THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCE

MULTI-GPU SOLUTIONS OF GEOPHYSICAL PDES WITH RADIAL BASIS FUNCTION-GENERATED FINITE DIFFERENCES

By

EVAN F. BOLLIG

A Dissertation submitted to the
Department of Scientific Computing
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Fall Semester, 2012

Evan F. Bollig defended this dissertation on October 1, 2012.

The members of the supervisory committee were:

Gordon Erlebacher
Professor Directing Thesis

Natasha Flyer
Outside University Representative

Mark Sussman
University Representative

Dennis Slice
Committee Member

Ming Ye
Committee Member

Janet Peterson
Committe Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with the university requirements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

AN ALTERNATIVE STENCIL GENERATION ALGORITHM FOR RBF-FD

Like all RBF methods, RBF-FD is designed to handle irregular node distributions, so the emphasis in literature focuses on how the method manages point clouds. While nothing prevents implementations of RBF-FD from utilizing existing meshes/lattices, most work in the field concentrates on simple geometries to better understand properties of the method and develop extensions. Without mesh/lattice connectivity available, stencils are generated by choosing the n-nearest neighbors to a center node, inclusive of the center. This is known more formally as a *k-nearest neighbor (k-NN)* problem [96] (a.k.a. ℓ-nearest neighbor search [107]). Here "nearest" is defined with the Euclidean distance metric, although it is possible to generalize to other metrics (see e.g., [3]).

In comparison to the RBF-FD method, global RBF methods with infinite support connect all nodes to all other nodes, so there is no need for neighbor queries. On the other hand, compact RBF methods require all nodes—with no limit on the count—that lie within the support/radius of the RBF centered at each node. This type of neighbor query is referred to as a *ball query* (a.k.a. range query [107]) due to the closed ball created by the radius of support for a compact RBF (see Equation **??**).

The k-NN and ball query share many similarities, but the former can be harder to solve. Consider, for example, the scenario in Figure 1.1. Two ball queries around a blue stencil center are represented as dashed and dash-dot circles. The inner query returns four neighbors, and the outer returns six. If a stencil of size $n = 6$ is desired, then the outer query can be truncated to give the five required neighbors shown in blue. In this example the red node and the farthest blue node are equidistant from the center, and ties are broken arbitrarily. Although k-NN is simply a truncated ball query, the real challenge lies in finding the proper search radius to enclose at least the n desired neighbors. To find the radius in practice depends on the choice of data structure used to access node locations.

A naïve approach for neighbor queries would be a brute-force search that checks distances from all nodes to every other node. Obviously the cost of such a method is high: $O(N^2)$ for all stencils. Multi-dimensional data structures, such as those discussed here, can limit the scope of searching and reduce the cost of stencil generation to $O(N \log N)$.

For the most part, investigations in RBF communities that delve into efficient neighbor queries are limited to ball queries. For example, the Partition of Unity method for approximation (e.g., [106, 107]), and particle methods like the Fast Multipole Method (e.g.,

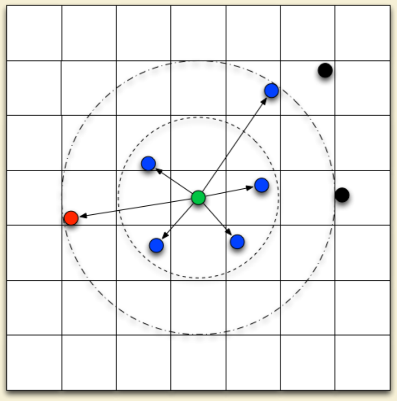


Figure 1.1: A stencil center in blue finds neighboring stencil nodes in blue. Two ball queries are shown as dashed and dash-dot circles to demonstrate the added difficulty of finding the right query radius to obtain the k-nearest neighbors.

[52, 117]) or Smoothed Particle Hydrodynamics (e.g., [65]). Examples of fast algorithms employed in these fields include the fixed-grid method [65, 107], k-D Trees [107], Range Trees [106, 107], and 2^d-Trees (i.e., Quad- and Octrees) [52, 117]. Surprisingly, while other communities continue the quest for fast neighbor queries, RBF collocation and RBF-FD communities have been slow on the uptake. For many years, the standard in the community has been to use k-D Trees (see e.g., [30, 35, 43]).

This chapter considers the use of an alternative neighbor query algorithm to generate RBF-FD stencils. It is based loosely on the fixed-grid method from [51, 59, 65]. [82] would classify the algorithm as a *fixed grid "bucket" method with one-dimensional spatial ordering.* The fixed-grid method loosens the requirements for finding the k-nearest neighbors (k-NN) stencils to accept k-"approximately nearest" neighbors (k-ANN). It also reorders nodes according to space-filling curves. In what follows, the fixed-grid method is compared to an efficient implementation of k-D Tree available for use in C++ and MATLAB ([96]). Benchmarks demonstrate that, with the proper choice of parameters for the fixed-grid, the method can be up to 2x faster than k-D Tree, and it comes with a free bonus: up to 5x faster SpMV performance due to the impact of spatial reordering that occurs during stencil generation.

1.1 k-D Tree

A k-D Tree is a spatial data structure that generally decomposes a space/volume into a small number of cells. All k-D Trees are binary and iteratively subdivide volumes and sub-volumes at each level into two parts. The "k" in k-D Tree refers to the dimensionality of the data/volume partitioned—that is $k \equiv d$.

Given a set of points bounded by a d-dimensional volume, a k-D Tree applies a hierarchy of $(d-1)$-dimensional axis aligned *splitting planes* to cut the space. At each level of the hierarchy the splitting planes result in two new *half-planes* [89]. Consecutive splits intercept one another at a *splitting value.* k-D Trees do not require that half-planes equally subdivide a volume; more often it is the data contained within the volume that is equally partitioned. The choice of dimension for the splitting plane, in conjunction with a variety of methods for choosing the splitting values allows for many flavors of k-D Trees (see e.g., [24, 82, 89] for comprehensive lists). *Point k-D Trees*, 2^d *Trees* (i.e., quad-/octrees), *BSP-Trees*, and *R-trees* are all members of the general k-D Tree class [89, 117].

This work considers *Point k-D Trees* [82], which partition a set of discrete points/nodes as outlined by the recursive procedure in Algorithm 1.1. Point k-D Trees assume that splitting planes intercept nodes rather than occur arbitrarily along the half-plane. The splitting value at each level of the tree is set to the *median coordinate* of the points in the half-plane, which ensures the tree is well balanced on initial construction. All nodes with coordinate (in the current dimension) less than or equal to the splitting value are contained by the left half-plane, and all nodes with coordinate greater than the splitting value are contained by the right. Half-planes containing only one element correspond to leaves of the tree. The median coordinate of a half-plane is found by sorting the n node coordinates contained by the partition and selecting the $\lceil \frac{n}{2} \rceil$-th element [24].

Algorithm 1.1 BuildKDTree(P, *depth*)

```
1:  Input: A set of $d$-dimensional points $P$ and the current $depth$.
2:  Output: The root of the $k$-D Tree for $P$.
3:
4:  if $size(P) = 1$ then
5:      return a new leaf storing $P$
6:  end if
7:  $L_i := median(coord(P, depth))$
8:  $v_l :=$ BuildKDTree($coord(P, depth) \leq L_i$, $(depth + 1)$ modulo $d$)
9:  $v_r :=$ BuildKDTree($coord(P, depth) > L_i$, $(depth + 1)$ modulo $d$)
10: return A new node $v := \begin{pmatrix} \text{value} := L_i \\ \text{left} := v_l \\ \text{right} := v_r \end{pmatrix}$
```

The k-D Tree in Figure 1.2 is an example of a Point k-D Tree. Given a set of eight nodes in two dimensions, the tree is constructed by applying one-dimensional cuts along the x-dimension, then the y-dimension, then back to the x-dimension. This approach is referred to as *cyclic splitting*, as consecutive cuts are applied by iterating dimensions in a round-robin fashion [82]. The first cut, $L1$, shown in blue, splits the nodes into two sets on either side of A. The corresponding tree in the center of Figure 1.2 shows $L1$ as the tree root with all nodes having x-coordinates less than or equal to A to the left, and all nodes having x-coordinates greater than A to the right. The second level of the tree, $L2$ and $L3$ (in blue), splits the half-planes on either side of A at nodes B and C. The axis parallel splits for each half-plane intercept $L1$ independently to partition half-planes along the y-dimension; once again, nodes with coordinates less than or equal (i.e., below) to the splitting value branch left in the tree, and y-coordinates greater than (i.e., above) the value branch right. The third level (red) returns to splitting half-planes in the x-dimension. Nodes D and H are not intersected by a splitting plane; their half-planes contain only one node so they immediately become leaves of the tree. This process to build a Point k-D Tree has

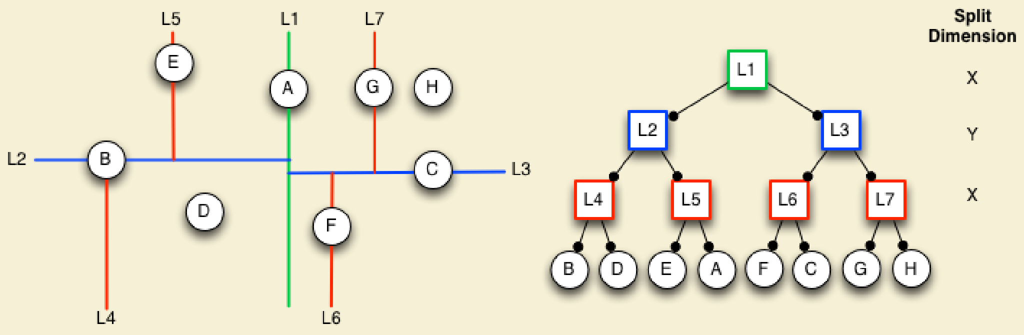


Figure 1.2: An example k-D Tree in 2-Dimensions. Nodes are partitioned with a cyclic dimension splitting rule (i.e., splits occur first in X, then Y, then X, etc.); all splits occur at the median node in each dimension.

a complexity of $O(N \log N)$ with $O(N)$ storage required [24, 82].

Frequently, the terms *k-D Tree* and *Point k-D Tree* are used synonymously by the RBF community (see e.g., [30, 35, 43]); the same is convention adopted here.

Generating an RBF-FD stencil with a k-D Tree can be efficiently accomplished in $O(n \log N)$ time—where n is the stencil size—following an approach introduced in [49], and presented in Algorithm 1.2. The k-NN search starts a depth-first recursive search of the k-D Tree to find the nearest neighbor to a query point, X_q. Traversal of the the tree occurs by following branches left or right based on comparison of X_q coordinates to the splitting value stored at each node of the tree, with the objective to find the smallest half-plane containing X_q. The search traverses the height of the tree in $O(\log N)$ steps to find the leaf that stores the nearest neighbor to X_q. The neighbor point and its distance from X_q are inserted into a global priority queue, pq. Points in the priority queue are sorted in descending order according to distance.

After finding the nearest neighbor the algorithm returns to the previous split in the tree and traverses onto the opposing half-plane (i.e., down the far branch) to look for other leaves. So long as the size of pq is at less than capacity (n) the search automatically adds points to the priority queue. If pq reaches capacity the algorithm starts to pop off excess points with the understanding that the action removes those points farthest from X_q.

In order to prune branches from the search and reduce complexity, Algorithm 1.2 makes use of a routine called "BoundsOverlapBall", which checks if any boundaries of the current level half-plane intersect/overlap with a closed ball centered at X_q. The ball is given a radius equal to the maximum distance in pq. Then, if the ball and a boundary intersect, the search will continue onto the half-plane on the opposite side of that boundary. This step handles the possibility that nearer nodes occur within the overlapped region in the other half-plane. If the ball and boundary do not intersect, the opposing half-plane and its related subtree are pruned from the search. Additional details on the implementation of "BoundsOverlapBall" can be found in [49, 97].

The authors of [49] find Algorithm 1.2 capable of efficiently querying the n-nearest neighbors with a complexity proportional to $O(\log N)$ (dominated by the cost of tree traversal). The relationship between stencil size n, and grid size, N, is better expressed as $O(n \log N)$

Algorithm 1.2 KNNSearchKDTree(X_q, n, $root$, $depth$)

```
Input: A query node $X_q$, number of desired neighbors ($n$), the current $root$ of the $k$-D Tree, and the current $depth$ of traversal.
Output: A global priority queue, $pq$, containing the $n$-nearest neighbors to $X_q$ sorted by distance from $X_q$ in descending order.
Assume: A routine named "BoundsOverlapBall" exists to determine if the boundaries of the current half-plane are intersected by the ball centered at $X_q$ with radius equal to the maximum distance in $pq$. As long as $pq.size < n$, "BoundsOverlapBall" defaults to true.

if $root$ is leaf then
    Insert $\{root, \text{dist}(X_q, root)\}$ into $pq$
    if $pq$.size $> n$ then
        $pq$.pop                                ▷ Keep only $n$-nearest neighbors
    end if
    return
end if

if coord$(X_q, depth)$ <= $root$.value then
    KNNSearchKDTree($X_q$, $n$, $root$.left, $(depth + 1)$ % $d$)
else
    KNNSearchKDTree($X_q$, $n$, $root$.right, $(depth + 1)$ % $d$)
end if

if coord$(X_q, depth)$ <= $root$.value then
    if BoundsOverlapBall($X_q$) then
        KNNSearchKDTree($X_q$, $n$, $root$.right, $(depth + 1)$ % $d$)
    end if
else
    if BoundsOverlapBall($X_q$) then
        KNNSearchKDTree($X_q$, $n$, $root$.left, $(depth + 1)$ % $d$)
    end if
end if
return
```

for one stencil.

RBF-FD only needs to generate stencils once, so the overall time for the step subsumes the cost of tree construction and N queries. The resulting total complexity of stencil generation for all stencils is then proportional to $O(N \log N)$.

1.2 A Fixed-Grid Algorithm

While a k-D Tree functions well for queries, the cost to build the tree structure is unnecessary overhead. Among the many data-structures that exist for nearest neighbor queries, alternatives like fixed-grid methods [82, 106, 107] (a.k.a. uniform grid [51, 65]) bypass much

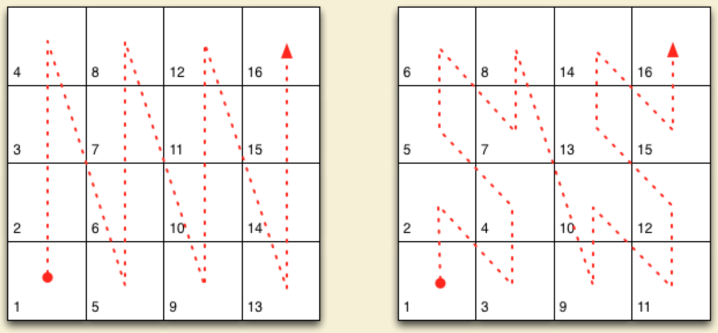


Figure 1.3: Two example space filling curves to linearize the same fixed-grid. Left: Raster-ordering (ijk); Right: Morton-/Z-ordering.

of the cost in construction with an assumption that only lower spatial dimensions (e.g., 2-D or 3-D) are significant for choosing neighbors. This discards the need to build a tree and shifts focus onto querying neighbors.

Fixed-grid methods get their name from a coarse 2-D or 3-D regular grid that is overlaid on the domain. The d-dimensional grid divides the domain's axis aligned bounding box (AABB)—that is, the minimum bounding box containing the entire domain with edges parallel to axes—into $(h_n)^d$ cells. Subdivisions are uniform, so one can easily identify the cell containing any sample point, p, given the coordinates of the AABB and $(h_n)^d$. For example, let (c_x, c_y, c_z) be the desired cell in 3-D, and $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$ be the minimum and maximum coordinates of the AABB (resp.). Then the cell coordinates are found by:

$$\begin{aligned}(dx, dy, dz) &= \left(\frac{(x_{max} - x_{min})}{h_n}, \frac{(y_{max} - y_{min})}{h_n}, \frac{(z_{max} - z_{min})}{h_n}\right)\\ (c_x, c_y, c_z) &= \left(\left\lfloor\frac{(p_x - x_{min})}{dx}\right\rfloor, \left\lfloor\frac{(p_y - y_{min})}{dy}\right\rfloor, \left\lfloor\frac{(p_z - z_{min})}{dz}\right\rfloor\right). \end{aligned} \tag{1.1}$$

Cells neighboring (c_x, c_y, c_z) are trivial to find by adding positive and negative offsets to each coordinate.

Fixed grid methods also make use of *space filling curves*. Space filling curves pass through every point in d-dimensional space, and through each point only once. Equivalently, space filling curves map d-dimensional space down to 1-D, where every point is converted to a unique index or traversal order based on its spatial coordinates. These mapping properties make space filling curves ideal for use as hash functions. Traversing the d-dimensional points (i.e., playing "connect the dots") draws the space filling curve. Figure 1.3 presents two common orderings of a 2-D fixed-grid. Note that one-dimensional orderings are not unique. On the left is a *Raster*-ordering (a.k.a. Scanline- or ijk-ordering): $f(c_x, c_y, c_z) = ((c_x * h_n) + c_y) * h_n + c_z$. The right half of Figure 1.3 shows an ordering known as Morton- or Z-ordering. Z-ordering construction is discussed later in this chapter. On both sides of Figure 1.3, the lower left corner of each cell indicates the mapped index. Traversing the cells in order produces the curves superimposed in red.

At a high level, fixed-grid methods have the following construction steps [65]:

1. Subdivide the domain with the overlay grid.

2. For each node, identify the containing cell coordinates.

3. For each node, use the cell coordinates as input to a spatial hash function (i.e., a space-filling curve).

4. Sort the nodes according to their spatial hash.

Particular details of how nodes are sorted, the choice of hashing function, the number of nodes allowed per cell, etc. determine the specific class of fixed-grid method and corresponding complexity. A comprehensive list of options and classifications can be found in [82].

1.2.1 Fixed-grid Construction

The algorithm in this work is inspired by fixed-grid approaches for GPU particle simulations ([51, 59, 65]). Particle methods require a ball query at each time-step. With time-steps often dominated by the cost of querying neighbors, the community understandably devotes significant effort to seek out the most efficient solutions possible [50]. The fixed-grid method is competitive for at least two reasons: a) by bypassing the need to build a tree, half the cost in querying neighbors is avoided; and b) nodes sorted according to a spatial hash reside closer in memory to nearby neighbors than in the case of unsorted nodes. The spatial locality results in a higher likelihood that data will be cached when required. Note that reordering by cell hash sorts nodes across cells but not within them—that is, nodes contained by the same cell are contiguous in memory, but remain arbitrarily ordered with respect one another. Fortunately, with contiguous groups of nodes, nearest neighbor queries can directly access all nodes per cell.

The authors of [51, 59, 65] assume a raster-ordering on cells, and that the uniform grid is sufficiently refined to ensure cells contain at most eight nodes. Particle interactions are limited to ball queries on the containing cell plus one valence of neighboring cells (i.e., 8 surrounding cells in 2-D and 26 cells in 3-D). Since the number of cells to check is fixed, the neighboring nodes can be obtained by direct access in constant time. A similar approach is taken by [50], but the authors opt for a Z-ordering of cells. As a point of difference in implementations, the authors of [50, 51, 65] leverage a fast radix sort algorithm to order nodes based on hash index, while [59] utilizes a slower bitonic sort algorithm. The fixed-grid in [106, 107] forgoes logic to refine the grid and enforce a maximum limit on the number of nodes per cell. The author also avoids sorting nodes based on cell hashes. Instead, a list is maintained that stores the indices of all contained nodes per cell.

The implementation presented here is a hybrid of the related algorithms. For example, cells are sorted based on raster-ordering, but without the restriction on max number of nodes per cell. Rather than a radix- or bitonic sort to reorder nodes, the list of node indices for each cell ([106, 107]) is constructed as part of a single-pass bucket sort. Finally, in stark contrast to [51, 59, 65, 106, 107], querying neighbors is not restricted to a fixed radius, or number of cell valences. To satisfy the k-NN query, this implementation iteratively increases the query radius to include a new valence of cells at each iteration. This multi-pass ball-query was demonstrated in Figure 1.1. The iteration terminates when the desired count of neighboring nodes is satisfied or exceeded.

Algorithm 1.3 BuildFixedGrid(P, h_n)

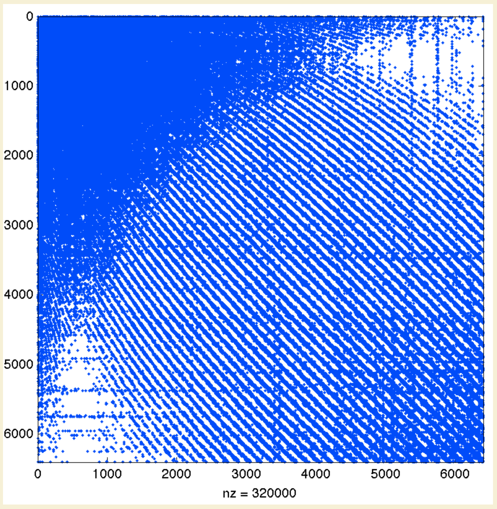
```
1: **Input:** A set points $P$, and the fixed-grid resolution, $h_n$.
2: **Output:** The reordered points in $P$, and corresponding cell buckets $Q$.
3:
4: Create $Q$: an $(h_n)^d$ array of empty buckets.
5: for point $p_i$ in $P$ do
6:     $c :=$ CellCoords($p_i$)
7:     $ind :=$ SpatialHash($c$)
8:     Append index $i$ onto $Q[ind]$
9: end for
10: for $j = 0, 1, ..., (h_n)^d$ do
11:     if $Q[j]$ is not empty then
12:         Append the set $P[Q[j]]$ onto $\hat{P}$
13:         Overwrite the set $Q[j]$ with new indices of $\hat{P}$
14:     end if
15: end for
16: $P := \hat{P}$
17: return
```

Algorithm 1.3 presents the fixed-grid build process. The routine starts with the allocation an array of empty buckets, Q. Next Q is populated based on the spatially hashed cell coordinates. The second for-loop in Algorithm 1.3 iterates through Q, looking for non-empty buckets. When one is found, nodes referenced by that bucket are transcribed/appended onto the "sorted" list of nodes $\hat{P}$. This way the nodes in each cell are contiguous, but maintain the original ordering with respect to one another. Additionally, node indices in $\hat{P}$ replace the old indices within Q.

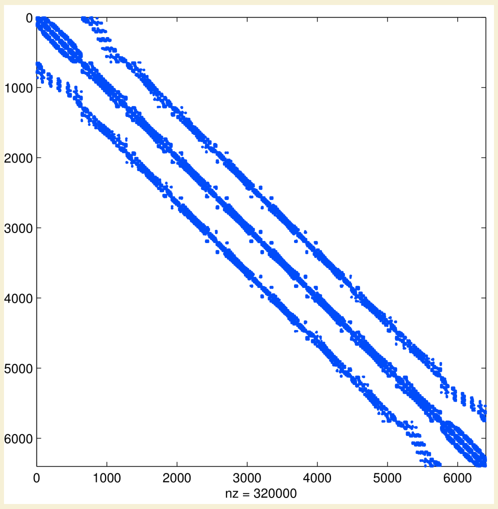
The entire build process complexity is proportional to $O(N)$, and requires $O((h_n)^d + N)$ storage. Samet [82] would classify this approach as a *fixed-grid bucket method with one-dimensional ordering*. The term *bucket* refers to the allowance for each cell to contain an arbitrary number of nodes. *One-dimensional ordering* is indicative of attempts later in the chapter to employ alternative space-filling curves in place of raster-ordering.

A special note: the final step of Algorithm 1.3 overwrites the original list of nodes with the sorted equivalent. The spatially sorted list is included in the cost of stencil generation, but available for reuse elsewhere. Since the first step in RBF-FD applications is to generate stencils, overwriting the input node set can guarantee that the node values throughout the entire life-cycle of an RBF-FD application will benefit from the same spatial locality as stencil generation. This benefit is (almost) free.

Consider Figure 1.4, which shows two differentiation matrices generated based on the same $N = 6400$ MD-node set (unit sphere), with each row representing an RBF-FD stencil of $n = 50$ non-zeros. The left matrix in Figure 1.4 is generated with stencils queried by a k-D Tree. The k-D Tree maintains the original ordering on P. The matrix on the right of Figure 1.4 is a permuted equivalent of the left matrix with fixed-grid cells sorted based on a raster-ordering for $h_n = 10$.



(a) k-D Tree



(b) Fixed-Grid

Figure 1.4: Example effects of node reordering for MD node set $N = 6400$ with $n = 50$. The differentiation matrices are permuted equivalents and roughly 0.78% full. Stencils generated based on k-D Tree maintain the original node ordering, while a fixed-grid with $h_n = 10$ condenses non-zeros for improved memory access patterns (i.e., cache reuse).

Looking at Figure 1.4 it should be obvious that reordering the nodes can improve memory access patterns for SpMV. If each row is applied as a sparse dot-product with a dense vector, the more condensed non-zeros are in the row, the more likely values from the dense vector will be resident in cache when needed. Likewise, non-zeros that appear on consecutive rows can benefit from cache reuse. Later in this chapter, the impact of spatial orderings are compared to determine how RBF-FD can benefit the most.

1.2.2 Fixed-Grid Neighbor Query

Querying the k-nearest neighbors for a single query node, X_q, is the subject of Algorithm 1.4. The process begins by finding X_q's containing cell, c. The hash value of c is used to identify (in Q) the list of nodes contained within that cell, all of which are appended to a vector of neighbor candidates, pq.

It is possible for the number nodes in c to exceed n; however, the algorithm conservatively assumes that it is necessary to search at least one valence of cells around it. This ensures that nodes near the cell boundaries will find nearby neighbors outside of c, and the stencils will balanced. For certain fixed-grid resolutions, a single valence may not satisfy the stencil size requirements, so the algorithm iterates outward onto new valences. As cells are checked, their nodes are appended onto pq.

The final stage of Algorithm 1.4 calculates the distance from all candidate nodes to X_q, and uses that metric to sort pq in ascending order. The first n nodes in pq are returned as the stencil.

Complexity of Algorithm 1.4 can vary based on the choice of h_n. For a sufficiently refined fixed-grid the k-ANN is dominated by the cost of the *while-loop* and behaves as

Algorithm 1.4 QueryFixedGrid(X_q, n, P, Q)

```
1: **Input:** A query point, $X_q$; the desired number of neighbors, $n$; a set of $d$-dimensional points $P$; and the matching cell bucket list, $Q$.
2: **Output:** The $n$-nearest neighbors list $pq$.
3:
4: $valence := 1$
5: $c :=$ CellCoords($X_q$)
6: $ind :=$ SpatialHash($cells$)
7: Append $P[Q[ind]]$ onto $pq$
8: while $pq$.size $< n$ OR $valence < 2$ do
9:     $cells :=$ NeighboringCellCoords($c, valence$)
10:    $inds :=$ SpatialHash($cells$)
11:    for each $q$ in $Q[inds]$ do
12:        if $q$ is not empty then
13:            Append node list $P[q]$ onto $pq$
14:        end if
15:    end for
16:    increment $valence$
17: end while
18: $dists :=$ ComputeDistances($pq$)
19: Sort $pq$ by $dists$
20: return the first $n$ nodes in $pq$
```

$O(\log h_n)$ per stencil. In the worst case, when h_n is small, the cost of sorting pq dominates, and is proportional to $O(N \log N)$ (using a C++ STL Sort) in the worst case. Results below demonstrate that proper choice of h_n can maintain logarithmic complexity similar to the k-D Tree query.

The fixed grid query algorithm is considered an *approximate nearest neighbor* (ANN) search. Consider again the nodes in Figure 1.1. A k-NN stencil of size $n = 8$ should contain the blue center, all blue nodes, plus the red node and one black node. The true k-NN would select the black node in the right-most column of the grid (i.e. the node closer to the dashed ball query). Under the fixed-grid method, however, the alternative black node is selected even though it is more distant. This is true because the more distant black node occupies the second valence of cells around the stencil center, whereas the true near neighbor is in the third valence. Algorithm 1.4 is able to truncate the search in the second valence by satisfying the requirement on n.

Similarly, if cells are rectangular in shape, the ball-query under fixed-grid functions as an ellipsoid. In this case, stencils are biased with more nodes in one direction. To combat this, and ensure spherical stencils, this work assumes the AABB bounding the domain is a cube (i.e., $dx = dy = dz$).

The difference between a true k-NN and k-ANN is insignificant from the perspective of RBF-FD. The method compensates automatically for differences in node locations when weights are calculated. On top of this, the only differences in stencils generated by k-NN versus k-ANN occur at nodes on the outermost reaches of the stencil (i.e., nodes with the least impact on the stencil center).

1.3 Performance Comparison

The implementation of k-D tree compared in this work, the *kd-Tree Matlab* library, was originally posted to the Matlab FileExchange in 2008 [96] and now maintained as an independent Google Code project [97]. The implementation is written in C++, but includes a MEX compiled interface, allowing for a consistent and efficient k-D Tree API in both languages. The original release of *kd*-Tree Matlab (pre-2012) was in use throughout the RBF community at the onset of this work. The dual language API was appealing for rapid-prototyping with MATLAB, and then porting applications to C++.

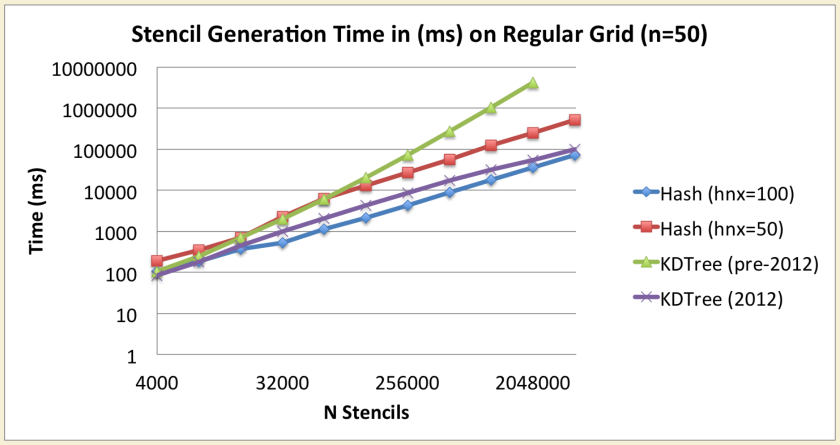
The pre-2012 implementation followed the $O(N \log N)$ expected complexity for neighbor queries, but the cost to build the tree was another story. Build times for small and medium sized grids (i.e., less than $N = 50000$ nodes) were small enough to be inconspicuous. However, large grid sizes were found to be prohibitively high, with the implementation behaving as $O(N^2)$. This issue ultimately led to work on a MATLAB prototype of the fixed-grid method ([9]) to test the concept of neighbor queries with low build costs.

The fixed-grid implementation, originally a pure MATLAB script, outperformed the "efficient" MEX-compiled k-D Tree for problem sizes $N > 20000$, and lead to the development of a faster C++ implementation tested here. However, with the 2012 release of *kd*-Tree Matlab on Google Code ([97]), the author has significantly improved the performance of the build process to achieve the $O(N \log N)$ behavior expected for a Point k-D Tree with cyclic splitting.

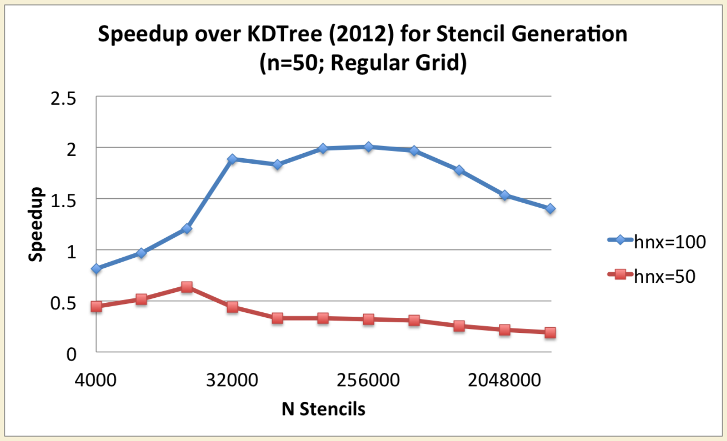
All benchmarks in this section were performed on the Itasca cluster at the Minnesota Supercomputing Institute. Itasca is an HPLinux cluster with 1,134HP ProLiant blade servers, each with two-socket, quad-core 2.8 GHz Intel Xeon processors sharing at least 24GB of RAM [2]. Both the k-D Tree and fixed-grid implementations are compiled with the Intel compiler toolchain (v13) and with the "-O3" optimizations for auto-vectorization, loop unrolling, etc.

Figure 1.5 demonstrates the performance of the k-D Tree and the fixed-grid method on increasing 3-D regular grid resolutions up to four million nodes. Both the current k-D Tree implementation (2012) and the original implementation (pre-2012) are shown on top to demonstrate the significant improvement in the latest release. For comparison, the C++ fixed-grid method is shown with two resolutions: $h_n = 50$ and $h_n = 100$. On the bottom, Figure 1.5 shows the associated speedups—defined as the ratio of time to compute k-D Tree stencils over the time for the same work in fixed-grid—achieved by the fixed-grid method over the 2012 release of *kd*-Tree Matlab. These timings cover the total time to build the data-structure and query all stencils. Also, the original node ordering is in raster-order, as is the fixed-grid.

Prior to the 2012 improvements, the C++ implementation was over 130x faster for four million nodes. With the newer and more reasonable k-D Tree benchmarks, the fixed-grid method is only 2x faster in the best case shown here. Although 2x is not as impressive, it is a notable improvement. One issue in Figure 1.5 is that the fixed-grid with $h_n = 50$ is consistently more than twice as slow as the k-D Tree. This is due to under-resolved cells with 32 nodes per cell; $h_n = 100$ has only 4 nodes per cell. For small N the k-D Tree is faster than both resolutions of fixed grid due to over-resolution. As N increases beyond one million nodes the $h_n = 100$ fixed-grid loses ground on the k-D Tree due to under-resolution.



(a) 3-D Regular Grid with stencil size $n = 50$



(b) Speedup of fixed-grid method versus k-D Tree

Figure 1.5: Querying the $n = 50$ nearest neighbors on a regular grid up to $N = 160^3$ demonstrates the gains achieved by the fixed-grid neighbor query method.

Similar behavior is seen in Figure 1.6 where the fixed-grid and k-D Tree are compared for various discretizations of the unit sphere. Each of the node sets described in Figure 1.6 are discussed in Chapter **??** and available for download ([10]). A range of resolutions are covered from a few hundred up to one million with some overlap. Each distribution (MD, Icosahedral, CVT) are generated differently and the nodes are naturally spatially sorted (icosahedral) or random (MD, CVT). Due to sorting in the fixed-grid method, node sets that are originally random exhibit more gain over k-D Tree. This is most evident in Figure 1.6, where different slopes appear for the segment corresponding to MD nodes and similar resolutions of Icosahedral nodes. The speedup curves for $h_n = 50$ and $h_n = 100$ demonstrate the dependence of fixed-grid method's success on the proper choice of h_n. When over-resolved (i.e., for $N < 10000$) the k-D Tree performs best. Under-resolution degrades performance rapidly starting at $N = 100000$ for $h_n = 50$, and $N = 500000$ for $h_n = 100$.

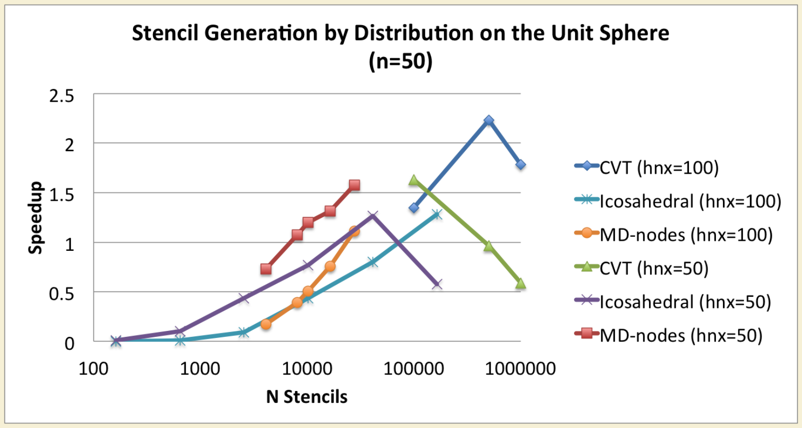


Figure 1.6: Fixed-grid speedup versus k-D Tree with stencil size $n = 50$.

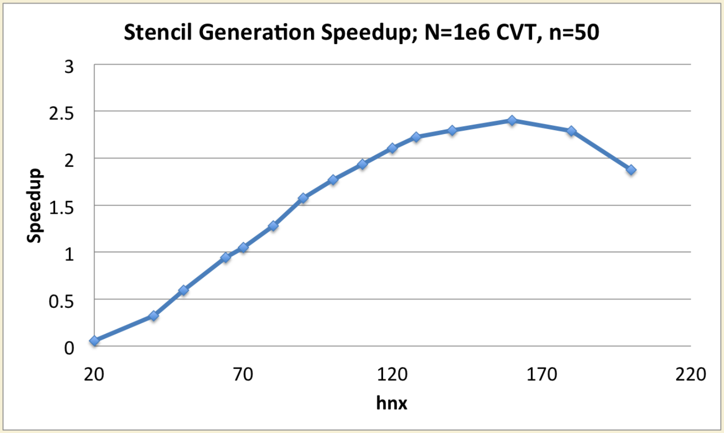
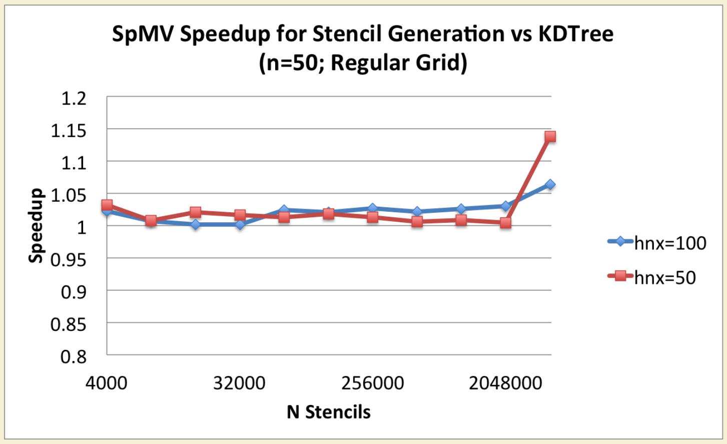


Figure 1.7: Fixed-grid speedup versus k-D Tree on a one-million node CVT (unit sphere), with stencil size $n = 50$.

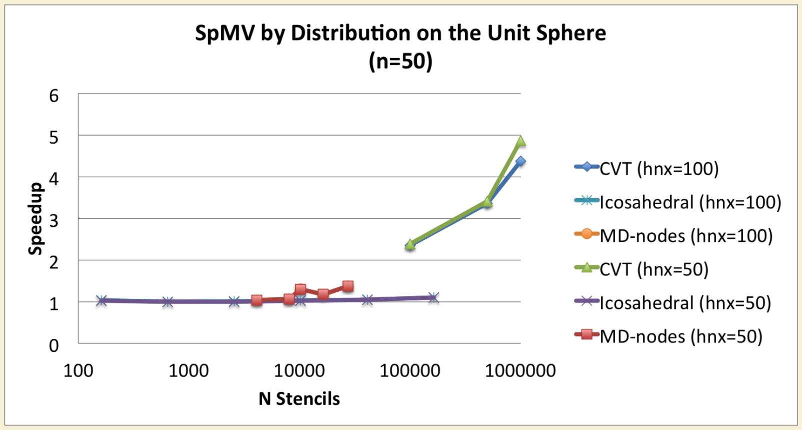
The speedup curves in Figures 1.5 and 1.6 hint at a maximum value h_n for which the fixed-grid will outperform its competitor, and with that value: a maximum speedup possible. In Figure 1.7, the $N = 10^6$ resolution CVT is used to generate stencils of $n = 50$, with range of values for h_n. The maximum of this curve shows that the fixed-grid can achieve up to 2.4x better than the k-D Tree for $h_n = 160$. In this case the number of cells, $(h_n)^3$, sufficiently resolves the domain for most cells intersecting the sphere to have one or two nodes. At $h_n = 70$ the fixed-grid is on par with k-D Tree; most cells have less than 8 nodes, which is consistent with the resolution sought by [50, 51, 65].

1.3.1 Impact on SpMV

Based on evidence so far, the fixed-grid method is not a big winner against the k-D Tree, but rather, it ekes out a small victory with the right choice of h_n for fewer than 8 nodes per cell. The reality is that since stencil generation only occurs once, and the difference between k-D Tree and fixed-grid is easily amortized by iterations during the bulk of the RBF-



(a) 3-D Regular Grid



(b) Unit Sphere

Figure 1.8: Fixed-grid impact on SpMV for stencil size $n = 50$.

FD application phase. However, as Figure 1.8 illustrates, the fixed-grid method includes another longer lasting benefit. Namely, the Sparse Matrix-Vector Multiply (SpMV)—the major overhead in RBF-FD applications—benefits positively due to the reordering that occurs during stencil generation. Reordering cells improves locality of nearby nodes and associated values, and leads to up to 5x speedup in the SpMV for random node distributions.

Note that regular grid and icosahedral test cases in Figure 1.8 see no more than 5% improvement, which is is to be expected as they are previously spatially sorted node distributions. CVT and MD node sets, however, see a maximum speedup of 5x and 1.4x respectively. Furthermore, none of the test cases result in SpMV slowdown, so a case can be made for use of the fixed-grid query as a general safety net to precondition the system when properties of the input grid are unknown.

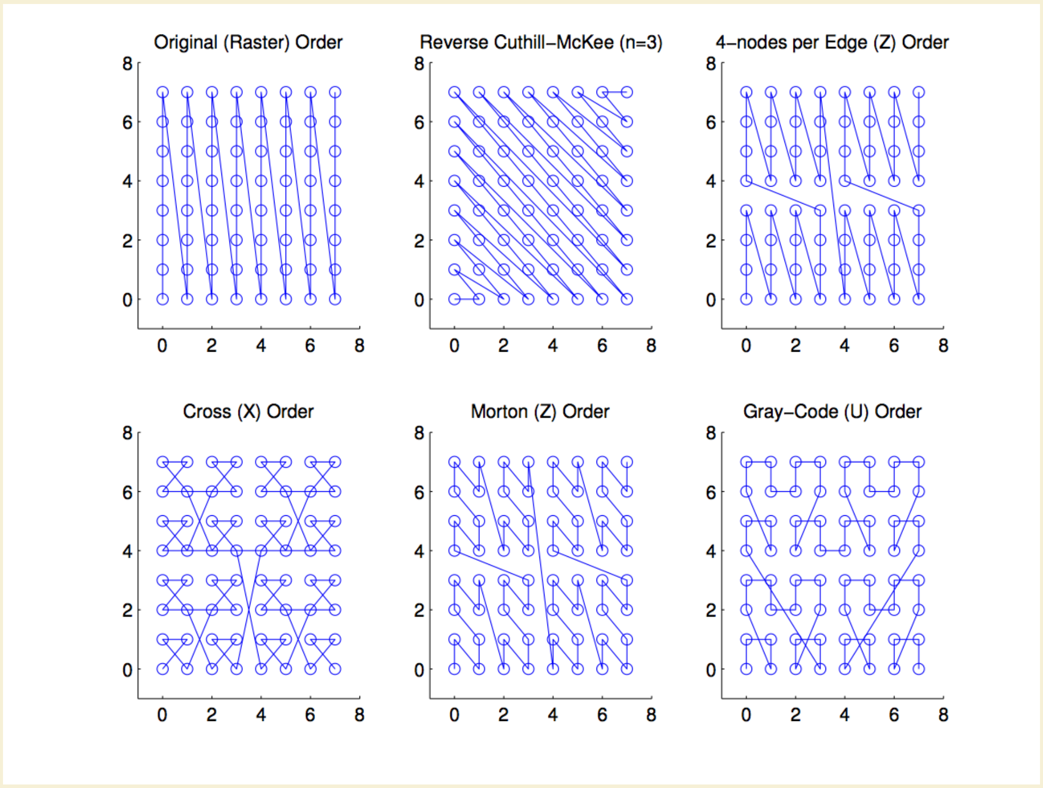


Figure 1.9: Example space filling curves used to reorder cells/nodes. The Raster, Z-, X-, U- and 4-nodes per Edge Z-orderings are space filling curves applied to reorder cells of the fixed-grid stencil queries. Reverse Cuthill-McKee (RCM) operates on output stencils and their associated adjacency graph. The RCM shown here is a special case with only 3 neighbors per node.

1.4 Alternative Orderings

As previously mentioned, the choice of space filling curve can impact the sparsity pattern of differentiation matrices, which in turn correlates to new memory access patterns. The use of Z-ordering for optimized memory access is common in fixed-grid methods (e.g., see the suggestions by [51, 59, 65] and implementation in [50]). While prototyping the fixed-grid method, a few attempts were made to reorder cells following Z-ordering and other space filling curves. Code to test ordering in this section is included in the MATLAB fixed-grid prototype[9].

Examples of those curves are presented in Figure 1.12. Here each node would represent a cell, and the orderings start at $(0, 0)$. Of the six cases shown in Figure 1.12, five of them (Raster, X, Z, U, 4-node Z) can be directly applied to reorder the cells of the fixed-grid. Each of the five tiles in Figure 1.12 get their name based on how the curve connects nodes. The sixth ordering in Figure 1.12, Reverse Cuthill-McKee, is a special case of space filling curve saved for later discussion in this section.

Excluding RCM, the remaining orderings in Figure 1.12 are constructed as a function of node coordinates, (c_x, c_y). Raster-ordering, discussed above, follows $(c_x * h_n) + c_y$. To

Table 1.1: 2-D Integer Dilation Masks for 32-Bit Integers

Level (i)	Mask (M_i)		S_i
0	$00000000000000001111111111111111_2$	0x0000$ffff$	32
1	$11111111000000000000000011111111_2$	0xff0000ff	16
2	$00001111000000001111000000001111_2$	0x0f00f00f	8
3	$11000011000011000011000011000011_2$	0xc30c30c3	4
4	$01001001001001001001001001001001_2$	0x49249249	2

produce the remaining curves, one turns to *bit-interleaving* (see e.g., [95]).

1.4.1 Bit-Interleaved Orderings

Bit-interleaving zips bits from two or more integers into one. For example, given two integers, X, and Y the bits are interleaved as:

$$\begin{aligned} X &= \{x_3x_2x_1x_0\} \\ Y &= \{y_3y_2y_1y_0\}, \\ interleave(X,Y) &= \{x_3y_3x_2y_2x_1y_1x_0y_0\}. \end{aligned} \tag{1.2}$$

Including a third value, Z, results in the following:

$$interleave(X,Y,Z) = \{x_3y_3z_3x_2y_2z_3x_1y_1z_3x_0y_0z_3\}.$$

Interleaving requires *integer dilation* modify an input integer value, so the output value is the original input bits spaced by new zeros. On a d-dimensional domain, bit-interleaving requires $d-1$ zeros between each bit. For example, dilating 1111_2 converts the four bit integer to 01010101_2 in 2-D, and 001001001001_2 in 3-D.

Dilating integers requires multiple bit-mask operations to spread the digits appropriately. For any b-bit integer, X_0, the dilation is given by the following iteration:

$$X_{i+1} = ((X_i \ll S_i) \mid X_i) \ \& \ M_i \qquad \text{for } i = 0, ..., \log_2(b) \tag{1.3}$$

where operators $\ll$, $|$, and & indicate left-shift, bit-wise OR, and bit-wise AND operators respectively. In general, a b-bit integer requires $\log_2(b)$ iterations through the dilation kernel. The necessary masks (M_i) and shifts (S_i) necessary to dilate a 32-bit integer are provided in Table 1.1 for 2-D, and Table 1.2 for 3-D. Both binary and hexadecimal representations are provided for M_i. Observe that the shifts and bit-masks applied during each iteration depend on the size of the integer and the dimension (refer to [95] for more details).

It is possible to lose bits in translation when dilating values that require greater than ${}^{b}/_{d}$ bits. In the case of 32-bits, input Equation 1.3 must be representable in 16-bits or less (i.e., ≤ 65535) for 2-D, and 10-bits or less (i.e., ≤ 1023) for 3-D. Note that these limits then become the maximum resolution possible for h_n in the fixed-grid method.

With dilation available, the process for 2-D bit-interleaving reduces to:

$$\begin{aligned} (d_x, d_y) &= (dilate(X), dilate(Y)) \\ interleave(X,Y) &= (d_x \ll 1) \mid d_y. \end{aligned} \tag{1.4}$$

Table 1.2: 3-D Integer Dilation Masks for 32-Bit Integers

Level (i)	Mask (M_i)		Shift (S_i)
0	$00000000000000001111111111111111_2$	0x0000$ffff$	48
1	$00000000000000000000000011111111_2$	0x000000ff	24
2	$00000000000011110000000000001111_2$	0x000f000f	12
3	$00000011000000110000001100000011_2$	0x03030303	6
4	$00010001000100010001000100010001_2$	0x11111111	3

Table 1.3: Integer Dilation Interleaving Operators

Ordering	2-D	3-D
IJK	$(X * h_n) + Y$	$((X * h_n) + Y) * h_n + Z$
Z	$interleave(X, Y)$	$interleave(X, Z, Y)$
U	$interleave(X, X \oplus Y)$	$interleave(X, Z, Z \oplus Y)$
X	$interleave(X \oplus Y, X)$	$interleave(X \oplus Y, Z, Y)$
4-Node Z *	$(d_x \ll 2) \mid d_y$	$((d_x \ll 4) \mid (d_z \ll 2)) \mid d_y$

Or in 3-D:

$$\begin{aligned} (d_x, d_y, d_z) &= (dilate(X), dilate(Y), dilate(Z)) \\ interleave(X, Y, Z) &= ((d_x \ll 2) \mid (d_y \ll 1)) \mid d_z. \end{aligned} \tag{1.5}$$

Equations 1.4 and 1.5 are used as hash functions to produce a vast number of space filling curves (see [95]).

By default, Equation 1.4 produces the Z-ordering in Figures 1.12 and 1.3. Consider for example:

$$\begin{aligned} (c_x, c_y) &= (5, 3) = (0101_2, 0011_2) \\ dilate((c_x, c_y), 2) &= (d_x, d_y) = (00010001_2, 00000101_2) \\ interleave(c_x, c_y) &= (00100111_2) = 39. \end{aligned} \tag{1.6}$$

Here, the cell coordinates, $(5, 3)$, are mapped to the value 39. This Z-index can be verified by counting traversals required to get from $(0, 0)$ to $(5, 3)$ on the Z-order curve in Figure 1.12.

The X- and U- orders shown in Figure 1.12 also result from Equation 1.4 with only slight modifications to the input. Table 1.3 provides operators and inputs in 2-D and 3-D for each space filling curve. The U and X orderings depend a bit-wise XOR operator, $\oplus$, to combine coordinates before interleaving.

The 4-node Z-curve on the last row of Table 1.3 is a special case. It is the same in theory as a standard Z-order, but during dilation it requests one extra zero per bit as though the dilation were for the next higher dimension. Then, interleaving the dilated coordinates following the operators provided in Table 1.3 gives the following interleave pattern:

$$interleave_{4node}(X, Y) = \{x_5 x_4 y_5 y_4 x_3 x_2 y_3 y_2 x_1 x_0 y_1 y_0\}. \tag{1.7}$$

By reserving two bits for each dimension the space filling curve is able to traverse twice as many nodes in the Y direction before taking its first step in the X direction.

Often, interleaved cell coordinates near the boundaries of the domain do not produce contiguous hash indices and may balloon to values greater than the number of nodes/cells in the domain. In these situations, missing indices reference coordinates outside the domain that would complete the hierarchical power-of-2 curves before entering the domain again. When sorted and traversed least to greatest, the hashed indices give a proper ordering of the domain, and missing indices are skipped.

1.4.2 Cuthill-McKee

The set of orderings considered up to this point have focused on spatially sorting cells during the stencil generation phase. The Cuthill-McKee family of algorithms change that focus to reorder nodes *after* stencils are generated.

reordering nodes focuses on the differentiation matrix.

directly to reduce bandwidth. A commonly used algorithm for this is called *Reverse Cuthill-McKee* (RCM) reordering. RCM takes as input a square, symmetric adjacency graph and returns a vector of indices indicating the new order for each row of the matrix.

The RCM algorithm first finds the row with fewest non-zeros corresponding to the node in the graph with the lowest degree of connectivity. traverses the adjacency graph

The Cuthill McKee algorithms can be equated to a breadth-first search over the adjacency graph. The algorithm starts by

The result array will be interpreted like this: R[L] = i means that the new label of node i (the one that had the initial label of i) will be L.

The algorithm queues nodes in order of degree at each level of the search and traverses the lowest degree priority. The Reverse variant of Cuthill-McKee inverts the node order so that the lowest degree and top level node are at the end of the matrix rather than the beginning. Aside from ordering, the Reverse and Standard Cuthill McKee algorithms are identical processes. However, RCM is the more popular of the variants due to reduced fill-in that occurs when some matrix decompositions are applied to the reordered matrix ([69]).

Cuthill McKee implementations assume the algorithm operates on square, symmetric adjacency graphs.

Rather than complete a full analysis of the impact from space filling curves, we only consider the ideal cases from CVT and RG (because they are used in the thesis). The integer dilation is costly and does not appear to significantly improve the performance of the best cases.

RCM is effective and can be used as post-processing once stencils are generated and/or DMs have been assembled.

1.4.3 Impact of Node Orderings

The ideal differentiation matrix, corresponding to discretization of a line, would have all non-zeros on the first $\frac{n-1}{2}$ to each side of the diagonal. For 2- or 3-D domains the ideal case is not possible to achieve, but the nodes can be reordered with the goal to condense At the end of BuildFixedGrid the original set of nodes, P is overwritten by $\hat{P}$. While nothing requires that $\hat{P}$ replace the usage of P

[71] found that reodering nodes via RCM and space filling curves offer similar benefits in terms of reduced TLB misses and better cache coherency.

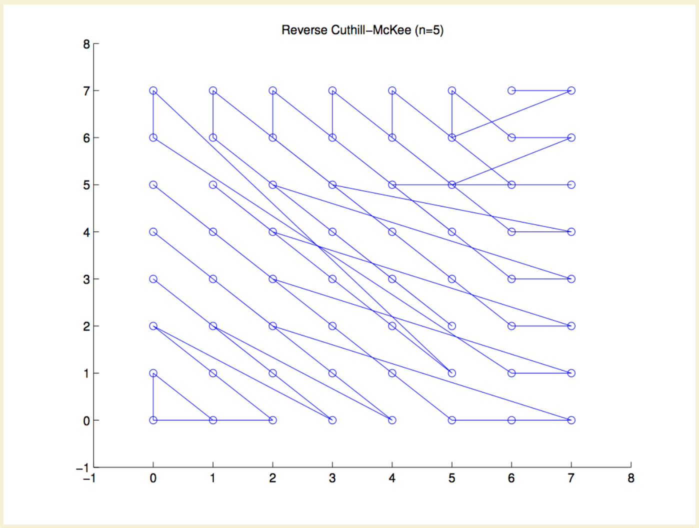


Figure 1.10: An RCM reordering of nodes based on stencils $n = 5$ nearest neighbors per node.

The RCM algorithm is a breadth first search over stencils to order them based on the minimum degree (number of connections per stencil). The zig-zag ordering shown in Figure 1.12 is an ideal case where all nodes are connected to two neighbors (i.e., $n = 3$). In practice the curve trends diagonally across the domain due to the BFS property, but the ordering often appears to connect nodes randomly

Obviously, the ideal case for bandwidth is when all rows contain the $\frac{n}{2}$ nodes corresponding to solution value to either side of u_j. In 1-D this corresponds to every node containing the $\frac{n}{2}$ nodes to the left and right of x_j. In 2-D this is only possible if the nodes in the domain are properly indexed such that stencils contain the proper set of neighbors—a stringent requirement that will

impact from ordering on matrix sparsity. Bandwidth impact. Bandwidth impact on condition considered in future chapter.

One frequently hears that ordering via space filling curves like Morton Ordering and/or gray codes can benefit memory access patterns.

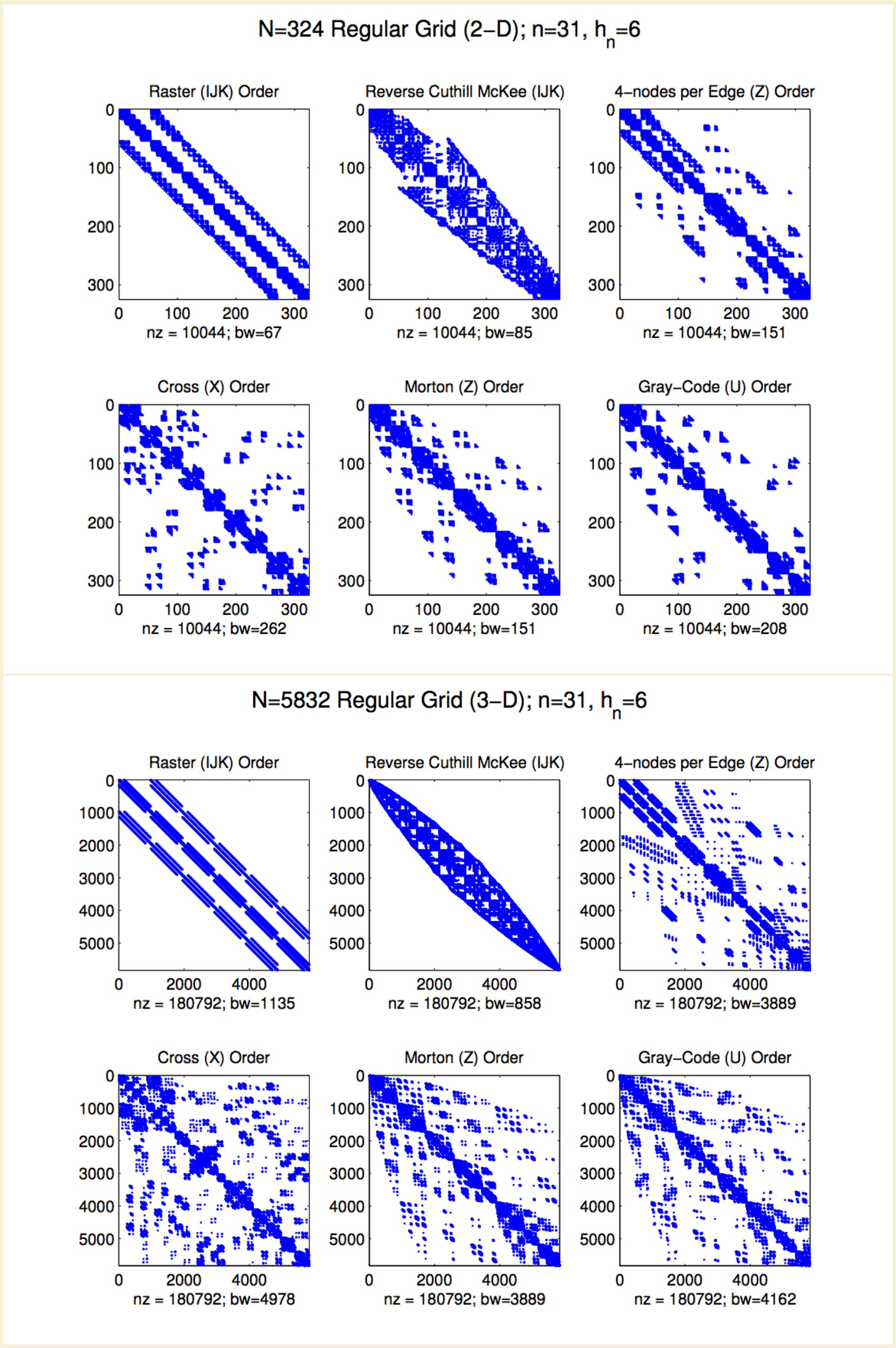


Figure 1.11: In order: a) node ordering test cases; b) original ordering of regular grid (raster); c) coarse grid overlay for hash functions ($hnx = 6$); d) example stencil ($n = 31$) spanning multiple Z's; e) spy of DM after orderings. 3) Spy impact on MD node set $N = 4096$, stencil size $n = 31$, $h_n = 6$

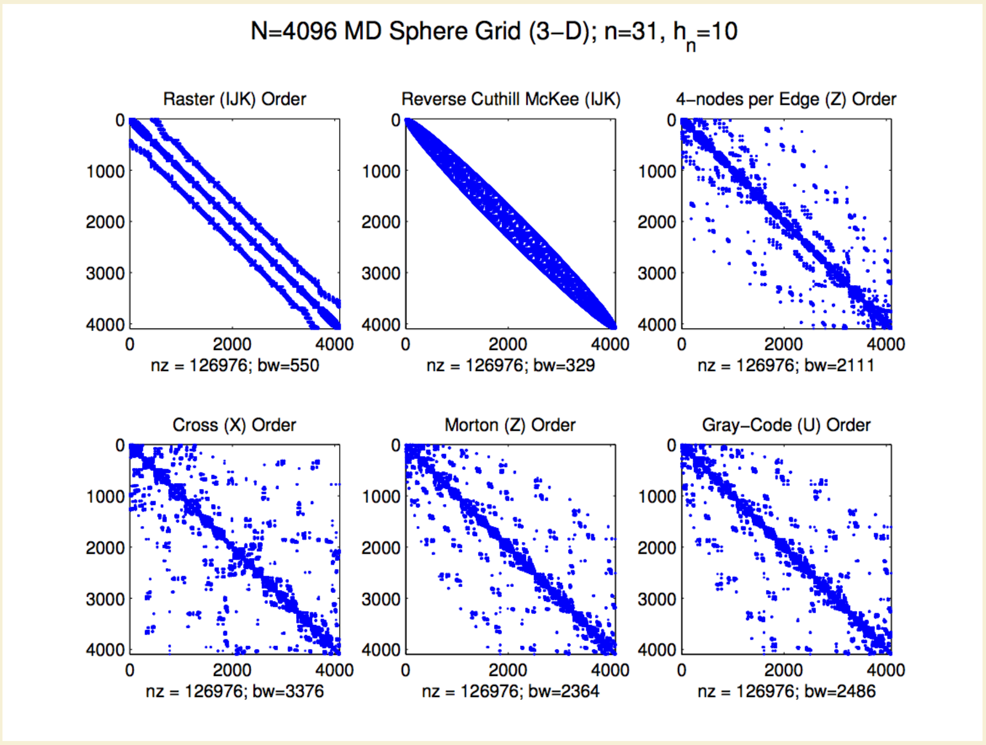


Figure 1.12: The impact of reordering on sparsity patterns for $N = 4096$ MD nodes on the sphere. Stencil size $n = 31$, fixed-grid resolution per dimension $h_n = 10$.

1.5 Conclusion and Future Work

Although RBF-FD only requires neighbor queries once, the results that follow reveal a long lasting positive impact on memory with a fixed-grid method, which is sufficient to justify its use. Investigations into moving node coordinates and/or local refinements for RBF methods (e.g., [34]) would find the fixed-grid method significantly more beneficial. As of this writing no known applications of RBF-FD consider moving nodes

Due to the limited significance of stencil generation under RBF-FD, the overhead in implementing and debugging the fixed-grid method on the GPU is difficult to justify. The implementation tested here was developed as a pure CPU prototype with minimal attention to optimization. The added complexity in reproducing the efficient fixed-grid method on the GPU could be the subject of future work for moving nodes.

[18] could benefit the algorithm by sorting nodes completely based on floating point Z-ordering.

It is worth noting that recent work by Connor and Kumar [18] has developed operators capable of producing Z-orderings based on floating point coordinates.

For quasi-regular distributions and small to medium sized grids the k-D Tree performs well enough in comparison to the fixed-grid method. However, the difference in

There is an ideal h_n.

1.5.1 Future Work

While more efficient implementations are possible, the savings demonstrated by Figure **??** the savings are significant with the right choice of h_n. Generating stencils for RBF-FD is a preprocessing cost, so we do not dedicate an excessive amount of attention to this algorithm. However, a few ideas that would improve: hilbert ordering, choose AABB resolution based on N not user parameters, faster sorting, GPU implementation

Open item from to this section include:

- An efficient GPU k-NN implementation for the investigation of RBF-FD in a moving nodes or adaptive mesh refinement situation. Examples include [18, 50, 78]
- Space filling curves are limited to a MATLAB prototype at the moment. These can be implemented in C++ for integers, with a potential investigation into floating point dilation and orderings [18].
-

[81] mentions the impact of ordering on conditioning.
TODO:

- Node orderings based on integer dilation
  - RCM algorithm high level (?)
  - Define bandwidth metric
  - bandwidth implications on memory access (lower bandwidth is usually better)
  - table showing the impact of orderings on bandwidths (already shown in figure, for N=4096).

BIBLIOGRAPHY

[1] HMPP Data Sheet. http://www.caps-entreprise.com/upload/article/fichier/64fichier1.pdf, 2009.

[2] Itasca (hp proliant bl280c g6 linux cluster). `https://www.msi.umn.edu/hpc/itasca`, June 2013. 11

[3] Kdtreesearcher class. MATLAB R2013a Documentation (`http://www.mathworks.com/help/stats/kdtreesearcherclass.html`), Aug 2013. 1

[4] AccelerEyes. *Jacket User Guide - The GPU Engine for MATLAB*, 1.2.1 edition, November 2009.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[6] Sylvie Barak. Gpu technology key to exascale says nvidia. `http://www.eetimes.com/electronics-news/4230659/GPU-technology-key-to-exascale-says-Nvidia`, November 2011.

[7] R. K. Beatson, W. A. Light, and S. Billings. Fast Solution of the Radial Basis Function Interpolation Equations: Domain Decomposition Methods. *SIAM J. Sci. Comput.*, 22(5):1717–1740, 2000.

[8] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, (1):1, 2009.

[9] E. F. Bollig. A fixed-grid prototype for rbf stencil generation with various space filling curves. `https://github.com/bollig/rbf_hash_query`, 2012. 11, 15

[10] E. F. Bollig. Sphere grids. `https://github.com/bollig/sphere_grids`, Aug 2012. 12

[11] Evan F. Bollig, Natasha Flyer, and Gordon Erlebacher. Solution to PDEs using radial basis function finite-differences (rbf-fd) on multiple GPUs. *Journal of Computational Physics*, 231(21):7133 – 7151, 2012.

[12] Andreas Brandstetter and Alessandro Artusi. Radial Basis Function Networks GPU Based Implementation. *IEEE Transaction on Neural Network*, 19(12):2150–2161, December 2008.

[13] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM.

[14] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth Surface Reconstruction from Noisy Range Data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–ff, New York, NY, USA, 2003. ACM.

[15] Tom Cecil, Jianliang Qian, and Stanley Osher. Numerical Methods for High Dimensional Hamilton-Jacobi Equations Using Radial Basis Functions. *JOURNAL OF COMPUTATIONAL PHYSICS*, 196:327–347, 2004.

[16] G Chandhini and Y Sanyasiraju. Local RBF-FD Solutions for Steady Convection-Diffusion Problems. *International Journal for Numerical Methods in Engineering*, 72(3), 2007.

[17] P P Chinchapatnam, K Djidjeli, P B Nair, and M Tan. A compact RBF-FD based meshless method for the incompressible Navier–Stokes equations. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 223(3):275–290, March 2009.

[18] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16:599–608, 2010. 21, 22

[19] CD Correa, D Silver, and M Chen. Volume Deformation via Scattered Data Interpolation. *Proceedings of Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 91–98, 2007.

[20] A Corrigan and HQ Dinh. Computing and Rendering Implicit Surfaces Composed of Radial Basis Functions on the GPU. *International Workshop on Volume Graphics*, 2005.

[21] N Cuntz, M Leidl, TU Darmstadt, GA Kolb, CR Salama, M Böttinger, D Klimarechenzentrum, and G Hamburg. GPU-based Dynamic Flow Visualization for Climate Research Applications. *Proc. SimVis*, pages 371–384, 2007.

[22] Salvatore Cuomo, Ardelio Gallettiy, Giulio Giuntay, and Alfredo Staracey. Surface reconstruction from scattered point via rbf interpolation on gpu. *CoRR*, abs/1305.5179, 2013.

[23] Oleg Davydov and Dang Thi Oanh. On the optimal shape parameter for gaussian radial basis function finite difference approximation of the poisson equation. *Computers Mathematics with Applications*, 62(5):2143 – 2161, 2011.

[24] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008. 3, 4

[25] E Divo and AJ Kassab. An Efficient Localized Radial Basis Function Meshless Method for Fluid Flow and Conjugate Heat Transfer. *Journal of Heat Transfer*, 129:124, 2007.

[26] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[27] Qiang Du, Max D. Gunzburger, and Lili Ju. Voronoi-based Finite Volume Methods, Optimal Voronoi Meshes, and PDEs on the Sphere. *Computer Methods in Applied Mechanics and Engineering*, 192:3933–3957, August 2003.

[28] G. E. Fasshauer. RBF Collocation Methods and Pseudospectral Methods. Technical report, 2006.

[29] Gregory E. Fasshauer. Solving Partial Differential Equations by Collocation with Radial Basis Functions. In *In: Surface Fitting and Multiresolution Methods A. Le M'ehaut'e, C. Rabut and L.L. Schumaker (eds.), Vanderbilt*, pages 131–138. University Press, 1997.

[30] Gregory E. Fasshauer. *Meshfree Approximation Methods with MATLAB*, volume 6 of *Interdisciplinary Mathematical Sciences*. World Scientific Publishing Co. Pte. Ltd., 5 Toh Tuck Link, Singapore 596224, 2007. 2, 4

[31] Gregory E. Fasshauer and Jack G. Zhang. On choosing "optimal" shape parameters for rbf approximation. *Numerical Algorithms*, 45(1-4):345–368, 2007.

[32] A. I. Fedoseyev, M. J. Friedman, and E. J. Kansa. Improved Multiquadric Method for Elliptic Partial Differential Equations via PDE Collocation on the Boundary. *Computers & Mathematics with Applications*, 43(3-5):439 – 455, 2002.

[33] Natasha Flyer and Bengt Fornberg. Radial basis functions: Developments and applications to planetary scale flows. *Computers & Fluids*, 46(1):23–32, July 2011.

[34] Natasha Flyer and Erik Lehto. Rotational transport on a sphere: Local node refinement with radial basis functions. *Journal of Computational Physics*, 229(6):1954–1969, March 2010. 21

[35] Natasha Flyer, Erik Lehto, Sebastien Blaise, Grady B. Wright, and Amik St-Cyr. Rbf-generated finite differences for nonlinear transport on a sphere: shallow water simulations. *Submitted to Elsevier*, pages 1–29, 2011. 2, 4

[36] Natasha Flyer and Grady B. Wright. Transport schemes on a sphere using radial basis functions. *Journal of Computational Physics*, 226(1):1059 – 1084, 2007.

[37] Natasha Flyer and Grady B. Wright. A Radial Basis Function Method for the Shallow Water Equations on a Sphere. In *Proc. R. Soc. A*, volume 465, pages 1949–1976, December 2009.

[38] B Fornberg, T Driscoll, G Wright, and R Charles. Observations on the behavior of radial basis function approximations near boundaries. *Computers & Mathematics with Applications*, 43(3-5):473–490, February 2002.

[39] B Fornberg, N Flyer, and JM Russell. Comparisons Between Pseudospectral and Radial Basis Function Derivative Approximations. *IMA Journal of Numerical Analysis*, 2009.

[40] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Computers & Mathematics with Applications*, 48(5-6):853 – 867, 2004.

[41] Bengt Fornberg and Natasha Flyer. Accuracy of Radial Basis Function Interpolation and Derivative Approximations on 1-D Infinite Grids. *Adv. Comput. Math*, 23:5–20, 2005.

[42] Bengt Fornberg, Elisabeth Larsson, and Natasha Flyer. Stable Computations with Gaussian Radial Basis Functions. *SIAM J. on Scientific Computing*, 33(2):869—-892, 2011.

[43] Bengt Fornberg and Erik Lehto. Stabilization of RBF-generated finite difference methods for convective PDEs. *Journal of Computational Physics*, 230(6):2270–2285, March 2011. 2, 4

[44] Bengt Fornberg and Erik Lehto. Stabilization of rbf-generated finite difference methods for convective pdes. *J. Comput. Physics*, 230(6):2270–2285, 2011.

[45] Bengt Fornberg, Erik Lehto, and Collin Powell. Stable calculation of gaussian-based rbf-fd stencils. Technical Report 2012-018, Uppsala University, Numerical Analysis, 2012.

[46] Bengt Fornberg and Cécile Piret. A Stable Algorithm for Flat Radial Basis Functions on a Sphere. *SIAM Journal on Scientific Computing*, 30(1):60–80, 2007.

[47] Bengt Fornberg and Cécile Piret. On Choosing a Radial Basis Function and a Shape Parameter when Solving a Convective PDE on a Sphere. *Journal of Computational Physics*, 227(5):2758 – 2780, 2008.

[48] Richard Franke. Scattered Data Interpolation: Tests of Some Method. *Mathematics of Computation*, 38(157):181–200, 1982.

[49] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977. 4

[50] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive sph simulation and rendering on the gpu. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 55–64, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association. 7, 13, 15, 22

[51] S. Green. Cuda particles. NVidia Whitepaper, 2010. 2, 5, 7, 13, 15

[52] N. A. Gumerov, R. Duraiswami, and E. A. Borovikov. Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions. Technical Report UMIACS-TR-2003-28, University of Maryland (College Park, Md.), Apr 2003. 2

[53] R Hardy. Multiquadratic Equations of Topography and Other Irregular Surfaces. *J. Geophysical Research*, (76):1–905, 1971.

[54] Y. C. Hon and R. Schaback. On unsymmetric collocation by radial basis functions. *Appl. Math. Comput.*, 119(2-3):177–186, 2001.

[55] Yiu-Chung Hon, Kwok Fai Cheung, Xian-Zhong Mao, and Edward J. Kansa. A Multiquadric Solution for the Shallow Water Equations. *ASCE J. Hydraulic Engineering*, 125:524–533, 1999.

[56] A. Iske. *Multiresolution Methods in Scattered Data Modeling*. Springer, 2004.

[57] L. Ivan, H. De Sterck, S. A. Northrup, and C. P. T. Groth. Three-Dimensional MHD on Cubed-Sphere Grids: Parallel Solution-Adaptive Simulation Framework. In *20th AIAA CFD Conference*, number 3382, pages 1325–1342, 2011.

[58] R. Jakob-Chien, J.J. Hack, and D.L. Williamson. Spectral transform solutions to the shallow water test set. *Journal of Computational Physics*, 119(1):164–187, 1995.

[59] I. Johnson. Real-time particle systems in the blender game engine. Master's thesis, Florida State University, November 2011. 2, 7, 15

[60] E J Kansa. Multiquadrics–A scattered data approximation scheme with applications to computational fluid-dynamics. I. Surface approximations and partial derivative estimates. *Computers Math. Applic*, (19):127–145, 1990.

[61] E J Kansa. Multiquadrics–A scattered data approximation scheme with applications to computational fluid-dynamics. II. Solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers Math. Applic*, (19):147–161, 1990.

[62] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[63] Khronos OpenCL Working Group. *The OpenCL Specification (Version: 1.0.48)*, October 2009.

[64] G Kosec and B Šarler. Solution of thermo-fluid problems by collocation with local pressure correction. *International Journal of Numerical Methods for Heat & Fluid Flow*, 18, 2008.

[65] Øystein E. Krog. GPU-based Real-Time Snow Avalanche Simulations. Master's thesis, Norwegian University of Science and Technology, June 2010. 2, 5, 6, 7, 13, 15

[66] Elisabeth Larsson and Bengt Fornberg. A Numerical Study of some Radial Basis Function based Solution Methods for Elliptic PDEs. *Comput. Math. Appl*, 46:891–902, 2003.

[67] Shaofan Li and Wing K. Liu. *Meshfree Particle Methods*. Springer Publishing Company, Incorporated, 2007.

[68] Yuxu Lin, Chun Chen, Mingli Song, and Zicheng Liu. Dual-RBF based surface reconstruction. *Vis Comput*, 25(5-7):599–607, May 2009.

[69] Wai-Hung Liu and Andrew H. Sherman. Comparative analysis of the cuthill-mckee and the reverse cuthill-mckee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, 13(2):pp. 198–213, Apr 1976. 18

[70] X. Liu, G.R. Liu, K. Tai, and K.Y. Lam. Radial point interpolation collocation method (RPICM) for partial differential equations. *Computers & Mathematics with Applications*, 50(8-9):1425 – 1442, 2005.

[71] John Mellor-crummey, David Whalley, and Ken Kennedy. Improving memory hierarchy performance for irregular applications using data and computation reorderings. In *International Journal of Parallel Programming*, pages 425–433, 2001. 18

[72] Mohamed F. Mokbel, Walid G. Aref, and Ibrahim Kamel. Performance of multi-dimensional space-filling curves. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, GIS '02, pages 149–154, New York, NY, USA, 2002. ACM.

[73] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[74] C.T. Mouat and R.K. Beatson. RBF Collocation. Technical Report UCDMS2002/3, Department of Mathematics & Statistics, University of Canterbury, New Zealand, February 2002.

[75] Ramachandran D. Nair and Christiane Jablonowski. Moving Vortices on the Sphere: A Test Case for Horizontal Advection Problems. *Monthly Weather Review*, 136(2):699–711, February 2008.

[76] R.D. Nair, S.J. Thomas, and R.D. Loft. A discontinuous Galerkin transport scheme on the cubed sphere. *Monthly Weather Review*, 133(4):814–828, April 2005.

[77] NVidia. *NVIDIA CUDA - NVIDIA CUDA C - Programming Guide version 4.0*, March 2011.

[78] Jia Pan and Dinesh Manocha. Fast GPU-based Locality Sensitive Hashing for K-Nearest Neighbor Computation. *Proceedings of the 19th ACM SIGSPATIAL GIS '11*, 2011. 22

[79] Portland Group Inc. *CUDA Fortran Programming Guide and Reference*, 1.0 edition, November 2009.

[80] DA Randall, TD Ringler, and RP Heikes. Climate modeling with spherical geodesic grids. *Computing in Science & Engineering*, pages 32–41, 2002.

[81] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial Mathematics, second edition, 2003. 22

[82] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. 2, 3, 4, 5, 7, 8

[83] C. Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

[84] Robert Schaback. Multivariate Interpolation and Approximation by Translates of a Basis Function. In C.K. Chui and L.L. Schumaker, editors, *Approximaton Theory VIII–Vol. 1: Approximation and Interpolation*, pages 491–514. World Scientific Publishing Co., Inc, 1995.

[85] J. Schmidt, C. Piret, B.J. Kadlec, D.A. Yuen, E. Sevre, N. Zhang, and Y. Liu. Simulating Tsunami Shallow-Water Equations with Graphics Accelerated Hardware (GPU) and Radial Basis Functions (RBF). In *South China Sea Tsunami Workshop*, 2008.

[86] J. Schmidt, C. Piret, N. Zhang, B.J. Kadlec, D.A. Yuen, Y. Liu, G.B. Wright, and E. Sevre. Modeling of Tsunami Waves and Atmospheric Swirling Flows with Graphics Processing Unit (GPU) and Radial Basis Functions (RBF). *Concurrency and Computat.: Pract. Exper.*, 2009.

[87] C. Shu, H. Ding, and K. S. Yeo. Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 192(7-8):941 – 954, 2003.

[88] C Shu, H Ding, and N Zhao. Numerical Comparison of Least Square-Based Finite-Difference (LSFD) and Radial Basis Function-Based Finite-Difference (RBFFD) Methods. *Computers and Mathematics with Applications*, 51(8):1297–1310, 2006.

[89] Steven Skiena. *The Algorithm Design Manual (2. ed.).* Springer, 2008. 2, 3

[90] Ian H. Sloan and Robert S. Womersley. Extremal systems of points and numerical integration on the sphere. *Adv. Comput. Math*, 21:107–125, 2003.

[91] D Stevens, H Power, M Lees, and H Morvan. The use of PDE centres in the local RBF Hermitian method for 3D convective-diffusion problems. *Journal of Computational Physics*, 2009.

[92] David Stevens, Henry Power, Michael Lees, and Herve Morvan. A Meshless Solution Technique for the Solution of 3D Unsaturated Zone Problems, Based on Local Hermitian Interpolation with Radial Basis Functions. *Transp Porous Med*, 79(2):149–169, Sep 2008.

[93] David Stevens, Henry Power, and Herve Morvan. An order-N complexity meshless algorithm for transport-type PDEs, based on local Hermitian interpolation. *Engineering Analysis with Boundary Elements*, 33(4):425 – 441, 2008.

[94] L. Stocco and G. Schrack. Integer dilation and contraction for quadtrees and octrees. In *Communications, Computers, and Signal Processing, 1995. Proceedings., IEEE Pacific Rim Conference on*, pages 426–428, 1995.

[95] L.J. Stocco and G. Schrack. On spatial orders and location codes. *Computers, IEEE Transactions on*, 58(3):424–432, 2009. 16, 17

[96] Andrea Tagliasacchi. kd-tree for matlab. `http://www.mathworks.com/matlabcentral/fileexchange/21512-kd-tree-for-matlab`, Sep 2010. 1, 2, 11

[97] Andrea Tagliasacchi. kd-tree matlab. `https://code.google.com/p/kdtree-matlab`, Jun 2012. 4, 11

[98] A. I. Tolstykh and D. A. Shirobokov. On using radial basis functions in a "finite difference mode" with applications to elasticity problems. In *Computational Mechanics*, volume 33, pages 68 – 79. Springer, December 2003.

[99] A.I. Tolstykh. On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations. In *Proceedings of the 16 IMACS World Congress, Lausanne*, pages 1–6, 2000.

[100] Robert Vertnik and Božidar Šarler. Meshless local radial basis function collocation method for convective-diffusive solid-liquid phase change problems. *International Journal of Numerical Methods for Heat & Fluid Flow*, 16(5):617–640, 2006.

[101] J.S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili. Keeneland: Bringing heterogeneous GPU computing to the computational science community. *IEEE Computing in Science and Engineering*, 13(5):90–95, 2011.

[102] B. Šarler and R. Vertnik. Meshfree Explicit Local Radial Basis Function Collocation Method for Diffusion Problems. *Computers and Mathematics with Applications*, 51(8):1269–1282, 2006.

[103] Richard Vuduc, James W Demmel, and Katherine A Yelick. Oski: A library of automatically tuned sparse matrix kernels. In *Institute of Physics Publishing*, 2005.

[104] J. G. Wang and G. R. Liu. A point interpolation meshless method based on radial basis functions. *Int. J. Numer. Methods Eng.*, 54, 2002.

[105] M Weiler, R Botchen, S Stegmaier, T Ertl, J Huang, Y Jang, DS Ebert, and KP Gaither. Hardware-Assisted Feature Analysis and Visualization of Procedurally Encoded Multifield Volumetric Data. *IEEE Computer Graphics and Applications*, 25(5):72–81, 2005.

[106] Holger Wendland. Fast evaluation of radial basis functions: Methods based on partition of unity. In *Approximation Theory X: Wavelets, Splines, and Applications*, pages 473–483. Vanderbilt University Press, 2002. 1, 2, 5, 7

[107] Holger Wendland. *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005. 1, 2, 5, 7

[108] Geoffrey Womeldorff. Spherical Centroidal Voronoi Tessellations: Point Generation and Density Functions Via Images. Master's thesis, Florida State University, 2008.

[109] R. Womersley. Extremal (maximum determinant) points on the sphere S^2. `http://web.maths.unsw.edu.au/~rsw/Sphere/Extremal/New/index.html`, Oct 2007.

[110] Robert S. Womersley and Ian H Sloan. How good can polynomial interpolation on the sphere be?, 2001.

[111] Grady Wright and Bengt Fornberg. Scattered node mehrstellenverfahren-type formulas generated from radial basis functions. In *The International Conference on Computational Methods*, December 15-17 2004.

[112] Grady B. Wright. *Radial Basis Function Interpolation: Numerical and Analytical Developments*. PhD thesis, University of Colorado, 2003.

[113] Grady B. Wright, Natasha Flyer, and David A. Yuen. A hybrid radial basis function–pseudospectral method for thermal convection in a 3-d spherical shell. *Geochem. Geophys. Geosyst.*, 11(Q07003):18 pp., 2010.

[114] Grady B. Wright and Bengt Fornberg. Scattered node compact finite difference-type formulas generated from radial basis functions. *J. Comput. Phys.*, 212(1):99–123, 2006.

[115] Z M Wu. Hermite-Birkhoff interpolation of scattered data by radial basis functions. *Approx. Theory Appl*, (8):1–10, 1992.

[116] Xuan Yang, Zhixiong Zhang, and Ping Zhou. Local Elastic Registration of Multimodal Medical Image Using Robust Point Matching and Compact Support RBF. In *BMEI '08: Proceedings of the 2008 International Conference on BioMedical Engineering and Informatics*, pages 113–117, Washington, DC, USA, 2008. IEEE Computer Society.

[117] Lexing Ying. A kernel independent fast multipole algorithm for radial basis functions. *Journal of Computational Physics*, 213(2):451 – 457, 2006. 2, 3

[118] Rio Yokota, L.A. Barba, and Matthew G. Knepley. PetRBF — A parallel O(N) algorithm for radial basis function interpolation with Gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25-28):1793–1804, May 2010.

[119] Hao Zhang and Marc G. Genton. Compactly supported radial basis function kernels, 2004.