs by removing at most $O(\alpha k^{1/d} n^{1-1/d})$ vertices. Moreover, such a partitioning can be computed in linear time sequentially and in parallel $O(n/p)$ time with p processors.*

The key observation is the following theorem.

Theorem 10.5.2 *Let $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ be a point set in $\mathbb{R}^d$ that is μ-non-uniform. Then the set of boxes $B(P)$ of hierarchical tree of P is a $(\log_{2^d} n + \mu)$-ply box-system and $FM^{\beta}(P)$ and $BH_P^{\beta}(P)$ are subgraphs of the 3β-overlap graph of $B(P)$.*

Therefore,

Theorem 10.5.3 *Let G be an N-body communication graph (either for BH or FMM) of a set of particles located at $P = \{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ in $\mathbb{R}^d$ ($d = 2$ or 3). If P is μ-non-uniform, then G can be partitioned into two equal sized subgraphs by removing at most $O(n^{1-1/d}(\log n + \mu)^{1/d})$ nodes. Moreover, such a partitioning can be computed in linear time sequentially and in parallel $O(n/p)$ time with p processors.*