2007

Parallel methods for large-scale applications in computational electromagnetics and materials science

Sudip K. Seal
Iowa State University

Parallel methods for large-scale applications in computational electromagnetics and materials science

by

Sudip K. Seal

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Major: Computer Engineering

Program of Study Committee:
Srinivas Aluru, Major Professor
Shanker Balasubramaniam
Suraj Kothari
Krishna Rajan
Jiming Song

Iowa State University

Ames, Iowa

2007

DEDICATION

To my family and well-wishers

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

ACKNOWLEDGEMENTS x

ABSTRACT xi

CHAPTER 1. INTRODUCTION 1

1.1 Contributions of This Research 3

1.2 Organization of This Dissertation 4

CHAPTER 2. SPACE FILLING CURVES : A PARALLEL METHOD FOR DOMAIN DECOMPOSITION 6

2.1 Background and Literature Review 7

2.2 Analysis of Z-Space Filling Curves 11

2.3 Geometric Properties of Common Space Filling Curves 20

2.4 Modeling the Problem of Parallel Domain Decomposition 23

2.5 Nearest Neighbor Distance 24

2.6 Parallel Nearest Neighbor Queries 27

2.7 Extending to κ Nearest Neighbors 32

2.8 Parallel Spherical Region Queries 36

2.9 Discussion 38

CHAPTER 3. REVISITING A DISTRIBUTION INDEPENDENT ALGORITHM FOR FAST MULTIPOLE METHOD 40

3.1 Background and Literature Review 40

3.2 Space Filling Curves and Compressed Octrees 46
3.3 Performance Results of a Parallel ACE-based FMM 49
3.4 A Modified Interaction List Building Algorithm 54
3.5 Discussion 56

CHAPTER 4. CLUSTERING ANALYSIS OF ATOM PROBE TOMOGRAPHY DATA 58

4.1 Background and Literature Review 59
4.2 Autocorrelation 60
4.3 Parallel Algorithm 63
4.4 Results 68
4.4.1 Run-time Results 68
4.4.2 Materials Science Results 70
4.5 Discussion 74

CHAPTER 5. CONCLUSIONS AND FUTURE WORK 76

APPENDIX

Nearest Neighbor Distance in $\mathcal{D}$-dimensions 78

BIBLIOGRAPHY 80

LIST OF TABLES

Table 3.1 Runtimes (in seconds) on a cluster platform for various stages of the FMM algorithm using 4 million points. 49

Table 3.2 Runtimes (in seconds) on 32 processors of a cluster platform for various stages of the FMM algorithm with varying problem sizes. 50

Table 3.3 Runtimes (in seconds) on the BlueGene/L for various stages of the FMM algorithm using 4 million points. 52

Table 3.4 Runtimes (in seconds) on 128 processors of a BlueGene/L for various stages of the FMM algorithm with varying problem sizes. 53

Table 3.5 Convergence of numerical results. 53

Table 3.6 Time (in seconds) to build interaction lists using the original and modified algorithm for 4 million points with varying number of processors. 56

LIST OF FIGURES

Figure 2.1 Examples of SFCs for $k = 1, 2$ and 3. (a) Z-curve. (b) Hilbert curve. (c) Gray code curve. 7

Figure 2.2 (a) A 8×8 decomposition of 2D space containing 12 points. The points are labeled in the order they are visited using a Z-SFC. (b) Partitioning the SFC mapped linearized points across 4 processors. 8

Figure 2.3 Example of a range query in a domain that is mapped using (a) Z-SFC and (b) Hilbert curve. 10

Figure 2.4 Interleaving the bit representations of the coordinates of the cells yields their order according to the Z-SFC. 12

Figure 2.5 Contribution to $S_{r,t}(k+1)$ from $S_{r,t}(k)$ and $\Delta_{r,t}(k+1)$. 14

Figure 2.6 Partitioning a $2^4 \times 2^4$ decomposition across 16 processors using a Z-space filling curve. 16

Figure 2.7 Data partitioning due to the conventional algorithm on a 16 processor hypercube. D_i denotes the sub-domain assigned to processor P_i. 18

Figure 2.8 Data partitioning due to the proposed algorithm on a 16 processor hypercube. The same domain labeling is used as in Fig. 2.7. 19

Figure 2.9 (a) Example of standard squares in 2D: $a_1a_2a_3a_4$, $b_1b_2b_3b_4$ and $c_1c_2c_3c_4$ are standard squares obtained at recursive levels 3,2 and 2, respectively. $a'_1a'_2a'_3a'_4$ and $b'_1b'_2b'_3b'_4$ are examples of non-standard squares. (b) The Z-SFC enters and exits a standard square only once unlike a non-standard square. 21

Figure 2.10 A 2D example illustrating Definition 2 and Definition 4 of $C(u, d)$ and $A(u, d)$. 22

Figure 2.11 The bounding cube is S_l. The dashed inner region is composed from fusing three cuboids, each of dimensions $2d \times (2^{k-l} - 2d) \times (2^{k-l} - 2d)$. For a cell u in this region, $A(u, d)$ is S_l. 23

Figure 2.12 The filled cells are shaded. (a) SFC-array distance between filled cells u and v is 1. (b) SFC-array distance between filled cells u and v is 2 since cell w is encountered by the SFC between u and v. 29

Figure 2.13 (a) Cells that are within δ distance from the left and right edges of any partition do not have their δ-SFC-neighborhood present locally. (b) Bounding $\Pr\{T_u|\overline{W}_u\}$. 29

Figure 2.14 Definition of $X_{u\kappa}$. 33

Figure 3.1 A quadtree built on a set of 10 points in two dimensions. 42

Figure 3.2 Illustration of partial local expansion calculation. Cells inside R' but outside R are required for C's partial local expansion. 44

Figure 3.3 A compressed quadtree corresponding to the quadtree of Fig. 3.1. . . . 46

Figure 3.4 Bit interleaving scheme for a hierarchy of cells. 48

Figure 3.5 Performace on a Linux cluster: (a) Runtime *vs.* number of processors. (b) Runtime *vs.* problem size. (c) Speedup *vs.* number of processors. (d) Efficiency *vs.* number of processors. 51

Figure 3.6 Performance on BlueGene/L: (a) Runtime *vs.* number of processors. (b) Problem size *vs.* number of processors. 53

Figure 3.7 A compressed quadtree for an exponential distribution. 54

Figure 4.1 (a) A square domain containing two types of atoms. (b) Computational domain decomposed into an array of cells. The cells are numbered in the same way as matrices; i.e., the cell at top left corner is labeled (0,0), rows are numbered consecutively from top to bottom, and columns are numbered consecutively from left to right. (c) The set in each cell denotes the number of atoms of each type that is contained within it. . 61

Figure 4.2 (a) Autocorrelation of the matrix in Fig. 4.1(c) for $s_x = 3$ and $s_y = 1$. Note that one of the atom types is suppressed and only the cell population of atoms represented by the unfilled circles is shown. (b) Clusters present in Fig. 4.1 are revealed through the autocorrelation function computation. 62

Figure 4.3 (a) Runtimes with varying problem sizes. (b) Runtimes with varying number of processors. 68

Figure 4.4 (a) Runtimes with varying number of processors. (b) Runtimes with varying number of shifts. 69

Figure 4.5 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 0$ sec. The y axis and the s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$. . . . 71

Figure 4.6 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 210$ sec. The y and the s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$. 72

Figure 4.7 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 3600$ sec. The y and s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$. 73

ACKNOWLEDGEMENTS

I would like to thank my adviser, collaborators and members of my committee for their enthusiastic support. Past and present members of our research group will be severely missed. They made this stint infinitely more exciting and happier for me. My special thanks to those very special people who have been with me through thick and thin during this eye-opening experience. Finally, words alone can never fully describe my gratitude for the many hardships endured by my wife as well as the bottomless patience demonstrated by my family during this time. Their unyielding support can never be thanked in mere words, but still, “Thank you”.

ABSTRACT

In most parallel algorithms for scientific applications, domain decomposition constitutes an important first step that greatly influences subsequent inter-processor communication patterns. This in turn affects the overall performance of the application itself. Space filling curves (SFCs) are a popular tool to partition multidimensional data across processors because of their ability to preserve data locality while incurring little computational overhead. Despite their wide usage, the efficiency of SFCs for parallel data partitioning has always been empirically justified due to the challenge that a rigorous analysis poses. The first part of this thesis presents such a formal analysis of SFCs. The second part highlights their use in the construction of compressed octrees - a hierarchical tree data structure that forms the basis of an efficient parallel algorithm for the fast multipole algorithm (FMM). FMM offers the only linear time numerical solution of the n-body problem encountered in various areas of scientific research. The motivation for our renewed interest in the FMM is due to a recently invented numerical formulation based on accelerated Cartesian expansions (ACE), that generalizes this method to include a far larger class of potentials than originally designed for. We present runtime and scaling results of a parallel ACE-based FMM implementation on two different parallel architectures. Finally, an efficient parallel algorithm to analyze time series three dimensional atom probe (3DAP) microscopy data is presented. Analysis of such data is used to characterize properties of material samples. Due to recent advances in microscopy, typical sizes of 3DAP data sets are as large as $\sim 10^{8-9}$ atoms. Faced with such enormous volumes of data, the challenge has shifted to the design of algorithms that can analyze such large data sets and yield clustering information of the various atomic species present therein. A novel algorithm for this purpose will be described and its performance results on the BlueGene/L discussed.

CHAPTER 1. INTRODUCTION

Rapid advances have been made in the field of parallel computing over the last three decades. As a result, a wide variety of application areas such as electromagnetics, fluid dynamics, molecular dynamics, high energy physics and astrophysics have fallen within the purview of parallel computing research. In addition, new and more specialized fields, such as computational biology and combinatorial materials science, which have the potential of pushing the demand for high performance computing even further are continually emerging. Such trends underscore the continuing need for efficient and scalable parallel methods for deployment at the frontiers of cutting-edge science and engineering.

Availability of more powerful and cost-effective hardware enable application scientists to develop increasingly sophisticated computational models that simulate real life scientific phenomena with even greater numerical accuracy. This, in turn, not only drives the demand for more efficient data structures and better algorithms that can support the growing needs of the application scientists but also the necessity to gain better understanding of already existing methodologies with a view to accurately assessing the merit of their continued use in larger simulations. Motivated by this, the work presented in this thesis consists of a component that analyzes the efficiency of a popular parallel method (space filling curves) for their use in larger parallel systems. In the second component, a related tree data structure is used to implement a distribution independent algorithm for the fast multipole method that has been recently reformulated to include a larger class of potentials than the original formulation. The final component includes the design and implementation of a novel algorithm for the efficient analysis of atom probe microscopy data in materials science. A brief introduction to each problem discussed in this thesis follows.

Many scientific computing applications are based on an underlying spatial domain which can either be a predefined interval of space-time coordinates or an interval in some other abstract space. For example, in most electromagnetics or fluid-dynamics applications, the problem domain is an interval of space and time over which the governing equation, usually a partial differential or an integral equation, is defined. On the other hand, in problems such as the n-body problem or molecular dynamics (MD), the domain is composed of a set of particles represented as points in three dimensional space. The very first task of any parallel algorithm designed for such applications is to partition the problem domain across multiple processors. This initial data partitioning step, called *parallel domain decomposition*, determines subsequent communication patterns and load balance of the resulting parallel algorithm and as such affects its overall runtime. One of the most popular methods used for parallel domain decomposition is called a *space filling curve (SFC)*. The first part of this thesis will present a formal analysis of SFCs and study their efficiency as parallel methods for domain decomposition.

The *fast multipole method (FMM)* continues to remain an extremely powerful algorithm of paramount interest to a wide variety of applications such as VLSI design, molecular dynamics and astrophysical simulations to mention only a few. *Compressed octree* is a hierarchical tree data structure that has been shown to yield a very efficient parallel algorithm for the FMM. As will be shown later, SFCs lie at the heart of compressed octrees. Our renewed interest in the FMM stems from a recently announced accomplishment that uses a fundamentally new numerical formulation of the FMM that is based on what are called *accelerated Cartesian coordinates (ACE)*. An ACE-based FMM has been shown to include a far larger class of potentials than the original formulation. In the second part of the thesis, a parallel implementation of an ACE-based FMM using compressed octrees is studied for its scaling behavior on both a commodity cluster and the state-of-the-art BlueGene/L platform.

The final component of this thesis presents a novel algorithm for clustering analysis of atom probe data. An atom probe is a microscope that is used in materials science to produce near atomic scale images of material samples. This allows further study of such materials for their structural properties. *Atom probe tomography (APT)* is the process of rendering 3D reconstructions of the atom probe data. The technology that drives APT has rapidly advanced in the past five years. At present, the state-of-the-art Local Electrode Atom Probe (LEAP) yields APT data that consists of $\sim 10^8$ atoms, their coordinates and type. It is expected that in the near future, the yield will be $\sim 10^9$ atoms. Faced with such large data sets, the challenge has shifted to the design of algorithms that can analyze such enormous amounts of atomic data and yield clustering information of the various atomic species present in the sample.

1.1 Contributions of This Research

The main contributions of the work presented in this thesis are the following:

Space filling curves: Despite the popularity of SFCs as tools for parallel domain decomposition, their effectiveness in preserving data locality has never been studied formally. The justification for their use is almost always presented empirically through good scaling and runtime results. This thesis includes the first formal analysis of SFCs for parallel domain decomposition. The Z-SFC is formally analyzed when the spatial data is dense. A closed form expression for a distance metric that captures the notion of locality in terms of spatial queries commonly encountered in scientific computing is derived and its ramification on conventional data partitioning using SFC is studied. This result is then generalized to include various other SFCs and the dense case analysis is relaxed to reflect more realistic data distributions. The common geometric properties of the most popular SFCs are exploited to present bounds on communication overheads incurred in commonly occurring spatial queries on data that is partitioned across multiple processors using SFCs.

Fast multipole method: The fast multipole method has been traditionally applied to problems that involve computing pair-wise interactions between particles that interact through a R^{-1} potential where R denotes the inter-particle distance. The renewed interest in FMM is motivated by recent advances that extend the FMM to include all potentials of the form R^{ν} where $\nu \in \mathbb{R}$. In light of this, a compressed octree based implementation of the fast multipole method is studied with an emphasis on its performance on larger parallel systems. A modification to the original compressed octree based algorithm has also been suggested and its advantage in terms of runtime is compared with the original implementation.

Cluster analysis algorithm for APT data: Existing techniques for clustering analysis of 3D atom probe (3DAP) data are serial and require $O(n^2)$ work for n atoms. As a result, for data size with $n \sim 10^{8-9}$ produced by current generation of atom probe microscopes, such techniques take prohibitively long runtime. In this thesis, an $O(n)$ work autocorrelation based technique will be presented. This technique not only reveals clustering of constituent atoms and spatial associations between them but also lends itself to an efficient parallelization with $O(n/P)$ runtime and $O(n/P)$ storage on P processors. Runtime and scaling results on a 1,024 node Blue Gene/L are presented. This is the first parallel algorithm for analyzing 3DAP data.

1.2 Organization of This Dissertation

Each chapter in this dissertation begins with a background of the problem presented in the chapter along with related literature review. The chapters are organized as follows:

Chapter 2 introduces the concepts of parallel domain decomposition and SFCs. The use of SFCs for parallel domain decomposition is discussed with a particular emphasis on their ability to preserve spatial locality. The connection between proximity preserving properties of SFCs and the communication overhead incurred during parallel neighborhood and region queries are investigated. When possible closed form formulas are derived for proximity related metrics. A rigorous analysis that establishes bounds on the communication overhead incurred due to spatial queries commonly encountered in parallel scientific applications is presented.

Chapter 3 revisits the well-known fast multipole method. It briefly describes the methodology underlying a FMM application including a short primer on the compressed octree data structure that forms the basis of an efficient parallel algorithm for the FMM. The inter-relationships between SFCs and compressed octrees are highlighted. Runtime and scaling results of an implementation of the novel ACE-based formulation of the FMM is studied on two different parallel platforms. In doing so, performance bottlenecks are identified and their source isolated. A modification to the underlying compressed octree based algorithm is implemented with a view to improving the performance on large parallel machines.

Chapter 4 begins by describing the computational challenge that besets materials scientists when analyzing very large atomic data sets acquired through advanced atom probe microscopy. A novel algorithm that detects nanometer sized clustering features of the constituent atoms hidden in enormously large data sets is described in detail. A runtime analysis of the underlying algorithm is presented along with supporting scaling results on a cluster platform as well as on a 1,024 processor BlueGene/L. The parallel software is then used to analyze three large sets of data for the aluminum alloy Al-1.9Zn-1.7Mg, each corresponding to a different aging condition. The evolution of nano-clusters of the constituent atoms are tracked and the results presented.

Chapter 5 summarizes the research presented in this thesis, the main results as well as their limitations and implications. Possible directions for future work are also discussed.

CHAPTER 2. SPACE FILLING CURVES : A PARALLEL METHOD FOR DOMAIN DECOMPOSITION

The primary purpose of any scientific application is most often to solve an equation that describes a system or process that is being modeled or simulated. In this context, the term *domain decomposition* is often used to refer to the process of partitioning the underlying domain of the governing equation(s) such that it provides a more accurate result and/or take fewer numerical steps to achieve a predetermined degree of accuracy of the result. In parallel scientific computing, the same term is used to refer to the process of partitioning the underlying domain of the problem across processors in a manner that attempts to balance the work performed by each processor while minimizing the number of communications between them.

When designing parallel algorithms, parallel domain decomposition is almost always the first step. Computations within any sub-region often require information from other sub-regions of the original domain. For example, in the finite element method, computations at a particular element need information available in the elements surrounding it. Since the original domain is distributed across different processors, information from neighboring sub-domains may not necessarily be available locally in every processor. In such cases, processors need to initiate communications with other processors. As communication is significantly slower than computation, efficient parallel algorithms attempt to minimize inter-processor communications and, in fact, try to overlap computation and communication for even better performance. Another important design goal is to achieve load balance. The load on a processor refers to the amount of computation that it is responsible for. Simultaneously achieving minimum inter-processor communication and maximum load balance is often the most challenging part of designing efficient parallel algorithms.

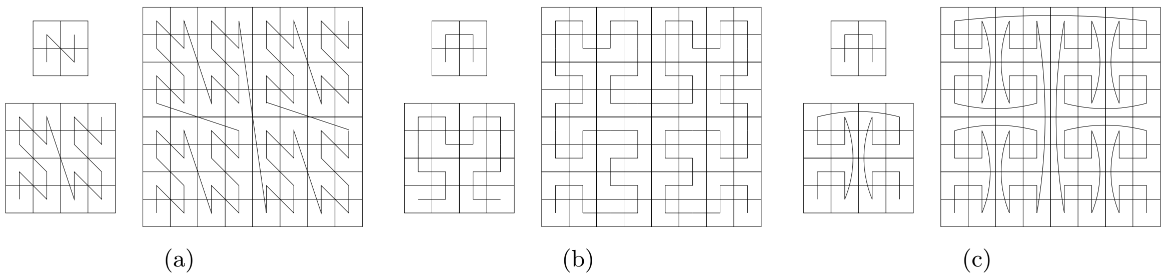


Figure 2.1 Examples of SFCs for $k = 1, 2$ and 3. (a) Z-curve. (b) Hilbert curve. (c) Gray code curve.

Over the years, one of the tools for this purpose has been the use of space filling curves (SFCs). Despite their wide usage, the efficiency of SFCs as a tool for parallel domain decomposition has always been empirically justified due to the challenge that a rigorous analysis poses. This chapter will present what is believed to be a first such formal analysis of SFCs.

2.1 Background and Literature Review

Peano introduced space filling curves (SFCs) as curves that pass through every point of a closed unit square to demonstrate that the number of points in a unit interval has the same cardinality as the number of points in a unit square. Since then SFCs have been generalized to higher dimensions. A SFC is defined as a mapping $\mathbb{R}^d \rightarrow \mathbb{R}$. Though in their original form SFCs are continuous, in this chapter we are primarily interested in discrete SFCs. The reader is referred to [Sagan (1994)] for an in depth exposition of continuous SFCs. For our purpose, we will adopt the following definition of a SFC: Consider a d dimensional hypercube. Bisecting this hypercube k times recursively along each dimension results in a d dimensional array of $2^k \times 2^k \times \cdots \times 2^k = 2^{dk}$ non-overlapping hypercells of equal size. These hypercells belongs to $\mathbb{R}^d$ since each is defined by d coordinates. A SFC is a mapping of this set of 2^{dk} cells to a one dimensional linear ordering. The process of mapping discrete multidimensional data into a one dimensional ordering is often referred to as *linearization.* There are many different mappings that yield a linearization of multidimensional data. Three common examples of linearization

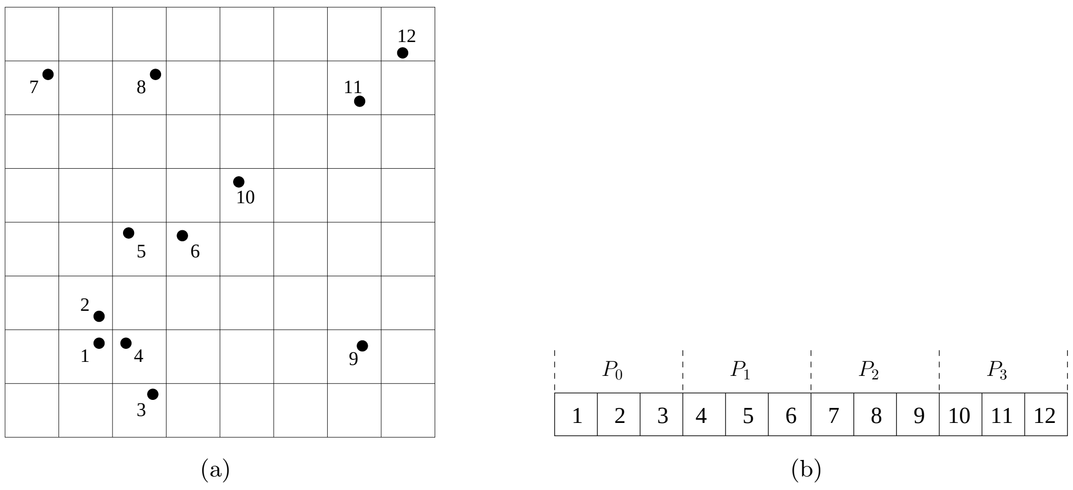


Figure 2.2 (a) A 8×8 decomposition of 2D space containing 12 points. The points are labeled in the order they are visited using a Z-SFC. (b) Partitioning the SFC mapped linearized points across 4 processors.

of two dimensional data are shown in Fig. 2.1 for $k = 1$, 2 and 3. As can be easily observed from the figure, each curve is recursively drawn in that a $2^{k+1} \times 2^{k+1}$ SFC contains four $2^k \times 2^k$ SFCs. For a Hilbert curve [Hilbert (1891)], appropriate rotations need to be performed as well. The above curves can be generalized to higher dimensions.

In general, a SFC based decomposition of a domain that contains n points is carried out by first enclosing the points in a hypercube of side length, say L. Consider the decomposition of this hypercube into 2^{kd} hypercells, each of side length $L/2^k$, such that each cell is occupied by at most one point. The *resolution* of the resulting decomposition is defined by k. A SFC is then used to map the non-empty cells into a one dimensional ordering. An example with $k = 3$ in two dimensions is shown in Fig. 2.2(a). In this figure, the cells that are numbered are considered to be occupied by one point. The rest of the cells are empty. A Z-SFC is used to visit each of the cells. The order in which the occupied cells are visited along the Z-SFC is indicated by the numbers in the cells. The resulting one dimensional ordering is divided into p equal partitions which are assigned to processors. In practice, the value of k is usually fixed and multiple points that reside in a cell are listed in an arbitrary order in the SFC ordering.

Note that the underlying computational domain of a scientific problem need not necessarily be a discrete set of points as shown in the example above. For example, when solving a 3D partial differential equation using finite element method (FEM), the domain is a continuous sub-interval in $\mathbb{R}^3$ that is discretized very often into triangular elements (though the shape of each element may vary with different application). For the analysis that is to follow soon, such a set of discretized space-time sub-intervals will be abstracted by a discrete set of points, each representing a unique space-time sub-interval. For example, a triangular FEM discretization can be abstracted by a discrete point set consisting of the centroids of each triangular element.

Communications in parallel scientific computing are often generated when spatial queries needed for local computations are not satisfied since part of the query region is available in one or more remote processor(s). Common spatial queries that are encountered in many scientific applications are nearest neighbor queries and spherical region queries. *Nearest neighbor queries* are spatial queries in which each point needs information from its nearest neighbor(s). Such queries arise in almost all finite element and finite difference methods [Bank and Jimack (2001); Zhuang and Sun (2005)]. In a *spherical region query*, each point requires information from all other points that lie within a specified radius, say r, from it. Spherical region queries are common to molecular dynamics simulations [Eisenhauer and Schwan (1996); Hayashi and Horiguchi (2000); Nyland et al. (1997); Plimpton (1995)].

In any mapping from a higher dimension to a lower dimension, proximity information will be inherently lost. For example, in Fig. 2.2(a), both points 2 and 4 are nearest neighbors of point 1 in the two dimensional space. But, along the linear array [see Fig. 2.2(b)], point 2 continues to remain a nearest neighbor of point 1 while point 4 does not. As such, if the 12 points in the example are partitioned across four processors as shown in Fig. 2.2(b), then both points 1 and 2 will be local to a processor whereas point 4 will be in a different one. If a nearest neighbor query is generated by point 1, then it can only be satisfied through a communication with another processor to retrieve point 4. It is easy to see that a spherical region query generated by point 1 with a radius greater than or equal to the side-length of the smallest sub-square will also result in a similar communication. Typical database applications

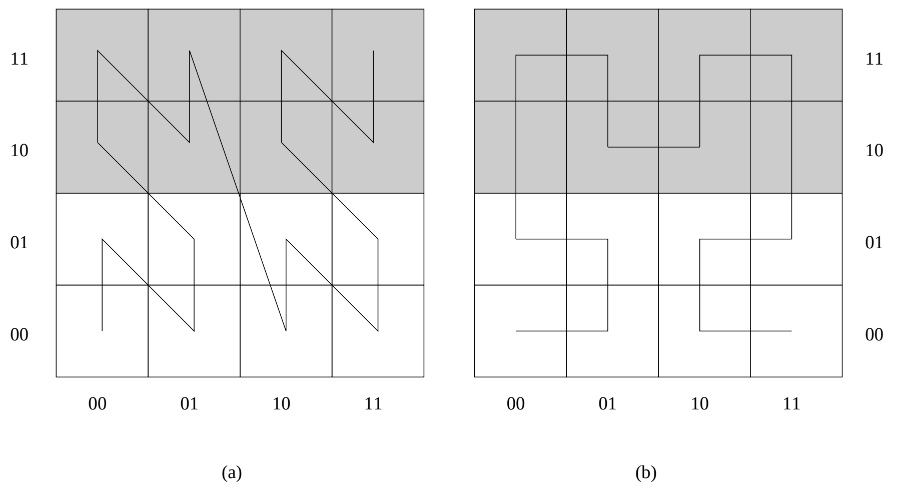


Figure 2.3 Example of a range query in a domain that is mapped using (a) Z-SFC and (b) Hilbert curve.

and geographic information systems often generate yet another kind of queries, called *range queries*, in which all points within a specified hyper-rectangular region of the underlying domain are to be reported. In Fig. 2.3(a) and Fig. 2.3(b), the shaded regions represent such a range query in two dimensions. Clearly, the space filling curve on the left (Z-SFC) generates two non-contiguous portions of the linearized array compared to the one on the right (Hilbert curve) which contains the query region in one contiguous sub-array.

Therefore, it comes as no surprise that different space filling curves exhibit different locality preserving properties. Over the years, a few results have been reported in this context. Different measures of locality have been proposed and locality properties of different SFCs based on such definitions have been studied. Locality of multidimensional SFCs have been studied in [Gotsman and Lindenbaum (1996)] based on a measure which reflects the extent to which two points that are close along the linearized data can be far apart in the original multidimensional space under a Euclidean norm. They showed that the Hilbert curve comes closest to achieving optimal locality based on the above measure. Clustering properties have been analyzed in [Abel and Mark (1990)] for two dimensions and for higher dimensions in [Jagadish (1990)]. Closed form formulas for the number of clusters in a query region of arbitrary shape have been derived for the Hilbert curve in]Moon et al. (2001)].

Note that SFC based partitions can result in non-contiguous regions of the domain to be mapped to the same processor. For example, in Fig. 2.2(a), partitioning the 12 points across four processors results in points 7, 8 and 9 to be mapped to a single processor. Unlike the other three partitions, the region of space that maps to the processor owning the above three points is disjoint. Even if the mapped domain is continuous, it may be of a complicated shape. Complicated shapes of partitions naturally raises the question of how good or bad is the computation to communication ratio for a SFC based parallel domain decomposition. Parallel applications [Dennis (2003); Griebel and Zumbusch (2002); Parashar and Browne (1996); Pilkington and Baden (1996); Steensland et al. (2002)] that use SFCs to partition the problem domain exhibit good scaling and runtime results. Empirical justification remains the mainstay of the popularity of SFCs.

As a first step towards a formal analysis of SFC based parallel domain decomposition, we analyze the quality of parallel partitioning of a set of uniformly distributed points in 2D using a Z-SFC in the next section.

2.2 Analysis of Z-Space Filling Curves

The Z-curve lends itself to a particularly elegant implementation based on bit interleaving that has often been used [Warren and Salmon (1993); Hariharan et al. (2002)]. Consider a two dimensional $2^k \times 2^k$ decomposition. Each of the 2^{2k} cells can be specified using two coordinates (i, j) where $0 \leq i, j \leq 2^k - 1$. The order in which the cells are visited by the Z-curve, also called the *Morton order* [Morton (1966)], can be shown to be identical to the order imposed on the set of distinct integers that is obtained by interleaving the bit representation of the coordinates of each cell . As an example [see Fig. 2.4], for a cell whose coordinates are $(2, 3) \equiv (10, 11)_2$, interleaving the bits of the coordinates results in the integer $1101 \equiv 13$, which is the order in which this cell is visited by the Z-SFC. This convenient interleaving scheme can be generalized into arbitrary dimensions; i.e., if a cell in d dimensions is specified by the coordinates $(x_1, x_2, \cdots, x_d)$, then the order in which the cell is visited along the Z-curve is obtained by interleaving the bits of $x_1, x_2, \cdots, x_d$.

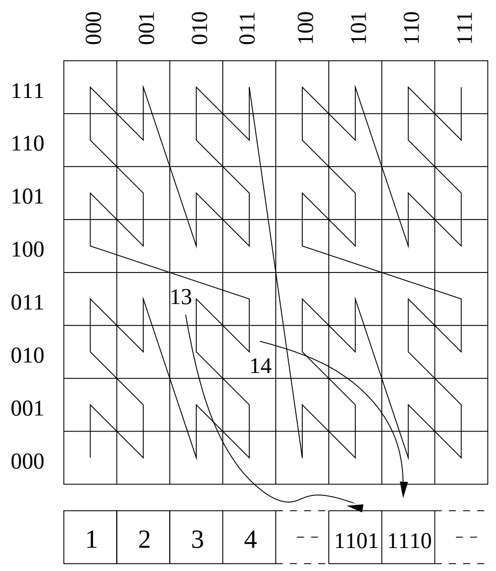


Figure 2.4 Interleaving the bit representations of the coordinates of the cells yields their order according to the Z-SFC.

Let L_{ij} denote the integer obtained as a result of interleaving the bits of the binary representations of i and j where $0 \leq i, j \leq 2^k - 1$ for some k. Clearly, L_{ij} consists of $2k$ bits in which the odd positions are occupied by the bits from the binary representation of i and the even positions by those of j with the understanding that the least significant bit is at position 0. Let b_{ij}^q denote the bit value at position q of L_{ij}.

Observation 1 *The distance, denoted by $d(L_{ij}, L_{uv})$, along a Z-space filling curve between the cells with coordinates (i, j) and (u, v) in a $2^k \times 2^k$ decomposition of a square is:*

$$d(L_{ij}, L_{uv}) = \sum_{q=0}^{2k-1} (b_{ij}^q - b_{uv}^q) 2^q \tag{2.1}$$

Lemma 1 *For a $2^k \times 2^k$ decomposition of a square:*

$$d\left(L_{2^{k-1}j}, L_{(2^{k-1}-1)j}\right) = \frac{1}{3}\left(4^k + 2\right)$$

for all $0 \leq j \leq 2^k - 1$.

Proof From the bit interleaving operation as described above, it follows that the even bits are the same for $L_{2^{k-1}j}$ and $L_{(2^{k-1}-1)j}$. Thus, we have :

$$d\left(L_{2^{k-1}j}, L_{(2^{k-1}-1)j}\right) = \sum_{q=0}^{2k-1}\left(b^q_{2^{k-1}j} - b^q_{(2^{k-1}-1)j}\right)2^q = \sum_{w=0}^{k-1}\left(b^{2w+1}_{2^{k-1}j}\ b^{2w+1}_{(2^{k-1}-1)j}\right)2^{2w+1}$$

In $L_{2^{k-1}j}$, the most significant odd bit is 1 and the other odd bits are 0 while in $L_{(2^{k-1}-1)j}$, the most significant odd bit is 0 and the other odd bits are 1. Therefore:

$$d\left(L_{2^{k-1}j}, L_{(2^{k-1}-1)j}\right) = 2^{2k-1} - \sum_{w=0}^{k-2} 2^{2w+1} = \frac{1}{3}\left(4^k + 2\right)$$

which concludes the proof. ∎

Lemma 2 *For a* $2^k \times 2^k$ *decomposition of a square:*

$$d\left(L_{i2^{k-1}}, L_{i(2^{k-1}-1)}\right) = \frac{1}{6}\left(4^k + 2\right)$$

for all $0 \le i \le 2^k - 1$.

Proof Similar to that of Lemma 1.

A cell is considered *adjacent* to each of its neighbors along the four directions east, west, north and south, respectively. Let $S_{\rightarrow}(k)$ denote the sum of the distances along the SFC array between each cell and its adjacent cell to the east in a $2^k \times 2^k$ decomposition of a square. For all cells in the rightmost column of the cell decomposition, the distance to the adjacent cell to the east is defined to be zero.

Lemma 3 $S_{\rightarrow}(k)$ *satisfies the following recurrence relation:*

$$S_{\rightarrow}(k+1) = 4S_{\rightarrow}(k) + \frac{8}{3}\cdot 2^k \cdot \left(4^k + \frac{1}{2}\right)$$

with $S_{\rightarrow}(0) = 0$.

Proof Consider Fig. 2.5 which shows a $2^{k+1} \times 2^{k+1}$ decomposition of a square. The first term in eqn (2.2) results directly from the fact that the $2^{k+1} \times 2^{k+1}$ square consists of four $2^k \times 2^k$

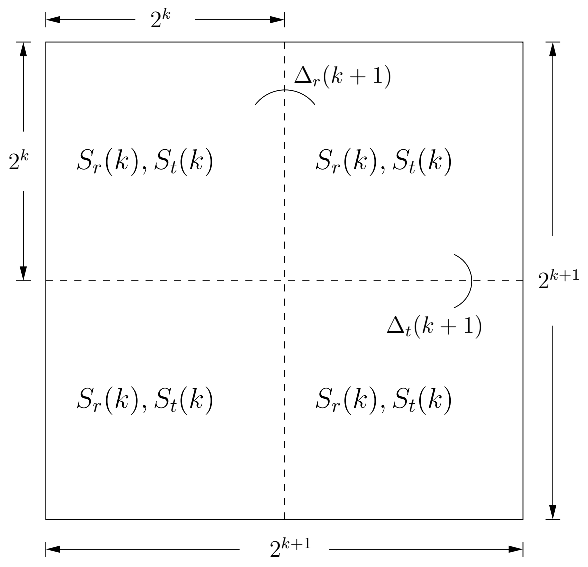


Figure 2.5 Contribution to $S_{r,t}(k+1)$ from $S_{r,t}(k)$ and $\Delta_{r,t}(k+1)$.

decompositions of its quadrants, each of which contribute $S_{\rightarrow}(k)$ to $S_{\rightarrow}(k+1)$. In going from a $2^k \times 2^k$ decomposition to $2^{k+1} \times 2^{k+1}$ decomposition, one has to include the contribution from the additional 2^{k+1} distances to the east adjacent neighbor of the cells in the column numbered $2^k - 1$. Let us denote this additional contribution by $\Delta_r(k+1)$. Using Lemma 1, we have :

$$\Delta_r(k+1) = 2^{k+1} \cdot d\left(L_{2^k j}, L_{(2^k-1)j}\right) = 2^{k+1} \cdot \frac{1}{3}\left(4^{k+1}+2\right) = \frac{8}{3} \cdot 2^k \cdot \left(4^k + \frac{1}{2}\right)$$

which proves the lemma. ∎

Lemma 4 *The solution of eqn (2.2) is:*

$$S_{\rightarrow}(k) = \frac{2}{3} 2^k (4^k - 1)$$

Proof By induction. Clearly, $S_{\rightarrow}(0) = 0$. Assume that $S_{\rightarrow}(q) = \frac{2}{3} 2^q (4^q - 1)\ \forall\ 0 \leq q \leq k$. Then, for $q = k+1$:

$$S_{\rightarrow}(k+1) = \frac{2}{3} 2^{k+1}\left[4^{(k+1)} - 1\right] = 4S_{\rightarrow}(k) + \frac{8}{3} \cdot 2^k \cdot \left(4^k + \frac{1}{2}\right)$$

which proves the lemma. ▮

In a similar manner, one can define $S_{\uparrow}(k)$ to denote the sum of the distances along the SFC array between each cell and its adjacent cell to the north in a $2^k \times 2^k$ decomposition of a square. For a cell in the topmost row of the cell decomposition, the distance to its adjacent cell to the north is defined to be zero. Since the proofs for $S_{\uparrow}(k)$ are almost identical to the ones for $S_{\rightarrow}(k)$, we state the following lemmas without explicitly proving them.

Lemma 5 *$S_{\uparrow}(k)$ satisfies the following recurrence relation:*

$$S_{\uparrow}(k+1) = 4S_{\uparrow}(k) + \frac{4}{3} \cdot 2^k \cdot \left(4^k + \frac{1}{2}\right)$$

with $S_{\uparrow}(0) = 0$.

Lemma 6 *The solution of eqn (2.2) is:*

$$S_{\uparrow}(k) = \frac{1}{3}2^k(4^k - 1)$$

Like $S_{\rightarrow}(k)$ and $S_{\uparrow}(k)$, let the corresponding sums of the distances to the adjacent cells to the west and south be denoted by $S_{\leftarrow}(k)$ and $S_{\downarrow}(K)$, respectively. Also, denote the sum of the distances along the SFC array to all four adjacent cells by $S(k)$. In other words, $S(k) = S_{\rightarrow}(k) + S_{\uparrow}(k) + S_{\leftarrow}(k) + S_{\downarrow}(k)$. Then:

Lemma 7 *For a $2^k \times 2^k$ Z-SFC, $S(k) = 2^{k+1}(4^k - 1)$.*

Proof Since the distance between a cell and its adjacent cell to the right is equal to the distance between that neighbor and its adjacent cell to the left for all cells, $S_{\leftarrow}(k) = S_{\rightarrow}(k)$. Similarly, $S_{\downarrow}(k) = S_{\uparrow}(k)$. Therefore, using Lemma 4 and Lemma 6, we have:

$$S(k) = 2S_{\rightarrow}(k) + 2S_{\uparrow}(k) = 2^{k+1}(4^k - 1).$$

This proves the lemma. ▮

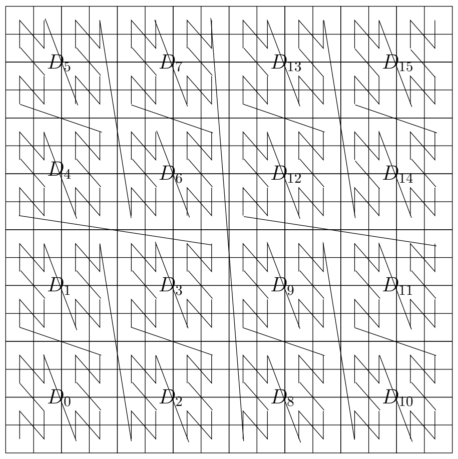


Figure 2.6 Partitioning a $2^4 \times 2^4$ decomposition across 16 processors using a Z-space filling curve.

If we denote the average distance along the SFC of a cell to an adjacent cell in a $2^k \times 2^k$ decomposition of a square by $\bar{S}(k)$, then the following theorem is immediate:

Theorem 8 *The average distance along a $2^k \times 2^k$ Z-SFC to an adjacent cell is:*

$$\bar{S}(k) = 2\sqrt{n}\left(1 - \frac{1}{n}\right) = \Theta\left(\sqrt{n}\right)$$

Proof Follows from Lemma 7 and the fact that $\bar{S}(k) = S(k)/n$ where $n = 4^k$. ∎

The above theorem can be extended to include adjacent cells at the corners of each cell. Whereas the resulting analysis becomes more complicated, $\bar{S}(k)$ yields the same asymptotic behavior as Theorem 8 and all arguments based on it remain unaltered. The generalization of $\bar{S}(k)$ to higher dimensions is straightforward and yields $\bar{S}(k) = \Theta\left(n^{\frac{\nu-1}{\nu}}\right)$ in ν-dimensions.

Types of accesses required vary from application to application but an immediate ramification of Theorem 8 can be understood by considering one that generates a nearest neighbor query at every filled cell as is likely the case for any finite difference method based application. The conventional way of using Z-SFCs for parallel domain decomposition is as follows:

1. Order the $n = 2^{2k}$ cells into a linear array using a Z-space filling curve.

2. Partition the resulting array equally among P processors so that processor with rank i, denoted P_i, gets the $\frac{n}{P}$ cells with SFC ranks $i\frac{n}{P}, i\frac{n}{P}+1, \cdots, (i+1)\frac{n}{P}-1$ where $0 \leq i \leq P-1$.

The expectation is that the proximity preserving properties of the space filling curve will keep most of the accesses local. Consider the partitioning of a $2^k \times 2^k$ Z-SFC across $P = 2^{2m}$ processors. Each processor receives a $2^{k-m} \times 2^{k-m}$ square domain of cells. An example with $k = 3$ and $m = 2$ is shown in Fig. 2.6. In executing an iteration of the parallel application, a processor communicates with the four other processors containing adjacent cells to fetch nearest neighbor information. In the dense case such as the one considered here, the number of cells assigned to a processor is quadratic in 2^{k-m} while the number of cells requiring communication is linear in 2^{k-m}, the cells along the boundaries of the domain assigned to a processor.

The same domain decomposition in the processor space can be viewed in the following way: consider a coarser $2^m \times 2^m$ decomposition of the domain of $2^k \times 2^k$ cells. Each cell in the $2^m \times 2^m$ decomposition is the sub-domain assigned to a processor. The rank of the processor which is assigned a sub-domain in the conventional partitioning scheme is the same as the SFC rank of the sub-domain assigned to it in the $2^m \times 2^m$ processor space decomposition. Viewed another way, one can think of the SFC as ordering both the data and the processor space.

From this observation, the induced mapping in the processor space respects the same properties as those of the $2^k \times 2^k$ decomposition of the data space presented earlier. Therefore, on an average, to satisfy the nearest neighbor queries for the cells in the rows and columns at the boundaries of the processors, each processor will have to communicate with processors whose ranks are an average distance of $\Theta\left(\sqrt{P}\right)$ away in the induced linear ordering of the $2^m \times 2^m$ processor space (see Theorem 8). To illustrate the effect of this induced mapping in the processor space, consider a hypercube topology (see Fig. 2.7). Consider the sub-domain given by the cell number $(2^{m-1}-1, j)$ in the $2^m \times 2^m$ processor space decomposition, for any j. The east neighbors of cells along the east boundary of this domain can be found in the processor whose sub-domain is specified by the cell number $(2^m, j)$. From Observation 1, these two processors have ranks that differ in $m = \Theta(\log P)$ bits. They are as far apart as the

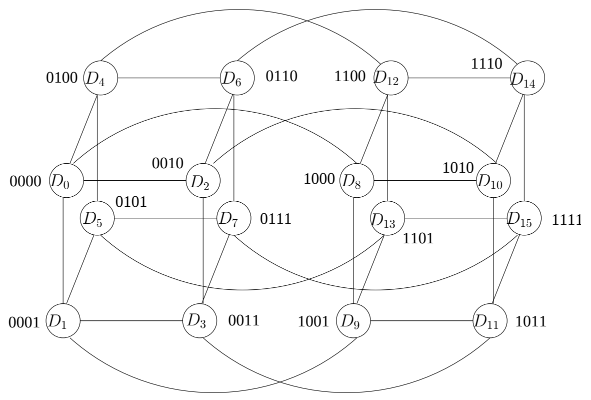


Figure 2.7 Data partitioning due to the conventional algorithm on a 16 processor hypercube. D_i denotes the sub-domain assigned to processor P_i.

diameter of the hypercube.

It is well known that application performance is improved when the topology implied by the communication within the application is appropriately mapped to the topology of the physical architecture. In a recent study, [Kumar et al. (2006)] identified the mismatch between the default communication topology assumed by their NAMD molecular dynamics code and the IBM BlueGene/L architecture as one of the main sources of performance bottleneck, and show significantly improved performance by matching the two. Despite the popularity of SFC based decompositions, topology aware mapping of space filling curves has so far not been considered. It is clear that a different mapping of the sub-domains to processors can ensure that all communicating processors are neighbors in a hypercube architecture. Consider the following assignment strategy:

1. Order the $n = 2^{2k}$ cells into a linear array using a Z-space filling curve.

2. Partition this SFC array into $P = 2^{2m}$ equal segments $D_0, D_1, \ldots, D_{P-1}$.

3. Assign D_i to the processor whose label represents i in the binary reflected Gray code ordering.

This strategy imposes a Z-space filling curve order to map the data but uses a Gray code

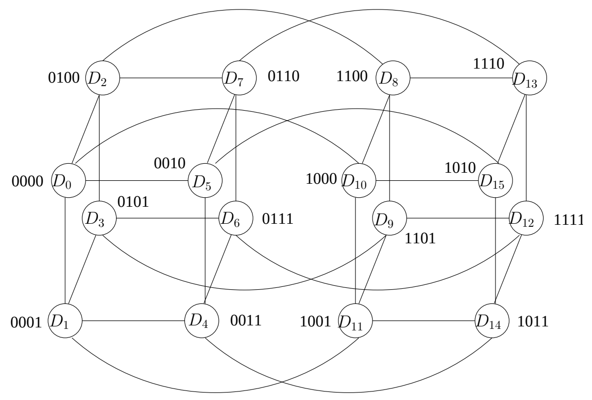


Figure 2.8 Data partitioning due to the proposed algorithm on a 16 processor hypercube. The same domain labeling is used as in Fig. 2.7.

curve to map the partitions to the processors. Note that the mapping used is the same as the classic embedding of a mesh (in this case the $2^m \times 2^m$ sub-domain space) in a hypercube. Thus, this mapping has the property that two adjacent sub-domains are assigned processors whose ranks differ in exactly one bit. Thus, all resulting communication is between nearest neighbors in the hypercube. The new assignment algorithm when applied to the example in Fig. 2.6 yields the data distribution in Fig. 2.8.

Such topology-aware mapping of SFCs can be carried out for other architectures. For example, using an embedding strategy of a two dimensional mesh into two or higher dimensional mesh architectures, a topology aware SFC mapping can be carried out on such target architecture. A nice advantage of this approach is that it can directly utilize the considerable research results that are known in embeddings.

Clearly, the above conclusions were arrived at using an analysis of Z-SFCs that took advantage of their bit-interleaved representation. For other popular SFCs such as the Hilbert SFC or the Gray code SFC, corresponding bit representations exist [Pilkington and Baden (1996)] though not as elegant. Consequently, a similar approach is not immediately tenable. In addition, the above results presume a distribution of points in which every cell of the spatial decomposition is filled. This assumption limits the scope of the analysis since in a realistic problem domain, the computing elements represented by the points or filled cells most often are not distributed evenly in space as is supposed above. The goal in the next section is to be able to generalize such an analysis to include a broader class of SFCs while capturing more realistic point distributions.

2.3 Geometric Properties of Common Space Filling Curves

A SFC-based domain decomposition is initiated by first enclosing the discrete set of three dimensional points that represent the domain within the smallest cube. This cube is then recursively bisected k number of times until the result three dimensional array of cells are such that each cell contains at most one point. As stated in section 2.1, k is called the resolution of the decomposition. Such a decomposition results in a $2^k \times 2^k \times 2^k$ array of cells, some occupied by a point while others are empty. We will denote the set of $m = 2^{3k}$ cells, occupied as well as unoccupied, by σ. As a result of the above recursive bisections, two kinds of cubes are generated at each level - ones that are a result of the bisections and others that are not. We will distinguish between them.

Definition 1 *Standard cubes refer to all the intermediate cubes that result from the process of recursive bisection of a cube along each dimension.*

A 2D example is shown in Fig. 2.9. In Fig. 2.9(a), squares $c_1c_2c_3c_4$, $b_1b_2b_3b_4$ and $a_1a_2a_3a_4$ are squares obtained during the first, second and third recursive bisections, respectively, of the outermost square. As such, they are all standard squares. On the other hand, the squares $a'_1a'_2a'_3a'_4$ and $b'_1b'_2b'_3b'_4$ cannot be obtained using the recursive bisections and are therefore not treated as standard squares. An important property satisfied by all standard cubes is that an SFC enters and exits such cubes exactly once. This is not true for non-standard cubes. Fig. 2.9(b) shows this for a Z-SFC though this is true for both the Hilbert SFC and the Gray code SFC. We state this fact formally as:

Observation 2 *Z, Hilbert and Gray code space filling curves enter and exit standard cubes exactly once implying that the cells in a standard cube form a contiguous sub-array in the resulting linear array.*

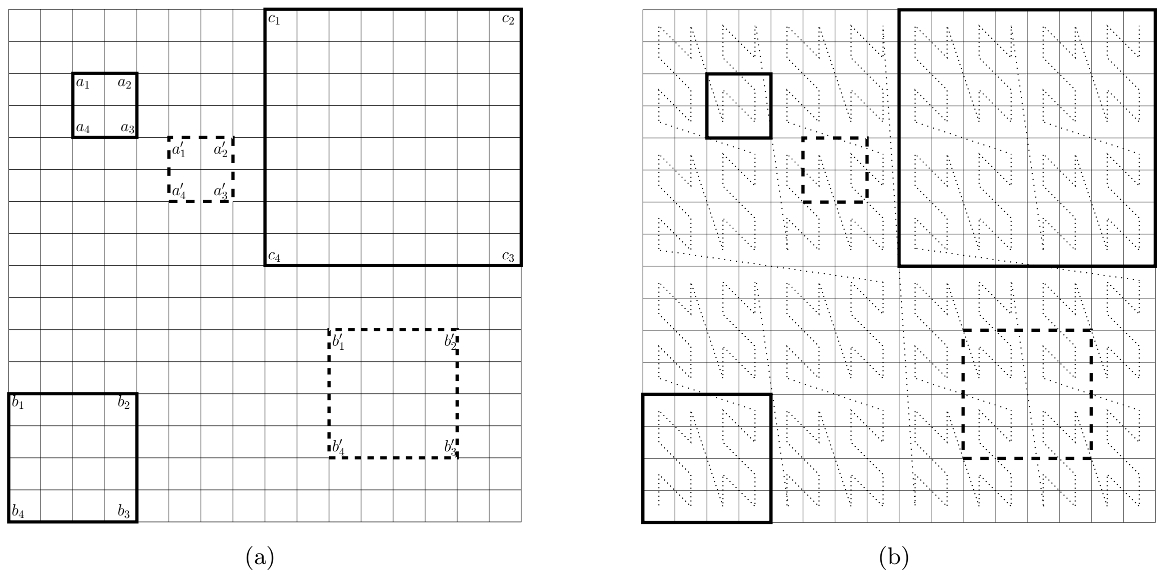


Figure 2.9 (a) Example of standard squares in 2D: $a_1a_2a_3a_4$, $b_1b_2b_3b_4$ and $c_1c_2c_3c_4$ are standard squares obtained at recursive levels 3,2 and 2, respectively. $a'_1a'_2a'_3a'_4$ and $b'_1b'_2b'_3b'_4$ are examples of non–standard squares. (b) The Z-SFC enters and exits a standard square only once unlike a non-standard square.

In addition, we need to define the following geometric constructs in order to be able to extract the common properties exhibited by a large class of SFCs.

Definition 2 *$C(u, d)$ is defined as the region of overlap between a sphere of radius d centered at cell u and the $2^k \times 2^k \times 2^k$ array of cells. $|C(u, d)|$ denotes the number of cells in $C(u, d)$.*

Definition 3 *A cell v is said to be contained in $C(u, d)$ iff the center of v is contained within $C(u, d)$.*

Definition 4 *For cell u, let $A(u, d)$ denote the smallest standard cube that encloses the region $C(u, d)$. $|A(u, d)|$ denotes the number of cells in $A(u, d)$.*

Note that the size of $C(u, d)$ depends on the position of u in the $2^k \times 2^k \times 2^k$ array of cells. A 2D example is shown in Fig. 2.10. The region of overlap between three circles of the same radius but centered at cells labeled u, v and w are shown. It can be easily seen that

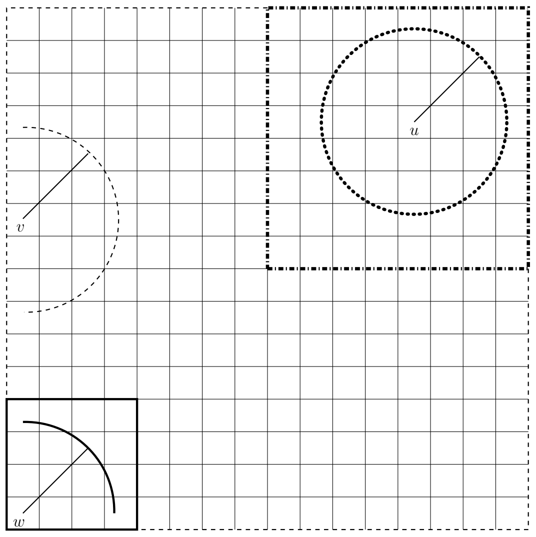


Figure 2.10 A 2D example illustrating Definition 2 and Definition 4 of $C(u,d)$ and $A(u,d)$.

$|C(u,d)| \neq |C(v,d)| \neq |C(w,d)|$. The smallest standard square enclosing $C(u,d)$, $C(v,d)$ and $C(w,d)$ are accordingly different whence $|A(u,d)| \neq |A(v,d)| \neq |A(w,d)|$. Based on the above geometrical definitions, the following two lemmas follow:

Lemma 9 *There are at least $\frac{d^3}{8}$ cells contained in $C(u,d)$, excluding u.*

Proof Consider a coordinate system with origin at the center of cell u. Irrespective of the location of u, we can find a point (x,y,z) such that $|x| = |y| = |z| = \frac{d}{\sqrt{3}}$ and (x,y,z) is on the surface of $C(u,d)$. Let $w = \lfloor \frac{d}{\sqrt{3}} \rfloor$. Since w is integral, the cube of side w with one corner at u and another corner at (x,y,z) has $(w+1)^3$ grid cell centers inside, or on the boundary of the cube. But since $w+1 > \frac{d}{\sqrt{3}} > d/2$, we have $(w+1)^3 > d^3/8$, which proves the lemma. ▮

Lemma 10 $\sum_{u\in\sigma} |A(u,d)| = O\left(d \cdot m^{\frac{5}{3}}\right)$

Proof Let S_l denote a cube at level l. For all the cells u located in a cuboid of dimensions $2d \times (2^{k-l} - 2d) \times (2^{k-l} - 2d)$ placed symmetrically at the center of S_l (see Fig. 2.11), $A(u,d)$ is S_l. Hence, for each cell u within the region in S_l formed from the fusion of the three cuboids, $|A(u,d)| = 8^{k-l}$. The number of such cells in S_l is at most the total number of cells in the

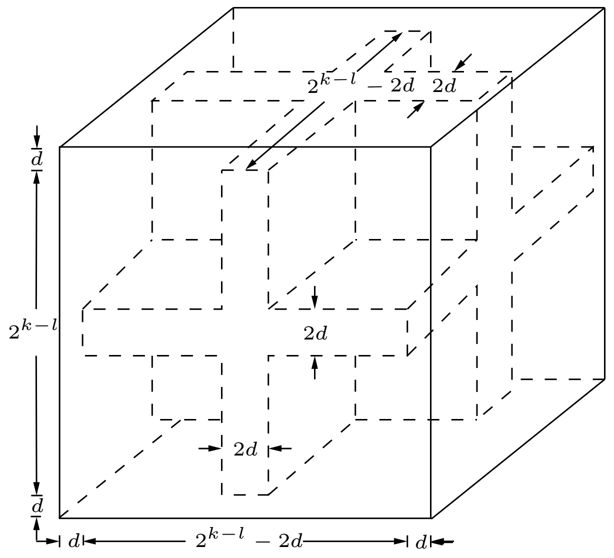


Figure 2.11 The bounding cube is S_l. The dashed inner region is composed from fusing three cuboids, each of dimensions $2d \times (2^{k-l} - 2d) \times (2^{k-l} - 2d)$. For a cell u in this region, $A(u, d)$ is S_l.

three cuboids, which is $3 \cdot 2d \cdot 2^{k-l} \cdot 2^{k-l} = 6 \cdot d \cdot 4^{k-l}$. If σ_l denotes the set of cells $u \in \sigma$ for which $A(u, d) = 8^{k-l}$, then $|\sigma_l| = 8^l \cdot (6 \cdot d \cdot 4^{k-l})$. Since for all $u \in \sigma_l$, $|A(u, d)| = 8^{k-l}$, it follows that:

$$\sum_{u\in\sigma} |A(u,d)| = \sum_{l=0}^{k} \sum_{u\in\sigma_l} |A(u,d)| = \sum_{l=0}^{k} (8^l) \cdot (6 \cdot d \cdot 4^{k-l}) \cdot (8^{k-l}) = O\left(d \cdot m^{\frac{5}{3}}\right) \quad \blacksquare$$

Note that the counting results embodied in Lemma 9 and Lemma 10 are independent of the particular SFC that may be used to linearize the $2^k \times 2^k \times 2^k$ array of cells. As such, they hold for all linear mappings of the 3D array of cells induced by any SFC.

2.4 Modeling the Problem of Parallel Domain Decomposition

Given $m = 2^{3k}$ cells in a $2^k \times 2^k \times 2^k$ array of cells, we populate the cells by independently considering each cell, and placing a point in it with probability p $(0 < p < 1)$. A cell is called *occupied* if a point is placed in it, and is called *unoccupied* otherwise. We assume the point in an occupied cell is placed at its center. Let σ denote the set of all m cells, and $\sigma(m,p)$ denote the set of occupied cells. A SFC is then used to arrange the cells in $\sigma(m,p)$ into a one dimensional array, called the *SFC-array*. Let $|\sigma(m,p)|$ denote the number of points. The problem size $n = E[|\sigma(m,p)|] = mp$. Parallel domain decomposition is accomplished by splitting the SFC-array evenly across processors using block decomposition. Let $P = n^{\alpha}$ $(0 < \alpha \leq 1)$ denote the number of processors. Each processor receives at most $\left\lceil \frac{|\sigma(m,p)|}{P} \right\rceil$ elements.

2.5 Nearest Neighbor Distance

Using the above model, we will bound the average SFC-array distance between a point and its nearest neighbor and compare the result with that stated in Theorem 8. An analysis of the average SFC-array distance yields insights into the communication overhead incurred due to several commonly encountered spatial queries when a computational domain is partitioned across processors using and SFCs, as will be presented in subsequent sections.

Definition 5 *For cell $u \in \sigma$, the Euclidean distance to a nearest neighbor, denoted by d_u, is defined as:*

$$d_u = \begin{cases} 0 \text{ , } \textit{if } u \textit{ is unoccupied} \\ 0 \text{ , } \textit{if all other cells are unoccupied} \\ \textit{distance to the nearest neighbor, otherwise} \end{cases}$$

A point may have multiple nearest neighbors. We will only consider the nearest neighbor that lies farthest along the SFC-array.

Definition 6 *For cell $u \in \sigma$, define X_u as:*

$$X_u = \begin{cases} 0 \text{ , } \textit{if } u \textit{ is unoccupied} \\ 0 \text{ , } \textit{if all other cells are unoccupied} \\ \textit{maximum SFC-array distance to a nearest neighbor, otherwise} \end{cases}$$

Definition 7 *Let Z denote the average of the maximum SFC-array distance to a nearest neighbor, where the average is taken over the expected number of occupied cells.*

$$Z = \frac{1}{pm} \sum_{u \in \sigma} X_u$$

By linearity of expectation, we have

$$\mathbf{E}\left[Z\right] = \frac{1}{pm} \sum_{u \in \sigma} \mathbf{E}\left[X_u\right] \tag{2.2}$$

Using the definition of X_u:

$$\begin{aligned} \mathbf{E}\left[X_u\right] &= \Pr\{u \text{ is occupied }\} \cdot \Pr\{u \text{ is not the only occupied cell }\} \cdot \mathbf{E}\left[X_u | X_u \neq 0\right] \\ &\leq p \sum_{1 \leq d \leq d_{max}} \Pr\{d_u = d\} \cdot \mathbf{E}\left[X_u \mid d_u = d\right] \end{aligned} \tag{2.3}$$

where d_{max} is the largest value that d can take given a $2^k \times 2^k \times 2^k$ block of cells. Combining eqn (2.2) and eqn (2.3), we have:

$$\mathbf{E}\left[Z\right] = \frac{1}{m} \sum_{u} \sum_{d} \Pr\{d_u = d\} \, \mathbf{E}\left[X_u \mid d_u = d\right] \tag{2.4}$$

where we adopt the notation used for the rest of this chapter in which the summation of u implies that it is over all $u \in \sigma$ and that of d implies that it is over all $1 \leq d \leq d_{max}$ unless otherwise specified.In Sections 5, 6 and 7, we analyze SFC-based parallel domain decomposition with respect to nearest neighbor, κ nearest neighbor and spherical region queries.

Lemma 11 $\mathbf{E}\left[X_u | d_u = d\right] \leq p|A(u,d)|$

Proof Let Y_{ud} denote the length of the SFC-subarray that contains all occupied cells in $A(u,d)$ (see Observation 2). If $d_u = d$, then $X_u \leq Y_{ud}$, since both u and its nearest neighbor lie in an SFC-subarray of length Y_{ud}. Thus, it follows that $\mathbf{E}\left[X_u | d_u = d\right] \leq \mathbf{E}\left[Y_{ud}\right] = p|A(u,d)|$, since every cell in $A(u,d)$ is occupied with probability p. ▮

Lemma 12 *For any $u \in \sigma(m,p)$ and $d \in [1, d_{max}]$,* $\Pr\{d_u = d\} \leq (1-p)^{d^3/8}$

Proof The event $d_u = d$ implies that $C(u,d)$ has no occupied cells. It follows from Lemma 9 that at least $d^3/8$ cells must be unoccupied. The probability of this event is no more than $(1-p)^{d^3/8}$. ∎

Lemma 13

$$\sum_{1\le d\le d_{max}} d(1-p)^{\frac{d^3}{8}} \le \frac{16}{3p}$$

Proof Using $(1+1/n)^n < e$, we get:

$$S = \sum_{1\le d\le d_{max}} d(1-p)^{\frac{d^3}{8}} \le \sum_{1\le d\le d_{max}} de^{\frac{-pd^3}{8}}$$

Note that d is not necessarily an integer. However, d^2 is an integer because $d^2 = x^2+y^2+z^2$ for integers x, y and z. Letting $r = d^2$, we have:

$$S \le \sum_{r=1}^{\infty} \sqrt{r} e^{\frac{-pr\sqrt{r}}{8}} \le \int_0^{\infty} \sqrt{r} e^{\frac{-pr\sqrt{r}}{8}} dr$$

Using a change of variables, let $y = p^{\frac{1}{3}}\sqrt{r}/2$. Thus: Then, $dy = p^{\frac{1}{3}} dr/4\sqrt{r}$.

$$S \le \frac{16}{p}\int_0^{\infty} y^2 e^{-y^3} dy = \frac{16}{p}\frac{\Gamma(1)}{3} = \frac{16}{3p}$$

where $\Gamma(q) = \int_0^\infty e^{-t}t^{q-1}dt$ is the Gamma function. ∎

Theorem 14 $\mathbf{E}[Z] = \Theta(n^{2/3})$.

Proof Using eqn (2.4), Lemma 10, Lemma 11 and Lemma 12, we get:

$$\mathbf{E}[Z] \le p^{\frac{1}{3}} n^{\frac{2}{3}} \sum_{1\le d\le d_{max}} d(1-p)^{\frac{d^3}{8}} \le \frac{p^{-\frac{2}{3}}}{3} n^{\frac{2}{3}} = \Theta\left(n^{\frac{2}{3}}\right)$$

which proves the theorem. ∎

As shown in the appendix, this result can be generalized to show that the nearest neighbor of a point in ν dimensions is located at an expected SFC-array distance of $\Theta\left(n^{\frac{\nu-1}{\nu}}\right)$. Clearly, this measure of locality deteriorates with increasing dimensionality. This is not surprising since the number of neighboring cells grows exponentially as the number of dimensions increase. As a result, capturing locality with any one dimensional ordering becomes increasingly difficult. The large expected nearest neighbor distance results from the fact that equal weight is assigned to cells, u for which the corresponding $|A(u,d)|$ is large compared to those for which $|A(u,d)|$ is small, though there are many more of the latter than the former. The three dimensional result above seemingly indicates that having fewer than $\Theta\left(n^{2/3}\right)$ points per processor might result in all remote accesses, as not even the expected distance of a nearest neighbor falls within the same processor. In turn, this would imply that the number of processors would be restricted to $O(n^{1/3})$. A more careful analysis presented in the next section, will show that this is not really the case. In fact, it will be shown that the number of remote access remains sublinear unless $P = \Theta(n)$.

2.6 Parallel Nearest Neighbor Queries

As stated in section 2.4, parallel domain decomposition using SFCs is carried out by partitioning the SFC-array across processors using block decomposition. Nearest neighbor queries are often encountered in finite element or finite difference based applications. Clearly, not all nearest neighbor queries generated by each occupied cell can be satisfied locally due to the partial loss of spatial locality resulting from the SFC linearization. Occupied cells whose queries are not satisfied locally need to communicate with remote processors. In this section, we will bound the communication overhead due to nearest neighbor queries generated by each occupied cell. To be more precise, we will compute the expected number of occupied cells which have at least one nearest neighbor on a remote processor since those are the cells that will generate communication. In this context, we define the following indicator random variable:

Definition 8 *For $u \in \sigma$, the random variable $\mathcal{N}_u$ indicates if any nearest neighbor of u is* *remote:*

$$\mathcal{N}_u = \begin{cases} 0 \ , \ \textit{if cell } u \textit{ is unoccupied} \\ 0 \ , \ \textit{if all NNs of } u \textit{ are available locally} \\ 1 \ , \ \textit{otherwise} \end{cases} \tag{2.5}$$

Definition 9 *Let $\mathcal{N}$ be the number of points that have at least one nearest neighbor on a remote processor.*

$$\mathcal{N} = \sum_{u \in \sigma} \mathcal{N}_u$$

Intuitively, $\mathcal{N}$ corresponds to the number of input points that will result in interprocessor communication during a parallel nearest neighbor query. We wish to compute $\mathbf{E}[\mathcal{N}]$. While doing so, we will assume that $|\sigma(m,p)| > mp/2$. Noting that $\mathbf{E}[|\sigma(m,p)|] = n$, a Chernoff bound [Mitzenmacher and Upfal (2005)] yields $\Pr\{|\sigma(m,p)| \leq n/2\} = O\left(e^{-n/8}\right)$. Since this condition is true with very high probability, it can be seen through conditional probabilities that assuming this condition to be true will not change the asymptotic value of $\mathbf{E}[\mathcal{N}]$.

Note that there are two measures of distance in this analysis - the distance between two cells u and v in the original three dimensional space and that between the same two cells along the SFC-array. In Fig. 2.12(a), the SFC-array distance between the filled cells u and v is 1 while it is 2 in Fig. 2.12(b), though the physical distance between u and v are the same in both cases. As can be seen even in this small example, when the resulting SFC-array is block partitioned across three processors, u's nearest neighbor v is local in Fig. 2.12(a) but not in Fig. 2.12(b). In order to be able to capture the inter-dependence between these two distances, it it important to establish the following definition:

Definition 10 *For any point $u \in \sigma(m,p)$, and integer δ, the δ-SFC-neighborhood of u is defined as the set of all points whose SFC-array distance from u is less than or equal to δ.*

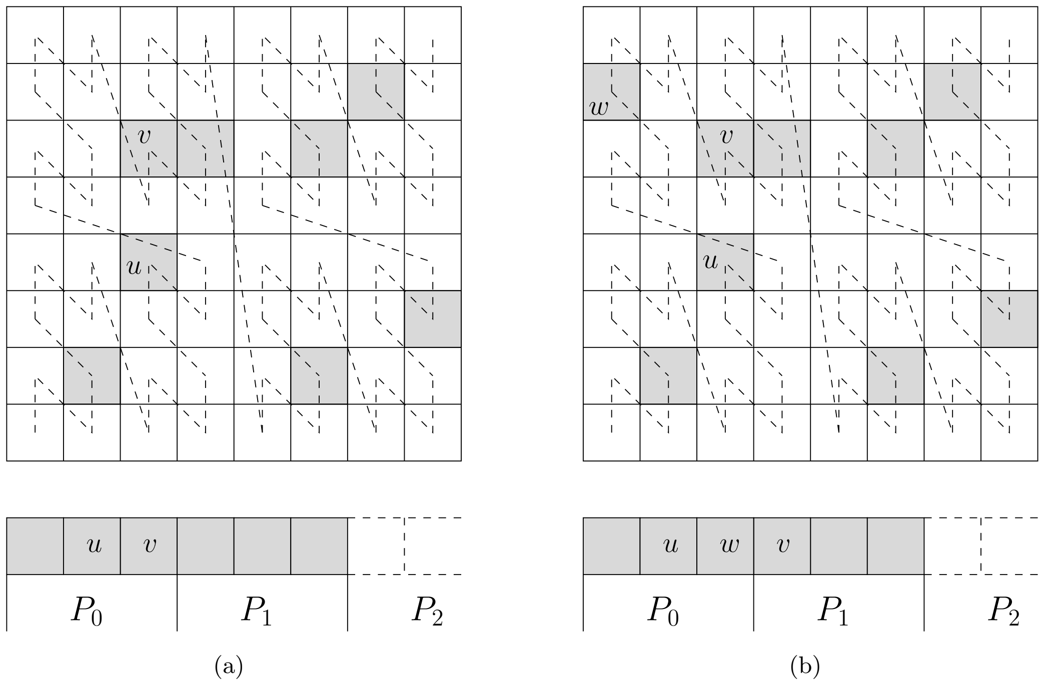


Figure 2.12 The filled cells are shaded. (a) SFC-array distance between filled cells u and v is 1. (b) SFC-array distance between filled cells u and v is 2 since cell w is encountered by the SFC between u and v.

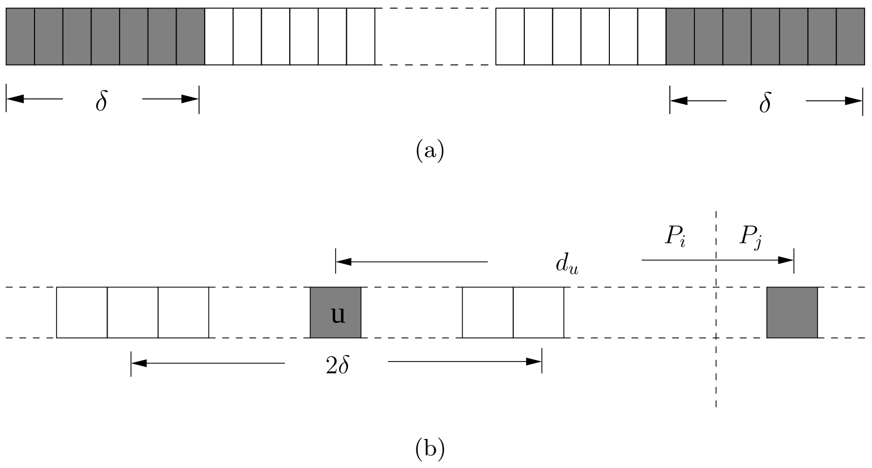


Figure 2.13 (a) Cells that are within δ distance from the left and right edges of any partition do not have their δ-SFC-neighborhood present locally. (b) Bounding $\Pr\{T_u|\overline{W}_u\}$.

In terms of the δ-SFC-neighborhood of a cell u, let us define the following two events:

W_u : event that the δ-SFC-neighborhood of u is not available locally, given that u is occupied

T_u : event that $\mathcal{N}_u = 1$, given that u is occupied

Using eqn (2.5), one can write:

$$\mathbf{E}\left[\mathcal{N}_u\right] = \Pr\{\mathcal{N}_u = 1\} = p\Pr\{T_u\} \leq p\left[\Pr\{W_u\} + \Pr\{T_u|\overline{W}_u\}\right] \tag{2.6}$$

We will bound $\Pr\{W_u\}$ and $\Pr\{T_u|\overline{W}_u\}$ separately. As shown in Fig. 2.6(a), for any given partition, only those cells that are within a distance δ from the edges do not have their δ-SFC-neighborhood present locally. There are 2δ such cells. Since the size of each partition is at least $n/2$ (see assumption above) and the number of partitions is $P = n^\alpha$, we have:

$$\Pr\{W_u\} \leq \frac{2\delta}{n/2P} = \frac{4\delta}{n^{1-\alpha}} \tag{2.7}$$

In order to bound the second term in eqn (2.6), consider the event $\{T_u|\overline{W}_u\}$ which is shown pictorially in Fig. 2.6(b). Consider a filled cell u. For any $d > 0$, all points that are at a distance less than or equal to d are contained in the standard square $A(u,d)$. All occupied cells in $A(u,d)$ lie in a contiguous portion of the SFC array (see Observation 2). The number of occupied cells in $A(u,d)$ is the binomial random variable $\mathcal{B}(|A(u,d)|,p)$ i.e. the number of heads in $|A(u,d)|$ coin tosses, where the probability of a head on each toss is p. Recall that random variable d_u is the physical distance from u to its nearest neighbor. If the nearest neighbor of u is remote [see Fig. 2.6(b)] and the δ-SFC-neighborhood of u is local, then it must be true that $A(u,d_u)$ has more than δ cells. Taking into account all possible values of the nearest neighbor distance, we get:

$$\Pr\{T_u|\overline{W}_u\} \leq \sum_d \Pr\{d_u = d\} \cdot \Pr\{\mathcal{B}(|A(u,d)|,p) > \delta\} \tag{2.8}$$

Using eqn (2.6), eqn (2.7) and eqn (2.8) yields:

$$\mathbf{E}\left[\mathcal{N}_u\right] \leq p\left[\frac{4\delta}{n^{1-\alpha}} + \sum_d \Pr\{d_u = d\} \cdot \Pr\{\mathcal{B}(|A(u,d)|, p) > \delta\}\right] \tag{2.9}$$

Using linearity of expectation, we get:

$$\begin{aligned}\mathbf{E}\left[\mathcal{N}\right] = \sum_{u\in\sigma}\mathbf{E}\left[\mathcal{N}_u\right] &\leq \sum_{u\in\sigma} p\left[\frac{4\delta}{n^{1-\alpha}} + \sum_d \Pr\{d_u = d\} \cdot \Pr\{\mathcal{B}(|A(u,d)|, p) > \delta\}\right] \\ &= 4\delta n^{\alpha} + p\sum_{u\in\sigma}\sum_d \Pr\{d_u = d\} \cdot \Pr\{\mathcal{B}(|A(u,d)|, p) > \delta\} \\ &= 4\delta n^{\alpha} + p\sum_d \Pr\{d_u = d\} \cdot \sum_{u\in\sigma} \Pr\{\mathcal{B}(|A(u,d)|, p) > \delta\} \qquad (2.10)\end{aligned}$$

The first summation in the above expression can be bound using Lemma 9. The following lemma bounds the second summation.

Lemma 15 *For* $r > 0$ *and* $\delta > \log k$*,*

$$\sum_{u\in\sigma} \Pr\{\mathcal{B}(|A(u,r)|, p) > \delta\} \leq 13mr\frac{p^{1/3}}{\delta^{1/3}}$$

Proof We partition σ into $k+1$ sets, $\sigma_0, \sigma_1, \ldots, \sigma_k$ as follows. For each cell $u \in \sigma_i$, $|A(u,r)| = 8^{k-i}$. From Lemma 10, we know $|\sigma_i| \leq 6r \cdot 4^k \cdot 2^i$. Therefore:

$$\begin{aligned}\sum_{u\in\sigma} \Pr\{\mathcal{B}(|A(u,r)|, p) > \delta\} &= \sum_{i=0}^{k}\sum_{u\in\sigma_i} \Pr\{\mathcal{B}(|A(u,r)|, p) > \delta\} \\ &= \sum_{i=0}^{k} 6r4^k 2^i \Pr\{\mathcal{B}(8^{k-i}, p) > \delta\} \\ &= 6r4^k \sum_{i=0}^{k} 2^i \Pr\{\mathcal{B}(8^{k-i}, p) > \delta\}\end{aligned}$$

Let i^* be the smallest integer such that $8^{k-i^*}p < \delta/6$. Thus $2^{i^*} \leq 2^{k+1}\left(\frac{6p}{\delta}\right)^{1/3}$. We can separate the above sum as follows:

$$\begin{aligned}\sum_{u\in\sigma}\Pr\{\mathcal{B}(|A(u,r)|,p)>\delta\} &= 6r4^k\sum_{i=0}^{i^*}2^i+6r4^k\sum_{i=i^*+1}^{k}\Pr\{\mathcal{B}(8^{k-i},p)>\delta\}\\ &\le 6r4^k2^{i^*+1}+6r4^k\sum_{i=i^*+1}^{k}2^{-\delta}\\ &\le 12r4^k2^{i^*}+6r4^kk2^{-\delta}\end{aligned}$$

We have used the following Chernoff bound: if X is a binomial random variable whose expectation is μ, then for any $\gamma > 6\mu$, $\Pr\{X \ge \gamma\} < 2^{-\gamma}$. Since $\delta > \log k$, the above expression is no more than $13r4^k2^{i^*} = 13r8^k\frac{p^{1/3}}{\delta^{1/3}} = 13mr\frac{p^{1/3}}{\delta^{1/3}}$. ∎

Using Lemma 9, Lemma 13 and Lemma 15 in eqn (2.10), we get:

$$\mathbf{E}[\mathcal{N}] \le 4\delta n^\alpha + \frac{13np^{1/3}}{\delta^{1/3}}\cdot\frac{16}{3p} = 4\delta n^\alpha + \frac{208n}{3\delta^{1/3}p^{2/3}} \tag{2.11}$$

Using eqn (2.11), the following theorem can proven:

Theorem 16 *For $P = n^\alpha$, $\mathbf{E}[\mathcal{N}] = O(n^{3/4+\alpha/4})$.*

Proof Let $\delta = n^\epsilon$. Rewriting eqn (2.11) in terms of ϵ yields:

$$\mathbf{E}[\mathcal{N}] \le 4n^{\alpha+\epsilon} + \Theta\left(n^{1-\epsilon/3}\right)$$

Minimizing the right hand side with respect to ϵ, we get $\mathbf{E}[\mathcal{N}] = O(n^{3/4+\alpha/4})$. ∎

A discussion of the implications of the above theorem is deferred until the end of this chapter.

2.7 Extending to κ Nearest Neighbors

In this section, we will extend the results of the previous section to include κ nearest neighbors (κ-NN) which we define as follows:

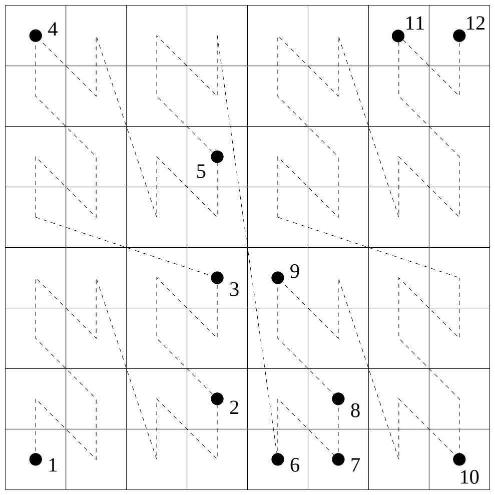


Figure 2.14 Definition of $X_{u\kappa}$.

Definition 11 *For any $u \in \sigma(m,p)$, the κ-nearest neighbor set of u is defined as the set of κ filled cells, $\mathcal{K}_u \subseteq \sigma(m,p)$, such that for any $v \in \mathcal{K}_u$ and any $w \in \sigma(m,p)\backslash\mathcal{K}_u$, $D(u,v) \leq D(u,w)$ where $D(i,j)$ denotes the Euclidean distance between cells i and j.*

Note that $|\mathcal{K}_u| = \kappa$ for all $u \in \sigma(m,p)$. Consider point 3 in Fig. 2.14. In this example, a κ-NN query with $\kappa = 2$ for point 3 contains the point set $\{2,5,9\}$. Note that points 2 and 5 are equidistant from point 3. In such cases, ties are broken arbitrarily. Thus, in the above example, for $\kappa = 2$, both sets $\mathcal{K}_3 = \{5,9\}$ as well as $\mathcal{K}'_3 = \{2,9\}$ are equally valid. Note that $|\mathcal{K}_3| = |\mathcal{K}'_3| = \kappa = 2$.

Though points 2 and 5 are at the same distance from point 3, for $\kappa = 2$ we will refer to the point 5 as the *farthest κ-NN* since the SFC-array distance between points 3 and 5 is larger than that between points 3 and 2. Before we bound the communication overhead due to parallel κ-NN queries, we first bound the average SFC-array distance to the farthest κ-NN. With this in mind, we define the following variables:

Definition 12 *For cell $u \in \sigma$, the Euclidean κ-NN distance, denoted by $d_{u\kappa}$, is defined as:*

$$d_{u\kappa} = \begin{cases} 0 \text{ , } \textit{if } u \textit{ is unoccupied} \\ 0 \text{ , } \textit{if all other cells are unoccupied} \\ \max\{D(u,v) \,|\, v \in \mathcal{K}_u\} \end{cases}$$

In the above example [see Fig. 2.14], for the point 3 and $\kappa = 2$, if the set $\mathcal{K}_3 = \{5,9\}$ is considered, then $d_{32} = D(3,5)$; otherwise, if $\mathcal{K}_2 = \{2,9\}$ is considered, then $d_{32} = D(3,2)$. Note that since $D(3,2) = D(3,5)$, d_{32} is uniquely defined.

Definition 13 *For cell $u \in \sigma$, define $X_{u\kappa}$ as:*

$$X_{u\kappa} = \begin{cases} 0 \text{ , } \textit{if } u \textit{ is unoccupied} \\ 0 \text{ , } \textit{if all other cells are unoccupied} \\ \textit{maximum SFC-array distance to a farthest } \kappa\textit{-NN, otherwise} \end{cases}$$

In the above example [see Fig. 2.14], though both points 2 and 5 are at the same distance from point 3, we will use point 5, the farthest κ-NN, to define X_{32}.

Definition 14 *Let Z_κ denote the average of the maximum SFC-array distance to a farthest κ-NN, where the average is taken over all filled cells.*

$$Z_\kappa = \frac{1}{pm} \sum_{u \in \sigma} X_{u\kappa}$$

Using linearity of expectation values yield:

$$\mathbf{E}[Z_\kappa] = \frac{1}{pm} \sum_{u \in \sigma} \mathbf{E}[X_{u\kappa}] \tag{2.12}$$

From the definition of $X_{u\kappa}$, we have:

$$\begin{aligned} \mathbf{E}[X_{u\kappa}] &= \Pr\{u \text{ is occupied }\} \cdot \Pr\{u \text{ is not the only occupied cell }\} \cdot \mathbf{E}[X_{u\kappa} | X_{u\kappa} \neq 0] \\ &\leq p \sum_{1 \leq d \leq d_{max}} \Pr\{d_{u\kappa} = d\} \cdot \mathbf{E}[X_{u\kappa} \mid d_{u\kappa} = d] \end{aligned} \tag{2.13}$$

Combining eqn (2.12) and eqn (2.13), we have:

$$\mathbf{E}\left[Z_\kappa\right] = \frac{1}{m}\sum_d \Pr\{d_{u\kappa} = d\} \cdot \sum_u \mathbf{E}\left[X_{u\kappa} \mid d_{u\kappa} = d\right] \tag{2.14}$$

Lemma 17 *For any $u \in \sigma(m,p)$ and $d \in [1, d_{max}]$, $\Pr\{d_{u\kappa} = d\} \leq p^{\kappa-1}(1-p)^{d^3/8-\kappa+1}$*

Proof The event $d_{u\kappa} = d$ implies that there are exactly $\kappa - 1$ filled cells and at least $d^3/8 - \kappa + 1$ unfilled cells within $C(u,d)$ (from Lemma 9). The probability of this event is at most $p^{\kappa-1}(1-p)^{d^3/8-\kappa+1}$. ∎

The second summation can still be bounded by using Lemma 11 since in the case of κ-NN queries, the contiguous SFC-subarray formed by the occupied cells in $A(u, d = d_{u\kappa})$ still contains all the elements in $\mathcal{K}_u$ and the arguments in the proof for Lemma 11 remain equally valid for κ-NN. The following theorem follows from the same line of reasoning as in the case of parallel nearest neighbor queries:

Theorem 18 $\mathbf{E}\left[Z_\kappa\right] = \Theta\left(\gamma^{\kappa-1} \cdot n^{\frac{2}{3}}\right)$ *where* $\gamma = \frac{p}{1-p}$.

Proof The proof follows from eqn (2.14), Lemma 10, Lemma 13 and Lemma 17. ∎

As expected, theorem 18 reduces to theorem 14 when $\kappa = 1$.

In a parallel κ-NN query, the κ nearest neighbors of each cell in the partitioned SFC-array needs to be reported back. Clearly, in a domain partitioned using SFCs, communications are generated by each cell in the SFC-array whose κ-NN are not available locally. Based on our model, we wish to bound the communication overhead incurred due to unsatisfied parallel κ-NN queries. Like before, we can generalize the definitions of the corresponding random variables to reflect κ-nearest neighbors as follows:

Definition 15 *For $u \in \sigma$, the random variable $\mathcal{N}_{u\kappa}$ indicates if any κ-nearest neighbor of u*

is remote:

$$\mathcal{N}_{u\kappa} = \begin{cases} 0 \text{ , } \textit{if cell } u \textit{ is unoccupied} \\ 0 \text{ , } \textit{if } \mathcal{K}_u \textit{ is available locally} \\ 1 \text{ , } \textit{otherwise} \end{cases} \tag{2.15}$$

Definition 16 *Let $\mathcal{N}_\kappa$ be the number of points that have at least one κ-nearest neighbor on a remote processor.*

$$\mathcal{N}_\kappa = \sum_{u \in \sigma} \mathcal{N}_{u\kappa}$$

Let:

$$T_{u\kappa} \quad : \quad \text{event that } \mathcal{N}_{u\kappa} = 1\text{, given that } u \text{ is occupied}$$

In terms of the δ-SFC-neighborhood of each cell and the event W_u as defined in section 2.6, we get:

$$\mathbf{E}\left[\mathcal{N}_{u\kappa}\right] = \Pr\{\mathcal{N}_{u\kappa} = 1\} = p \Pr\{T_{u\kappa}\} \leq p\left[\Pr\{W_u\} + \Pr\{T_{u\kappa} | \overline{W}_u\}\right] \tag{2.16}$$

Using the same reasoning as for theorem 16, we get the following for parallel κ-nearest neighbor query which we present without proof:

Theorem 19 *For $P = n^\alpha$, $\mathbf{E}\left[\mathcal{N}_\kappa\right] = O\left(\gamma^{\kappa-1} \cdot n^{(3/4+\alpha/4)}\right)$ where $0 < \alpha < 1$.*

Proof Using Lemma 17 instead of Lemma 12, the proof is similar to that of Theorem 16. ∎

Theorem 19 reduces to Theorem 16 when $\kappa = 1$, as expected.

2.8 Parallel Spherical Region Queries

In molecular dynamics simulations, the Lennard-Jones potential between atoms u and v is computed iff their physical separation $d(u, v) \leq r$, where r is a user provided cut-off distance. Therefore, it is desirable that such pairs of atoms reside on the same processor to minimize communication overhead. In this section, we analyze the total number of points that require remote accesses for such a spherical region query generated by points in a domain that has been partitioned across P processors using a SFC. We assume that the cutoff radius r is a constant that is independent of m, p or P.

Definition 17 *For a point $u \in \sigma(m,p)$, the ϕ-Euclidean-neighborhood of u is defined as the set $\{v \in \sigma(m,p) \,|\, D(u,v) \leq \phi\}$.*

Definition 18 *For $u \in \sigma$, the random variable R_u corresponds to whether the point in u has remote interactions or not, and is defined as:*

$$R_u = \begin{cases} 0 \ , \ \textit{if cell } u \textit{ is unoccupied} \\ 0 \ , \ \textit{if } r\textit{-neighborhood of } u \textit{ is local} \\ 1 \ , \ \textit{otherwise} \end{cases} \tag{2.17}$$

Definition 19 *The random variable R which corresponds to the fraction of points that have remote interactions is defined as:*

$$R = \frac{1}{mp} \sum_{u \in \sigma} R_u \tag{2.18}$$

Clearly, smaller the value of R, lesser the number of remote interactions and greater the efficiency of computation. Our goal is to bound $\mathbf{E}[R]$. As in Section 2.6, in the remainder of this section, $\mathbf{E}[R]$ is computed subject to the condition $|\sigma(m,p)| > mp/2$. Let us define the following event:

$$V_u \quad : \quad \text{event that } R_u = 1 \text{, given that } u \text{ is occupied}$$

Like before, in terms of the δ-SFC-neighborhood of each cell and the event W_u as defined in section 2.6, we write:

$$\mathbf{E}[R_u] \leq p \left\{ \Pr\{W_u\} + \Pr\{V_u | \overline{W}_u\} \right\}$$

The bound on the first term was established in section 2.6 [see eqn (2.7)] which yields:

$$\mathbf{E}\left[R\right] = \sum_{u\in\sigma} \mathbf{E}\left[R_u\right] \leq 4\delta n^{\alpha} + p\sum_{u\in\sigma} \Pr\{V_u|\bar{W}_u\} \tag{2.19}$$

To bound the second term, consider a filled cell u. All points that are in a r-Euclidean-neighborhood of u are contained in the standard square $A(u,r)$. If the r-Euclidean-neighborhood of u is not fully local while the δ-SFC-neighborhood of u is local, then it must be true that $A(u,r)$ has more than δ cells. Therefore:

$$\sum_{u\in\sigma} \Pr\{V_u|\overline{W}_u\} \leq \sum_{u\in\sigma} \Pr\{\mathcal{B}(|A(u,r)|,p) > \delta\} \leq 13mr\frac{p^{1/3}}{\delta^{1/3}} \tag{2.20}$$

where we have used Lemma 15. Combining eqn (2.19) and eqn (2.20), we get:

$$\mathbf{E}\left[R\right] \leq 4\delta n^{\alpha} + 13mr\frac{p^{1/3}}{\delta^{1/3}} \tag{2.21}$$

Theorem 20 *For $P = n^{\alpha}$, $\mathbf{E}\left[R\right] = O(n^{3/4+\alpha/4})$.*

Proof Let $\delta = n^{\epsilon}$ and consider r to be a constant. Rewriting eqn (2.21) in terms of ϵ, we get:

$$\mathbf{E}\left[R\right] \leq 4n^{\alpha+\epsilon} + \Theta\left(n^{1-\epsilon/3}\right)$$

Minimizing the right hand side with respect to ϵ, we get $\mathbf{E}\left[R\right] = O(n^{3/4+\alpha/4})$. ∎

2.9 Discussion

Recall that one of the motivations for analyzing the efficiency of SFCs as tools for parallel domain decomposition was to evaluate the merits of their continued use in large parallel systems. As shown in the above sections, the total number of points requiring remote accesses grows as $O\left(n^{3/4+\alpha/4}\right)$ for nearest neighbor, κ-nearest neighbor and spherical region queries alike. Intuitively, one would expect that when a computational domain is partitioned across fewer processors, the locality preserving property would significantly increase thereby considerably reducing the communication overhead due to typical neighborhood or region queries such as those analyzed in this chapter. By the same reasoning, communication overhead is expected to overwhelm computation if the number of processors is increased sufficiently. One of the main conclusions from the above analysis indicates that communication overhead does not vary significantly as the number of processors is increased. As $P \to 1$ ($\alpha \to 0$), the average communication overhead already starts as $O(n^{3/4})$. As the number of processors P is increased, the rate at which total remote accesses grow scales only as $\sqrt[4]{P}$. These results favor the use of large parallel systems.

On the other hand, the total number of remote accesses grows sublinearly with the problem size n, as long as the number of processors used is sublinear in n. This implies that the computational complexity is greater than the communication complexity. Once the points within a query region are found, there is typically $O(1)$ computation per point. Even assuming that there are only a constant number of points in each query region, the total computational complexity grows as $\Theta(n)$. Thus, the ratio of total computation cost to total communication cost is given by $\Theta\left(n^{(1-\alpha)/4}\right)$. This ratio increases with increasing n, thus improving the situation as n increases. A critical issue for a parallel algorithm to be practically useful is the ability to limit the communication overhead in relation to the computational costs. As the ratio of computational complexity to communication complexity is an increasing function of n, a lower percentage of time spent in communication can be achieved for a fixed number of processors by increasing n. These analyzes demonstrate that SFCs are useful for domain decomposition on large parallel systems.

CHAPTER 3. REVISITING A DISTRIBUTION INDEPENDENT ALGORITHM FOR FAST MULTIPOLE METHOD

A number of problems in a wide variety of research areas such as astrophysics, electromagnetics and fluid dynamics can be modeled after what is called the n-body problem. Given n bodies that mutually interact with each other through a $1/R$ potential, where R is the distance between the interacting bodies, the n-body problem is to compute the net force on each body due to all the others.

3.1 Background and Literature Review

Historically, the n-body problem was first encountered in the context of simulations for the study of physical phenomena such as astrophysical structure formation and quickly rose to challenge the limits of available computational resources. In such simulations, the size of the problem as defined by the number of interacting particles is enormous. Any realistic simulation requires several millions, often billions, of such particles that interact with each other through their mutual forces of interaction. Clearly, a direct solution would involve computing the force on each of the n bodies due to the remaining bodies - a computation that scales as $\Theta(n^2)$ assuming that the cost of computing each pair-wise *particle-particle* interaction is $\Theta(1)$. Clearly, such a quadratic runtime is prohibitive when n is large.

One of the earliest works that resulted from the need to reduce the number of computations is the Barnes-Hut algorithm [Barnes and Hut (1986)] and is based on an earlier work [Appel (1985)]. The algorithm reduces the number of operations by computing *particle-cluster* interactions, rather than direct pair-wise particle-particle interactions, when the interacting particles are sufficiently far from each other. It exploits the fact that the net force on each particle can be viewed as the sum of forces due to nearby particles (*nearfield interactions*) and those from particles that are far away (*farfield interactions*) for a suitably chosen parameter that decides the degree of nearness. Since there are $O(1)$ number of particles in the nearfield of each particle, these interactions are computed directly. To compute the remaining farfield interactions, particles that are sufficiently distant are organized into clusters and their contributions to the net effect on any faraway particle are computed using an aggregate effect due to the clusters which are assumed to be located at their respective centers of mass. Algorithms that exploit such particle-cluster interactions to reduce the number of computations are based on hierarchical tree data structures that organize multidimensional points using a recursive decomposition of the space containing them. Such a tree is called a *quadtree* in two dimensions, *octree* in three dimensions and hyperoctree in higher dimensions. In this chapter, we will simply refer to them as octrees irrespective of the dimensionality, d, of the domain space. The dimensionality will be clear from the context. An octree is constructed in the following manner: consider a hypercube enclosing the given n multidimensional points. This domain is recursively bisected in a manner that is similar, though not identical, to a SFC decomposition. The domain enclosing all the points forms the root of the octree. This is subdivided into 2^d sub-regions of equal size by bisecting along each dimension. Each of these regions that contain at least one point is represented as a child of the root node. The same procedure is recursively applied to each child of the root node terminating when a sub-region contains at most one point. The resulting tree is called a region octree to reflect the fact that each node of the tree corresponds to a non-empty sub-domain. An example is shown in Fig. 3.1. In practice, the recursive subdivision is stopped when a predetermined level of resolution is reached, or when the number of points in a sub-region falls below a pre-established constant threshold. Clearly, the number of leaves in an octree built in the above manner on n particles is $\Theta(n)$. Note that each node of an octree represents a unique cluster of particles enclosed within a unique hypercubic region of space which will be referred to as the *cell* corresponding to the node.

Algorithms based on particle-cluster interactions proceed by carrying out the following procedure for each particle: the octree is traversed starting from the root cell. Consider a cell

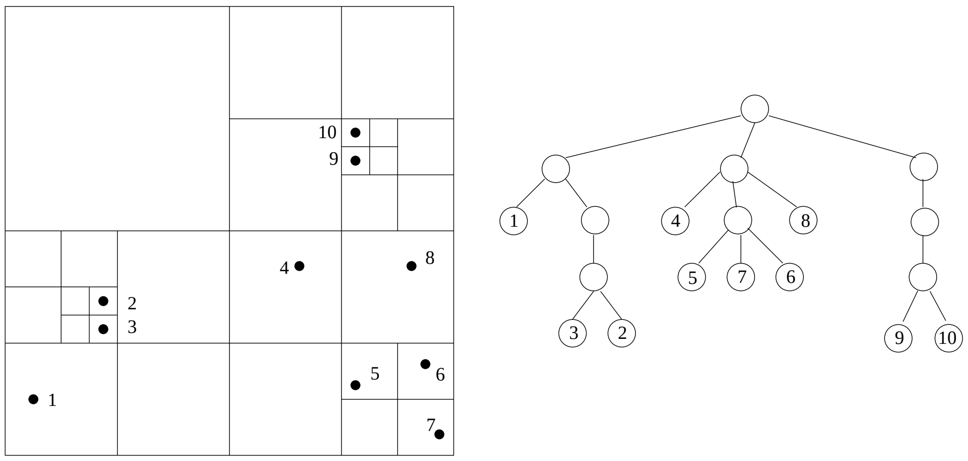


Figure 3.1 A quadtree built on a set of 10 points in two dimensions.

C encountered at some point during the traversal. If C is sufficiently far away from the particle as measured by the ratio of its side length to its distance to the particle, the potential due to the cell on the particle is directly computed. Otherwise, each of C's child cells are evaluated recursively in the same manner. Finally, the contribution to the net force on the particle from nearby particles is computed directly. For more details on these pioneering works, the reader is referred to the original sources [Appel (1985); Barnes and Hut (1986)].

The effect of a cluster of particles, denoted by B_i, on another particle at a distant point in another cluster A can be represented by a truncated infinite series called the *multipole expansion*, $\phi_p(B_i)$, where p is the order of the truncated approximation. To approximately compute the farfield force on particles in A due to particles in B_i, $\phi_p(B_i)$ is evaluated at each particle location in A. Therefore, using particle-cluster interactions, the effect on particles in A due to k distant clusters $B_1, B_2, \cdots, B_k$ is proportional to $|A|(|B_1| + |B_2| + \cdots |B_k|)$.

The fast multipole method (FMM), pioneered in [Zhao (1987); Greengard (1988); Anderson (1992)], is based on an additional level of refinement to the above approach which extends the particle-cluster interactions to *cluster-cluster* interactions. In this method, a second truncated infinite series, called the *local expansion,* $\psi_p(B_i)$, centered at the centroid of A can be efficiently computed from $\phi_p(B_i)$. The local expansion reflects the effect of distant clusters on the particles in A. The cluster-cluster interaction approach proceeds by first transforming $\phi_p(B_i)$ into $\psi_p(B_i)$

for each cluster B_i which is then used to compute $\psi_p(A) = \sum_{i=1}^{k} \psi_p(B_i)$. Subsequently, the net farfield effect at each particle in A is computed using $\psi_p(A)$, thereby making the number of operations proportional to $(|A| + |B_1| + |B_2| + \cdots |B_k|)$.

FMM is a robust mathematical method with guaranteed error bounds that computes cell-cell interactions directly. Each such cell is associated with a multipole expansion and a local expansion, as indicated earlier. For more details, see [Zhao (1987); Greengard (1988)]. The number of operations required to compute the multipole expansion due to a cluster of particles is proportional to the number of particles in the cluster [Greengard (1988)]. As mentioned in the last section, the number of points in the clusters represented by the leaves of an octree is $\Theta(1)$ and the number of leaves is $\Theta(n)$. The first step of the FMM algorithm is to compute the multipole expansions due to each leaf cell directly from the particles in them. Thus, this step requires $O(n)$ time.

Since the sizes of clusters represented by the internal nodes become progressively larger closer to the root, directly computing the multipole expansions at the internal nodes is expensive. Instead, the multipole expansions at the internal nodes are computed by a bottom-up traversal of the octree in which the multipole expansions of the children are aggregated to form the multipole expansion at the parent in time proportional to the order, p, of the expansion. This phase of the FMM algorithm is called the *upward accumulation*. Therefore, the total time for computing the multipole expansions at the internal nodes (upward accumulation) is proportional to the number of nodes in the octree.

The local expansion at a cell C is obtained by appropriately combining the multipole expansions of the cells that converge at every point in cell C. This phase is called *translation*. In Figure 3.2, the local expansion at cell C should include the effect of all particles outside region R. However, the effect of all particles outside region R' are included in the local expansion of C's parent C'. When considering cell C, we only compute the effect of particles that are outside region R but within region R'. We call this the *partial local expansion* at cell C. The set of cells that lie in the region inside R' but outside region R is called the *interaction list* of cell C. The partial local expansion due to the interaction list is added to the local expansion at cell

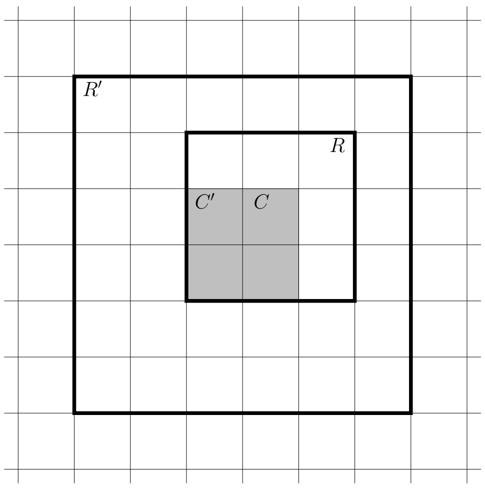


Figure 3.2 Illustration of partial local expansion calculation. Cells inside R' but outside R are required for C's partial local expansion.

C' to compute the local expansion at cell C. Once the partial local expansions are calculated for each node in the tree, the local expansions can be calculated in a top-down traversal, in a reversal of the procedure used for computing multipole expansions. This top-down traversal of the octree is called the *downward accumulation*. After the downward accumulation phase is completed, the contribution from the nearfield particles are computed directly at each leaf. For further details on FMM, the reader is referred to the original sources [Greengard and Rokhlin (1987); Greengard (1988)]. Thus, the FMM algorithm consists of the following five tree operations:

Step I : Computing the multipole expansion ϕ_p at each leaf.

Step II : Computing the multipole expansion ϕ_p at each internal node using an upward accumulation.

Step III : Computing the partial local expansions at each node in the tree from their respective interactions lists.

Step IV : Computing the local expansion ψ_p at each node using a downward accumulation.

Step V : Computing the nearfield interactions at each leaf directly and adding them to the farfield interactions (local expansion, ψ_p at the leaf).

Whereas the run-times of step I and V depend on the number of leaves ($\Theta(n)$), step II through step IV require time proportional to the size of the tree. Thus, it is clear that the run-time of the FMM algorithm depends on the size of the octree. On the other hand, note that in Fig. 3.1, the length of the chains in the octree can be made arbitrarily long by bringing the particles (represented by the corresponding leaves at the end of the chain) closer and closer because more recursive bisections will be required to separate the points when they are closer. This indicates that the size of an octree is dependent on the distribution of the points which in turn implies that the run-time of a FMM algorithm based on an octree is itself dependent on the *distribution* of the points rather than the *number* of points, n. For an elegant exposition on the ramifications of this distribution dependence on the run-time of FMM, the reader is referred to [Aluru (1996)].

A modified spatial data structure, called the *compressed octree*, which yields a distribution independent run-time of the FMM algorithm was first proposed in [Aluru et al. (1998)]. The fundamental insight that forms the basis of a compressed octree data structure is the following: though multiple nodes on a chain represent different regions of space (see Fig. 3.1), they all contain the same points. Thus, in any application where points are associated with elements of interest (particles, grid cells, finite elements etc.) and region subdivision is for purposes of convenience or enabling faster algorithms, different nodes on a chain essentially contain the same information. As such, no information is lost if the chains are compressed. However, such a compressed node should still encapsulate the fact that it represents multiple regions of space unlike the nodes that are not compressed. To encapsulate the spatial information otherwise lost in the compression, two cells are stored in each node v of a compressed octree, *large cell of* v and *small cell of* v, denoted by $L(v)$ and $S(v)$, respectively. The large cell is defined as the largest cell that encloses all the points the node represents. Likewise, the small cell is the smallest cell that encloses all the points that the node represents. If a node is not a result of compression of a chain, then the large cell and the small cell of that node are the same; otherwise, they are different. A compressed octree is simply an octree with each of its chains compressed into a single node. Thus, each node in a compressed octree is either a leaf or has at

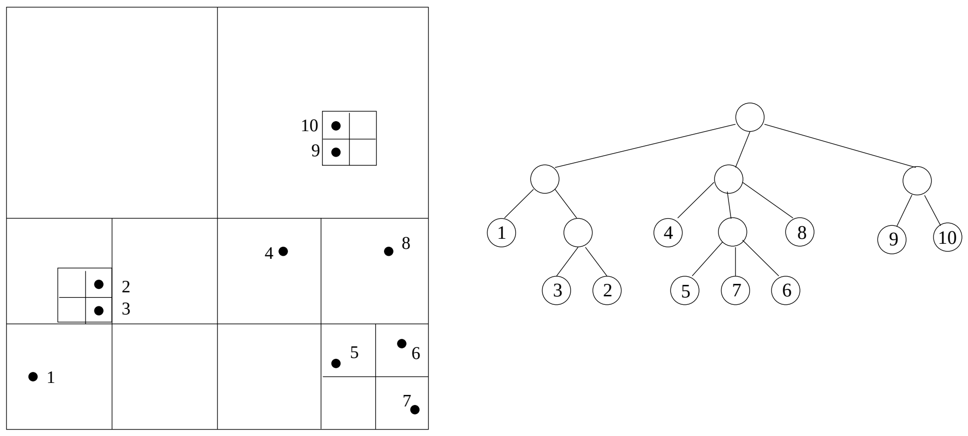


Figure 3.3 A compressed quadtree corresponding to the quadtree of Fig. 3.1.

least two children. This ensures that the size of the resulting compressed octree is $O(n)$ and is independent of the spatial distribution of the points. The compressed octree corresponding to the octree in Fig. 3.1 is shown in Fig. 3.3.

It has been shown that compressed octrees on n input points can be built in $O(n \log n)$ optimal time. Using some careful analysis, it can also be shown that a FMM algorithm based on compressed octrees results in $O(n)$ runtime per iteration [Aluru et al. (1998); Sevilgen et al. (2000)]. In parallel, the entire FMM computation, including parallel domain decomposition, can be effectively achieved by using parallel compressed octrees which can be built in $O\left(\frac{n}{P} \log \frac{n}{P}\right)$ with $O\left(\frac{n}{P} + k\right)$ storage where P and k denote the number of processors and the highest level of spatial resolution in the octree, respectively [Hariharan et al. (2002)].

3.2 Space Filling Curves and Compressed Octrees

One of the primary advantages of a parallel compressed octree based FMM is that the same scheme is used for both domain decomposition as well as the FMM solution. This is in stark contrast to earlier parallelization approaches in which domain decomposition was affected using techniques such as orthogonal recursive bisection [Berger and Bokhari (1987); Salmon (1990)] or SFC [Warren and Salmon (1993)] while the numerical computations of the application itself relied on octrees. Apart from the complexities in reconciling the two different schemes, both in methodology and software complexity, there is no way to prove one method minimizes communication to bring non-local data essential to build the other. These problems can be overcome by directly using the same scheme for both needs. The similarity in decomposition used by octrees and SFCs can be exploited to unify ideas on SFC based parallel domain decomposition and parallel octrees. Given a cell, the corresponding *cell space* is defined as the set of all cells obtained by dividing the root cell into cells of the same size. Octree nodes represent cells which belong to cell spaces at different resolutions. SFCs order the cell space at a particular level of resolution. Thus, octrees can be viewed as multiple SFCs at various resolutions. For convenience, suppose we pick the Z-SFC. When drawing an octree, we can draw the children on a node in the order in which Z-SFC visits the subcells represented by the children. By doing so, we ensure that the order of octree nodes at the same level is the same as the SFC order of the corresponding subcells. The membership test is easy: A cell at a particular resolution is present in an octree if the cell is not empty (i.e., it has one or more points). The same concepts apply for a compressed octree except that a cell is present as a small cell at a node in the compressed octree only if the cell is not empty and none of its immediate subcells contains all the points in the cell. Although the Z-SFC is used here, it is possible to use any SFC order while applying the same concepts.

Apart from establishing a linearization of cells at a particular resolution, it is also beneficial to define a linearization that cuts across multiple levels. Note that given any two cells, they are either disjoint or one is contained in the other. This observation can be exploited to establish a total order on the cells of an octree [Sevilgen et al. (2000)]: given two cells, if one is contained in the other, the subcell is taken to precede the supercell; if they are disjoint, they are ordered according to the order of the immediate subcells of the smallest supercell enclosing them. A nice property that follows from these rules is the resulting linearization of all cells in an octree (or compressed octree) is identical to its postorder traversal.

As each node in an octree is uniquely described by the corresponding cell, it can be represented by its index in SFC linearization, as shown in Fig. 3.4. However, ambiguity may arise when distinguishing indices of cells at different levels of resolution. For example, it is

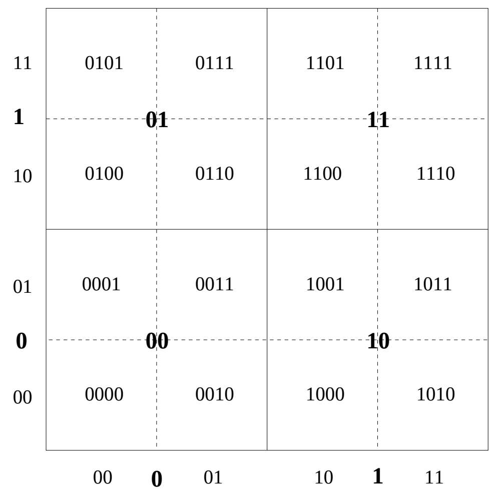


Figure 3.4 Bit interleaving scheme for a hierarchy of cells.

not possible to distinguish between 00 (cell of length $\frac{D}{2}$ with coordinates (0,0)), and 0000 (cell of length $\frac{D}{4}$ with coordinates (00,00)), when both are stored in, say, standard 32-bit integer variables. A simple mechanism suggested to overcome this is to prepend the bit representation of an index with a '1' bit [Warren and Salmon (1993)]. With this, the root cell is 1, the cells with Z-SFC indices 00 and 0000 are now 100 and 10000, respectively.

The process of assigning indices to cells can be viewed hierarchically. A cell at resolution i can be described using i-bit integer coordinates. The first $i-1$ of these bits are the same as the coordinates of its immediate supercell. Thus, the index of a cell can be obtained by taking the least significant bit of each of its coordinates, concatenating them into a d-bit string, and appending this to the index of its immediate supercell. Note that bit representations of cells is meaningful under the assumption that the resolution of an octree is small and fixed, which is valid in practice. The advantage of such bit representation is that it allows primitive operations on cells using fast bit operations [Hariharan et al. (2002)]. Since an octree can be viewed as a linearization of cells across multiple cell spaces at different resolutions, the communication overhead related results of Chapter 2 are valid for each such cell space.

The next section discusses the runtime and scaling results of a compressed octree based FMM algorithm on two different parallel architectures. The FMM computations in this implementation use accelerated Cartesian expansions to represent the farfield interactions.

3.3 Performance Results of a Parallel ACE-based FMM

Our renewed interest in parallel FMM stems from a very fundamental result [Shanker and Huang (2006)] that generalizes the FMM algorithm to all potentials of the form R^ν where $\nu \in \mathbb{R}$. Additionally, it shows that reformulating the FMM in terms of what the authors call *accelerated Cartesian expansions (ACE)*, the upward and downward accumulation phases of the FMM can be rendered independent of ν. The dependence on ν is restricted to the translation phase. This is a remarkable result since applications in which particles interact with each other through more than one potential, for example in molecular dynamics where atoms interact with each other through both the electromagnetic potential ($\nu = -1$) as well as the Lennard-Jones potential ($\nu = -6, -12$), can now fall within the purview of a common FMM based algorithm. In practice, an ACE-based FMM implementation for the different types of potential need only modify the code fragment for the translation phase since the remaining phases are insensitive to the different types of potentials being simulated.

A parallel ACE-based FMM algorithm was implemented and studied on two different parallel architectures - a Linux cluster of dual Intel Xeon processors with 2GB RAM running at 3.06 GHz at each node and the more specialized BlueGene/L platform, each node of which contains two dual-core PPC440 CPUs running at 700Mhz, with 512MB of RAM per node. The performance results are reported below.

Table 3.1 Runtimes (in seconds) on a cluster platform for various stages of the FMM algorithm using 4 million points.

No. of Procs.	Tree Const.	Build Interaction Lists	Upward Pass	Translation Phase	Downward Pass	Total	Speedup
1	19.70	1342.63	0.47	80.46	0.49	1443.75	1.00
2	9.69	685.37	0.22	41.11	0.28	736.67	1.96
4	4.02	395.75	0.11	21.0	0.14	421.02	3.43
8	1.97	224.78	0.07	11.21	0.08	238.11	6.06
16	1.02	126.69	0.03	5.57	0.04	133.35	10.83
32	0.56	64.29	0.03	2.99	0.02	72.89	19.81
64	0.31	41.88	0.03	1.54	0.02	43.78	32.98

In Table 3.1, runtimes of the algorithm during its various stages on the Linux cluster is reported for a problem of size, $n = 4,000,000$. Fig. 3.5(a) plots the total runtime versus number of processors for a fixed problem size. Clearly, the runtime is dominated by the time to build the interaction lists of each node in the tree. This observation will discussed at the end of this section.

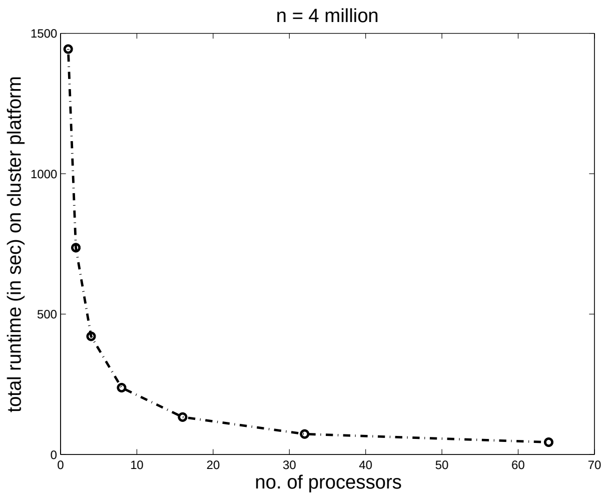
Table 3.2 Runtimes (in seconds) on 32 processors of a cluster platform for various stages of the FMM algorithm with varying problem sizes.

Problem Size	Tree Const.	Build Interaction Lists	Upward Pass	Translation Phase	Downward Pass	Total
32,768	0.10	11.99	0.01	0.33	0.01	12.44
262,144	1.80	83.21	0.02	5.60	0.02	90.65
2,048,383	8.20	792.56	0.02	39.37	0.02	840.17

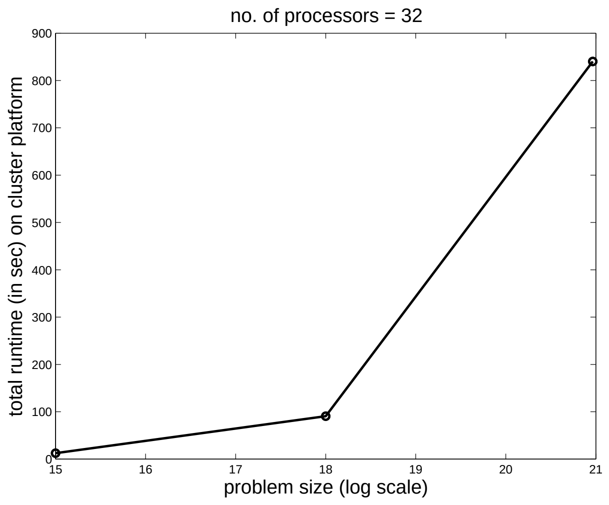
The last column in Table 3.1 lists the speedup when compared to running the code on one processor of the cluster. Ideally, speedup is defined with respect to the best sequential algorithm. In the absence of such a sequential code, we use the above strategy to derive some quantitative conclusions about the performance of the above implementation on the cluster platform. The speedup versus number of processors is plotted in Fig. 3.5(c). The observed speedup (solid line) is compared to the the speedup factor modeled according to Amdahl's law:

$$\text{Speedup, } S(P) = \frac{P}{1 + (P-1)f}$$

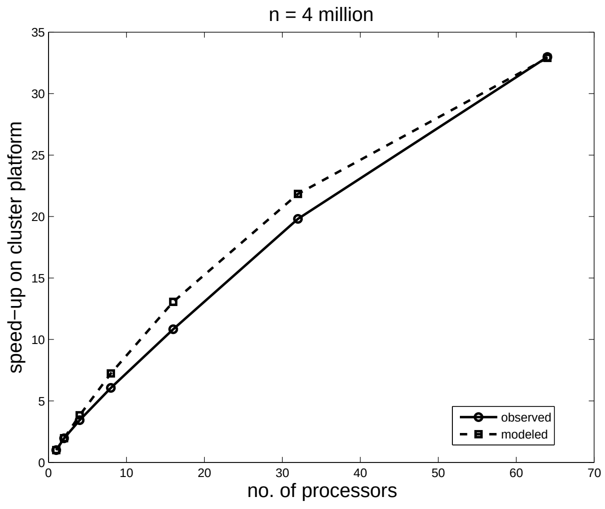
where P is the number of processors and f is the fraction of the computation that cannot be divided into concurrent parts, that is, the sequential section of the overall parallel algorithm. The observed behavior is close to the modeled speedup with $f \sim 1.5\%$ (dashed line) indicating that almost every step of the underlying sequential algorithm has been efficiently parallelized confirming theoretical runtime results in [Aluru et al. (1998)]. Fig. 3.5(d) plots the efficiency



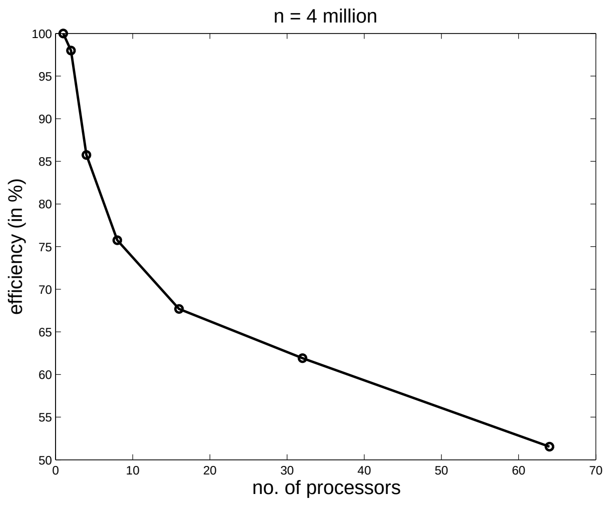
(a)



(b)



(c)



(d)

Figure 3.5 Performace on a Linux cluster: (a) Runtime *vs.* number of processors. (b) Runtime *vs.* problem size. (c) Speedup *vs.* number of processors. (d) Efficiency *vs.* number of processors.

of the implementation computed according to:

$$\text{Efficiency, } E(P) = \frac{S(P)}{P}\,\%$$

As can be seen, the efficiency drops close to 50% on 64 processors. The reason for the relatively rapid decrease in the efficiency can possibly be attributed to the interaction list building algorithm which, as implemented, generates a large number of parallel queries. A modification to the original interaction list building algorithm that addresses this possible performance bottleneck is discussed in the next section.

The scaling of the run-time with problem size for a fixed number of processors is shown in Table 3.2 and plotted in Fig. 3.5(b). Note that the runtime for $n = 2,048,383$ in Table 3.2 is larger than that for $n = 4,000,000$ in Table 3.1. This is because the computational domain of the points in Table 3.1 is smaller than that of Table 3.2. In addition, the number of points per leaf is about 16 in the former compared to about one point per leaf in the latter. This results in a much smaller tree for the first distribution and, hence, a smaller runtime.

Table 3.3 Runtimes (in seconds) on the BlueGene/L for various stages of the FMM algorithm using 4 million points.

No. of Procs.	Tree Const.	Build Interaction Lists	Upward Pass	Translation Phase	Downward Pass	Total
32	1.96	453.66	0.04	27.71	0.02	483.39
128	0.56	155.85	0.02	7.71	0.02	164.16
512	0.18	61.04	0.01	2.23	0.01	63.47

Table 3.3 and Table 3.4 show similar runtime and scaling results of the software on the BlueGene/L platform. They are plotted in Fig. 3.6(a) and Fig. 3.6(b), respectively.

Finally, Table 3.5 shows the convergence of the numerical results as a function of the precision parameter p that governs the order of approximation used in the accelerated Cartesian expansion formulation of the FMM.

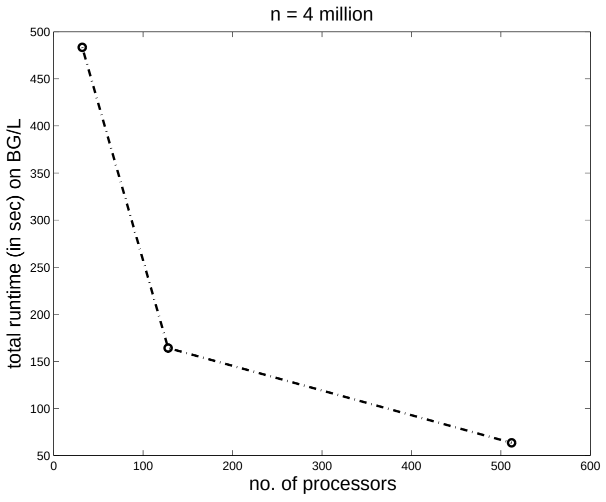
The runtime results in Table 3.1 and Table 3.3 presented above indicate that the current

Table 3.4 Runtimes (in seconds) on 128 processors of a BlueGene/L for various stages of the FMM algorithm with varying problem sizes.

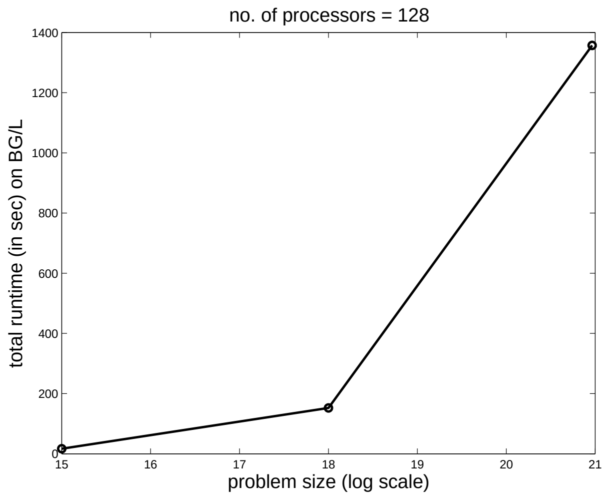
Problem Size	Tree Const.	Build Interaction Lists	Upward Pass	Translation Phase	Downward Pass	Total
32,768	0.02	15.47	0.01	1.05	0.02	16.57
262,144	0.07	144.47	0.02	7.60	0.03	152.19
2,048,383	0.34	1295.35	0.03	61.24	0.05	1357.01

Table 3.5 Convergence of numerical results.

Precision	Error (%)
0	4.689165310434e-003
1	3.338517604346e-003
2	1.417123499997e-004
3	1.296137525199e-004
4	5.282658787624e-006
5	5.279494129853e-006
12	1.314129857198e-011



(a)



(b)

Figure 3.6 Performance on BlueGene/L: (a) Runtime *vs.* number of processors. (b) Problem size *vs.* number of processors.

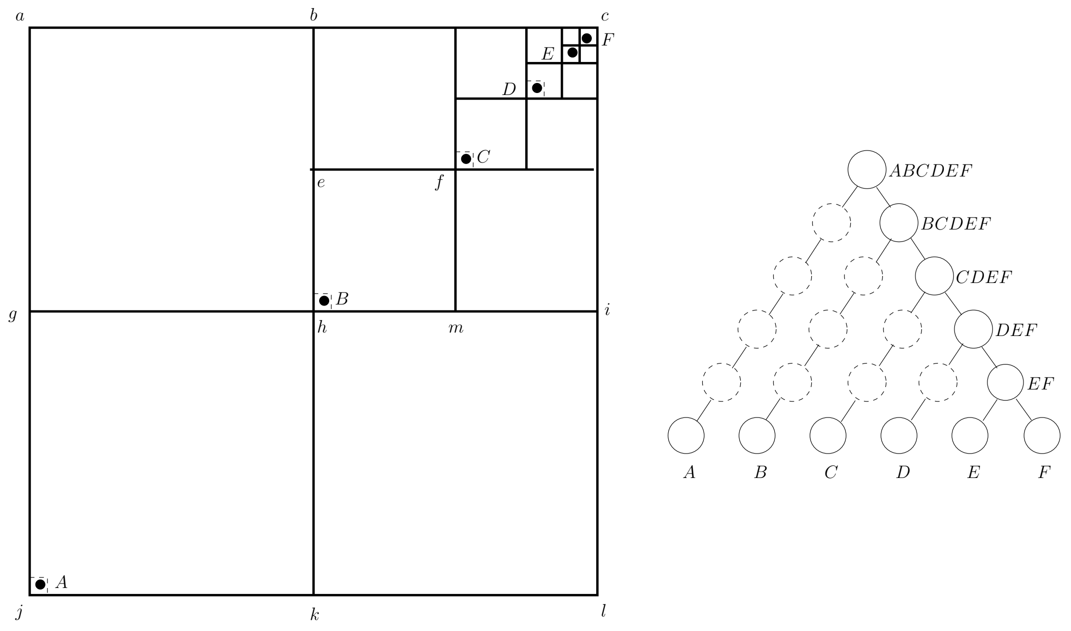


Figure 3.7 A compressed quadtree for an exponential distribution.

parallel FMM implementation needs improvement to the interaction list building phase. During this phase, each node in the tree generates a significant number of query cells [Hariharan et al. (2002)], a great number of which may not be available locally. Therefore, the number of query cells generated affect both the number of communications as well as as the size of each message. An improvement to this phase of the algorithm should therefore attempt to minimize the number of query cells while not compromising the numerical accuracy of the final result. A modified algorithm for this purpose is the subject of the next section.

3.4 A Modified Interaction List Building Algorithm

In principle, when a compressed octree is built according to the algorithm described in [Hariharan et al. (2002)], the sizes of the leaf cells can vary. For example, in the example of Fig. 3.7, the region of space corresponding to the leaf nodes are the boxes with solid lines that enclose each of the points. In practice, however, the leaf cell sizes are kept equal and are generally determined by the underlying physical constraints that govern the problem [Hariharan et al. (2002)]. This is shown by the boxes with dashed lines in Fig. 3.7. Effectively, this implies that the number of levels of the tree to be built on the n input points is known beforehand. This has an important ramification on the performance a parallel FMM when compressed octrees with equal sized leaf cells are used.

As stated earlier, during the translation phase that precedes the downward accumulation phase of a FMM transforms, multipole expansions of cells that belong to the interaction list, denoted by $\mathcal{I}_C$, of a cell C, are converted into local expansions that converge at every point in C. The algorithm for building the interaction list in [Hariharan et al. (2002)] is as follows: for any cell C in the tree:

1. Compute the parent cell of C by computing the smallest cell that encloses C and its adjacent cell in the postorder traversal order of the compressed octree.
2. Obtain the cells in the nearfield of the parent cell using bit arithmetic operations.
3. Compute subcells of the cells obtained in Step 2 such that size of each subcell is equal to the size of C. Discard those subcells that are in the nearfield of C.
4. Partition these subcells into two arrays for convenience : an array Local for subcells that are local to the processor and an array Remote for subcells that are remote to the processor. The Remote array will thus have all the nodes whose information will be fetched at the translation phase of each iteration.

Consider the leaf cell $efmh$ containing B in Fig. 3.7. The above algorithm will divide each of the cells $abhg$, $ghkj$ and $hilk$ into four immediate subcells and finally yield the set $\mathcal{I}_B = \{A\}$ leaving the points C, D, E and F to interact with B through nearfield interactions in the last step of the FMM. As mentioned earlier, in practice the leaf cell containing the point B is the one shown by the square with dashed lines. Consequently, step 3 of the above algorithm divides the cells $abhg$, $ghkj$ and $hilk$ into cells of the same same size as the one with dashed lines bounding B. This yields a far larger number of cells to be queried than is required in step 4. For example, if the side-length of the root box in Fig. 3.7 is assumed to be 32 and that of the smallest cell as 1, then the number of cells to be queried to ascertain $\mathcal{I}_B$ is 768 cells (each with the size of the dashed leaf cell containing B) when that number should only be 12 (each with the size of the leaf cell $efmh$ containing B). The difference is exponentially more in three dimensions. The unnecessarily larger number of query cells stems from the fact that the small and large cells of leaves in a compressed octree are assumed to be of equal size though in practice this may not be the case. In essence, the leaves of a compressed octree are not necessarily uncompressed. The modification to the original algorithm is, therefore, immediate - modify step 3 above to read:

Step 3 : *Compute subcells of the cells obtained in Step 2 such that size of each subcell is equal to the size of the large cell of C. Discard those subcells that are in the nearfield of C.*

The following table shows a runtime comparison of the two interaction list building algorithms for a non-uniform distribution of $n = 4$ million particles on varying number of processors:

Table 3.6 Time (in seconds) to build interaction lists using the original and modified algorithm for 4 million points with varying number of processors.

Algorithm	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$	$P = 64$
Original	685.37	395.75	224.78	126.69	64.29	41.88
Modified	31.36	17.14	8.79	5.11	3.27	1.96

In the present implementation, the above performance gain comes at a cost of some numerical accuracy. For example, when compared to Table 3.5, convergence is slower by about an order of magnitude.

3.5 Discussion

In this chapter, an ACE-based parallel FMM was studied for its runtime and scaling behavior on two different parallel architectures - a cluster platform and the more specialized BlueGene/L. Whereas the speedup obtained for a fixed problem size on varying number of processors indicate that the parallel algorithm is very efficient and scalable, performance bottleneck was detected in the interaction list building phase of the FMM algorithm. A modification to the original algorithm significantly reduces this bottleneck. However, the resulting runtime saving affects the accuracy of the numerical results by about an order of magnitude. Balancing the two is a subject of ongoing and future work.

CHAPTER 4. CLUSTERING ANALYSIS OF ATOM PROBE TOMOGRAPHY DATA

Last five years have seen rapid advances in the design of atom probe microscopes. The state-of-the-art LEAP microscope is at the cutting edge of 3DAP tomography. At the present rate of data acquisition, the volume of data generated by a routine atom probe experiment varies from 10^7 to 10^8 atoms, their identities and positions in three dimensions. With continued innovation of experimental techniques such as ultra-fast pulsed laser assistance to evaporation and wider field-of-view detectors, these data sizes are anticipated to grow by several orders of magnitude in the very near future. Visualization tools that render 3D APT data provide striking insight into the nature of material microstructure. However, the enormous amounts of point data that are generated in a single experiment and the fine scale at which subtle features such as atomic clustering occur, make it impossible to discern the nature of nanoscale ordering present in a sample through mere visual inspection. Additionally, to investigate the development of nanostructure, e.g. the evolution of solute clustering in an alloy subject to thermal treatment, requires the experimental acquisition and analysis of APT data sets at numerous distinct times during the treatment. This results in a time series of very large data sets that require analysis to determine structural properties and track their evolution across the changing treatment conditions. Such enormous amounts of data quickly become prohibitively large for the application of conventional serial techniques. The sheer volume of data itself makes parallel approaches a natural choice. In summary, the need for the development of efficient parallel algorithms that can quickly analyze 3DAP data while providing accurate nanostructural descriptions of a given material is considered to be the next challenge in APT research [Larson and Kelly (2006)].

4.1 Background and Literature Review

Different techniques have been developed for specific nanoscale analyses of APT data in the past few years including maximum separation, nearest neighbor, Fourier-based, correlation and contingency table techniques [Vurpillot et al. (2004); Moody et al. (2007); Miller and Kenik (2004); Miller et al. (1996)]. In nearest neighbor analyses the immediate vicinity surrounding each atom of a specific species is searched within some prescribed radius, or until some maximum number of neighboring atoms have been identified. Clustering information is inferred from statistical analyses of atomic identities reported back from the neighborhood searches. The maximum separation method is based on the observation that the mean characteristic distance separating solute atoms in a cluster is less than that separating solute atoms occurring in the matrix in general. A critical distance parameter, d_{max}, is chosen such that any two solute atoms separated by a distance less than d_{max} are considered to be clustered together. A difficulty inherent in both of these methods is that the pre-established critical distance parameter that is required is often chosen arbitrarily with little analytical justification. Direct approaches like Fourier based methods have several advantages since they are independent of size, distribution and identities of the atoms but they scale as $O(n^3 \log n)$ in three dimensions. More recently, grid based autocorrelation and contingency table techniques have been proposed [Vurpillot et al. (2004); Moody et al. (2007)]. In these approaches, the spatial domain enclosing the atoms is decomposed into an array of non-overlapping cells. Each cell is then associated with a number(s) and subsequent inferences are drawn based on computations on the number(s) associated with the cells instead of the original data set of atoms.

In general, any agglomerative clustering algorithm [Murtagh (1983)] based on a Euclidean metric can potentially reveal clustering information in the data but the $O(n^2)$ work of such algorithms is prohibitively large for the targeted data sizes, whether the algorithms are serial or parallel. Parallel hierarchical algorithms for various models of computing have been reported over the years. Algorithms with $O(n \log n)$ runtime using n SIMD hypercubic processors was reported in [Li and Fang (1989)]. An algorithm with $O(n \log n)$ runtime using $n/\log n$ processors on Parallel Random Access Machine (PRAM), butterfly and tree models is presented in [Olson (1995)]. Other parallel clustering algorithms geared towards different computing or networking models are available; for example, see [Li (1990); Rajasekaran (2005)]. Provably efficient, scalable parallel clustering algorithms for distributed memory machines, however, continue to remain an open area of research. In this chapter, we will develop a radically new approach to the problem. It is based on the concept of autocorrelation and results in a simple, yet very powerful serial algorithm for the analysis of very large 3DAP data sets in optimal time and space. Additionally, it can be parallelized effortlessly.

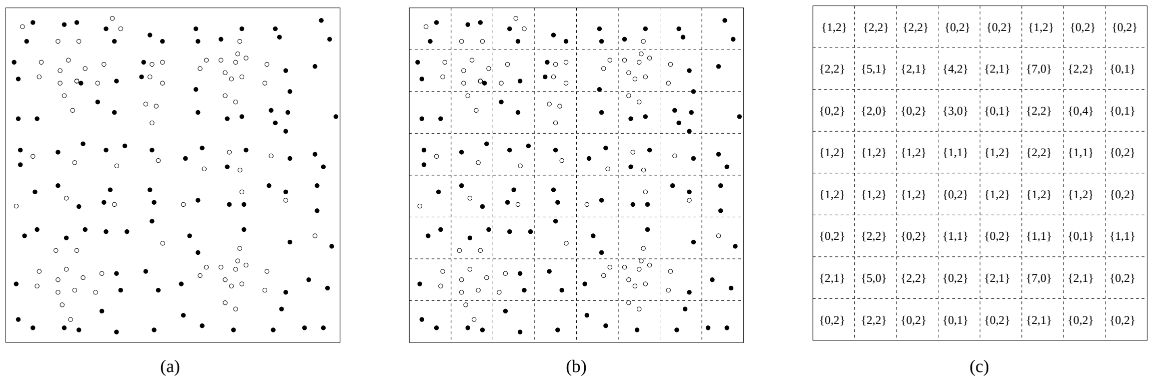
4.2 Autocorrelation

Autocorrelation functions have multiple definitions depending on the nature of the problem to which they are applied. For example, the definition of autocorrelation for stationary processes differs from that for ergodic processes. The exact analytical expression of an autocorrelation function is, therefore, problem dependent. The definition adopted in [Vurpillot et al. (2004)] is:

Definition 20 *The autocorrelation function $g(\mathbf{x}')$ of a function $f(\mathbf{x})$ is defined by:*

$$g(\mathbf{x}') = \iiint f(\mathbf{x}) \cdot f(\mathbf{x} + \mathbf{x}')\, dV \tag{4.1}$$

and bears close resemblance to the one adopted in this thesis. All autocorrelation functions including the one defined above exhibit several interesting properties such as: (a) the autocorrelation of a periodic function is periodic with the same period (b) an autocorrelation function reaches its peak at the origin and (c) the autocorrelation of the sum of two completely uncorrelated functions is the sum of the autocorrelations of each function separately. From the standpoint of computational complexity, the choice of the function $f(\mathbf{x})$ that is autocorrelated is critical. The requirement for $f(\mathbf{x})$ to accurately describe the physical property of interest must be balanced with the ability to compute it fast. In [Vurpillot et al. (2004)], $f(\mathbf{x})$ is a structure function represented by a sum of Dirac delta operators carried out over all atom indices from 0 to $n-1$. This leads to an algorithm that scales as $O(n^2)$ which is prohibitive



{1,2}	{2,2}	{2,2}	{0,2}	{0,2}	{1,2}	{0,2}	{0,2}
{2,2}	{5,1}	{2,1}	{4,2}	{2,1}	{7,0}	{2,2}	{0,1}
{0,2}	{2,0}	{0,2}	{3,0}	{0,1}	{2,2}	{0,4}	{0,1}
{1,2}	{1,2}	{1,2}	{1,1}	{1,2}	{2,2}	{1,1}	{0,2}
{1,2}	{1,2}	{1,2}	{0,2}	{1,2}	{1,2}	{1,2}	{0,2}
{0,2}	{2,2}	{0,2}	{1,1}	{0,2}	{1,1}	{0,1}	{1,1}
{2,1}	{5,0}	{2,2}	{0,2}	{2,1}	{7,0}	{2,1}	{0,2}
{0,2}	{2,2}	{0,2}	{0,1}	{0,2}	{2,1}	{0,2}	{0,2}

Figure 4.1 (a) A square domain containing two types of atoms. (b) Computational domain decomposed into an array of cells. The cells are numbered in the same way as matrices; i.e., the cell at top left corner is labeled (0,0), rows are numbered consecutively from top to bottom, and columns are numbered consecutively from left to right. (c) The set in each cell denotes the number of atoms of each type that is contained within it.

for 3DAP data sizes. Instead, we use the following discrete autocorrelation function to derive an $O(n)$ work algorithm:

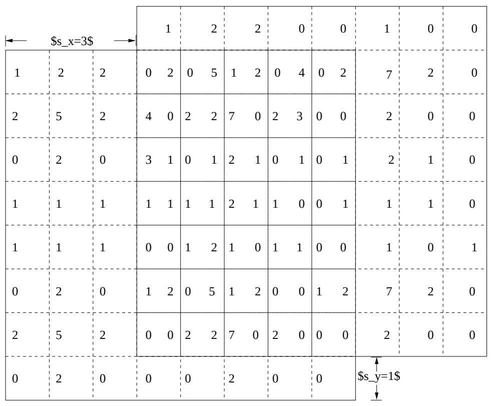
Definition 21 *Given a matrix M of dimensions $n_x \times n_y \times n_z$, the three dimensional autocorrelation of M at the integer values s_x, s_y and s_z is defined as:*

$$\sigma(s_x, s_y, s_z) = \frac{1}{\mathcal{N}} \sum_{i=s_x}^{n_x-1} \sum_{j=s_y}^{n_y-1} \sum_{k=s_z}^{n_z-1} M[i-s_x, j-s_y, k-s_z] \cdot M[i,j,k] \tag{4.2}$$

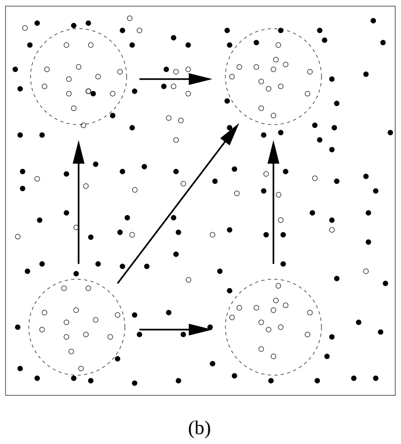
where $0 \le s_x < n_x$, $0 \le s_y < n_y$, $0 \le s_z < n_z$ and $\mathcal{N}$ is a normalization factor that will be described shortly.

Two (or one) dimensional autocorrelations of M are similarly evaluated from the above expression by setting one (or two) of s_x, s_y and s_z equal to zero.

The motivation for adopting the above definition is very intuitive and can be easily understood using a two dimensional example. Consider a square domain that contains n atoms of two different types as is shown in Fig. 4.1(a) using filled and unfilled circles. Even in this small example, cluster identification through visual inspection of the atoms in the domain is difficult, if at all possible. In order to reveal the embedded clustering features of atoms, consider Fig.



(a)



(b)

Figure 4.2 (a) Autocorrelation of the matrix in Fig. 4.1(c) for $s_x = 3$ and $s_y = 1$. Note that one of the atom types is suppressed and only the cell population of atoms represented by the unfilled circles is shown. (b) Clusters present in Fig. 4.1 are revealed through the autocorrelation function computation.

4.1(b) which shows the same domain decomposed into a 8×8 array of non-overlapping cells. Let this array of cells be denoted by M. Each cell is indexed by its column index, $i \in [0, 1, \cdots, 7]$ and its row index, $j \in [0, 1, \cdots, 7]$. The matrix M can be made to capture the number density profile of the entire original domain by associating a set of two integers with each cell in M as shown in Fig. 4.1(c). Each entry m_{ij} of M is a set of two integers $\{n_0, n_1\}_{ij}$ which reflects the number of atoms of the respective type that are enclosed in the volume represented by the cell (i, j). This matrix immediately reveals the fact that cells (1,1), (1,5), (6,1) and (6,5) are particularly rich in atoms denoted by unfilled circles. The autocorrelation function defined in eqn (4.2) simply corresponds to the sum of the product of all those entries of M that overlap when it is overlayed upon itself after one of the copies is shifted by s_x, s_y and s_z units along the x, y and z dimensions, respectively. For example, the autocorrelation function for $s_x = 3$ and $s_y = 1$ is the sum of the product of the elements in the overlapping cells shown in Fig. 4.2(a) which uses the same matrix as in the example of Fig. 4.1(c) with the population of one of the two types of atoms suppressed for clarity. Clearly, the autocorrelation value will exhibit local maxima for those values of s_x and s_y at which highly populated cells overlap. In Fig. 4.2(a), local maxima will be encountered for $(s_x = 4, s_y = 0)$, $(s_x = 0, s_y = 5)$ and $(s_x = 4, s_y = 5)$ shown by the directions of the arrows in Fig. 4.2(b). From the locations of the maxima, the relative displacements of the clusters can be ascertained as shown in Fig. 4.2(b). On the other hand, if the shifts are such that regions of high density overlap with regions of extremely low densities, the resulting autocorrelation value will be close to a local minima. Similar conclusions can be arrived at for the second type of atoms if the addition and multiplication operations in eqn (4.2) are carried out independently but simultaneously on both elements of the sets associated with each cell. In fact, spatial correlations between different types of atoms can be found when the matrix corresponding to one type is correlated with that of another. These correlations, called *cross-correlations*, reveal the relative spatial densities of different types of atoms. Thus, clusters of both types of atoms present in the domain and also the spatial correlations between clusters of different types of atoms can be revealed using the above definition of correlation. If the density of atoms is uniform for any species, its corresponding correlation function will show little variation. Further spatial properties of clusters of different (or same) types of atoms present in a sample can be gathered using the properties of correlation functions stated earlier. The choice of the cell size is important not only for the runtime of the algorithm, but also for the resolution of the desired clustering features. Both these points will be elucidated in the following section.

4.3 Parallel Algorithm

Let n denote the total number of atoms in the data set and $p = p_x \cdot p_y \cdot p_z$ denote the total number of processors. Further, let $L_x \times L_y \times L_z$ denote the dimensions of the smallest cuboid that encloses the n points where $L_x = n_x \cdot l$, $L_y = n_y \cdot l$ and $L_z = n_z \cdot l$ and l is the sidelength of the smallest cell. The total number of input points n is to be distinguished from N which is defined as $n_x \times n_y \times n_z$. Let ν be the total number of atomic species present in the input data set. The number of different types of atoms in a sample is always much smaller than the size of the data set. Hence, $\nu << n$. The algorithm for parallel cluster correlation proceeds as

Algorithm 1 Parallel Cluster Correlation Algorithm, parCCA (n, l, ν)

```
1: $p$ processors are assigned a $p_x \times p_y \times p_z$ Cartesian topology.
2: Allocate $n/p$ points to each processor. Compute spatial extent of data.
3: Compute the global cell indices of all local points based on their coordinates.
4: Each processor sends global cell index information to destination processors.
5: Each processor creates the local portion of the global 3D array of cells based on its Cartesian
   rank and local domain boundaries.
6: Population vector for each local cell is computed based on global cell index information.
7: Receive population vectors from processors whose Cartesian rank differs by one along at
   least one of the dimensions.
8: Compute local autocorrelation function.
9: Aggregate partial autocorrelation.
```

shown in algorithm 1.

A $p_x \times p_y \times p_z$ three dimensional Cartesian topology is imposed on the p processors and each processor is assigned n/p points. The smallest and largest values of the x, y and z coordinates of the local points are communicated to all processors using an *MPI_Allreduce* communication primitive which allows the computation of the three dimensional spatial extents, L_x, L_y and L_z, of the data. These tasks are accomplished in steps 1-2. Based on its Cartesian rank and the spatial extents of the computational domain, each processor computes its local domain boundaries. An *MPI_Allgather* operation communicates the boundaries to all processors. This information is stored in each processor. Based on this boundary information and the coordinates of the data point, each processor determines the destination processor to which each local point belongs and the indices in the global array of cells to which the point belongs. This is accomplished in step 3. It is not necessary to send the data points themselves to the destination processors. Only the information about their atomic type and the global cell indices of the cell they belong to. This information is communicated using a call to *MPI_Alltoall* in step 4. Based on the information received from all other processors, each processor assigns a *population vector*, $v(i, j, k)$, consisting of $(u < \nu)$ components to each cell (i, j, k) it is responsible for in step 5. Each component n^u_{ijk} of $v(i, j, k)$ maintains a count of the number of atoms of type u enclosed in the volume of the domain represented by the cell (i, j, k). Once the population vectors are computed in step 6, the original point set can be discarded. The autocorrelation function in eqn (4.2) is defined for matrices of integers. In this algorithm, M is a matrix of population vectors for which we define:

$$\sigma(s_x, s_y, s_z) = \frac{1}{\mathcal{N}} \sum_{i=s_x}^{n_x-1} \sum_{j=s_y}^{n_y-1} \sum_{k=s_z}^{n_z-1} v^T(i,j,k) \cdot v(i-s_x, j-s_y, k-s_z) \tag{4.3}$$

where $v^T(i,j,k) = (n^0_{ijk}, n^1_{ijk}, \cdots, n^{\nu-1}_{ijk})$ is a row vector. The autocorrelation function in eqn (4.3) is merely a sum of products. As such, the part of the sum that is local to a processor can be computed independently if the lagged elements that reside in remote processors are available. Let s_{xm}, s_{ym} and s_{zm} denote the maximum shifts in each direction. Note that for every intermediate set of values for s_x, s_y and s_z, the autocorrelation function depends on values whose indices lag behind by the shift values. For local cells within the volume $[0, s_{xm}] \times [0, s_{ym}] \times [0, s_{zm}]$, the corresponding lagged elements reside in processors whose Cartesian rank differs by one along at least one of the dimensions. This volume of cells is requested and brought from the neighboring processors through one round of communication in step 7 before the autocorrelation function computation is begun in order to avoid multiple startup overheads of communication primitives. The additional amount of data received in this step by each interior processor $P(i,j,k)$ in the Cartesian topology from its seven Cartesian neighboring processors is listed in the following table:

$P_{i-1,j,k}$	$P_{i,j-1,k}$	$P_{i,j,k-1}$	$P_{i-1,j-1,k}$	$P_{i,j-1,k-1}$	$P_{i-1,j,k-1}$	$P_{i,j,k}$
$s_{xm}\frac{n_y}{p_y}\frac{n_z}{p_z}$	$s_{ym}\frac{n_x}{p_x}\frac{n_z}{p_z}$	$s_{zm}\frac{n_x}{p_x}\frac{n_y}{p_y}$	$s_{xm}s_{ym}\frac{n_z}{p_z}$	$s_{ym}s_{zm}\frac{n_x}{p_x}$	$s_{xm}s_{zm}\frac{n_y}{p_y}$	$s_{xm}s_{ym}s_{zm}$

Note that each element of the partial autocorrelation array computed in the previous step corresponds to a unique combination of mass and shifts. As such, the local partial autocorrelations can be aggregated without mixing the masses and shifts using *MPI_Allreduce* operations in the last step. As will be demonstrated shortly, the resulting m length vector contains all the autocorrelation information which can be processed using any plotting software to reveal the spatial correlations of the atomic species present in the sample of n atoms.

Runtime Analysis

The computation time for steps 1 through 3 is clearly $O\left(\frac{n}{p}\right)$. The spatial resolution of atom probe data is limited by that of the LEAP microscope. Thus, if the resolution of the microscope is denoted by λ, then inter-atomic distances that are smaller than λ cannot be resolved by it and hence will not be present in the resulting data. This automatically sets a lower bound on the side length l of a cell. This eliminates the need to find the smallest cell length, usually a factor of the smallest inter-atomic distance, which in turn would have required $O\left(\frac{n}{p}\log\frac{n}{p}\right)$ to compute. In general, l is chosen to be equal to $\kappa\lambda$ where $\kappa > 1$. If the density of atoms in any sub-volume of the material is always within a constant factor of the density in any other sub-volume, as is the case for almost all materials, then for $l = \kappa\lambda$, we have:

$$n_x \cdot n_y \cdot n_z = O(n) \tag{4.4}$$

In step 5, the local part of the $n_x \times n_y \times n_z$ array of cells is computed. The computation time for this step is proportional to the size of the local array and, hence, proportional to $\frac{n_x}{p_x} \cdot \frac{n_y}{p_y} \cdot \frac{n_z}{p_z} = O\left(\frac{n}{p}\right)$. Note that if the global array of cells of size $n_x \times n_y \times n_z$ were known *a priori*, each processor would have been allocated $O(n_x n_y n_z / p_x p_y p_z)$ sized portion of it. The global index information received in the previous step accomplishes precisely that. As such, computing the population vector for each local cell in step 6 simply requires updating the components of the population vector in each local cell to reflect the number of atoms of each type whose global index was received in the previous step. This takes constant time for each atom position. Thus, the computation time for step 6 is $O\left(\frac{n}{p}\right)$. Time required to compute the partial autocorrelations for a set of shifts $[0, s_{xm}] \times [0, s_{ym}] \times [0, s_{zm}]$ in step 8 is proportional to the overlap volume of the three dimensional matrices. If the total number of computations for shift values of (s_x, s_y, s_z) is denoted by $C(s_x, s_y, s_z)$, then:

$$C(s_x, s_y, s_z) = \nu \cdot \left(\frac{n_x}{p_x} - s_x\right) \cdot \left(\frac{n_y}{p_y} - s_y\right) \cdot \left(\frac{n_z}{p_z} - s_z\right) = O\left(\nu \frac{n_x n_y n_z}{p_x p_y p_z}\right) = O\left(\frac{n}{p}\right) \tag{4.5}$$

since $\nu = O(1)$. Thus, the total number of computations over all possible shifts is:

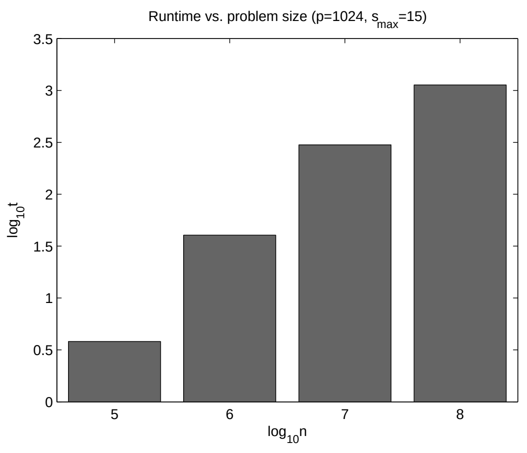
$$C = \sum_{s_x=1}^{s_{xm}} \sum_{s_y=1}^{s_{ym}} \sum_{s_z=1}^{s_{zm}} O\left(\frac{n}{p}\right) = O\left(s^3 \frac{n}{p}\right) \tag{4.6}$$

where $s = \max\{s_{xm}, s_{ym}, s_{zm}\}$. Cluster features in material samples almost never exhibit long range correlations. To be more precise, separation between clusters are almost always much smaller than the spatial extent of the sample material. Typically, for most samples, $s = O(1)$. As such, the computation time of step 8 is $O\left(\frac{n}{p}\right)$. Step 9 aggregates the local portions of the autocorrelation function using a *MPI_Allreduce* which takes $O\left(\frac{n}{p}\right)$ computation time. Combining the runtimes of all the steps, the total computation time for the algorithm is $O\left(\frac{n}{p}\right)$. Communication time is incurred from one *Allgather*, one *Alltoall*, two *AllReduce* and one *SendRecv*.

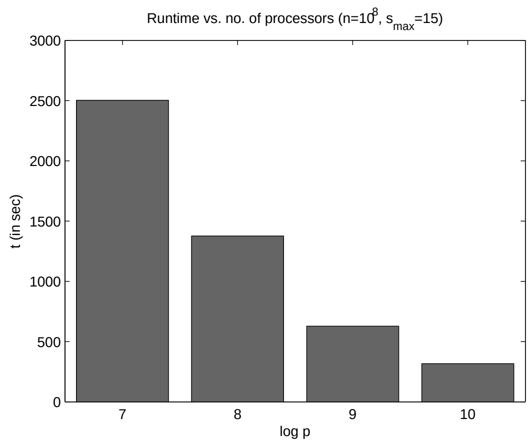
Memory Requirements and Load Balance

Each processor is initially assigned $\frac{n}{p}$ points in step 1. Once the global indices are stored, it is not required to store the points themselves. In general, depending on the distribution of points in the domain, it is possible that a processor may not own any point at all. However, in material samples the density of atoms in any sub-volume never differs from another sub-volume by more than $\Theta(1)$. This fact ensures that each processor receives $\Theta\left(\frac{n}{p}\right)$ information in step 4. The local array of cells encodes the number density profile of the spatial sub-domain owned by each processor. The size of this local array is $\frac{n_x}{p_x} \cdot \frac{n_y}{p_y} \cdot \frac{n_z}{p_z} = \Theta\left(\frac{n}{p}\right)$. All subsequent computation are carried out using this local submatrix. It can be easily seen from the above table that the extra storage required in step 7 is $\Theta\left(\frac{n}{p}\right)$. In step 6, the partial autocorrelation is stored in an integer array of length $\nu \cdot s_{xm} \cdot s_{ym} \cdot s_{zm} \leq \nu s^3 = O(1)$. Thus, the total storage requirement of this algorithm is $\Theta\left(\frac{n}{p}\right)$.

Steps 1-4 impose equal load on every processor. Load imbalance in step 5 may stem from highly non-uniform distribution of points in the domain. As mentioned earlier, this is not the case for APT data sets and as such the load in step 5 is $O(n/p)$. Since all operations after the computation of the array of population vectors in step 6 are carried out on the global array of cells distributed equally across the p processors, the load is $O(n_x n_y n_z / p_x p_y p_z)$=$O(n/p)$ on each processor. The only remaining source of load imbalance is the varying number of computations from one set of shifts (s_x, s_y, s_z) to another. Because length scale of spa-



(a)



(b)

Figure 4.3 (a) Runtimes with varying problem sizes. (b) Runtimes with varying number of processors.

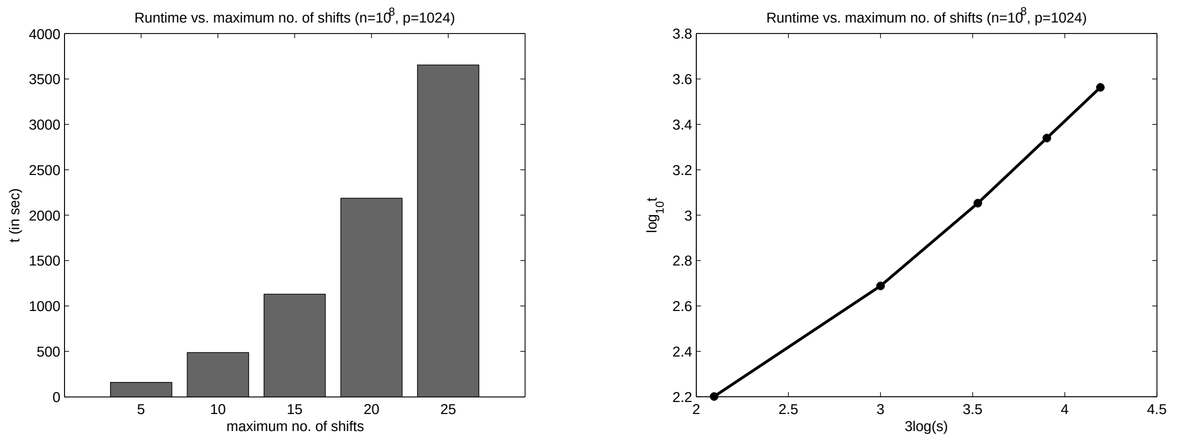
tial correlations in material samples of interest are much smaller than their spatial extents, $\max\{s_{xm}, s_{ym}, s_{zm}\} = s$ remains much smaller than the size of the local sub-domain. As such, $s^3 << n/p$ which restricts the imbalance only to the processors at the boundaries of the topology.

4.4 Results

The above algorithm was implemented on the BlueGene/L platform and used to track the evolution of atomic clusters in an aluminum alloy Al-1.9Zn-1.7Mg over three different aging conditions. The results of this implementation are reported in the following sections.

4.4.1 Run-time Results

The parameters against which we studied the runtime are the number of processors (p), size of the data set (n) and the maximum number of spatial autocorrelations (s). Fig. 4.4 (a) shows the runtime of the parallel algorithm on varying problem sizes when p and s are kept constant. As can be seen, a problem size of 10^8 (which is a typical size of current 3DAP data sets) can be analyzed in about 20 minutes. By extrapolating the above graph, it can be easily



(a) (b)

Figure 4.4 (a) Runtimes with varying number of processors. (b) Runtimes with varying number of shifts.

seen that problem sizes $\sim 10^9$ can be analyzed in about 3.5 hours on a full rack (1024 nodes) BlueGene/L. Analysis of time series 3DAP data, therefore, falls within a realistic time frame.

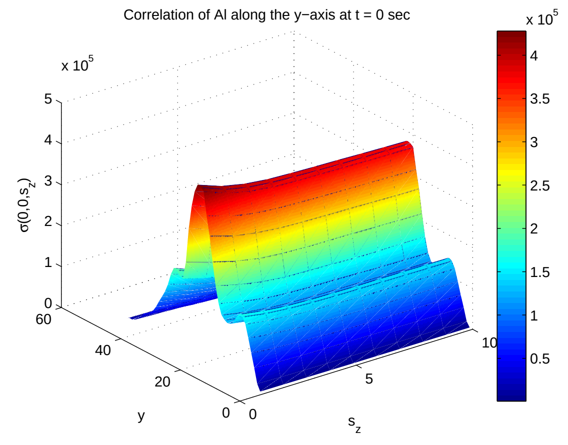
Fig. 4.4 (b) shows the runtimes of the algorithm on varying number of processors for a constant problem size and constant s. The plot clearly shows that the runtime of the algorithm is linear in p^{-1}. This implies that a 3DAP data set of $\sim 10^9$ atoms can be analyzed in about 1.75 hours on 2048 processors on the BlueGene/L. The runtime of the algorithm shows strongest dependence on the maximum number of shifts chosen for the autocorrelation function as can be seen Fig. 4.4.1 (a). For this plot, the maximum number of shifts was varied from 5 to 25 for a problem size of 10^8. As expected from eqn (4.6), the runtime scales as s^3 as shown in Fig. 4.4.1 (b). The above results clearly demonstrate the promise of this algorithm in handling large 3DAP data sets. In the next section, some key observations are made regarding the clustering properties of atoms in a material sample. These results are by no means exhaustive but only serve to demonstrate the ability of this algorithm to reveal both qualitative as well as quantitative nanoscale structural features from large 3DAP data.

4.4.2 Materials Science Results

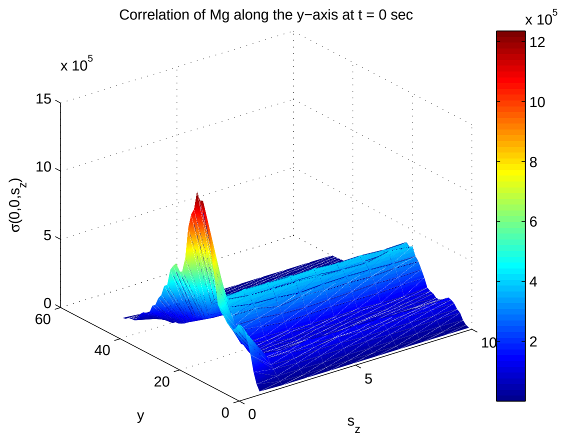
The experimental APT data used corresponds to three aging conditions of the aluminum alloy Al-1.9Zn-1.7Mg. Specimens were solution treated at 460C, water quenched at $\sim 10^{o}$C and heat treated at 150^{o} C for 0 seconds, 210 seconds and 3600 seconds, respectively. After heat treatment, the samples were electro-polished using 80% nitric acid solution and fine-polished with 2% perchloric acid and 2-butoxyethelene solution. Atom probe tomography was performed using the commercial local electrode atom probe.

Result of the autocorrelation based algorithm on three APT data sets under different aging conditions are shown in Fig. 4.5, Fig. 4.6 and Fig. 4.7. The figures reveal the spatial clustering properties of Mg, Al and Zn as the alloy is aged. The normalization factor $\mathcal{N}$ in eqn (4.3) is chosen to be the square of the average density of each atomic species. This choice was motivated by the fact that the concentration of the various species of atoms in the sample can vary by several orders of magnitude. As such, normalization of the respective autocorrelation by this value allows comparison of the three species on the same scale without introducing any artificial and unphysical meaning. The evolution of solute nanostructure, i.e. co-clustering of magnesium and zinc ions, with increasing aging time is clearly evident. Visual inspection can only ascertain the approximate presence of solute clustering. Hence, statistical analysis tools are required to either confirm a complete lack of nanostructure or to identify and characterize the presence of ordering indiscernible to the eye. After aging for 210 seconds, Fig. 4.6 reveals that clusters of the solute atoms have developed. The magnesium-zinc co-segregation continues to evolve with aging time as seen in Fig. 4.7 which corresponds to aging for 3600 sec. The y-profile of the correlation functions of Al,Mg and Zn at $s_z = 2$ is shown in the fourth plot of Fig. 4.5, Fig. 4.6 and Fig. 4.7. It shows how the different species in the alloy correlate under different aging conditions.

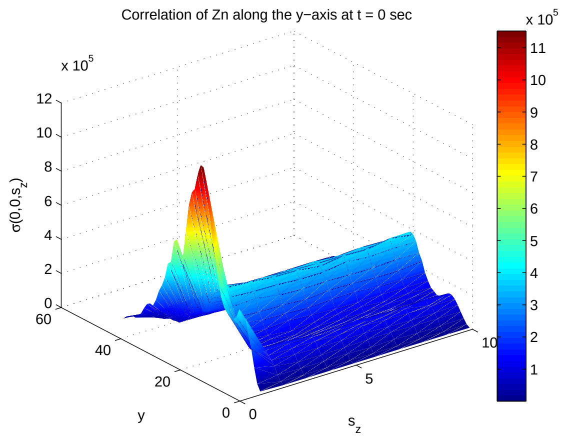
The autocorrelation function of Al along the y-axis reveals that the variation stems from two sources: a decrease in the direction of the geometrical edges due to the finite nature of the data set and oscillations due to gaps in the data set where regions containing artifacts were removed pre-analysis. Along the s_z-axis, the variation is expected to be almost completely



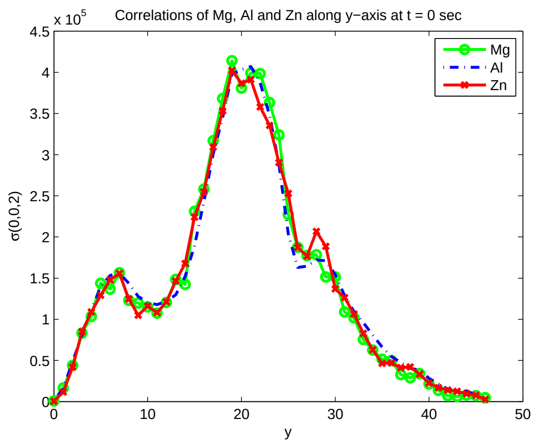
(a)



(b)



(c)



(d)

Figure 4.5 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 0$ sec. The y axis and the s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$.

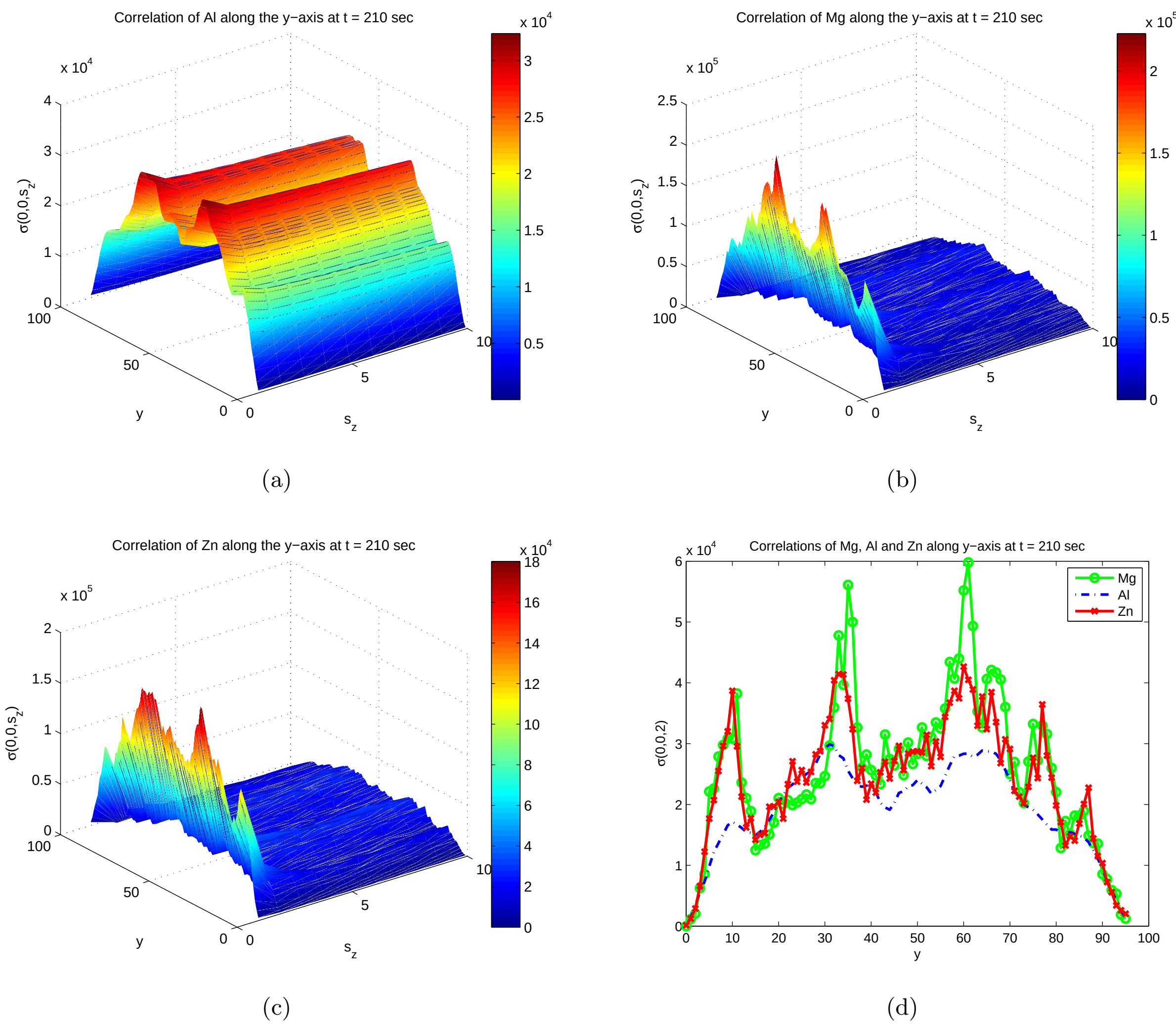
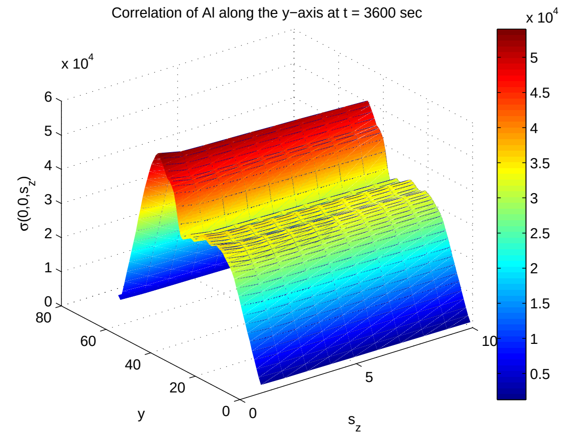
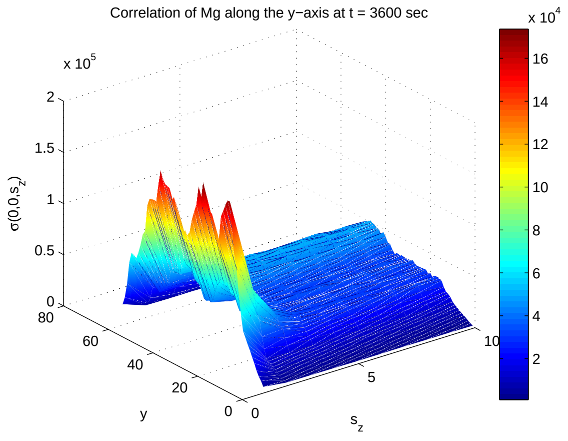


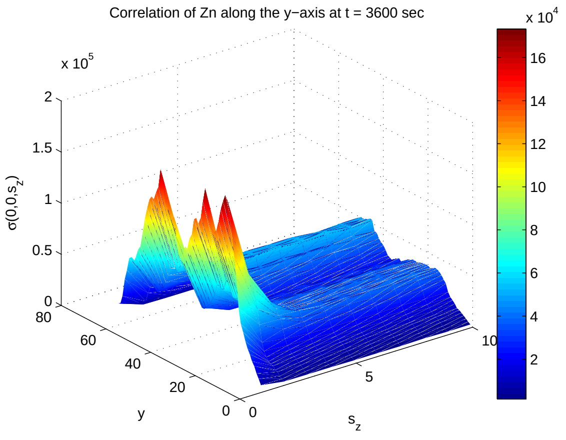
Figure 4.6 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 210$ sec. The y and the s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$.



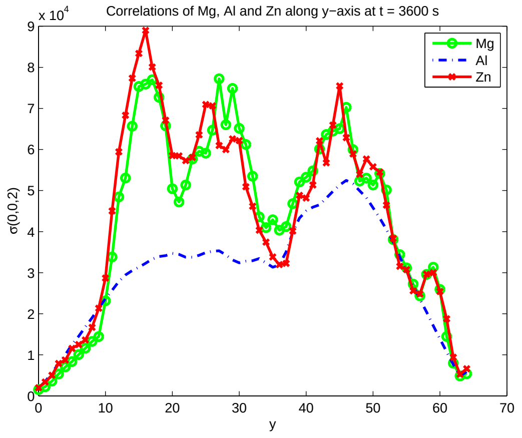
(a)



(b)



(c)



(d)

Figure 4.7 The autocorrelation coefficient map of the sample sliced along the xz-plane with shifts along the z-direction at time, $t = 0$ sec. (a) Al (b) Mg (c) Zn (d) This plot shows the profiles for Al, Mg and Zn at $s_z = 2$ superimposed on each other to demonstrate the relative spatial correlations of the three types of atom at $t = 3600$ sec. The y and s_z axes are shown in units of the smallest cell side length, $l \approx 1nm$.

uniform as can be seen in the Al plots. The Al plots can be used in comparison to the ones generated for Zn and Mg, providing the basis to discern correlations in the distribution of the solute atoms. In Fig. 4.5(d), all three plots are in good agreement implying that there is negligible segregation of the solute in the xz plane in the aged for t=0 secs data. However, in Fig. 4.6(d) and Fig. 4.7(d), there are a series of peaks in the Mg and Zn plots signifying pronounced departures from the Al autocorrelation profile. The peaks reveal regions in the xz plane where solute atoms of a particular ion type tend to cluster. The fact that the peaks in both Zn and Mg are highly correlated along the y-axis implies co-segregation in the distribution of the solute atoms along the xz plane. Similar autocorrelation analysis across remaining orthogonal directions, the corresponding plots for which have not been included in this manuscript for lack of space, facilitate further characterization of the clustering.

The spacing of the Zn and Mg peaks across the y-axis provide insight into the manner in which the solute clusters are distributed in the material. In addition, analysis of full-width at half maximum of the peaks of the correlation graphs can indicate a characteristic size of the clusters present. Comparing Fig. 4.6(d) and Fig. 4.7(d), there is a greater departure in the profiles of the solute atoms from that of Al in the latter. This implies that as aging time is increased from 210 secs to 3600 secs, Zn and Mg continue to segregate. It is also apparent that in particular Zn becomes increasingly more segregated over the extended aging interval.

In summary, the drastically reduced data that results from using the rather simple yet novel autocorrelation approach presented in this paper can reveal structural features at near atomic scales without the large computational overhead of existing methods.

4.5 Discussion

This chapter presented an autocorrelation based parallel algorithm specific to the analysis of nanostructures present in very large APT data. This is the first parallel algorithm for cluster identification in APT microscopy data. It is based on an $O(n)$ work autocorrelation formulation that should effectively scale to the largest APT data sets available both now and in the near future. The runtime of the parallel algorithm is $O(n/p)$ on p processors for data sets containing n atoms. The inherent simplicity of the algorithm renders it scalable to a very large numbers of processors. The memory signature of the algorithm is $O(n/p)$ thus making this approach suitable for data sets of the order of 10^9 that are expected to be available in the very near future. The algorithm was implemented on the BlueGene/L platform and applied to APT data from an aluminum alloy Al-1.9Zn-1.7Mg that was obtained using a LEAP microscope. Nano-features of different atomic species present in the sample were tracked at three different conditions of aging revealing their spatial correlations under the varying conditions.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

In the first part of this thesis, a formal analysis of the well known and much used technique of space filling curves for parallel domain decomposition was presented. The rigorously derived results bound the communication overhead generated during common spatial queries encountered in typical parallel scientific computing applications whose computational domain is partitioned across processors using SFCs. The dependence of the bounds on the problem size and the number of processor justify the use of SFCs for large parallel systems. Though these results constitute a first such analysis, they are not without their limitations in terms of their implications. The analysis exploits the common geometrical properties that are exhibited by the most popular SFCs in order to arrive at the results. Such an approach, however, cannot distinguish between the relative merits or demerits of using the individual SFCs themselves. For example, it does not address the question whether a Hilbert SFC is more efficient than the Z-SFC. A more detailed analysis in this direction remains an open problem. In addition, the analysis presented is based on a uniform distribution of points. Generalization to non-uniform distributions has the potential to yield even further insights into the performance of SFCs as parallel partitioning tools on large systems

The ACE-based implementation of FMM that is presented in the second part of the thesis uses SFCs to partition the domain. In addition, the benefits of the SFC representation of the underlying compressed octree data structure are exploited for efficient parallel searches and other computations that are frequently encountered within the overall FMM algorithm. The renewed interest in FMM originates from the work of Shanker and Huang (2006) that generalizes the FMM algorithm to all potentials of the form R^{ν} where $\nu \in \mathbb{R}$ using a formulation based on accelerated Cartesian coordinates. An efficient and scalable parallel implementation of this numerical algorithm is expected to have far reaching affect in areas of scientific computing that require numerous pair-wise force computations originating from various potentials of the form $R^{-\nu}\ \forall\, \nu \in \mathbb{R}$ simultaneously, for example, in molecular dynamics problems in which atoms interact through both the long range electromagnetic as well as the short-range Lennard-Jones potentials. Such an implementation is presented and its performance on two different parallel architectures is studied. Whereas the application scales well on both platforms, improvement was necessary in one of the phases of the underlying algorithm.

The final part of the thesis describes a novel algorithm for the analysis of extremely large data sets produced in advanced atom probe microscopy experiments. The demand for efficient algorithms for such analysis that can track the evolution of clusters of atoms of different species across enormous time series data sets of atoms is immediate. A simple, yet powerful, $O(n)$ work algorithm that enables such analyses was presented. This algorithm was parallelized and implemented on the BlueGene platform. Since the application requires $O(n/P)$ work on P processors and $\Theta(n/P)$ storage, it scales very well to very large data sets containing billions of points as are expected in the very near future. The resulting implementation was used to successfully track the evolution of clusters in an aluminum alloy Al-1.9Zn-1.7Mg over three different aging conditions. Ongoing and future work in this project include isolating artificial artifacts, if any, of using the autocorrelation based analysis that forms the basis of the algorithm. Though straightforward, an extension to include cross-correlation analysis of the clusters has the potential to reveal further spatial correlation information between clusters of the different atomic species that constitute a given materials sample.

The results included in this thesis constitute a body of work that spans both formal analyses as well as implementation and related performance results of parallel algorithms for two distinct areas of scientific research. A sincere attempt was made to present the material in as seamless a manner as possible.

APPENDIX

Nearest Neighbor Distance in $\mathcal{D}$-dimensions

Lemma 21 *There are at least* $\left(\frac{\sqrt{\mathcal{D}-1}}{\mathcal{D}} \cdot d\right)^{\mathcal{D}}$ *cells contained in* $C(u,d)$*, excluding* u*.*

Proof Consider a $\mathcal{D}$-dimensional coordinate system with origin at the center of cell u. Irrespective of the location of u, we can find a point $(x_1, x_2, \cdots, x_{\mathcal{D}})$ such that $|x_1| = |x_2| = \cdots = |x_{\mathcal{D}}| = \frac{d}{\sqrt{\mathcal{D}}}$ and $(x_1, x_2, \cdots, x_{\mathcal{D}})$ is on the surface of $C(u,d)$. Let $w = \lfloor \frac{d}{\sqrt{\mathcal{D}}} \rfloor$. Since w is integral, the cube of side w with one corner at u and another corner at $(x_1, x_2, \cdots, x_{\mathcal{D}})$ has $(w+1)^3$ grid cell centers inside, or on the boundary of the cube. But since $w+1 > \frac{d}{\sqrt{\mathcal{D}}} > d/(\mathcal{D}/\sqrt{\mathcal{D}-1})$, we have $(w+1)^{\mathcal{D}} > \left(\frac{\sqrt{\mathcal{D}-1}}{\mathcal{D}} \cdot d\right)^{\mathcal{D}}$, which proves the lemma. ∎

Lemma 22 $\sum_{u\in\sigma} |A(u,d)| = O\left(d \cdot m^{2-\frac{1}{\mathcal{D}}}\right)$

Proof Let S_l denote a hypercube at level l. For all the cells u located in a hypercuboid of dimensions $2d \times (2^{k-l} - 2d)_1 \times (2^{k-l} - 2d)_2 \times \cdots \times (2^{k-l} - 2d)_{\mathcal{D}-1}$ placed symmetrically at the center of S_l (see Fig. 2.11), $A(u,d)$ is S_l. Hence, for each cell u within the region in S_l formed from the fusion of the $\mathcal{D}$ hypercuboids, $|A(u,d)| = (2^{\mathcal{D}})^{k-l}$. The number of such cells in S_l is at most the total number of cells in the three hypercuboids, which is less than $\mathcal{D} \cdot 2d \cdot 2^{k-l} \cdot 2^{k-l} \cdots 2^{k-l} = 2\mathcal{D} \cdot d \cdot 2^{(k-l)\cdot(\mathcal{D}-1)}$. If σ_l denotes the set of cells $u \in \sigma$ for which $A(u,d) = (2^{\mathcal{D}})^{k-l}$, then $|\sigma_l| = (2^{\mathcal{D}})^l \cdot 2\mathcal{D} \cdot d \cdot 2^{(\mathcal{D}-1)\cdot(k-l)}$. Since for all $u \in \sigma_l$, $|A(u,d)| = (2^{\mathcal{D}})^{k-l}$, it follows that:

$$\sum_{u\in\sigma} |A(u,d)| = \sum_{l=0}^{k} \sum_{u\in\sigma_l} |A(u,d)| = \sum_{l=0}^{k} 2^{\mathcal{D}l} \cdot 2\mathcal{D} \cdot d \cdot 2^{(\mathcal{D}-1)\cdot(k-l)} \cdot 2^{\mathcal{D}(k-l)} = O\left(\mathcal{D} \cdot d \cdot m^{2-\frac{1}{\mathcal{D}}}\right)$$

which proves the lemma. ∎

Lemma 23

$$\sum_{1 \le d \le d_{max}} d(1-p)^{c_{\mathcal{D}} \cdot d^{\mathcal{D}}} = \Theta(1)\ , \quad \textit{where } c_{\mathcal{D}} = \left(\frac{\sqrt{\mathcal{D}-1}}{\mathcal{D}}\right)^{\mathcal{D}}$$

Proof Let

$$S = \sum_{1 \le d \le d_{max}} d(1-p)^{c_{\mathcal{D}} \cdot d^{\mathcal{D}}} = \sum_{1 \le d \le d_{max}} d e^{-p c_{\mathcal{D}} d^{\mathcal{D}}}$$

We know that $d^2 = x_1^2 + x_2^2 + \cdots + x_{\mathcal{D}}^2$ is an integer though d may not be.

Case I: Let $\mathcal{D} = 2a+1$. Then, $d^{\mathcal{D}-1} = d^{2a}$ is an integer greater than d. Thus:

$$S < \sum_{1 \le d \le d_{max}} d^{\mathcal{D}-1} e^{-p c_{\mathcal{D}} d^{\mathcal{D}}}$$

Let $r = d^{\mathcal{D}}$. Then:

$$S < \frac{1}{\mathcal{D}} \int_0^{\infty} e^{-p c_{\mathcal{D}} r} dr = \frac{1}{\mathcal{D}} \cdot \frac{1}{p c_{\mathcal{D}}} = \Theta(1)$$

Case II: Let $\mathcal{D} = 2a$. Then, $d^{\mathcal{D}}$ is an integer greater than d. Thus:

$$S < \sum_{1 \le d \le d_{max}} d^{\mathcal{D}} e^{-p c_{\mathcal{D}} d^{\mathcal{D}}} < \frac{1}{\mathcal{D}} \cdot \frac{1}{(p c_{\mathcal{D}})^{1+\frac{1}{\mathcal{D}}}} \int_0^{\infty} e^{-z} z^{\frac{1}{\mathcal{D}}} dz$$

which yields:

$$S < \frac{1}{\mathcal{D}} \cdot \frac{1}{(p c_{\mathcal{D}})^{1+\frac{1}{\mathcal{D}}}} \Gamma(1 + \frac{1}{\mathcal{D}}) = \Theta(1)$$

which proves the lemma. ∎

Theorem 24 $\mathbf{E}[Z] = \Theta(n^{\frac{\mathcal{D}-1}{\mathcal{D}}})$.

Proof Follows from eqn (2.4), Lemma 11, Lemma 21 and Lemma 22. ∎

BIBLIOGRAPHY

Abel, D. J. and Mark, D. M. (1990). A Comparative Analysis of Some Two-Dimensional Orderings. *Int'l J. Geographical Information Systmes*, 4(1):21–31.

Aluru, S. (1996). Greengard's N-body Algorithm is Not Order N. *SIAM Journal on Scientific Computing*, 17(3):773–776.

Aluru, S., Gustafson, J., Prabhu, G. M., and Sevilgen, F. E. (1998). Distribution Independent Hierarchical Algorithms for the N-Body Problem. *Journal of Supercomputing*.

Anderson, C. (1992). An Implementation of the Fast Multipole Method without Multipoles. *SIAM, J. Sci. Stat. Comput.*, 13(4):923–947.

Appel, A. W. (1985). An Efficient Program for Many-body Simulation. *SIAM Journal of Scientific and Statistical Computing*, 6:85–103.

Bank, R. E. and Jimack, P. K. (2001). A New Parallel Domain Decomposition Method for the Adaptive Finite Element Solution of Elliptic Partial Differential Equations. *Concurrency and Computation: Practice and Experience*, 13(5):327–350.

Barnes, J. and Hut, P. (1986). A Hierarchical $O(N \log N)$ Force-calculation Algorithm. *Nature*, 324(4):446–449.

Berger, M. J. and Bokhari, S. H. (1987). A Partitioning Strategy for Non-uniform Problems on Multiprocessors. *IEEE Trans. Comput.*, 36(5):570–580.

Dennis, J. M. (2003). Partitioning with Space-Filling Curves on the Cubed-Sphere. In *Intl. Parallel and Distributed Processing Symposium*, pages 269–269.

Eisenhauer, G. and Schwan, K. (1996). Design and Analysis of a Parallel Molecular Dynamics Application. *Journal of Parallel and Distributed Computing*, 35(1):76–90.

Gotsman, C. and Lindenbaum, M. (1996). On the Metric Properties of Discrete Space Filling Curves. *IEEE Transactions on Image Processing*, 5(5):794–797.

Greengard, L. and Rokhlin, V. (1987). A Fast Algorithm for Particle Simulations. *Journal of Computational Physics*, 73:325–348.

Greengard, L. F. (1988). *The Rapid Evaluation of Potential Fields in Particle Systems*. MIT Press.

Griebel, M. and Zumbusch, G. (2002). Hash Based Adaptive Parallel Multilevel Methods with Space Filling Curves. In *Proc. Neumann Institute for Computing Symposium*, pages 479–492.

Hariharan, B., Aluru, S., and Shanker, B. (2002). A Scalable Parallel Fast Multipole Method for Analysis of Scattering from Perfect Electrically Conducting Surfaces. In *Proc. Supercomputing*, page 42.

Hayashi, R. and Horiguchi, S. (2000). Efficiency of Dynamic Load Balancing Based on Permanent Cells for Parallel Molecular Dynamics Simulation. In *Int'l Parallel and Distributed Processing Symposium*, page 85.

Hilbert, D. (1891). Uber die stegie Abbildung einer Linie auf Flachenstuck. 38:459–460.

Jagadish, H. V. (1990). Linear Clustering of Objects with Multiple Attributes. In *Procs. ACM SIGMOD*, pages 332–342.

Kumar, S., Huang, C., Almasi, G., and Kale, L. V. (2006). Achieving Strong Scaling with NAMD on Blue Gene/L. In *Proc. Intl. Parallel & Distributed Processing Symposium*.

Larson, D. J. and Kelly, T. K. (2006). Nanoscale Analysis of Materials using a Local Electrode Atom Probe. *Microscopy and Analysis*, 20(3):59–62.

Li, X. (1990). Parallel Algorithms for Hierarchical Clustering and Clustering Validity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11):1088–1092.

Li, X. and Fang, Z. (1989). Parallel Clustering Algorithms. *Parallel Computing*, 11(3):275–290.

Miller, M. K., Cerezo, A., Hetherington, M. G., and Smith, G. D. W. (1996). *Atom Probe Field Ion Microscopy*. Monographs on the Physics and Chemistry of Materials. Oxford University Press.

Miller, M. K. and Kenik, E. A. (2004). Atom Probe Tomography: A Technique for Nanoscale Characterization. *Microscopy and Microanalysis*, 10:336 – 341.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Moody, M., Stephenson, L. T., Liddicoat, P. V., and Ringer, S. P. (2007). Contingency Table Techniques for Three Dimensional Atom Probe Tomography. *Microscopy Research and Technique*, 70.

Moon, B., Jagadish, H. V., Faloutsos, C., and Saltz, J. H. (2001). Analysis of the Clustering Properties of Hilbert Space-Flling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141.

Morton, G. (1966). A Computer Oriented Geodetic Database and a New Technique in File Sequencing. Technical report, IBM, Ottawa, Canada.

Murtagh, F. (1983). A Survey of Recent Advances in Hierarchical Clustering Algorithms. *The Computer Journal*, 26:354–359.

Nyland, L., Prins, J., Yun, R. H., Hermans, J., Kum, H., and Wang, L. (1997). Achieving Scalable Parallel Molecular Dynamics Using Dynamic Spatial Domain Decomposition Techniques. *Journal of Parallel and Distributed Computing*, 47(2):125–138.

Olson, C. F. (1995). Parallel Algorithms for Hierarchical Clustering. *Parallel Computing*, 21(8):1313 – 1325.

Parashar, M. and Browne, J. C. (1996). On Partitioning Dynamic Adaptive Grid Hierarchies. In *Hawaii Intl. Conf. On System Sciences*, pages 604–613.

Pilkington, J. and Baden, S. (1996). Dynamic Partitioning of Non-uniform Structured Workloads with Space Filling Curves. *IEEE Transaction on Parallel and Distributed Systems*, 7(3):288–300.

Plimpton, S. (1995). Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1–19.

Rajasekaran, S. (2005). Efficient Parallel Hierarchical Clustering Algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):497–502.

Sagan, H. (1994). *Space Filling Curves*. Springer-Verlag.

Salmon, J. K. (1990). *Parallel Hierarchical N-body Methods*. PhD dissertation, California Institute of Technology.

Sevilgen, Aluru, and Futamura (2000). A provably optimal, distribution-independent parallel fast multipole method. In *Intl. Parallel and Distributed Processing Symposium*, pages 77 – 84.

Shanker, B. and Huang, H. (2006). Accelerated Cartesian Expansions – an $O(N)$ Method for Computing of Potentials of the Form R^{ν} for all real ν. Technical Report MSU-ECE-2006-6, Michigan State University.

Steensland, J., Chandra, S., and Parashar, M. (2002). An Application-centric Characterization of Domain-based SFC Partitioners for Parallel SAMR. *IEEE Transactions on Parallel and Distributed Systems*, 13(12):1275–1289.

Vurpillot, F., Geuser, F. D., Costa, G. D., and Blavette, D. (2004). Application of Fourier Transform and Autocorrelation to Cluster Identification in the Three-dimensional Atom Probe. *Journal of Microscopy*, 216(3):234–240.

Warren, M. S. and Salmon, J. K. (1993). A Parallel Hashed Oct-Tree N-Body Algorithm. In *Proc. Supercomputing*, pages 12–21.

Zhao, F. (1987). An $O(N)$ Algorithm for Three-dimensional N-body Simulations. Technical Report TR 995, MIT, Cambridge, MA.

Zhuang, Y. and Sun, X. (2005). A Highly Parallel Algorithm for the Numerical Simulation of Unsteady Diffusion Processes. In *Int'l Parallel and Distributed Processing Symposium*, page 16a.