Empirical Analysis of Space–Filling Curves for Scientific Computing Applications

Daryl DeFord
Department of Mathematics
Washington State University
Pullman, WA 99164
Email: daryl.deford@email.wsu.edu

Ananth Kalyanaraman
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164
Email: ananth@eecs.wsu.edu

***Abstract*—Space-Filling Curves are frequently used in parallel processing applications to order and distribute inputs while preserving proximity. Several different metrics have been proposed for analyzing and comparing the efficiency of different space-filling curves, particularly in database settings. In this paper, we introduce a general new metric, called *Average Communicated Distance,* that models the average pairwise communication cost expected to be incurred by an algorithm that makes use of an arbitrary space–filling curve. For the purpose of empirical evaluation of this metric, we modeled the communications structure of the Fast Multipole Method for n–body problems.**

Using this model, we empirically address a number of interesting questions pertaining to the effectiveness of space-filling curves in reducing communication, under different combinations of network topology and input distribution settings. We consider these problems from the perspective of ordering the input data, as well as using space-filling curves to assign ranks to the processors. Our results for these varied scenarios point towards a list of recommendations based on specific knowledge about the input data. In addition, we present some new empirical results, relating to proximity preservation under the average nearest neighbor stretch metric, that are application independent.

***Index Terms*—Space–Filling Curves; Fast Multipole Method; Proximity Preservation; Scientific Computing; Performance Evaluation; Average Communicated Distance.**

I. Introduction

Many applications of parallel computing rely on distributing codependent portions of a given problem onto multiple processors. Thus, to complete a particular step in an algorithm, data may have to be exchanged between many pairs of processors. This communication behavior often limits the performance of algorithms in practice, as each processor's computations cannot be performed without the data, but generally all of the processors are trying to communicate at the same time over the same network.

A related problem exists in distributed processing and data selection applications, where data indexed across multiple dimensions must be ordered in such a fashion as to optimize ranged searches through the data. Whether this data is distributed among multiple processors or stored in a serial database, being able to access the data in an efficient fashion remains an important concern. This has assumed a higher prominence in the current "Big Data" era, where data movement is acknowledged as a serious deterrent to scalability.

Unfortunately, there are no methods to map $n-$dimensional data into a $1-$dimensional order without separating some points that have some parameters in common. A preferred approach to tackle this challenge invokes the theory of space–filling curves. A Space–Filling Curve (SFC) is a mapping from a multi–dimensional space to a linear ordering that allows for unique indexing of the points in that space. A similar, basic type of mapping occurs when the elements of a matrix are stored linearly in an array, using the familiar row/column–major ordering. In general, the order generated by the more sophisticated SFCs tends to preserve proximity to a higher degree, especially when the SFC is applied to a complex set of multivariate data.

In the context of parallel applications, there are two ways in which SFCs can be used. The first way, which represents the more common use–case, is to deploy SFCs for linear ordering the set of input points (or "particles") from a multi-dimensional space and subsequently, identify chunks from that ordered data that will locally reside on individual processors. The second way in which an SFC can be deployed under a parallel setting is for processor rank assignment — i.e., how to label the p processors on a given network with unique ranks $[1 \ldots p]$. As this rank assignment problem becomes one of linear ordering the set of p processors from the given multi–dimensional network, SFCs can be used here too.

Traditionally, on distributed memory computers, this task of rank assignment is generally performed by the underlying communication library/framework, independent of the application layer. A recent paper of Bhatele et al. evaluates the effects of different node selections for communication intensive parallel applications [1]. In addition, with the emergence of massively parallel on-chip network architectures (e.g., [2], [3]), programmers have a better control over labeling the cores (or tiles of cores). Consequently, in this paper, we study both these types of SFCs, and henceforth refer to them as "*particle–order SFCs*" and "*processor–order SFCs*".

Many analytical results have been constructed and proved for a variety of particle–order SFCs and specific applications e.g. [4]–[10]. These results tend to be asymptotic in nature, and there does *not* appear to have been a significant amount of empirical testing comparing the efficacies of the different SFCs for the presented models. In this paper, we will consider

0190-3918/13 $26.00 © 2013 IEEE
DOI 10.1109/ICPP.2013.26

four discrete SFCs that are commonly studied and used in a wide variety of applications: the row/column–major order, the Z–curve [11], the Gray order [12], and the Hilbert Curve [13] — as candidates for particle– and processor–order SFCs.

Similarly, several different metrics have been used to evaluate the efficiency of SFC use. In problems with multi-dimensional data, the most commonly used metric is the number of "clusters" accessed, which measures the number of times an SFC leaves and reenters a rectilinear region of interest corresponding to a range query [10]. The better the ordering, the smaller the average number of clusters that needs to be accessed for any particular query. Thus, recursively constructed, continuous curves often perform very well under this metric. When applied to parallel processing applications, the clustering metric can provide a way to estimate network communications required for range queries under different SFC settings.

In 2012, Xu and Tirthapura introduced a different metric called the Average Nearest Neighbor Stretch (ANNS) [14]. This metric evaluates the multiplicative change in distance between points that are adjacent in the space, as they are mapped into an SFC's linear ordering. As opposed to the notion of clustering, this metric is more generic and provides asymptotic data on the relative efficiency of the curves themselves, disassociated from any particular application. As such, theoretical results describing average nearest neighbor stretch behavior may be applied to a wide variety of situations, independent of any particular algorithm, hardware, or application.

A. Contributions

In this paper, we propose a new metric — one that is more relevant to parallel computing — called ***Average Communicated Distance (ACD)****, for evaluating the efficacy of using different SFCs in parallel scientific applications.* The metric provides a way to arrive at an estimate for the expected communication delay as imposed by a particular implementation of an algorithm.

Definition *1 (ACD Metric):* Given a particular problem instance, the *Average Comunicated Distance (ACD)* is defined as the average distance for every pairwise communication made over the course of the entire application. The communication distance between any two communicating processors is given by the length of the shortest path (measured in the number of hops) between the two processors along the network intraconnect.

For evaluation purposes, we model an abstraction of the communication structure of the Fast–Multipole Method (FMM) [15], which is one of the most widely used methods in scientific computing for solving the classical n–body problem. This algorithm takes a set of particles in space with associated "force" values (usually gravitational mass, or electromagnetic charge) and computes local pairwise interactions directly, and long range interactions collectively. A more thorough explanation of the FMM algorithm can be found in [16].

The empirical model described in this paper represents a new approach to analyze SFCs — by attempting to model and quantify expected communication under three different parameter dimensions, viz. SFCs, network topologies and input distributions. Our empirical model covers both traditional (particle–ordering) and emerging (processor–ordering) use–cases of SFCs. To the best of our knowledge, this is the first attempt at analyzing both use-cases in tandem.

More specifically, we address the following list of research questions using our empirical model:

Q1) *What is the nearest-neighborhood preservation efficacy achieved by different particle–order SFCs?*

Q2) *What is the effect of different combinations of* $\{$*particle-order, processor-order*$\}$ *SFCs on the Average Communicated Distance metric?*

Q3) *What is the performance of each of the particle-order SFCs under the ACD metric, for a given network topology? Similarly, what is the performance of each of the network topologies under the ACD metric, for a given input distribution?*

Q4) *How does the Average Communicated Distance vary as a function of processor size, input size and input distribution, for each SFC?*

For the last two questions, we use the same SFC curve for both particle– and processor–ordering. For simplicity, we used 2D space in all our experiments.

Our findings suggest both theoretical avenues of inquiry for future research and practical applications of particular SFCs, both for distributing the input data among parallel processors, and for canonical labeling of processors on a particular network topology, with an overall goal of minimizing communication network usage under a non–contention setting. In addition, we empirically corroborate most of the observations made in [14] and also present some surprising results relating to the more well-studied average nearest neighbor stretch metric. Collectively taken, these findings along with the proposed empirical methodology can be expected to serve as a design guide for algorithm developers and parallel programmers in scientific computing. Although the effects of network contentions on our findings cannot be ignored, they are not studied as part of this paper.

Our work in this paper differs from previously published approaches in several ways.

- First, the metric that we have defined (ACD) does not appear to have been considered previously in the literature. Also, as demonstrated in this paper using the FMM application, the metric can be made to more closely model the *expected communication behavior* of any target parallel application — something for which currently available metrics such as ANNS and clustering are not suitable.
- Secondly, this appears to be the first paper that studies the use of SFCs for both ordering and separating data points as well as distributing the sets of points onto the processors using a possibly independent SFC. Also, our empirical results can serve as a design reference to assist researchers and application scientists interested in applying these SFCs to similar types of problems to those

considered in this paper.
- Finally, our generalizations of the nearest neighbor considerations introduced by Xu and Tirthapura provide an intermediate measure of SFC performance between the ANNS and all neighbors stretch.

B. Related Work

There exists a significant body of literature focused on the use of SFCs for database applications under the clustering metric [7], [8], [10], [17]. In these problems, the better the ordering, the smaller the average number of clusters that needs to be accessed for any particular query. Note that these studies are concerned with particle–ordering exclusively. In 1990, Jagadish applied the Hilbert curve to this problem and presented some analytic and empirical results showing that the Hilbert curve outperformed both the Gray order and Z–curves [8]. Later, by restricting attention to two–dimensional spaces, he was able to give a closed–form expression for the clustering number of the Hilbert curve using its recursive properties [7]. The general case of two–dimensional curves was considered by Asano et al. [5]. They considered general combinatorial properties of these SFCs and devised a construction of an SFC with improved worst case performance.

These results are extended in a more recent work of Moon et al. that extends the results on Hilbert clustering numbers to rectilinear surfaces in $n-$dimensions [10]. This paper mentions a number of applications of this particular use of SFCs and has become an influential paper in the field, having been cited several hundred times since its publication.

More recently, significant advances have been made on this problem by Xu and Tirthapura [17]. This important paper provides both a lower bound for the optimal clustering number of any given SFC in $n-$space, and also shows that under certain realistic assumptions, *all* continuous SFCs are optimal with regards to clustering. This is a particularly surprising result as it implies that for these types of problems the Hilbert curve offers no asymptotic advantage over even the simple "snake scan" ordering (the continuous analog of the basic row/column order).

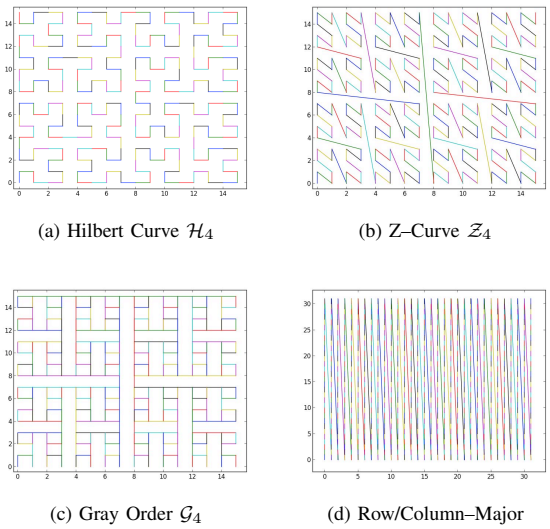
In another recent paper [14], Xu and Tirthapura consider an interesting generalization of this problem defining the average nearest neighbor stretch of a SFC to be the multiplicative increase in distance between points that are adjacent in the $n-$space after they are mapped into a linear ordering. They define other metrics such as the all-pairs stretch and maximum nearest neighbor stretch to provide a more complete picture of the efficiency of any particular mapping. Besides giving a lower bound for the efficiency of any SFC this paper provides another surprising result; it is shown that the Z–curve and the row major ordering are asymptotically equivalent under their metric and that both of these SFCs are within a constant factor of the optimal lower bound that they provide. Both papers of Xu and Tirthapura contain excellent historical surveys of the motivating results for their problems [14], [17].

Organization of the Paper: The rest of the paper takes the following format. In Section II, we define the terms that we will use throughout the paper and give explicit examples of the SFCs, distributions, and topologies used in our analysis. Section III contains an overview of our empirical algorithm and describes our experimental methodology, while Section IV describes our algorithm for calculating the ACD metric for FMM applications. The remaining sections contain the results of our experiments. Specifically, Section V describes our results related to generalization of Xu and Tirthapura's ANNS metric and considers our research question 1. Section VI contains our results on minimizing communication distance in parallel FMM instances and answering research questions 2, 3 and 4. The full generality of the ACD metric is discussed in Section VII. Finally, Section VIII concludes the paper along with possible extensions and experiments for future study.

II. Definitions and Terminology

A. Space–Filling Curves

In this paper, we evaluate the efficiency of four SFCs, the Hilbert curve, the Z–curve, the Gray order, and the simple row–major order. Constructions of these SFCs proceed as follows. Given a $2^k \times 2^k$ universe of points, or spatial resolution, each particle is assigned a unique natural number from $\{1, 2, 3 \ldots, 4^k\}$ by the particular curve's ordering. Here we introduce and describe the basic constructions of the SFCs examined in this paper. All of these curves are well known and have been frequently studied for similar purposes. Complete constructions and definitions can be found in [5].



(a) Hilbert Curve $\mathcal{H}_4$ (b) Z–Curve $\mathcal{Z}_4$

(c) Gray Order $\mathcal{G}_4$ (d) Row/Column–Major

Fig. 1. An example illustration of the Space-Filling Curves considered in our study.

1) Hilbert Curve: The Hilbert curve was originally defined by David Hilbert [13] as a specific example of a wide class of SFCs originally discovered by Giuseppe Peano [18]. It is a recursively constructed SFC where each iteration of the curve contains four copies of the previous iteration, rotated so that the entry and exit points align.

We will consider discrete iterations of the Hilbert curve, where $\mathcal{H}_k$ represents the k^{th} iterations, while the analytic, continuous Hilbert curve is the SFC obtained by taking $\lim_{k\to\infty} \mathcal{H}_k$. For the discrete case, $\mathcal{H}_{k+1}$ is constructed from four copies of $\mathcal{H}_k$ in a 2×2 grid where the individual $\mathcal{H}_k$ are rotated to align their entry and exit points. Due to this construction process, Hilbert curves have many symmetries and combinatorial properties. Figure 1(a) shows $\mathcal{H}_4$.

2) Z–curve and Gray Order: The Z–curve is obtained by taking the binary representations of the coordinates of each point and interleaving the bits together to construct a single integer representation. This ordering can also be constructed recursively in a similar fashion to the Hilbert Curve, but $\mathcal{Z}_{k+1}$ is obtained without rotating the $\mathcal{Z}_k$. The Gray order takes the Z–curve representations of each point and orders them by the Gray code, where each successive binary representation differs in exactly one place, instead of in a linearly increasing fashion. This leads to a recursive construction where the lower two $\mathcal{G}_k$ are not rotated and the upper two $\mathcal{G}_k$ are rotated $180°$. Figure 1(b) and Figure 1(c) show these recursively constructed SFCs respectively.

Note that it is more computationally efficient to compute the order of each point directly with bit operations than to use recursive techniques for all of these curves. However, for theoretical considerations, the combinatorial properties of the recursive constructions are more valuable for asymptotic analysis, especially for the clustering metric.

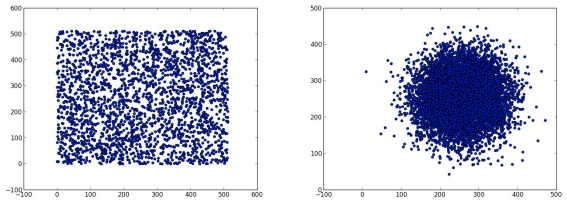
3) Row Major: The row major curve is the simplest of the SFCs that we will consider. To construct a row major ordering, simply assign the points in the first column the values from $\{1, 2, 3, \ldots 2^k\}$ while in general the points in the i^{th} column are numbered from $\{(i-1) \times 2^k + 1, \ldots i \times 2^k\}$.

B. Network Topologies

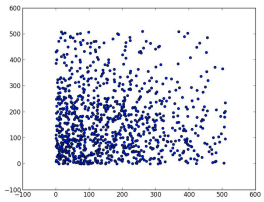
We studied the performance of six different communications network topologies. The simplest networks are bus and ring topologies, where each processor may only communicate with two direct neighbors. The bulk of our experiments focused on mesh/grid and torus topologies which are more common on HPC architectures. We also studied the quadtree topology, where each communication must travel up and down the tree, and the classical hypercube topology.

C. Probability Distributions for Input

In order to model random initial particle placements for our FMM algorithm we used three different types of probability distributions to populate our problems. The first distribution that we used is the uniform distribution, where each point in the spatial resolution has an equal probability of being selected (Figure 2(a)). To model centrally distributed problems we used a bivariate normal distribution with symmetric axes (Figure 2(b)). Finally, in order to model asymmetric or skewed distributions, we selected particles with an exponential distribution, which clusters the selected values in a single quadrant (Figure 2(c)). Figure 3 shows an example particle–ordering achieved for exponentially distributed points.

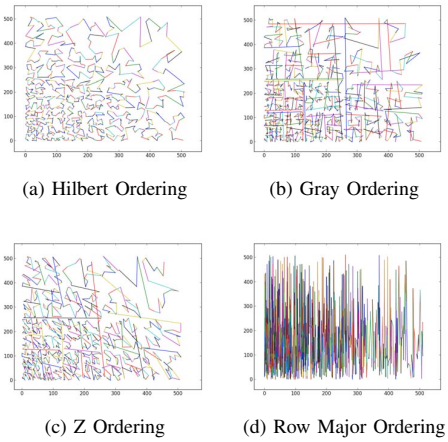


(a) Uniform Distribution (b) Normal Distribution



(c) Exponential Distribution

Fig. 2. A figure showing examples of the two dimensional probability distributions considered in this paper.



(a) Hilbert Ordering (b) Gray Ordering

(c) Z Ordering (d) Row Major Ordering

Fig. 3. As an example of particle–ordering SFCs, this figure shows the linear order of the particles displayed in Figure 2(c) by each of the SFCs respectively.

III. Modeling Communication for the FMM Algorithm

In what follows, we describe our model for the interprocessor communication in the FMM algorithm borrowing the terminology used by Hariharan *et al.* [19] in their work on implementing the FMM algorithm for computational electromagnetics. In this algorithm, the spatial domain is represented as a compressed quadtree for 2D (compressed octree for 3D), where the cells with particles at the finest resolution occupy leaf positions, and coarser cells are represented by internal nodes [20]. For the purpose of analysis, let us assume that a cell at the finest resolution may contain at most one particle.

The communication at every time step of the FMM al-

gorithm is dictated by two types of interactions: *near–field interactions (NFI) and far–field interactions (FFI).*

The near–field interaction list is computed for each particle and requires information from all particles within radius r. For instance, in 2D the number of nearest neighbors which share an edge/corner with a cell is bounded by 8 (corresponding to $r = 1$). These neighboring cells in the quadtree representation corresponds to communicating with at most 8 other leaf nodes. For particular linear orderings of particles and of processors on the network, the number of hops to communicate for each such near-field pairwise interaction can easily be computed.

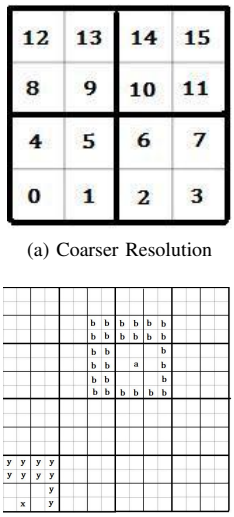
The computation of far-field interactions results in three different types of communications:

a) **Interpolation:** This maps to an upward accumulation on the quadtree, where values from the children of an internal node are used to calculate the value at that node.
b) **Interaction list:** This is a step that applies to all the internal nodes of the quadtree. Here, each cell at coarse resolutions interacts with all of the children of its parent's neighbors that are not adjacent to the cell at that resolution. From a communication perspective, this corresponds to a communication between the processor that holds that internal node in the quadtree with all other processors that hold its parent's children's cells' internal nodes.
c) **Anterpolation:** This maps to a downward accumulation on the quadtree, where the values at a parent node are percolated down to its children.

In order to calculate the Average Communicated Distance (ACD) required for the far–field interaction, we separate the interpolation and anterpolation steps from the interaction list connections. To compute the quadtree for the upwards and downwards accumulation steps, we separate the processors by the quadrants of the particles that they have been assigned. Then, we use the linear ordering of the processors to determine what transmissions are necessary to gather information from each particle to the top of the tree. By convention, we assume that for each level of resolution, the lowest ranked processor in a quadrant will collect the data from the cells at that level. This allows us to compute the shortest communication distances necessary to implement the interpolation and anterpolation interactions.

Calculating the interaction list is more complex. For each cell at each level of resolution, we construct a list containing the children of the cell's parent's neighbors that share no common edges or corners with the original cell, and are at the same level of resolution of the original cell (refer to Figure 4). Each cell in the list may contain multiple particles that have been distributed across multiple processors. We again adopt the convention that the processor that contains the lowest indexed particle in the linear ordering is responsible for communication of that cell. Thus, for each cell we compute the distance between the processor representing that cell and the processor representing each of the cells in its interaction list.

Near–field and far–field interactions lead to significantly different computational scenarios and place distinct burdens



(a) Coarser Resolution

(b) Finer Resolution

Fig. 4. Interaction Lists: Figure showing two partitioned spatial resolutions. In the coarse resolution image (a), the interaction list of node 0 is $\{2, 3, 6, 7, 8 - 16\}$, or every node that it not in its quadrant. However, the interaction list of node 6 is $\{0, 4, 8, 12, 13, 14, 15\}$. At the finer resolution, nodes in the interaction list of x are marked with y and nodes in the interaction list of a are marked with b.

on the processor communication network. Particularly, far–field interactions demand more resources at coarser resolutions since the communicating pairs of processors are separated by a larger distance across the network. On the other hand, the demand in communication for near–field interactions increase significantly with the number of particles n, as well as the radius r, and the relative density of the particles, which is determined by the distribution.

IV. Algorithm for Computing the ACD for FMM Applications

To effectively characterize the communication efficacies of different SFCs on to the FMM model, we study and evaluate the two interaction types — near-field and far-field — separately. The initial operation of our method is the same for either case and can be described as follows:

Given an initial distribution of n particles in a $2^k \times 2^k$ spatial resolution:

1) Order the particles linearly with the specified particle–order SFC;
2) Partition the particles into p consecutive chunks of size $\frac{n}{p}$ each;
3) Order the processors with the specified processor–order SFC (applies only to mesh and torus topologies);
4) Distribute chunk i to processor i, for $1 \leq i \leq n$.

For NFI, we compute the neighborhood of each particle and determine the distance between each communication that occurs. For FFI, we use a log–tree in each quadrant to contact each processor that contains at least one particle in the quadrant.

For the near–field interactions:

5) For each particle x, construct a list of all neighbors y, of x, such that $d(x, y) \leq r$.
6) For each (x, y) pair, determine the communicated distance as the shortest path distance along the network (possibly zero) between the processor that contains x and the processor that contains y. Note that this manner of calculating the distance renders our model contention-unaware.
7) Output the sum of these communication distances for all (x, y) as the ACD value corresponding to all near-field interactions.

For the far–field interactions:

5) For each quadrant containing at least one particle, compute an ordered list of all of the processors that contain at least one particle in that quadrant.
6) Construct a log–tree (quadtree in 2D) connecting the processors in each quadrant.
7) To capture the parent-child communication that happens during interpolation and anterpolation, we compute the shortest path distance along the network between the two corresponding processors.
8) Construct the interaction list for each processor at each level of resolution.
9) For each processor, compute the distance along the network between that processor and each other processor in its interaction list.
10) Output the sum over all the communication distances — Interpolation, Anterpolation, and Interaction List — as the ACD value corresponding to all far-field interactions.

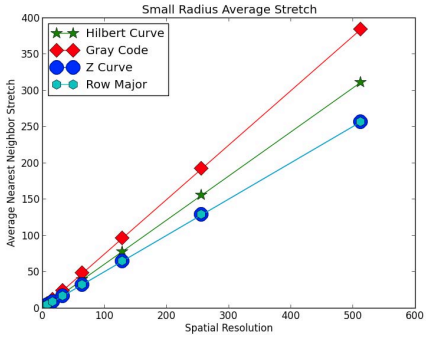
Notice that these two abstractions can be applied in much more general circumstances by noticing that these elements correspond to traditional archetypes of parallel communication. Nearest neighbor queries as modeled in the NFI step are common in parallel computing applications, and the FFI abstraction matches parallel prefix and collective broadcast type communications. In Section VII we describe how ACD can be applied to applications other than FMM.

V. Experimental Results: Nearest Neighbor Proximity Preservation

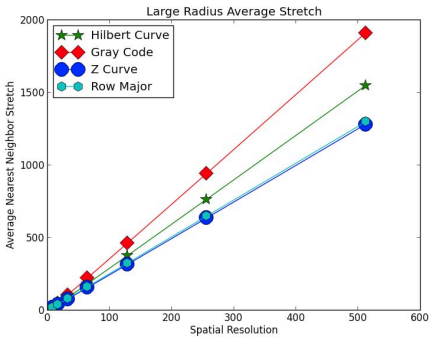
In this section we address our first research question: *What is the nearest–neighborhood preservation efficacy achieved by different particle–order SFCs?* Note that this question is oblivious to the underlying network topology. For this purpose we consider the metric introduced by Xu and Tirthapura called the average nearest neighbor stretch (ANNS) [14]. Given a particular spatial resolution and an SFC, the ANNS can be computed by summing over the distance in the linear ordering between each pair of nearest neighbors (points that are separated by a Manhattan distance of 1 in k–space). Thus, this provides a measure of the efficiency of an SFC that is application independent. The ANNS for SFCs can be easily modeled within our method.

Instead of taking a random subset of the points in a given spatial resolution as the input to our program, we input every point of the resolution. Then, setting the radius equal to 1 and computing the linear distance between each point and its neighbors, the ACD is given by the ANNS for that particular spatial resolution and SFC, in two dimensions. In their paper, Xu and Tirthapura gave analytic results for the Z–curve and the row major curve. We confirmed their analytical results and obtained empirical results for these curves as well as the Hilbert curve and Gray order.

Figure 5(a) shows our results for the four SFCs as the spatial resolution ranges from 4 points to 512×512 points for the standard neighborhood size $r = 1$. Our results clearly demonstrate that in two dimensions, the Z–curve and row major significantly *outperform* the Gray code and the Hilbert curve. This is surprising, since under most previously considered, clustering–type metrics the Hilbert curve is traditionally assumed to give the best results, due to its greater complexity especially in empirical evaluations [7], [8]. As the number of points in the spatial resolution increases, the relative ordering remains the same and the differences between SFC performances increases.



(a) Standard ANNS



(b) Large Radius ANNS

Fig. 5. A comparison of different particle-order SFCs on Average Nearest Neighbor Stretch.

We generalized this metric by expanding the definition of nearest neighbors to larger Manhattan radii. Thus, we calculated the multiplicative increase in distance from k–space to the linear ordering for all of the points within a fixed radius, not just the nearest neighbors. In each of the experiments we

performed, irregardless the radius used, the relative ordering of the curves was the same. Figure 5(b) shows similar results for a larger neighborhood size with $r = 6$.

These results verify the theoretical and asymptotic analyses of Xu and Tirthapura on the Z–curve and row major ordering. Their paper focused primarily on these two SFCs and showed that these curves were asymptotically equivalent in terms of the ANNS metric. Moreover, they proved that both SFCs performance is within a constant factor of the optimal lower bound for all continuous SFCs. Our empirical data suggests that analytical analysis of the Hilbert curve and Gray order is likely to demonstrate that these curves do not perform as well under this metric. From a theoretical perspective, the greater complexity of the Hilbert and Gray curves makes proving asymptotic results much more difficult for these curves. Similarly, our extension of the ANNS metric to larger nearest neighbor radii presents a broader picture of the proximity preservation properties of the SFCs that we have studied.

VI. Experimental Results: Evaluation under the Average Communicated Distance metric

We varied the probability distribution of the particles, the size of the problems, and the topology of the network to gain a more complete understanding of the relative performances of the SFCs for FMM-type applications. We report specifically on the results of three separate experimental designs, tailored to our research questions Q2-Q4 (in Section I). The results presented here are averages over multiple independent trials for each set of parameters.

A. Comparing Different Particle/Processor-Order SFC Combinations

We compared the effect of using different combinations of particle/processor-order SFCs on the ACD metric. All experiments were designed using a fixed input size of 250,000 particles, drawn from each of the three distributions (Uniform, Normal, Exponential) separately, using a spatial resolution of 1024×1024. For the network of processors, we assumed a set of 65,536 processors connected with a torus topology. Each of the particle/processor–order SFCs was chosen to be one of {Hilbert, Z–curve, Gray, Row major}. This resulted in 16 SFC pairing combinations.

Tables I and II show the results for the near–field and far–field interaction models, respectively. For the near–field interactions (Table I), the results are unanimously in favor of the Hilbert ordering for every particle distribution.

Note that although the relative performance of the curves is unchanged as the distribution varies, the recursively defined curves offer much better performance on uniformly distributed points than on the bivariate normal distribution. This is because in a bivariate normal distribution, the particles are clustered towards the center, which is the location of the largest discontinuities in each recursively constructed linear mapping, since the central particles all belong to separate quadrants of the SFC.

TABLE I
A comparison of different particle/processor-order SFC combinations for NFI under various distributions. The lowest ACD value within each row is displayed in **boldface**, while the lowest ACD value within each column is displayed in *italics*.

	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***4.008***	*4.308*	*4.939*	*13.117*
Z–Curve	**5.486**	5.758	6.573	18.127
Gray Code	**5.802**	6.010	6.970	19.220
Row Major	**9.126**	9.763	11.713	70.353

(a) Uniform Distribution

	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***8.561***	*9.297*	*10.123*	*20.340*
Z–Curve	**11.003**	11.551	12.984	26.842
Gray Code	**11.881**	12.595	13.249	28.188
Row Major	**20.143**	22.221	24.053	66.719

(b) Normal Distribution

	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***5.238***	*5.654*	*6.271*	*14.943*
Z–Curve	**6.943**	7.070	8.235	20.851
Gray Code	**7.276**	7.663	8.760	22.269
Row Major	**12.483**	13.017	15.289	61.227

(c) Exponential Distribution

TABLE II
A comparison of different particle/processor-order SFC combinations for FFI under various distributions. The lowest ACD value within each row is displayed in **boldface**, while the lowest ACD value within each column is displayed in *italics*.

	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***19.494***	*20.841*	*22.572*	*31.124*
Z–Curve	**24.217**	24.793	27.787	37.709
Gray Code	**24.622**	25.446	27.997	39.282
Row Major	**44.513**	48.762	50.118	57.880

(a) Uniform Distribution

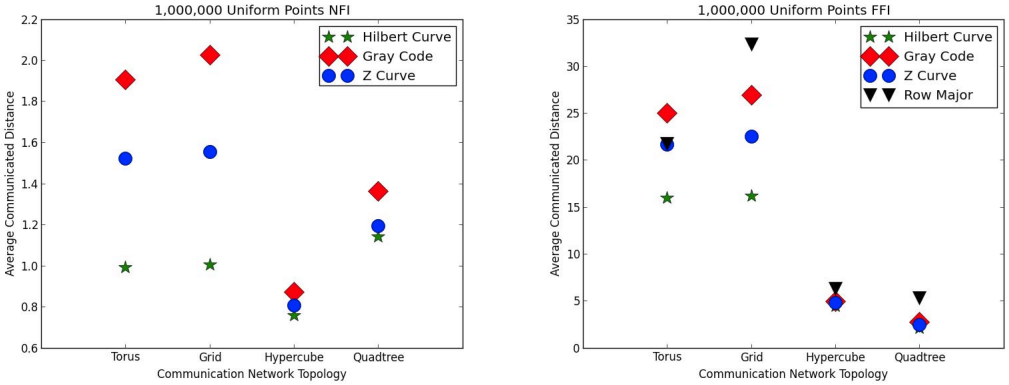
	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***26.336***	*26.824*	31.963	*32.542*
Z–Curve	29.160	**28.036**	34.241	36.663
Gray Code	29.449	**27.981**	*31.909*	37.291
Row Major	**43.639**	44.636	49.133	45.475

(b) Normal Distribution

	Particle Order			
Processor Order ↓	Hilbert Curve	Z–Curve	Gray Code	Row Major
Hilbert Curve	***18.960***	*19.841*	*23.007*	*31.368*
Z–Curve	24.672	**23.316**	26.315	37.576
Gray Code	**23.762**	24.076	27.973	37.863
Row Major	**42.447**	44.067	46.872	50.963

(c) Exponential Distribution

The difference in the ACD values under these two distributions is approximately a factor of 2. This is a significant variance considering the total number of communications that must be performed in a realistic implementation. However, since the relative performance of the curves is unchanged,



(a) Near–Field Interactions

(b) Far–Field Interactions

Fig. 6. The charts show the results of comparing different network topologies for a) the Near–Field; and b) Far–Field interactions, respectively. All experiments were performed using $1,000,000$ uniformly distributed particles on a 4096×4096 spatial resolution. For NNI, a radius of 4 was used. Results from the bus and ring topologies, as well as the row-major entries for the near–field interactions have been omitted because they are significantly larger than the other ACD values. This plot is representative of all the experiments we performed to evaluate the topologies.

there is *no* incentive to shift the ordering of particles between FMM iterations to reflect the dynamically changing particle distribution profile.

For the far–field interactions (Table II), the results are similar, although not identical. In this case, under the non–uniform distributions (i.e., Normal, Exponential), the Z–curve offers slightly lower ACD values than the Hilbert curve, when the processors are ranked either with the Gray or Z–Curves. In all other cases, the Hilbert curve provides superior results. Note that the difference in ACD performance between the distributions is significantly different than in the near–field interaction case. Particularly, particles that are distributed in an exponential distribution give better values than when the particles are distributed uniformly. This is because at the finer levels of interaction, particles in the sparser quadrants have smaller interaction lists, and fewer long–distance transmissions are necessary with the recursive SFCs.

Overall, these results suggest that for implementations of FMM-type algorithms, use of any recursively constructed SFC (i.e., Hilbert, Z, and Gray) can be expected to offer significant reduction in the overall communication. That said, the results in Tables I and II show a clear advantage of using either the Hilbert or Z-curve over the Gray curve under the ACD metric — i.e., if one were to order the efficacies of the different curves by their ACD values, then the following ordering is expected:

$$\{Hilbert \approx Z\} < Gray << Row\text{-}major.$$

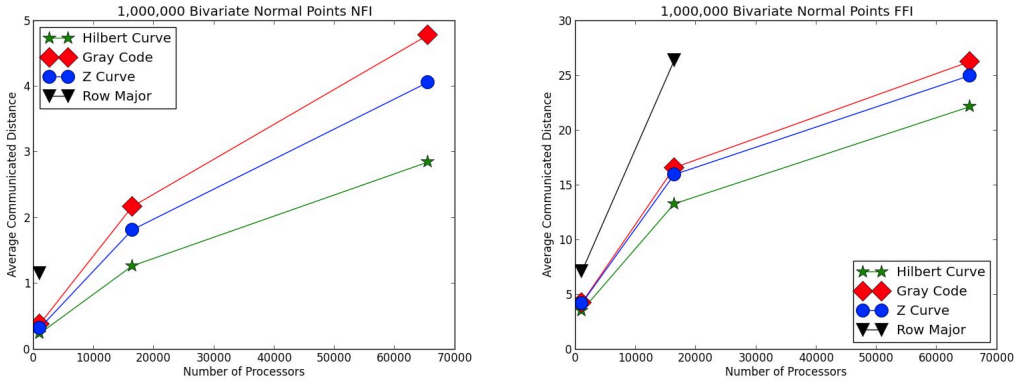
The results also point to the following recommendations: From a processor-ordering standpoint, the Hilbert curve is the clear winner over all other curves for the torus topology, regardless of the particle-ordering used. Although the results are not presented, this observation also holds for the mesh topology. On the other hand, the choice for the particle-ordering SFC is *not* as obvious. If the processors are ranked using the Hilbert/row-major scheme in the underlying mesh or torus topology, using the Hilbert curve to order the particles in likely to be the most communication-effective choice. Otherwise, both Hilbert and Z-curves offer comparably best choices.

B. *Effect of the Network Topology*

Next, we address the following research questions under the ACD metric: *Q3) What is the performance of each of the particle-order SFCs under the ACD metric, for a given network topology? Similarly, what is the performance of each of the network topologies under the ACD metric, for a given input distribution?* In the interest of keeping the number of studies combinatorially less-explosive, we used the same SFC within each experiment — e.g., in an experiment involving the Hilbert curve, both the particle and processor orderings were achieved using Hilbert curve. Consequently, this study generated 24 sub-cases: one for each {topology, SFC} pair. In our experiments, we used fixed sets of inputs and computed the ACD for each topology under each SFC.

Figure 6 shows the results for the Near–Field and Far–Field interactions. As in the other experiments, the results were fairly conclusive. The Hilbert curve offered the best performance across the different topologies, while the topologies themselves show a well-defined order. Unsurprisingly, for the near–field interactions, the hypercube gave the best results. However, this result should be taken with some caution because our experiments do *not* account for potential contention scenarios, which could degrade the performance for the hypercube and quadtree topologies more so than for others. The trends shown in this plot are representative of all the experiments we performed using different input sizes and distributions.

For far–field interactions, the quadtree topology leads to slightly smaller values than even the hypercube. This is to be



(a) NFI

(b) FFI

Fig. 7. These plots show ACD values for a) near–field, and b) far–field interactions, as a function of the number of processors and the SFC used. The input used was fixed at 1,000,000 uniformly distributed particles. Some of the row–major data has been excluded from these plots because for this SFC, the ACD values at larger processor numbers were significantly higher than the other data–points.

expected because the quadtree's layout mirrors the structure of communication incurred during FFI. Again, contention needs to be factored in before corroborating this trend. Also as expected, the performance of the bus and ring topologies was significantly worse than the other topologies. Similarly, the row-major performance was very poor compared to the other SFCs, and this data offers compelling reasons to utilize any other SFC for FMM–type implementations.

Notice that, for the recursively-defined SFCs, the results from the mesh and torus topologies are highly comparable for both interaction models, despite the wrapped around connections in the torus. This observation suggests that the proximity preserving properties of the recursively defined SFCs (viz. Hilbert, Z-curve, Gray) provide an equally effective mapping on to the mesh as on the torus — possibly also suggesting the lesser utility of the wrapped links in such cases. However, this analysis does not apply to the row–major ordering, which, as Figure 6(b) shows, returns markedly lower ACD values on a torus topology than on a mesh.

C. Other Parametric Studies

We also studied the effect of varying the processor size, number of particles, and the input distribution on the ACD metric (Q4). In all these experiments, we fixed the network topology as a torus. For the near–field interactions, we varied the nearest neighbor radius r for all cases, but this parameter change did not affect the ordering of the SFCs. Obviously, larger radii require more processor to processor communications and result in higher ACD values. However, since this affects all curves proportionately, it does not provide any incentive to select separate SFCs for larger radius values.

Figure 7 shows plots for both the near–field and far–field interactions as the number of processors varies. From this data it is easy to see that the Hilbert curve once again offers the best performance, for both the near–field and far–field interactions, while the Gray code and Z-order are approximately equivalent, and the row-major curve is very poor. Although not shown, this behavior also holds for increasing input sizes. For larger problem sizes, the gains in efficiency by selecting a better SFC increase significantly, especially if the original curve is the row major, whose ACD values are significantly greater than any of the other SFCs we tested. Our results suggest that this holds both as the number of particles is increased for a fixed number of processors and as the number of processors is increased for a fixed number of particles.

As for input distribution, the ACD achieved during NFI was observed to be the best for the uniform distribution, followed by exponential and normal distributions (in that order). On the other hand, for FFI, the effects of the input distributions were generally indistinguishable. These observations can also be inferred from Tables I and II.

VII. Generality of the ACD Metric

Although we have modeled the FMM algorithm in order to demonstrate the efficacy of the ACD metric, any communication bound parallel application can be evaluated with this metric. By abstracting different primitives of communications models, the ACD for most common types of parallel communication such as all-to-all and broadcast can be computed in advance for particular applications to allow algorithm designers to select the appropriate SFCs for data separation and processor ranking.

The algorithms presented in Section IV for computing the ACD for NFI and FFI interactions imposed by FMM provide insight into this process. For example, the calculation at each level of resolution for the FFI is equivalent to a log–tree broadcast communication, which is frequently used in parallel implementations. Working at this level of abstraction, the ACD can be calculated for any application with consistent, significant communication demands.

Given the appropriate input parameters, like the network topology, calculating the expected communication costs for

these primitives modified for the particular application can be done for each SFC under consideration. The curve that gives rise to the lowest ACD value can then be selected to minimize the losses due to communication in the full–scale computation. For a particular instance, all that is required is an abstraction of the communication demands and hierarchy over the topology. Then, the ACD value can be calculated for each type of communication, point–to–point, all–to–all, etc., and these can be combined to predict the performance of the implementation.

VIII. Conclusions

In this paper, we presented a new metric called Average Communicated Distance for evaluating the efficacies of different SFCs for parallel scientific computing applications. Using this metric, we modeled the communication characteristics of the classical FMM algorithm in its different stages. Consequently, we performed an extensive empirical evaluation of several standard SFCs under this model. This empirical methodology represents a new approach to analyze SFCs for parallel applications — by attempting to model and quantify expected communication under three different parameter dimensions, viz. SFCs, network topologies, and input distributions. Notice that although this paper focuses on the FMM model, the ACD metric may be used to evaluate any parallel algorithm or application. Our results empirically validate previously published results in theory. In addition, based on our results, we provided a list of recommendations that could serve as benchmarks for effective use of SFCs in FMM-type applications. Our findings suggest both theoretical avenues of inquiry for future research and practical applications of particular SFCs, both for distributing the input data among parallel processors, and for canonical labeling of processors on a particular network topology, with an overall goal of minimizing communication network usage.

Future research directions include:

i) Study the impact of data volume and network contention on communication efficiency, and into the modeling of the ACD metric;
ii) Validation of the communication trends projected by the ACD metric using real world application and using 3D; and
iii) Theoretical investigation to study a direct mapping function from multi–dimensional space to 2D/3D intraconnect network.

Acknowledgments

The authors would like to thank Professor Nairanjana Dasgupta and Professor Srikanta Tirthapura for their useful discussions. This research was partially supported by NSF grant IIS 0916463 and DOE award DE-SC-0006516.

References

[1] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci, M. Schulz, and C. H. Still, "Mapping applications with collectives over sub-communicators on torus networks," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 97:1–97:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389128

[2] T. G. Mattson, R. Van der Wijngaart, and M. Frumkin, "Programming the intel 80-core network-on-a-chip terascale processor," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 38:1–38:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413409

[3] T. Corporation, "Tilera," Sep. 2012. [Online]. Available: http://www.tilera.com

[4] S. Aluru and F. E. Sevilgen, "Parallel domain decomposition and load balancing using space-filling curves," in *Proceedings of the Fourth International Conference on High-Performance Computing*, ser. HIPC '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 230–. [Online]. Available: http://dl.acm.org/citation.cfm?id=523991.938911

[5] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures," *Theoretical Computer Science*, vol. 181, no. 1, pp. 3 – 15, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397596002599

[6] B. Hariharan and S. Aluru, "Efficient parallel algorithms and software for compressed octrees with applications to hierarchical methods," in *High Performance Computing - HiPC 2001 8th International Conference*. Proceedings, 2005, pp. 17–20.

[7] H. V. Jagadish, "Analysis of the hilbert curve for representing two-dimensional space," *Information Processing Letters*, vol. 62, pp. 17–22, 1997.

[8] ——, "Linear clustering of objects with multiple attributes," *SIGMOD Rec.*, vol. 19, no. 2, pp. 332–342, May 1990. [Online]. Available: http://doi.acm.org/10.1145/93605.98742

[9] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, "A massively parallel adaptive fast multipole method on heterogeneous architectures," *Commun. ACM*, vol. 55, no. 5, pp. 101–109, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2160718.2160740

[10] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, 2001.

[11] G. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966. [Online]. Available: http://books.google.com/books?id=9FFdHAAACAAJ

[12] F. Gray, "Pulse code communication," 1953.

[13] D. Hilbert, "Ueber die stetige abbildung einer line auf ein flchenstck," *Mathematische Annalen*, vol. 38, no. 3, pp. 459–460, 1891.

[14] P. Xu and S. Tirthapura, "A lower bound on proximity preservation by space filling curves," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, ser. IPDPS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1295–1305. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2012.118

[15] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, Dec. 1987. [Online]. Available: http://dx.doi.org/10.1016/0021-9991(87)90140-9

[16] R. Beatson and L. Greengard, "A short course on fast multipole methods," in *Wavelets, Multilevel Methods and Elliptic PDEs*. Oxford University Press, 1997, pp. 1–37.

[17] P. Xu and S. Tirthapura, "On the optimality of clustering properties of space filling curves," in *Proceedings of the 31st symposium on Principles of Database Systems*, ser. PODS '12. New York, NY, USA: ACM, 2012, pp. 215–224. [Online]. Available: http://doi.acm.org/10.1145/2213556.2213587

[18] G. Peano, "Sur une courbe, qui remplit toute une aire plane," *Mathematische Annalen*, vol. 36, no. 1, pp. 157–160, 1890.

[19] B. Hariharan, S. Aluru, and B. Shanker, "A scalable parallel fast multipole method for analysis of scattering from perfect electrically conducting surfaces," in *In Proceedings of Supercomputing, The SCxy Conference series*. ACM/IEEE, 2002.

[20] H. Sundar, R. S. Sampath, and G. Biros, "Bottom-up construction and 2:1 balance refinement of linear octrees in parallel," *SIAM J. Sci. Comput.*, vol. 30, no. 5, pp. 2675–2708, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1137/070681727