Improvements on Scientific System Analysis

Vladimir Grupchev
University of South Florida, vgrupcev@mail.usf.edu

Follow this and additional works at: http://scholarcommons.usf.edu/etd
Part of the Computer Engineering Commons

Scholar Commons Citation

This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Improvements on Scientific System Analysis

by

Vladimir Grupchev

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yi-Cheng Tu, Ph.D.
Sagar Pandit, Ph.D.
Yao Liu, Ph.D.
Michael Weng, Ph.D.
Wen-Xiu Ma, Ph.D.

Date of Approval:
December 13, 2014

Keywords: Molecular Simulations, Streaming, Push-Based, SDH, Big Data

Copyright © 2015, Vladimir Grupchev
Dedication

За Мајче...
Acknowledgments

Thank you Dr. Yi-Cheng Tu for your unconditional support and guidance throughout my Ph.D. studies. I appreciate your every second spent with me on my way to this milestone.

Thank you Dr. Sagar Pandit for your inputs on my research and the hours spent explaining how things work in the physics world.

Thank you Dr. Yao Liu, Dr. Wen-Xiu Ma and Dr. Michael Weng for being part of my committee and giving me invaluable input to better my dissertation.

Thank you Anand Kumar, Peyman Behzadnia, Mehrad Eslami and Ran Rui for making my Ph.D. studies more enjoyable.

Thank you Mama Lile, Tato Viktor, and my two brothers, Aleksandar and Dimitar for being what you are to me - my family.

Thank you Krecho Vladimir, Anatoli, Krecho Zoran, Meto, Zote, Klimo, Veljan... for not knowing what I do so far from home, yet still be my friends :).

Thank you Sashe, Mintie, Natasha and Mile, Irena and Scott, Elka and Mende for making this country my home away from home.

Very special thanks to Adi (who probably doesn’t care and wags his tail:) )

Oh and yes: I wouldn’t be here, writing all these "Thank you" notes if it weren’t for you, Angela Angeleska - no words can thank you enough...just my heart screams: Thank You!
Table of Contents

List of Tables iv
List of Figures v
Abstract vii

Chapter 1: Introduction 1
  1.1 Scientific Simulations 2
    1.1.1 Molecular Simulation (MS) 2
  1.2 Challenges Imposed by the Analysis of MS Data 3
    1.2.1 MS Data Analysis 3
      1.2.1.1 Spatial Distance Histogram Computation 5
      1.2.1.2 Motivation 5
    1.2.2 MS Data 8
      1.2.2.1 Data Access: Pull-based vs. Push-based 9
  1.3 Problem Statement 12
    1.3.1 Efficient Computation of SDH 12
    1.3.2 I/O Issues and High Throughput Data Analysis 13
  1.4 Our Approach 14
  1.5 Contribution and Roadmap of the Dissertation 15

Chapter 2: Related Work 19
  2.1 SDH Computation 19
    2.1.1 Force Computation Problem 19
    2.1.2 Data Stream Data Analysis 20
    2.1.3 K-nearest Neighbors Problem 21
    2.1.4 Space-Partitioning Trees Concept
      2.1.4.1 Quad-tree Approach 21
      2.1.4.2 R-trees 22
      2.1.4.3 kd-tree Approach 23
  2.2 Push-based System 26
    2.2.1 Streaming Data 26
Appendix A: Copyright Permissions

A.1 IEEE, License for Material in Chapter 3
List of Tables

Table 3.1 Notations used. 30
Table 3.2 The % of cell pairs that are expected to be resolved. 39
Table 3.3 Parameters with ranges. 43
Table 3.4 The three buckets affected by distance distribution. 52
Table 4.1 Popular analytical queries in MS. 77
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Calculating SDH.</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Examples of MS data.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>The difference in data access between push and pull system.</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Creating kd-tree from data space.</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>The density-map-based SDH algorithm.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Distance range of two resolvable cells overlap with three buckets.</td>
<td>41</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>The simulated hydrated bilayer system.</td>
<td>42</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Efficiency of ADM-SDH.</td>
<td>45</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Accuracy of ADM-SDH.</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Minimum and maximum distances between two cells.</td>
<td>51</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Pairs of cells that lead to total or partial error compensation.</td>
<td>54</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Distance compensation between two pairs of cells.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Approximation of $\frac{1}{x} - 1$ obtained in Matlab.</td>
<td>60</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Similar triangles ratio.</td>
<td>61</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Accuracy of SL-SDH using different bucket width.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>Accuracy of SL-SDH for synthetic data.</td>
<td>65</td>
</tr>
</tbody>
</table>
Figure 3.13  Efficiency of SL-SDH.  

Figure 3.14  Accuracy and performance tradeoffs of ADM-SDH and SL-SDH.  

Figure 4.1  MS modules structure.  

Figure 4.2  Computing minimum and maximum distance between two cells.  

Figure 4.3  Push-based system flow.  

Figure 4.4  Workload setup.  

Figure 4.5  Speed up over different levels of atom selection for 838K atoms.  

Figure 4.6  Speed up over different levels of atom selection for 2.5M atoms.  

Figure 4.7  Speed up over different levels of atom selections for 4.2M atoms.  

Figure 4.8  Speed up over different levels of atom selections for 8.8M atoms.  

Figure 4.9  Showing speedup for different workload intensity for 838K atoms.  

Figure 4.10  Showing speedup for different workload intensity for 2.5M atoms.  

Figure 4.11  Showing speedup for different workload intensity for 4.2M atoms.  

Figure 4.12  Showing speedup for different workload intensity for 8.8M atoms.  

Figure 4.13  Showing speedup for different data size over all workload.  

Figure 4.14  Showing speedup for different data size over all workload and groups.
Abstract

Thanks to the advancement of the modern computer simulation systems, many scientific applications generate, and require manipulation of large volumes of data. Scientific exploration substantially relies on effective and accurate data analysis. The shear size of the generated data, however, imposes big challenges in the process of analyzing the system. In this dissertation we propose novel techniques as well as using some known designs in a novel way in order to improve scientific data analysis.

We develop an efficient method to compute an analytical query called spatial distance histogram (SDH). Special heuristics are exploited to process SDH efficiently and accurately. We further develop a mathematical model to analyze the mechanism leading to errors. This gives rise to a new approximate algorithm with improved time/accuracy tradeoff.

Known MS analysis systems follow a pull-based design, where the executed queries mandate the data needed on their part. Such a design introduces redundant and high I/O traffic as well as cpu/data latency. To remedy such issues, we design and implement a push-based system, which uses a sequential scan-based I/O framework that pushes the loaded data to a number of pre-programmed queries.

The efficiency of the proposed system as well as the approximate SDH algorithms is backed by the results of extensive experiments on MS generated data.
Chapter 1: Introduction

People are always trying to learn more about nature and their surroundings. Early on this curiosity, people would observe nature and the processes happening around them. They would try to understand first how, then why these natural processes occur. They would also try to understand the inner mechanisms that drive such processes. By doing this kind of observation, people were able to interpret some of the mechanisms of nature. However, the observation has its limitation. Namely, the studied process is not repetitive by demand. People had to wait for its next occurrence in order to study it again. And when (or even if) it happened again, the circumstances surrounding it may be different, rendering the observation invalid. So, the next stage of acquiring knowledge was to try and recreate these natural processes in a controllable environment - giving the birth of the experiment. With experiments, people were able to study processes more closely and repeatedly, better understanding the mechanisms and making much more precise conclusions about why/how things were happening.

In the era of computers, people have more and more resources to use in order to recreate and study the natural processes. Nowadays, computers are so powerful, they can recreate very complicated natural systems and can simulate their inner workings in an almost carbon copy version of the original natural process.
1.1 Scientific Simulations

A computer simulation usually is a computer program that can be run on a single machine or a cluster of machines. It is used to simulate an abstract model of a studied system. In essence, a computer simulation represents the execution, or the running of that abstract model. Computer simulations are usually used when the regular, analytical method is too weak for the system on hand, or the system is too vast, that it would take enormous amount of time and human power to analyze the system. They are helpful in exploring new systems / technologies and gaining new knowledge of the insights of such systems. They can also be used to interpret possible future outcomes, like the projection of the stock market or weather patterns or to estimate the performance of a very complex system.

Scientists in many fields use computers to simulate and analyze very convoluted processes and systems. They build complex simulation systems and run extensive simulation experiments in order to achieve scientific discovery. These computer simulation systems are becoming more and more present across the board of science fields. The simulations generated on these systems are crucial and integral part of numerous engineering and scientific processes.

Examples of scientific fields that use such simulation systems include astrophysics (simulating parts of the Universe), physics and biophysics (Molecular Simulations), etc.

1.1.1 Molecular Simulation (MS)

In the fields of material sciences, physics, biophysics, astrophysics, etc., computer simulations are usually used to help with the mathematical modeling of natural systems. These simulations involve physical movement of large number of particles (atoms, stars, etc.). The particles move and interact among each other, following basic Newtonian motion laws (forces
between particles and potential energy are specified by molecular mechanics force fields). Some of the parameters of the Newton’s equations of motion are of variable nature in order to better understand the dynamics and features of these particles over certain period of time. Once the system reaches equilibrium (no change over time), the simulation stops.

Molecular simulations (often referred to as Molecular Dynamics (MD) simulations) are computer simulations of complex biological, physical or chemical structures. These simulations, based on theoretical models, are widely utilized as a basic research mechanism for studying the behavior of the natural systems. The basic units of such systems/simulations are natural particles (such as atoms, molecules, stars, etc.) and they interact among each other for a certain period of time following postulated classical forces. Scientists run such simulations to analyze the characteristics of natural systems using experiments based on some theoretical models [41, 68]. MS, motivated by a wide range of applications, have become an important research tool in material sciences [42], astrophysics [61], biomedical sciences, and biophysics [2]. They are proven and powerful tool for understanding the inner-workings of a scientific system, by supplying a model description of the physical, biophysical and/or biochemical processes that are being unfold at a particle’s scale.

1.2 Challenges Imposed by the Analysis of MS Data

1.2.1 MS Data Analysis

The fist challenge that we are going to bring to light and which is imposed to the data analysis systems is the type of queries through which the data is being accessed. Usually, this is done through high-level analytical queries which computation is a lot more complex than the computation of simple aggregates. In order to achieve objective discovery and to
explain the workings of the scientific systems, scientist must analyze the data produced by the MS. Quantities measured during the simulations are analyzed to test the theoretical model [25, 43]. These analysis oftentimes comprise of computation of very complex quantities that show statistical properties of the data. Such queries are of great importance to scientist because they are the basic assembly blocks for a series of critical quantities needed to outline the scientific systems [25]. Some of these queries are usually the bottleneck of the data analysis systems, because they take a lot of time (often many days) to be executed and the existing systems are not designed to handle multiple queries on the same data stream at the same time, thus decreasing the overall efficiency of the data analysis.

Some of the statistical quantities or functions that are being computed in the process of MS system analysis are: center of mass, electron density, system’s energy, mean square displacement of particles, dipole histogram, diffusion constant, force autocorrelation, velocity autocorrelation, radial distribution function, etc. There are two types of queries/functions among the ones used to analyze an MS system. The first type is so-called one-body functions. They only involve quantities (attributes) from a single atom at any given time in the process of computation. Each atom (atom’s attributes) is being processed a constant number of times, thus the total running time of such functions/queries is \( O(N) \) when the system’s particle count is \( N \). This type of functions will produce useful final result in a single run of the incoming data. Most of these functions are defined on a single frame of the MS data. Even though we implement such queries in our system, we don’t spend a lot of time explaining them since they are fairly simple mathematical functions.

The second type of functions is multi-body functions. These functions are holistic in nature and are of special interest to scientists. The computation of such functions involve more than
one atom’s attributes and cannot produce final result in a single run of the MS data (i.e., if traditional methods are used for their computation). If a function/query takes \( m \)-tuple subsets of the data and sees them as one unit, that function is called \( m \)-body correlation function. Such complex queries include the Radial Distribution Function (RDF) [25, 45, 61] as well as some quantities associated with chemical shifts [69]. Generally, such functions are computed through histograms. For instance, the RDF is obtained from a histogram of all pairwise atom distances (called Spatial Distance Histogram or SDH [70]).

### 1.2.1.1 Spatial Distance Histogram Computation

An SDH is a histogram of all distances between all pairs of particles in the simulated system. The SDH serves as discrete approximation of the continuous probability distribution function (RDF). Figure 1.1 represents the idea behind the SDH computation. This type of query is of great importance to scientist being the basic assembly block for a series of critical quantities needed to outline the scientific systems [25]. Even though SDH (RDF) is one of the most important queries in MS, there are not that many efficient algorithms that tackle it. The traditional, straightforward way of computing RDF / SDH is a very time consuming process. If a brute force method is used, such complex multi-body functions need \( O(N^2) \) computations for \( N \) particles [2].

In this dissertation, we design and implement an efficient algorithm for computing SDH.

### 1.2.1.2 Motivation

The SDH is a fundamental tool in the validation and analysis of particle simulation data. It serves as the main building block of a series of critical quantities to describe a physical system.
Specifically, SDH is a direct estimation of a continuous statistical distribution function called *radial distribution functions* (RDF) [7, 26, 61] which is defined as

\[ g(r) = \frac{N(r)}{4\pi r^2 \delta r \rho} \]  

(1.1)

where \( N(r) \) is the expected number of atoms in the shell between \( r \) and \( r + \delta r \) around any particle, \( \rho \) is the average density of particles in the whole system, and \( 4\pi r^2 \delta r \) is the volume of the shell. Since SDH directly provides the value for \( N(r) \), the RDF can be viewed as a normalized SDH.

The RDF is of great importance in computation of thermodynamic quantities about the simulated system. Some of the important quantities like total pressure,

\[ p = \rho kT - \frac{2\pi}{3} \rho^2 \int dr r^3 u'(r) g(r, \rho, T) \]
and energy

\[
\frac{E}{NkT} = \frac{3}{2} + \frac{\rho}{2kT} \int dr 4\pi r^2 u(r) g(r, \rho, T)
\]

can be derived in terms of structure factor that can be expressed using \(g(r)\) [34]. For monoatomic systems, the relation between RDF and the structure factor of the system [22] takes simple form, viz.

\[
S(k) = 1 + \frac{4\pi\rho}{k} \int_0^\infty (g(r) - 1) r \sin(kr) dr.
\]

The definitions of all notations in the above formulae can be found in [34] and [22]. To compute SDH in a straightforward way, we have to calculate distances between all pairs of particles and put the distances into bins with a user-specified width, as done in state-of-the-art simulation data analysis software packages [38, 61]. MS or N-body techniques generally consist of large number of particles. For example, the Virgo consortium has accomplished a simulation containing 10 billion particles to study the formation of galaxies and quasars [60]. This kind of scale prohibits the analysis of large datasets following the brute-force approach. From a database viewpoint, it would be desirable to make SDH a basic query type with the support of scalable algorithms.

Previous works [30, 70] have addressed this problem by developing algorithms that compute exact SDHs with time complexity lower than quadratic. The main idea is to organize the data in a space-partitioning tree and process pairs of tree nodes instead of pairs of particles (thus saving processing time). The tree structure used include \(kd\)-tree in [30] and region quad/oct-tree in our previous work [70], which also proved that the time complexity of such algorithms is \(O(N^{2d-1})\)
where \( d \in \{2, 3\} \) is the number of dimensions in the data space. While beating the naive solution in performance, such algorithms’ running time for large datasets can still be undesirably long. On the other hand, an SDH with some bounded error can satisfy the needs of users. In fact, there are cases where even a coarse SDH will greatly help the fine-tuning of simulation programs [26].

Generally speaking, the main motivation to process SDHs is to study the statistical distribution of point-to-point distances in the simulated system [26]. Since a histogram by itself is an approximation of the underlying distribution \( g(r) \) (Eq. (1.1)), an inaccurate histogram generated from a given dataset will still be useful in a statistical sense. Therefore, in this dissertation, we focus on approximate algorithms with very high performance but low error rates. In addition to experimental results, we also evaluate the performance/accuracy tradeoffs provided by the proposed algorithms in an analytical way. The running time of the proposed algorithms is only related to the desired accuracy. Our experimental results show significant improvement in performance/accuracy tradeoff of our approximate algorithms over previous SDH algorithms – the error rates in query results are very small even when the running time is reasonably short.

1.2.2 MS Data

On top of the CPU hungry analytical queries, another challenge that is being imposed to the data analysis systems is the shear data they need to work with. The amount of data produced in a Molecular Simulation is very large, often in gigabytes (sometimes in petabytes). MS involves systems of moving particles that interact with each other. The number of these particles (basic units) taking part in the molecular simulations is big, usually in the range of hundreds of thousands to millions. For instance, Fig. 1.2(a) shows a snapshot of a collagen fiber. This collagen structure alone consist of 890,000 particles. In the field of astrophysics, the systems
observed and analyzed include clusters of stars that can hold billions of stars. For example, Fig. 1.2(b) depicts the Globular Cluster M80 (NGC6903)[50], the densest star cluster in our own galaxy, the Milky Way. This cluster contains hundreds of thousands of stars.

And oftentimes, the simulations produce datasets consisting of more than one snapshot (frame) of the system’s current state at various time instants. Each of these frames contain all the particles, together with all their measurements, such as spatial coordinates, mass, charge, velocity, forces, etc. And usually, a big number of such frames (in the tens of thousands) are being produced and stored in the course of a typical simulation.

1.2.2.1 Data Access: Pull-based vs. Push-based

The big volume of data produced by scientific simulations is imposing a significant stress on the data management and analysis systems / software.

The state of the art data management systems (DBMS) are very well equipped to handle very big load of data. The data produced by an application is fed to the DBMS at the very instance it is produced and the DBMS takes care of all aspects of data management, including organization.
of storage, indexing, retrieving data, etc. However, even though DBMSs are designed with a large amount of data in mind, they are very specifically oriented and optimized for business applications. Thus, handling large quantities of scientific data still imposes challenges to the existing DBMSs [20, 31, 51]. We can conclude, for now at least, that the current DBMSs are not the best option for scientific data management and analysis.

On the other side, the existing data analysis systems (like GROMACS) are configured to deal with large volume of scientific data, but they are not optimized for high throughput data analysis. Also, often times, the data is available only for certain period of time (e.g., streams) so the analyzing system should react to it in a very efficient way. The methods by which this enormous amount of data is being accessed can either increase or decrease the efficiency of a data analysis system. The known data analysis systems use pull based type design in which the data is being fed to the queries only by demand. When a query needs the data, it requests it. The data is then pulled through the system and delivered to the query. This type of design, we believe, introduces two types of problematic issues: 1. overhead in I/O traffic by not allowing for multiple queries to be executed on the same stream of data at the same time, and 2. cpu/data latency incurred when a request for the data is sent to the system (the time it takes from a request to the time the query can actually use the data). In the push-based design, the queries do not request the data. Rather, the data is pushed onto the queries automatically by the system and is being processed by the active queries. This alleviates the I/O traffic overhead as well as the cpu/data latency incurred for each query that would have been introduced by a pull-based design. Such a system’s design is especially suitable for many scientific applications that share the following features. First, the scientific analysis often involves executing a number of analytical queries commonly processed on a large portion, if not all of the generated data.
Second, many of these fields use the same popular data analytics primitives as the basic blocks on which they build their discovery. And three, the scientific data, once stored is never modified. New data only appends to the existing dataset, making the data perfect contender for streaming. Figure 1.3 depicts the two designs and their essential difference.

Figure 1.3: The difference in data access between push and pull system.

In this dissertation we tackle the challenge imposed by the amount of data produced by a simulation and we present a system that is capable of high throughput data analysis based on a push-based design. We describe the idea and we design and implement such push-based system for efficient data analysis. We would like to point out that the work presented in this dissertation is an application of such push-based strategy that has been previously considered in the database community in the works presented in [4, 29, 36], as well as the idea of sharing same tuple of data or data scans across many queries previously introduced in [14, 66, 73].

To summarize, data-intensive applications often require compelling amount of storage space and intensive processing capability, but also need fast data access and high throughput data analysis. Therefore, the need of a system that will be optimized to access data fast, with high
throughput, as well as efficiently execute the analytical queries with a possibility of running multiple queries on the same data stream is of a great importance to the scientific community.

1.3 Problem Statement

As mentioned previously, in this work we tackle two problems imposed by the process of MS data analysis.

1.3.1 Efficient Computation of SDH

The problem can be defined as follows: given the coordinates of \( N \) points in space, we are to compute the counts of point-to-point distances that fall into a series of \( l \) ranges in the \( \mathbb{R} \) domain: \([r_0, r_1), [r_1, r_2), [r_2, r_3), \ldots, [r_{l-1}, r_l]\). A range \([r_i, r_{i+1})\) in such series is called a bucket, and the span of the range \( r_{i+1} - r_i \) is called the width of the bucket. In this dissertation, we focus our discussions on the case of standard SDH queries where all buckets have the same width \( p \) and \( r_0 = 0 \), which gives the following series of buckets: \([0, p), [p, 2p), \ldots, [(l-1)p, lp]\). Generally, \( lp \), which is the boundary of the last bucket, is set to be the maximum distance of any pair of points in the dataset. Although almost all scientific data analysis only require the computation of standard SDH queries, our solutions can be easily extended to handle histograms with non-uniform bucket width and/or arbitrary values of \( r_0 \) and \( r_l \).\(^1\) The answer to an SDH query is basically a series of non-negative integers \( h = (h_1, h_2, \ldots, h_l) \) where \( h_i \) (\( 0 < i \leq l \)) is the number of pairs of points whose distances are within the bucket \([(i-1)p, ip)\).

\(^1\)The only complication of non-uniform bucket width is that, given a distance value, we need \( O(\log l) \) time to locate the bucket instead of constant time for equal bucket width.
1.3.2 I/O Issues and High Throughput Data Analysis

As mentioned earlier, in order to describe the scientific system, scientist must analyze the critical statistical properties of the data produced by the simulation. These properties are usually computed through series of queries that are being executed on the whole, or some selection of the MS data. Most of the systems that are widely used today across many scientific fields for analysis of the MS data are part of software systems that also run the simulation. Such examples include GROMACS [38], VMD [40], MDAnalysis [49], Wordom [58], MD-TRACKS [67], SimulaidOne [48], and Charmm [6]. Once the simulation is run, the system produces a flat file with all the measurements of each particle. These systems take each query as a user’s input and apply them to the data uploaded from these flat files (called trajectories in GROMACS). The biggest I/O problem/performance issue that these systems impose is the fact that for every new query issued by the user, the system has to load a significant part of the dataset into main memory before executing the query. Such pull-based design involves random I/Os that considerably affect the data throughput in the system. And while the use of an index-based scan might seem better option to a sequential scan, this is the case only when the query accesses small portion of the data. All of this gives rise of redundant and high I/O traffic in the process of scientific data analysis in pull-based type systems. This adds greatly to the overall performance time of the analysis of the MS system. Having in mind the volume of data produced by the MS in a single frame (in gigabytes), and we add the number of frames simulated (tens of thousands) the total volume of data to be analyzed is of huge magnitude. So to have to load big chunks of this data from disk to the main memory every time a query is being executed degrades the overall
system’s performance greatly. In this dissertation we design and implement a push-based type system that allows high-throughput data analysis in the process of scientific discovery.

1.4 Our Approach

In this work, we present a system that incorporates improvements of both aforementioned problems.

As mentioned earlier, an SHD is a histogram of the distances of all pairs of particles in the system. The brute force way requires computing all pairwise distances, essentially looking at all particles in the system. This approach requires quadratic time (in terms of the number of particles). To remedy this problem, we have taken the following approach. First, we constructed a conceptual data structure called Density Map (DM). This DM splits the simulated system’s space into a framework of regions (cells) of equal size. A region quad tree was used to represent this DM. Every node of the tree represents one cell from the DM. To go from one level to the next in the tree, every node (cell of the DM) is divided into four equally sized parts (subregions). Each node records the number of all particles that are in that region of the simulated space, as well as the location of the cell (the coordinates of the corner points of the cell). Second, in order to improve the naive method for SDH computation, we treat each cell of the DM as a single processing unit. This way, we significantly reduce the number of calculations for SDH computation. Details of the algorithm can be found later in this dissertation.

Considering the fact that the running time of the many queries used in the scientific world to analyze the MS is minuscule in comparison to the loading time of the MS data, it is safe to say that the scientific world would benefit from a system that can remedy the aforementioned problem. Our idea is to build a system that would load the MS data from hard disk in the main
memory only once and then execute as many queries as needed/wanted on that data, without the need of reloading the same data over and over again. Considering the volume of data in a single MS frame as well as the number of frames produced during a single simulation, this type of system can save a lot of time for MS analysis. The system would contain some of the most used queries that can also be run as user’s input. These queries would be pre-programmed as separate modules in our system, but they would be able to take certain attributes, like data selection for instance, as user’s input. Once the system is run, all (or a selection) of these queries would be executed against the MS data or certain selection. The system would act as a type of push-based system, essentially pushing the loaded data to all the queries.

Another improvement that our system provides is the following: by thorough observation and analysis we have discovered that many of the quantities (functions/queries) that are being computed in order to analyze the MS system share some of the basic queries. In order to take advantage of that, our systems pre-computes these basic queries and has their results handy (stored in memory) whenever one of the more complex functions needs them. So, for instance, when the total mass of the system is needed, our system just pulls the pre-computed value out and serves the query that needs it immediately. The more queries use the same basic part, the bigger the time saving will be.

1.5 Contribution and Roadmap of the Dissertation

We have designed and implemented (simulated) a push-based system that can be used as a tool for analysis of MS systems. We have tested the system on real MS datasets. The experimental results evidently show the superiority of the proposed system over one of the most widely used MS analysis system across many scientific fields, i.e., GROMACS. The
efficiency improvement ranges anywhere from 3 to 100 times improvement (not including the RDF computation) per set of selected queries run on a different selections of the MS data. If we take in account the number of such queries scientists may run every day, we believe our system would be of great benefit to MS community. In addition to the higher efficiency that our system achieves, we also believe that our idea will initiate a new breed of database related, push-based systems that could be used to analyze the MS systems in a more efficient way. Having a push-based like system, we were also able to improve the computation of the more complex quantities. Namely, the computation of some complex quantities involves computation of simple functions (sub-parts) that are shared among number of these complex quantities. So, by computing the shared sub-quantities in advance, we were able to achieve, however minuscule, time gain when computing the more complex queries.

Included in the aforementioned system is an approximate algorithm for efficient SDH computation that we have designed and implemented based on our previous work. The experimental results show that our new approximate algorithm is very efficient (fast run time) while maintaining surprisingly low error rates. In addition to the experiments, we have also backed up our method by analytical evaluation of the efficiency/error rate tradeoffs. We also developed different mathematical model to analyze the mechanism that leads to small error rates. This model, aside for tightening the error bounds on the original method, gave us idea of how we can improve on the basic approximate algorithm. This gave rise to the faster, more improved single level approximate algorithm for SDH computation.
The major technical contributions presented here are:

1. Present an approximate SDH processing strategy that is derived from the basic exact algorithm, and this approximate algorithm has constant-time complexity and a provable error bound,

2. Develop a mathematical model to analyze the effects of error compensation that led to high accuracy of our algorithm,

3. Propose an improved approximate algorithm based on the insights obtained from the above analytical results,

4. Design the network (tree like) of the most commonly used queries in MS (physics);

5. Build the system: design and build the modules, representing the quantities to be computed in an efficient manner, following the already built network, and

6. Develop a scientific simulation database benchmark that can be used for evaluating similar systems and products.

The remainder of this dissertation is arranged as follows: in Chapter 2 we give a survey of work done on efficient data analysis and specifically efficient SDH computation as well as the systems used in the field of MS data analysis. In Chapter 3 we introduce an approximate algorithm for efficient SDH computation (in 3.3). In the same chapter, we present the results of numerous experiments (in 3.4) as well as the results of the analytical evaluation of the approximate algorithm (in 3.5). This leads to the introduction and evaluation of a new and improved approximate algorithm discussed in Section 3.6. We continue the dissertation with Chapter 4, in which we show the design of the query network built from the most frequently used
quantities in MS system analysis (in 4.3) and we describe our push-based system for MS data analysis (in 4.4). Then, in Section 4.5 we present the benchmark designed to test our system as well as the results attained through comprehensive experiments run on real MS generated data. At the end, we conclude this dissertation with Chapter 5 in which we give an overview of our possible future endeavors in the field of MS data analysis.
Chapter 2: Related Work

2.1 SDH Computation

Recently, there have been some efforts aimed at designing and building MS data management systems on top of relational databases. Such efforts are presented through projects like BioSimGrid [51] and SimDB [21] that were developed especially for molecular simulations. However, to the best of our knowledge, such systems still lack the efficiency needed for MS data management as well as efficient query processing strategies. As far as our knowledge goes, the computation of SDH in such software packages is done in a brute-force way, which requires $O(N^2)$ time.

2.1.1 Force Computation Problem

In particle simulations, the computation of (gravitational/electrostatic) force is similar to the SDH problem. Specifically, the force is the sum of all pairwise interactions in the system, thus requires $O(N^2)$ steps to compute. The simulation community has adopted approximate solutions represented by the Barnes-Hut algorithm that runs on $O(N \log N)$ time [8] and the Multi-pole algorithm [32] with linear running time. Although all above algorithms use a tree-like data structure to hold the data, they provide little insights on how to solve the SDH problem. The main reason is that these strategies take advantage of two features of force: 1) for any
pairwise interaction, its contribution to the force decreases dramatically when particle distance increases, and 2) the effects of symmetric interactions cancel out. However, neither feature is applicable to the SDH computation, in which every pairwise interaction counts and all are equally important. Another method for force computation is based on well-separated pair decomposition (WSPD) \cite{13} and was found to be equivalent to the Barnes-Hut algorithm. A WSPD is a collection of pairs of subsets of the data such that all point-to-point distances are covered by the collection. The pairs of subsets are also well-separated in that the smallest distance between the smallest balls covering the subsets (with radius $r$) is at least $sr$ where $s$ is a user-defined parameter. Although relevant by intuition, the WSPD does not produce fast solution for SDH computation.

2.1.2 Data Stream Data Analysis

It is worth mentioning that there has been work done on a broader problem of histogram computation in the context of data stream management \cite{28}. The data stream systems usually work with distributive aggregates \cite{28} such as COUNT, SUM, MAX and MIN, which may be computed incrementally using constant space and time. They also tackle so-called holistic aggregates such as TOP-k \cite{18}, \cite{46}, QUANTILE \cite{47}, and COUNT DISTINCT \cite{27}, \cite{56}. When computing the holistic aggregates they have utilized hash-based functions that produce histograms \cite{46}, \cite{24}. But the data stream community has never specifically worked on the problem of computing a histogram that will disclose the distance counts belonging to a particular range (a bucket), i.e., an SDH. After thoroughly reviewing their work, we believe that none of their proposed solutions is directly applicable to the problem of SDH computation stated in this dissertation.
2.1.3 K-nearest Neighbors Problem

Another similar problem to the SDH computation is to find $k$-nearest neighbors (kNN) in a high-dimensional space [71]. In such a problem, avoidance of distance computation is the primary goal in algorithmic design due to the high cost of such operations. The main technique is to choose a set of reference points (i.e., pivots) in the database and pre-compute distances between data points to the pivots. In processing kNN queries, the search space can be pruned based on the pre-computed distances. However, being a searching problem, kNN is very different from the counting-based SDH problem. As a result, the data structures and algorithmic details shown in [71] have little overlap with our solutions to the SDH problem.

2.1.4 Space-Partitioning Trees Concept

Although SDH is an important analytics, there is not much elaboration on efficient SDH algorithms. An earlier work from the data mining community [30] opened the direction of processing SDHs by space-partitioning trees. The core idea is to process all the particles in a tree node as one single entity to take advantage of the non-zero bucket width $p$. This lowers the processing time by avoiding computation of particle-to-particle distances.

2.1.4.1 Quad-tree Approach

A Quad-tree is a tree data structure that is a fully balanced. It was first introduced by Raphael Finkel and J.L. Bentley in 1974 [[23]. Each node in the tree (except leaf nodes) has four children. The Quad-tree recursively divides the particle space into four equally sized partitions. Each point from the dataset is only found into one such region of the data space, therefore only found in one tree node. This means that there are no overlaps between subregions of a Quad-
tree. When the building of the Quad-tree stops, depends on the application. But in general, the height of the tree is influenced by the number of particles that can be stored in a leaf node. The bigger the number of particles in the leaf nodes allowed, the smaller the number of point-to-point distance computations. This will be very beneficial when computing SDH.

Earlier work done by Tu et al. [70] proposed an SDH algorithm using Quad-tree structure. That work also shows rigorous mathematical analysis (not found in [30]) of the algorithm’s time complexity. Specifically, in [70], a novel algorithm was proposed (named \textit{DM-SDH}) to compute SDH based on a data structure called \textit{density map}, which can be easily implemented by augmenting a Quad-tree index. The mathematical analysis [15] has shown that the algorithm runs on $\Theta(N^3)$ for two-dimensional data and $\Theta(N^5)$ for three-dimensional data, respectively.

We will present the main idea behind the DM-SDH algorithm later in this dissertation, in Section 3.2. Even though this algorithm significantly improves on the brute force method for SDH computations, its running time is still tied to $N$, the size of the dataset. To fix this, we set our mind in designing an improved algorithm with higher efficiency and a running time not related to $N$. That work produced an improved, very efficient approximate SDH algorithm with high accuracy [68]. Most importantly, its running time is not related to the dataset size. We present that work later in this dissertation.

\subsection{R-trees}

R-tree is a dynamic data structure that enforces hierarchy of objects. It was first introduced by Antonin Guttman in 1984 [33]. R-trees are very similar to B-trees often seen in the field of database [9, 17] because both are based on the concept of a balanced search tree. Each of the nodes represents a partition (rectangle) of space with all the points in it and they correspond to
disk page if the index is disk resident. The search algorithm works by comparing these nodes (i.e., the rectangles) to a search rectangle. Since the R-tree enforces hierarchy when splitting the data space, a spatial search query would result in very small number of visited nodes. Thus, the R-trees are well suited for storing spatial objects and supporting fast spatial access methods, including indexing multi-dimensional information, like geographical coordinates. The index is being dynamically updated every time a new point is added/removed from the database, making the R-tree an adequate structure for database operations such as update, delete and insert without the need of repeated reorganization. But the drawback on the R-tree comes from the way its search algorithm works: the nodes are recursively accessed if they overlap with the search rectangle (the given search range). This may lead the search algorithm to visit more than one subregion of the tree. Because of this overlapping, if the R-tree were to be used in an SDH computation, it may end up with possibly large number of duplicate counts of distances that would have to be found and deleted. This makes the R-tree approach inadequate for efficient SDH computation.

2.1.4.3 kd-tree Approach

The data structure adapted in [30] is the kd-tree. A k-d tree is a data structure that is an abstraction of a binary search tree. The k-d tree structure was first introduced by Jon Louis Bentley in 1975 [10]. A k-d tree, also known as k-dimensional tree, partitions a k-dimensional data space and stores its data points. Since the k-d trees are based on binary tree, their root too represents the entire data collection. Every node of a k-d tree is a k-dimensional point, representing sub-collection of points in the space. Hyper planes perpendicular to the system’s axes are being used to split the space of the k-d tree along a specific dimension. This means that,
at the root, all the sub-nodes will be split based on whether their first dimension is smaller (put to the left) or bigger (put to the right) compared to that of the root. This process continues going down the tree, and at each level divides the space (points) on the next dimension. Because of the way it is being build, it is possible in the k-d tree to have non-leaf tree nodes that have same points. But, the k-d tree ensures that all the leaf nodes will only consist of distinct (partitions of) points. Figure 2.1 shows a k-d tree with distinct points in the leaf nodes. The leaf nodes are also referred to as k-d tree buckets. A k-d tree is a useful data structure that has several applications, most notably in search and sorting problems. One such application is in search problems involving multidimensional keys, like nearest neighbor searches or range searches. Many database problems have been solved using k-d tree structures. Its database applications have been observed in [11].

![Figure 2.1: Creating kd-tree from data space.](image)

K-d trees can be also useful in representing a simulation system. Particles of the simulation system are stored into the nodes of the k-d tree, each node representing a single cluster of points in the simulation space. The idea behind using any tree structure, k-d tree included, is to process the nodes with all particles they store as a whole (instead of processing individual
particles). This, obviously will be an improvement over the naive, brute force approach that computes all point-to-point distances in order to generate the spatial distance histogram.

But the k-d trees have their drawback: by their design, the k-d trees may have huge number of nodes storing very little number of particles. And, if used for SDH computation, this big number of nodes will essentially induce a large number of node pair’s computations. In other words, an approach using k-d tree structure to solve the SDH problem may fast degrade into brute force time consuming method. Thus, we believe k-d tree is not the right choice of structure to be used when efficient SDH algorithm is needed.

In this dissertation, we introduce the design of the approximate algorithm and its performance analysis. Technically, we emphasize the impacts of error compensation among different distribution operations. As a result, we do not require low error rates to be obtained from each operation, as we show the total error is low even when a primitive heuristics is used. It is worth mentioning that our group has recently published another paper [3] in this field. But that work focuses on algorithms that generate approximate results using different type of heuristics based on spatio/temporal uniformity of data items. In other words, these two methods, although both take approximate SDH processing as the basic theme, make their contributions at two different levels of the problem. Work in [3] focuses on improving accuracy of single distribution operations while this work, in addition to a systematic description of the algorithm, studies how errors from different distribution operations cancel out each other, and to what extent such error compensation affects the accuracy of the final results.
2.2 Push-based System

2.2.1 Streaming Data

The idea of data streaming has been broadly used in many fields. The main usage, however, is aimed at processing live data generated online. There is an ocean of references for data stream management, but we believe the presentation in [28] encapsulates the majority of the ideas, problems and solutions. In the past decade, however, the database community started to follow the data stream idea to process stored data. Processes can take advantage of the streaming data at any time the data is being pushed through the system. This gives rise to push-based type design for data management systems. The idea of such push-based design was previously considered in projects such as DataPath [4], Volcano [29] and QPipe [36] among others. These works present ideas in which the pull-based dataflow is compared to the push-based dataflow, showing the need for the later. They also talk about maximizing the data and work sharing among queries at runtime. Essentially, we incorporate such ideas in our design of the push-based system.

2.2.2 DBMS

On the other side, the scientific community has steadily progressed from processing massive data files towards employing database systems for the storage, acquisition, and analysis of large-scale scientific data [5, 63]. The widely used and popular relational database systems are conventionally designed and optimized to better manage the data produced by the business type applications. But such conventional database systems (DBMS) are not well equipped to deal with the type and quantity of scientific data, such as the data produced by the molecular
simulations. In the recent past, the DBMS community has made some attempts into the design and construction of database systems optimized for handling scientific data. Such examples include the BDBMS project [19] that deals with annotation and provenance of the sequence data in biosciences, and the PeriScope project [55] that is designed to efficiently handle declarative queries against bio sequences. On top of the aforementioned examples, there are also ideas for new DBMS frameworks aimed at the management of scientific data [12, 39, 62]. One of those systems is the SciDB [12] and it is closest to the idea presented in this dissertation. SciDB is a data management and analytics system that is primarily used in application domains involving very big scale array data. This system, like the one presented in this dissertation, is designed around a multi-dimensional array data-model and it uses arrays to store the data. SciDB stores petabytes of data on a number of machines and runs its queries on those machines. It is made for high performance, high-availability, fault tolerance, and scalability. However, to the best of our knowledge, it too follows the pull based design where its queries demand the data they need. As seen earlier in this dissertation, this type of design can impose I/O overhead and decrease the data throughput when doing the analysis. Aside the mentioned issue, the design and build of such DBMS optimized for scientific data management come with additional challenges. Such challenges as well as their probable resolution are outlined in [31].

2.2.3 Analysis Systems

Generally, the data produced in the process of molecular simulation is being stored in large, plain files with no structure whatsoever. Queries, which are implemented in a stand-alone program within simulation/analysis systems, are executed onto such files producing the quantities that scientist use to analyze the molecular system. Such simulation and/or analysis
packages include: Gromacs, VMD, MDAnalysis, Wordom, MD-Tracks, SimulaidOne, and Charmm. But to the best of our knowledge, all of these systems work on a similar basis: they take a user defined query and execute it against the MS simulated data. In order for the query to be executed, the data has to be loaded into the main memory. Then, the result is either produced onto the display or written to a file. When the next user query comes, the system again loads significant part of the dataset into the main memory and executes the query. We believe that there is a room for improvement of such systems, given the fact that many of the user defined queries are fairly static. In other words, there is a number of queries that a user would always want to execute on a given simulation data. Furthermore, the selections of MS data onto which such queries might be executed, are also fairly constant (i.e., oftentimes the user selects the same group of atoms (e.g., all hydrogen atoms) to calculate given quantity, like center of mass for instance). So, by pre-coding many such queries and running them automatically once the system has loaded the data into memory, we believe we can save a lot of time that otherwise would have been spent in loading the same data into main memory anytime a query is executed. On top of the automated query execution, our system can take user’s query as input as well. With this, we believe our system is an improvement over the MS analysis systems that are used today.
Chapter 3: Spatial Distance Histogram (SDH) Efficient Computation

Spatial distance histogram (SDH) is one of the most used types of queries in Molecular Simulation system’s analysis. Being the histogram of all point-to-point distances in the system, it is very time consuming and computationally costly, even when the system contains fairly small number of particles.

3.1 Preview

Many of the known algorithms use the naive, often brute-force method for SDH computation. This approach is tightly related to the size of the dataset used in the simulations system. If there were \( N \) particles in the system, the naive method to compute SDH would result in \( N^2 \) computations, since the SDH is a histogram of all point-to-point distances. This type of approach is highly inefficient and can take many days when \( N \) is large. There are some improved algorithms [15, 30, 64], as mentioned in section 2, that have beaten the \( N^2 \) running time. On top of the improved running time, these works have also provided tight bound on the errors. But they too fall short for big data size sets, considering the fact that their running time is related to the database size \( N \). In this chapter, we present an efficient algorithm for SDH computation of the MS system under analysis. This new algorithm is unrelated to the data size \( N \) and despite being an approximate algorithm, it exhibits fairly high accuracy rate, while keeping its running time very low. In our method presented here, we have incorporated the two main features
of a good approximate algorithm. First, we show that we can control the error bound of our algorithm and we can also prove it is smaller than a certain threshold. This feature assures that the user knows how far from the correct solution she/he is. Second, we elaborate on the analysis of the cost needed for the algorithm to achieve smaller error than a given error bound. The second feature assures that the user will reach the needed efficiency/accuracy trade offs.

3.2 Background: DM-SDH

Before we introduce the new algorithm later in this chapter, we first elaborate on an approach that will serve as the main building block for our algorithm. That approach was presented in [70] and was used to compute the exact spatial distance histogram. The basic techniques, as well as the analysis done for that algorithm will be the core of the new and improved approximate algorithm presented in this dissertation.

We begin with the presentation of all notations that will be used in this chapter. Table 3.1 lists these notations. Note that these are the general symbols. There might be some symbols defined and used in a local context. Those symbols are not found in this table.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>total number of particles in data</td>
</tr>
<tr>
<td>$d$</td>
<td>number of dimensions of data</td>
</tr>
<tr>
<td>$i$</td>
<td>an index symbol for any series</td>
</tr>
<tr>
<td>$p$</td>
<td>histogram bucket’s width</td>
</tr>
<tr>
<td>$l$</td>
<td>number of histogram buckets</td>
</tr>
<tr>
<td>$DM_i$</td>
<td>density map of level $i$</td>
</tr>
<tr>
<td>$H$</td>
<td>tree height, i.e. number of density maps</td>
</tr>
<tr>
<td>$h$</td>
<td>SDH with elements $h_i$ ($0 &lt; i \leq l$)</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>error bound for the approximate algorithm</td>
</tr>
</tbody>
</table>

Copyright © 2012, IEEE.
3.2.1 Overview of the Density Map-based SDH (DM-SDH) Algorithm

In order to beat the running time of an algorithm using the naive approach for SDH computation, which we saw earlier is $O(N^2)$, we have to somehow avoid the computation of all point-to-point distances in the system. The smaller the number of computations, the faster the algorithm will work. To keep this number under control and small, we point out one very crucial feature of a spatial distance histogram: each bucket of the histogram has width $p$ that is always greater than zero. So, in order to determine in which bucket of the histogram a pair of points belongs, we only need to 1) figure out a range in which the distance between those two points belongs to, and 2) determine if that range fits entirely within the boundaries of a certain bucket.

The core idea behind this algorithm is a conceptual data structure named density map. The density map essentially is a framework of equally sized partitions (cells). When we are working with a 2-dimensional (2D) system, a cell in the density map is a square and when working with a 3-dimensional (3D) system, the cell is a cube. In this dissertation, unless said otherwise, we only use 2D systems and grids to present the methods. Note that, as studied in [15], it would be straightforward to develop a 3D version of the methods discussed in this work.

Every cell of the density map stores the number of particles occupying the space in the system that is represented by that particular cell. It also records the four coordinates of the region in order to know the exact position and boundaries of that cell/region in the system’s space.

The size of the cells defines one feature of the density map, we call resolution. In fact, a resolution of a density map is the reciprocal value of the size of a cell. In the process of SDH computation, we generate a sequence of density maps with different resolutions. All of the
generated density maps are organized into a point region (PR) Quad-tree [53]. Each level (i.e., all nodes on that level) of this Quad-tree represents a density map of certain resolution. In the sequence of density maps we build, the resolution of a given density map is always doubled in comparison to the resolution of the previous density map. In terms of the Quad-tree, that translates into: each tree level \( i \) has four times the number of nodes as the previous level \( i - 1 \).

Figure 3.1 represents the pseudocode for the DM-SDH algorithm. The procedure called RESOLVETWOCELLS is the heart of the algorithm. It takes a pair of cells \( M_1 \) and \( M_2 \) from the same density map as input. This procedure first computes the range of minimum and maximum distances between all pairs of particles, one in \( M_1 \) and one in \( M_2 \) (line 1). Since each cells of the density map stores the spatial coordinates of the region it presents, line 1 is executed in a constant time. We say two cells \( M_1 \) and \( M_2 \) are resolvable on this density map, if the range of minimum and maximum distances fall in a single bucket \( i \) of the histogram. If this is the case, the algorithm (lines 2 - 5) updates the histogram by increasing the number of counts in the specific bucket \( i \) by \( n_1n_2 \) where \( n_1 \) and \( n_2 \) are the counts of particles in cells \( M_1 \) and \( M_2 \), respectively. If the two cells do not resolve on the current density map, we go to the next one in the sequence, one with doubled resolution to the previous one and repeat the previous step. But in this case we call the RESOLVETWOCELLS procedure for all four (children) partitions of \( M_1 \) with all those of \( M_2 \) (lines 12 - 16). So, if \( M_1 \) and \( M_2 \) are not resolvable on the current density map, we have \( 4 \times 4 = 16 \) recursive calls to the RESOLVETWOCELLS procedure. In the case that we reach the last density map in the sequence, yet the two cells \( M_1 \) and \( M_2 \) are not resolvable then we need to compute distances between all particles in the two non-resolvable cells (lines 6 - 11). The density map \( DM_o \) at which the DM-SDH algorithm starts (line 2) is the first density map in the sequence with a diagonal of the cells smaller than the width \( p \) of
Algorithm DM-SDH

Inputs: all data points, density maps built beforehand, and bucket width $p$

Output: an array of counts $h$

1. initialize all elements in $h$ to 0
2. find the first density map $DM_o$ with cells diagonal length $k \leq p$
3. for all cells in $DM_o$
   4. do $n \leftarrow$ number of particles in the cell
   5. $h_0 \leftarrow h_0 + \frac{1}{2}n(n - 1)$
6. for any two cells $M_j$ and $M_k$ in $DM_o$
   7. do RESOLVETWOCELLS ($M_j, M_k$)
8. return $h$

Procedure RESOLVETWOCELLS ($M_1, M_2$)

0. check if $M_1$ and $M_2$ are resolvable
1. if $M_1$ and $M_2$ are resolvable
   2. then $i \leftarrow$ index of the bucket $M_1$ & $M_2$ resolve into
      3. $n_1 \leftarrow$ number of particles in $M_1$
      4. $n_2 \leftarrow$ number of particles in $M_2$
      5. $h_i \leftarrow h_i + n_1n_2$
   6. else if $M_1$ & $M_2$ are on the last density map
      7. for each particle A in $M_1$
         8. for each particle B in $M_2$
            9. do $f \leftarrow$ distance between A and B
               10. $i \leftarrow$ the bucket $f$ falls into
                  11. $h_i \leftarrow h_i + 1$
     12. else
      13. $DM' \leftarrow$ next density map with higher resolution
      14. for each partition $M'_1$ of $M_1$ on $DM'$
         15. for each partition $M'_2$ of $M_2$ on $DM'$
            16. do RESOLVETWOCELLS ($M'_1, M'_2$)

---

Figure 3.1: The density-map-based SDH algorithm. Copyright © 2012, IEEE.
the histogram bucket. The reason for this is that there will be no resolvable cells in any density
map with resolution lower than $DM_0$, since the bucket width would be smaller than the cell size.
Starting at that particular $DM_0$, we ensure that any intra-cell distance between particles of same
cells would be smaller than $p$ and will correctly be placed into the first bucket of the histogram
with range $[0, p)$ (lines 3 - 5). Lines 6 - 7 depicts the processing of the inter-cell distances, by
calling RESOLVETWOCELLS) for all cell pairs in $DM_0$.

One important detail of the DM-SDH, or more precisely of the Quad-tree design that will
be of great importance to the new approximate algorithm is the height of the Quad-tree, or the
number of density maps of different resolution in the sequence. The main gain of the DM-SDH
over the naive method is that it saves time by working with cells (the RESOLVETWOCELLS)
instead of computing all distances between particles one at a time. But when the count in the
cells is very small, the time saved by resolving those cells decreases. If the count falls extremely
low (say 4 or less particles), then an SDH computation would not gain anything by splitting such
cells in four partitions and try resolving their children. In such case the cost of resolving the
cells may be bigger than the cost of retrieving the particles and computing the distances between
them (lines 7 - 11 in RESOLVETWOCELLS). This may degrade the running time of DM-SDH
to the degree of the brute-force or even worse.

In light of the previous reasoning, we set the number of density maps $H$ to be:

$$H = \left[ \log_{2^d} \frac{N}{\beta} \right] + 1 \quad (3.1)$$

where $d$ represents the number of dimensions, $2^d$ is the degree of tree nodes (4 for 2D; 8 for 3D
data) and $\beta$ is the desired average number of points in a leaf node. Based on our findings, we set
\( \beta \) to be marginally greater than 4 in 2D (8 for 3D data). Otherwise, the CPU cost of resolving two cells will be greater than that of computing distances between all cell pairs.

### 3.2.2 Performance Analysis of DM-SDH

By its design, to only consider particles as part of cells in the density map, it is obvious that DM-SDH processes a number of pairwise distances in a single shot. Thus, DM-SDH imposes significant improvement over the \( O(N^2) \) of a brute-force method. A scrupulous analysis of the DM-SDH performance, as well as its time complexity can be found in [15]. The focus of the analysis is on the number of pairwise distances that can be accounted by resolving cells. A closed form formulae for that number is generated through a geometric modeling, thus making the analysis of the running time possible. Since the DM-SDH algorithm is not the main focus of this dissertation, we leave the complex technical details of that analysis as a reference only. Here we just outline the most important and relevant analytical results, because these results will be part of our basic building blocks in the design and analysis of the proposed approximate algorithm.

**Theorem 3.1** For any given standard SDH query with bucket width \( p \), let \( DM_o \) be the first density map where the DM-SDH algorithm starts running, and \( \alpha(m) \) be the ratio of non-resolvable pairs of cells on a density map that lies \( m \) levels below \( DM_o \) (i.e., map \( DM_{o+m} \)) to the total number of cell pairs on that density map. We have

\[
\lim_{p \to 0} \frac{\alpha(m+1)}{\alpha(m)} = \frac{1}{2}.
\]

**Proof.** See Section 4 of [15].
The idea behind Theorem 3.1 is that the possibility for a pair of cells to be non-resolvable, decreases by half when the level of the density map increases by one. So for example, for a non-resolvable cell pair on $\text{DM}_j$ where $j \geq o$, we expect $16 \times \frac{1}{2} = 8$ pairs of cells to be resolvable on the next level (out of the total 16 pairs of cells). The analysis shows that this Theorem works well not only for big $l$ (small $p$, which is more relevant for analysis of the simulated data), but it also rapidly converges even for fairly small $l$. In addition, the result of this Theorem also applies to 3D data (as shown in Section 5.1 of [15]). The most important aspect of Theorem 3.1 is that it shows that the amount of non-resolvable pairs of cells decreases exponentially as the algorithm accesses more density map levels. This observation is crucial in analyzing the complexity of the running time of DM-SDH that is derived as follows. The starting level $\text{DM}_o$ of the algorithm is fixed for any given SDH query with a bucket width $p$. Let us suppose there are $I$ pairs of cells on $\text{DM}_o$ that are non-resolvable. The total number of pairs of cells on the next level $\text{DM}_{o+1}$ that are considered by the algorithm is $I2^{2d}$. And in agreement with Theorem 3.1, half of these cell pairs will be resolved. This leaves $I2^{2d-1}$ unresolved pairs of cells. On the next level in the density map, $\text{DM}_{o+2}$, the number of unresolvable cell pairs would become $\frac{I2^{2d-1}2^{2d}}{2} = I2^{2(2d-1)}$. Therefore, after going over $n+1$ levels of the density map (Quad-tree), the number of calls to the RESOLVE TWO CELLS procedure is:

\[
T_c(N) = I + I2^{2d-1} + I2^{2(2d-1)} + \cdots + I2^{n(2d-1)} \\
= \frac{I[2^{(2d-1)(n+1)} - 1]}{2^{2d-1} - 1}
\]  

(3.2)
When $N$ grows to $2^d N$, $n$ increments by 1. Following the previous equation, we get:

$$T_c(2^d N) = \frac{I[2^{(2d-1)(n+2)} - 1]}{2^{2d-1} - 1} = 2^{2d-1} T_c(N) - o(1)$$

which derives

$$T_c(N) = O\left(N^{\log_{2d} 2^{2d-1}}\right) = O\left(N^{\frac{2d-1}{d}}\right).$$

The second portion of the DM-SDH’s running time includes the number of computed distances, which too follows the previously stated recurrence relation. More details of these derivations are presented in Section 6 of [15].

### 3.3 The Approximate Density Map-Based SDH Algorithm

Here, we will present a modified DM-SDH algorithm that enables us to get approximate results but with better performance. Our method specifically aims at two crucial components of a good approximate algorithm: 1) error bound that is both provable and controllable - this gives the user an idea on how close she/he is to the correct results; and 2) cost analysis: foresee the cost needed to achieve smaller error than a certain error bound - this assures that the user will obtain the needed efficiency/correctness tradeoffs.

As described in the previous section, the DM-SDH algorithm has to: 1) continuously resolve pairs of cells until the leaf level has been reached; and 2) compute every point-to-point distance for all the particles in the non-resolvable cells in the leaf level of the tree. The idea to improve the efficiency of DM-SDH through an approximate SDH computation is as follows: *stop the resolving of the cells when certain level of the tree has been reached and totally disregard all the distance computations if we can be certain that the number of distances in the*
unvisited levels will be within some error tolerance bounds. We call this algorithm Approximate Density Map-based SDH, or ADM-SDH. ADM-SDH can be easily obtained by modifying the DM-SDH. The main modification is that we don’t go all the way to the leaf level to try resolving cells. Specifically, the recursive calls to RESOLVETWOC commerciels stop after visiting $m$ levels of the tree. The vital issue with this modification, however, will be determining a value for $m$ given a user-specific error tolerance threshold $\epsilon$. In this dissertation we use these metrics to evaluate the errors:

$$e = \frac{\sum_i |h_i - h'_i|}{\sum_i h_i}$$

where $i$ is an SDH bucket, $h_i$ is the correct distance count, and $h'_i$ is the distance count produced by our approximate algorithm. Clearly, we have $\sum_i h_i = \frac{N(N-1)}{2}$.

According to our analytical model (i.e., Theorem 3.1), for total number of $l$ SDH buckets and any density map $DM_{o+m}$, the percentage of non-resolvable cells will be given in $\alpha(m)$. The value for $\alpha(m)$ can be efficiently computed, thanks to the existence of the closed-form formulae found in Section 4.4 of [15]. Some of the values for the percentage of resolvable pairs of cells (or $1 - \alpha(m)$) are listed in Table 3.2.

For a user-defined error bound $\epsilon$, we can find (using Table 3.2) the relevant number of density map levels needed to be visited so that the number of distances contained in the unvisited pairs of cells is below $\epsilon N(N-1)/2$. For instance, if the total number of SDH buckets is 32 and the desirable error bound is $\epsilon = 3\%$, Table 3.2 tells us that $m = 5$. In other words, to guarantee an error bound of 3\%, the algorithm needs to visit only five tree levels (not counting the starting level $DM_{o}$), and no single point-to-point distance computation is necessary.

Table 3.2 exceptionally corroborates Theorem 3.1: when $m$ increases by 1, $\alpha(m)$ practically halves itself, even for really small $l$ (as small as 2). Because the values for $1 - \alpha(1)$ in the first
Table 3.2: The % of cell pairs that are expected to be resolved. Different levels of density maps and total number of histogram buckets considered. Calculated in Mathematica 6.0. *Copyright © 2012, IEEE.*

<table>
<thead>
<tr>
<th>Map levels</th>
<th>Total Number of Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>50.6565</td>
</tr>
<tr>
<td>2</td>
<td>74.8985</td>
</tr>
<tr>
<td>3</td>
<td>87.3542</td>
</tr>
<tr>
<td>4</td>
<td>93.6550</td>
</tr>
<tr>
<td>5</td>
<td>96.8222</td>
</tr>
<tr>
<td>6</td>
<td>98.4098</td>
</tr>
</tbody>
</table>

row of the table are close to 0.5, the appropriate choice of \( m \) for guaranteed error bound \( \epsilon \) would be:

\[
m = \lg \frac{1}{\epsilon} \quad (3.3)
\]

By its design, it is easy to see that the cost of the ADM-SDH algorithm only relates to resolving pairs of cells on \( m + 1 \) tree levels. The cost can be deduced from Eq. (3.2). The total number of recursive calls to \texttt{RESOLVE}/\texttt{TWOCELLS} made by ADM-SDH is:

\[
T_c(N) = I \left[ \frac{2^{(2d-1)(m+1)} - 1}{2^{2d-1} - 1} \right] \quad (3.4)
\]

Plugging Eq. (3.3) into Eq. (3.4), we get:

\[
T_c(N) \approx I2^{(2d-1)m} = I2^{(2d-1)\frac{\lg \frac{1}{\epsilon}}{\lg \frac{1}{\epsilon}}} = I \left( \frac{1}{\epsilon} \right)^{2d-1} \quad (3.5)
\]
where $I$ is exclusively determined by the width of SDH buckets, $p$. Thus, we can conclude that the running time of our approximate algorithm, ADM-SDH, is only related to the user-specified error bound $\varepsilon$ and the SDH bucket width $p$, but is totally unrelated to the input data size $N$.

### 3.3.1 Heuristic Distribution of Distance Counts

In this section, we talk about what will happen to the distances coming from the pairs of cells that were not resolvable after visiting $m + 1$ tree levels. When we were estimating the error bounds of the ADM-SDH algorithm, we cautiously assumed that all of those “unresolved” distances would be incorrectly placed in an SDH bucket they don’t belong to, thus resulting in an error. But, considering the fact that we can allocate those “unresolved” distances to SDH buckets using some kind of heuristics, it is possible for some of them to be put in the correct bucket. In fact, many of them will end up in the correct bucket, which will substantially lower the previously estimated error. For example, let us consider a pair of non-resolvable cells, $M_1$ and $M_2$ with $n_1$ and $n_2$ as their particle counts, respectively. The total number of distances between these two cells will be $n_1 n_2$. $[u, v]$ represents the range of distances between particles in $M_1$ and $M_2$, with $u$ being the minimum and $v$ being the maximum distance. It is to be noted that $u$ and $v$ have already been calculated when trying to resolve $M_1$ and $M_2$. Now, let us assume that this range, $[u, v]$, overlaps with more than one bucket on the histogram. Figure 3.2 represents one such case, where $[u, v]$ overlaps with three buckets.

Following the case shown in Figure 3.2, we will present the three heuristics we used in our ADM-SDH algorithm to distribute the total of $n_1 n_2$ “unresolved” distances into the appropriate buckets. We describe the heuristics in descending order of the error they produce:
Figure 3.2: Distance range of two resolvable cells overlap with three buckets. Copyright © 2012, IEEE.

(1) *Skew heuristic:* always insert all $n_1n_2$ distance counts into one, predetermined bucket (e.g., put all the counts to the middle bucket);

(2) *Even heuristic:* always distribute distance counts evenly into the three buckets overlapping with $[u,v]$, so that each of the three buckets gets $\frac{1}{3}n_1n_2$;

(3) *Prop heuristic:* always distribute the distance counts proportionally, according to the portion of the overlap between $[u,v]$ and the given bucket. Applying this heuristic to the example in Figure 3.2, each of the three buckets, $i$, $i+1$, and $i+2$, gets $n_1n_2\frac{ip-u}{v-u}$, $n_1n_2\frac{p}{v-u}$, and $n_1n_2\frac{v-(i+1)p}{v-u}$ distance counts, respectively. It is clear that by using this heuristic, we are “enforcing” uniform statistical distribution of the point-to-point distances between the two cells, $M_1$ and $M_2$. The name PROP comes from proportional.

Obviously, by accepting the uniform distribution of point-to-point distances in the Prop heuristic, we put an oversimplified constrain to the system. The more accurate method would be to distribute the distance counts based on the actual distribution of distances between particles. But in order to compute that distribution, we need to know the actual spatial distribution of particles in the cells. This approach is very involved and includes very complex statistical interpretation of the spatial distribution of the particles and is outside the limits of this dissertation. However, the reader can look into [70], where there is a description of such heuristic.
It is worth mentioning that no matter what heuristic the ADM-SDH algorithm uses, its running time of will remain the same. The reason for this is that the resolution of two cells will be done in a constant time using any of the described heuristics.

3.4 Empirical Evaluation of ADM-SDH

The proposed ADM-SDH algorithm was implemented in C programming language and was tested on numerous real and generated MS data sets. The experiments were conducted on an Apple Mac Pro workstation with 8 GB of physical memory and two Intel Xeon dual-core processors running at 2.66GHz. The Mac Pro was running OS X 10.5 Leopard operating system. In the experiments, we set the algorithm to stop resolving cells after going through different number of density map levels and then distribute the remaining distance counts based on the three heuristics we described in Section 3.3.1. Then, we compare the histogram obtained by our approximate algorithm to the exact SDH obtained by the regular DM-SDH algorithm. By doing this we observe the error imposed by the approximate approach in the SDH computation.

Figure 3.3: The simulated hydrated bilayer system. We can see two layers of hydrophilic head groups (with higher atom density) connected to hydrophobic tails (lower atom density) all surrounded by water molecules (red dots). Copyright © 2012, IEEE.

Numerous generated and real data sets were used in the experiments. The real datasets were obtained from molecular simulations done on bio-membrane structures. Figure 3.3 shows one
such structure. The size of the data in the MS system used in our experiments varies from 50,000 to 12,800,000 atoms.

The generated data, on the other hand, was simulated from datasets with two different types of spatial particle distribution: 1) uniform distribution, which simulates a system with evenly distributed particles in space; and 2) Zipf distribution with order 1, which introduces skewness to the spatial particle distribution in the system.

We have used different parameters throughout the experiments. Their default values, as well as their ranges can be found in Table 3.3. The algorithm’s code together with the datasets used in the experiments presented in this dissertation can be found in [65].

Table 3.3: Parameters with ranges. Length unit: picometer. Copyright © 2012, IEEE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom count</td>
<td>100,000 – 8,000,000</td>
</tr>
<tr>
<td>Domain size</td>
<td>25,000 × 25,000</td>
</tr>
<tr>
<td>Maximum distance</td>
<td>35,355.34</td>
</tr>
<tr>
<td>Bucket width</td>
<td>25 – 4,000</td>
</tr>
<tr>
<td>Number of buckets</td>
<td>9 – 1,415</td>
</tr>
<tr>
<td>Data distribution</td>
<td>Uniform; Zipf; Real</td>
</tr>
<tr>
<td>Values of m</td>
<td>1 – 5</td>
</tr>
</tbody>
</table>

The results presented in Figure 3.4 show the running time of both the regular DM-SDH as well as the approximate ADM-SDH algorithm. The running times of the ADM-SDH shown here are only for a single value of the bucket width $p$, $p = 2500.0$. The running time of the regular DM-SDH is presented with the ‘Exact’ line. It is easy to see that the running time for DM-SDH grows polynomially with the dataset size $N$ with a slope of approximately 1.5. Figure
3.4(a) shows the running time of the algorithms after the tree has been constructed. It is obvious that the running time of ADM-SDH is unrelated to the dataset size $N$ - it does not change when the size of the data grows. One exception to this is the case when $m = 5$. As we can see from the figure, in this case the running time of ADM-SDH increases when $N$ is fairly small, but stays pretty flat (constant) after a certain value of $N$. The reason for this behavior is that when $N$ is small (less than let say 500,000 atoms) the resulting tree is very bushy and the algorithm has less than 5 levels of that tree to visit. When $N$ increases above certain threshold, though, the running time is not anymore affected by the change of $N$.

Figure 3.4(b) presents the total running time of the algorithms, which includes the tree construction times as well. We can see that when the algorithm visits only few levels (i.e., $m$ is small), the time to construct the quad-tree dominates the overall running time. The reason behind this is that the construction time of the tree is closely related to the dataset size $N$ (specifically, it is $O(N \log N)$). But, when the number of visited levels $m$ is greater than three, the shape of the curve almost resembles that in Figure 3.4(a), which is a clear indication that the running time of RESOLVETWOTREES dominates the overall running time.

The results produced in our experiments showed surprisingly high level of accuracy achieved by the ADM-SDH. Figure 3.5 plots the error rates resulting from the experiments in which ADM-SDH uses the three heuristics described in Section 3.3.1 and it is run on three different datasets (two generated: uniform and zipf, and one real dataset). It is clear that when $m$ increases the error rates decline. One interesting fact observed in the results shown in Figure 3.5 is that the error rate is below 10% throughout the experiments (using different datasets, size of data, heuristics and cases of $m$ - even including the case when $m = 1$). Such error rates are significantly smaller than the error bounds obtained from Table 3.2. As is to be expected, the
accuracy of the PROP heuristic is highest among all three mentioned heuristics, with EVEN being better than SKEW. Furthermore, the PROP heuristic produces very small error rates even when the algorithm visits very few levels (i.e., small values of $m$). As can be seen in figure 3.5, the error rate of ADM-SDH does not change when the data size $N$ increases. Such trend remains in effect for all three different sets of data. Moreover, and most notably when $m$ is fairly large, the PROP heuristic delivers a trend of decreasing error rates as $N$ increases. We suspect this to be a result of a possible neat feature of the PROP heuristic. Our insight on this feature: when the size of the data is very small, distributing the distance counts in individual operations can result in fairly large error rate. For example, in the extreme case when we only have a single distance needed to be distributed into (obviously one of) two buckets: putting the distance in the wrong bucket will result in error rate of 100% in that particular individual operation. So, we believe, considering the fact that the error compensation feature of the PROP heuristic lowers the total error rate to very small values (as can be seen in Figure 3.5), PROP heuristic has higher
Figure 3.5: Accuracy of ADM-SDH. Copyright © 2012, IEEE.

sensitivity to errors caused by individual distribution operations compared to SKEW and EVEN heuristics.

We understand that this feature can be very beneficial to future SDH algorithms and more in-depth study should be considered. However, at the present time, we believe it is beyond the extent of this dissertation and we put our focus only on the upper limit for the error.
3.4.1 Discussions

Considering the results of the experiments conducted using the approximate ADM-SDH algorithm, we can draw a conclusion that the ADM-SDH is an elegant method for solving the problem of SDH computation. Looking closely at the results, we can see that our algorithm delivers solution with very small error rates, even when it only visits very few levels of the quad-tree (even when $m = 1$). Thus, ADM-SDH is a simple, yet very efficient and surprisingly accurate in practice solution. Another important feature of ADM-SDH is that its running time is (almost totally) unrelated to the data size $N$. Only for small values of $m$ is the total running time dominated by the construction time of the tree.

The ADM-SDH algorithm delivers much better results in practice as compared to those expected by its theoretical analysis - the error rates of the experimental results are significantly lower than the error produced by our basic analysis. For instance, when $m = 1$ we can see in Table 3.2 that the expected error rate should be approximately 48%. But the error we got through the experiments for the same value of $m$ is not bigger than 10%. Moreover, the PROP heuristic delivers even better error rates, bringing the error down to 0.5%. Our justification for these low levels of error rate is as follows: during individual distribution operations using the three heuristics mentioned earlier, the algorithm may incorrectly put more distance counts into an SDH bucket (say bucket $i$ of Figure 3.2) than required. But following such operation, that may add too many counts into bucket $i$, there may be another distribution operation that may put too few distance counts in the same bucket $i$ than required. Thus, these two operations may cancel each other’s errors (or portion of them). The crucial part of the design of ADM-SDH is that the total error of a bucket is calculated after all such distribution operations have con-
cluded, essentially taking in consideration all the positive and negative errors from all individual operations. Based on what it does to the errors, we name such feature error compensation.

In order to get more insights of such feature, we definitely need to conduct more experiments with bigger pool of parameters (data size, data sets, bucket width, visited levels, spatial particle distribution, etc.) which, again, would be out of the scope of this dissertation. However, at this point, we can shed some light on it through some analytical investigation/reasoning.

Given the discussion above, we conclude that the error bounds shown in Table 3.2 are loose at best. The real, more accurate error bound should be given as

$$\varepsilon = \varepsilon' \varepsilon''$$

(3.6)

where $\varepsilon'$ represents the % of unresolved distances found in Table 3.2 and $\varepsilon''$ is the error rate imposed by the heuristics’ error compensation feature. In the following section, we analytically model and study the effect the error compensation has on the accuracy of the ADM-SDH algorithm. It turns out that the error compensation phenomenon dramatically boosts the ADM-SDH accuracy.

### 3.5 Performance Analysis of ADM-SDH

Given the fact that the error produced by our approximate algorithm is closely related to the spatial distribution of particles in the datasets, it has proven really challenging to derive a tight(er) error bound. The main reason behind this is that it is not trivial at all in formulating the spatial distribution of particles. On top of that, it is probably close to impossible deriving a closed-form formula for the distribution of distances between particles, which would
give a really tight bound on the error. Therefore, in this dissertation, we take the route of developing an analytical model to qualitatively analyze ADM-SDH’S behavior, especially targeting the mechanisms behind the errors. To make the analysis easier, we will impose two assumptions/constraints in the design of the analytical framework: 1) we assume that the spatial distribution of particles in the data is uniform; and 2) we only visit one level of the tree (start level) and that is the first level of the tree in which a cell has a side length smaller then or equal to $\sqrt{2}p/2$.

3.5.1 Distribution of Two Cells’ Distance

Let us consider two cells on a density map, named cell A and cell B. Let’s assume cell A is positioned in the $(t, j)$ field of the DM grid and cell B is positioned in the $(k, l)$ field of the same DM grid. Let $u$ be the minimum distance between these two cells, and $v$ be the maximum distance between them. We present the following lemma:

**Lemma 3.1** The range $[u, v]$ overlaps with at most three SDH buckets. Or, in other words, $p \leq v - u \leq 2p$.

**Proof.** Taking in consideration the above description of the two cells A and B, their minimum and maximum distances, $u$ and $v$ can be presented as functions of the cells’ position coordinates $(t, j, k, \text{and } l)$. There are three possible situations that we consider.

The first situation is when the two cells, A and B, have different row and column coordinates respectively, i.e., $i \neq k$ and $j \neq l$, the following equations can be written:

$$u = p\sqrt{\frac{(i-k-1)^2}{2} + \frac{(j-l-1)^2}{2}}$$  \hspace{1cm} (3.7)
\[ v = p \sqrt{\frac{(i-k+1)^2}{2} + \frac{(j-l+1)^2}{2}} \]  

(3.8)

The second situation is when the two cells are positioned in the same row, but not the same column (i.e., \( i = k \) and \( j \neq l \)), we can write the following equations for \( u \) and \( v \):

\[ u = p \frac{|j-l-1| \sqrt{2}}{2} \]  

(3.9)

\[ v = p \sqrt{\frac{(j-l+1)^2}{2} + \frac{1}{2}} \]  

(3.10)

The third situation is when the two cells are positioned in the same column, but not the same row (i.e., \( i \neq k \) and \( j = l \)), we can write the following equations for \( u \) and \( v \):

\[ u = p \frac{|i-k-1| \sqrt{2}}{2} \]  

(3.11)

\[ v = p \sqrt{\frac{(i-k+1)^2}{2} + \frac{1}{2}} \]  

(3.12)

We examine two cases in order to attain the proof of lemma 3.1.

(1) Cell A is found in the same row or column as Cell B. By Equations (3.9), (3.10), (3.11) and (3.12), we write

\[ v - u > 2p \sqrt{2}/2 > p \]  

(3.13)

and

\[ v - u = p \frac{4 \sqrt{2}r + 5}{2v + 2u} < p \frac{4 \sqrt{2}r + 5}{4u} = \left( \sqrt{2} + \frac{5}{4u} \right) p \]  

(3.14)
In the case when $u \geq 2\sqrt{2}p$, it is apparent that $\left(\sqrt{2} + \frac{5}{4u}\right)p < 2p$, therefore $v - u < 2p$. But when $u < 2\sqrt{2}$, the value of the quantity described by $v - u$ needs to be examined in a case by case manner. Luckily, the values that $u$ can get are series of discrete values only, all of which are smaller than $2p$:

- $u = 0$ (i.e., cell A is adjacent to cell B), then $v - r = v = p\sqrt{10}/2$.
- $u = \frac{\sqrt{2}}{2}p$, $v = \sqrt{5}p$, $v - u \approx 1.646p < 2p$.
- $u = \sqrt{2}p$, $v = \frac{\sqrt{34}}{2}p$, $v - u \approx 1.52p < 2p$.
- $u = \frac{3\sqrt{2}}{2}p$, $v = \sqrt{13}p$, $v - u \approx 1.5p < 2p$.
- $u = 2\sqrt{2}p$, $v = \frac{\sqrt{74}}{2}p$, $v - u \approx 1.5p < 2p$.

(2) When the two cells are positioned in different rows and columns (like in Figure 3.6), we divide the line segments $AE$ and $CD$, representing $u$ and $v$ respectively, in two parts by point $B$. Expressed through equations, we can write $u = CB + BD$, and $v = AB + BE$. Because $AC + CB \geq AB$ and $BD + DE \geq BE$, it follows that $AC + DE + CD \geq AE$, which is identical to $AC + DE + u \geq v$. Since $AC = DE = p$, we can conclude that $v - u \leq 2p$. But, because $u$ represents the minimum distance between cells A and B, and thus $FG \geq u$, the following is also true: $v - u = AE - CD \geq AF + GE \geq \sqrt{2}p > p$. 

Figure 3.6: Minimum and maximum distances between two cells. Copyright © 2012, IEEE.
The outcomes of the two cases presented above complete the proof of lemma 3.1.

\[\square\]

It is clear from lemma 3.1 that the maximum distance \(v\) has to fall in, either the bucket with range \([\left\lfloor \frac{u}{p} \right\rfloor p + p, \left\lfloor \frac{u}{p} \right\rfloor p + 2p)\) or the bucket with range \([\left\lfloor \frac{u}{p} \right\rfloor p + 2p, \left\lfloor \frac{u}{p} \right\rfloor p + 3p)\).

Let’s assume that the distances between particles from two cells on the density map adhere to a cumulative distribution function \(F\) over the range \([u, v]\). Following this assumption, Table 3.4 presents the probabilities that a distance falls into the relevant bucket of the histogram.

Table 3.4: The three buckets affected by distance distribution. Two non-resolvable cells considered. Copyright © 2012, IEEE.

<table>
<thead>
<tr>
<th>Bucket Range</th>
<th>Cumulative Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\left\lfloor \frac{u}{p} \right\rfloor p, \left\lfloor \frac{u}{p} \right\rfloor p + p))</td>
<td>(F(\left\lfloor \frac{u}{p} \right\rfloor p + p))</td>
</tr>
<tr>
<td>([\left\lfloor \frac{u}{p} \right\rfloor p + p, \left\lfloor \frac{u}{p} \right\rfloor p + 2p))</td>
<td>(F(\left\lfloor \frac{u}{p} \right\rfloor p + 2p) - F(\left\lfloor \frac{u}{p} \right\rfloor p + p))</td>
</tr>
<tr>
<td>([\left\lfloor \frac{u}{p} \right\rfloor p + 2p, \left\lfloor \frac{u}{p} \right\rfloor p + 3p))</td>
<td>(1 - F(\left\lfloor \frac{u}{p} \right\rfloor p + 2p))</td>
</tr>
</tbody>
</table>

### 3.5.2 Compensating the Distance Counts in the Skew Method

As pointed out earlier, one phenomenon we observed during our experiments of the ADM-SDH is, what we called, error compensation. This is the mechanism because of which, we believe, our approximate algorithm produces significantly lower error rates than the expected, theoretically derived ones. And our explanation for it is that the errors produced by one distribution operation may be (partially) canceled by those of another distribution operation. In this dissertation, without loss of generality, we will use the Skew heuristic to probe this mechanism and derive some type of analytical evaluation for it.

When using the Skew heuristic, the algorithm will put all distance counts into one bucket. Assuming the Skew is designed such that it puts all distance counts to the bucket with the
smallest bucket index, the distances belonging to the range overlapping with the three buckets in Table 3.4 will be put into the bucket with range $\left[\lfloor \frac{u}{p} \rfloor p, \lfloor \frac{u}{p} \rfloor p + p \right)$. If only this single distribution operation is used to calculate the error, it is obvious that the resulting error will be high: if $e$ is the error, we get $e = 1 - F(\lfloor \frac{u}{p} \rfloor p + p)$ for the bucket $\left[\lfloor \frac{u}{p} \rfloor p, \lfloor \frac{u}{p} \rfloor p + p \right)$. In this example, the algorithm had overestimated the number of counts in the first bucket, i.e., the error $e$ is positive. But, there might be other distribution operations that will ‘take away’ distance counts out of that same bucket (i.e., put too few), essentially compensating for the overestimation of the positive error. For instance, let us consider a distribution operation with minimum distance $u_1 = u - p$. This operation would move out some of the counts belonging to the first bucket in Table 3.4, generating a negative error and therefore (partially) canceling out the positive error mentioned above. Following this idea, our goal will be to recognize such distribution operations that compensate each other’s errors and find out how much of error compensation we can have.

First order of business is to show that in an ideal set of circumstances, the error can be zero.

**Lemma 3.2** Given a distribution operation with minimum distance $u$, if there exists another distribution operation with minimum distance $u_1 = u - p$, the error produced by ADM-SDH using the Skew heuristic will be zero.

**Proof.** According to Table 3.4, the error imposed to the first SDH bucket (bucket $i$) by any distribution operation is $1 - F(\lfloor \frac{u}{p} \rfloor p + p)$. And this is an overestimation of the counts, i.e., the error is positive. Let us assume that there exists another distribution operation with a minimum distance $u_1 = u - p$. The error imposed to the same bucket $i$ by this new operation would be $F(\lfloor \frac{u}{p} \rfloor p + 2p) - F(\lfloor \frac{u}{p} \rfloor p + p)$. Note that this is an underestimation, i.e., the error will be negative. Following the same reasoning, the error imposed to the same bucket $i$ by a third operation with minimum distance $u_2 = u - 2p$, would be $1 - F(\lfloor \frac{u}{p} \rfloor p + 2p)$. This too is a
negative error. Summing up the values of the three aforementioned errors, it is clear that the resulting error of bucket $i$ generated by the three distribution operations is zero.

Figure 3.7 depicts one such example where the effects of the error compensation mechanism produces a zero error. Namely, cells $C$ and $C'$ will contribute to each other’s error canceling, when the minimum distances $AC$ and $AC'$ are computed.

No matter how good the previous analysis seems, it is not always the case that the errors will totally cancel each other out. The main problem is the lack of the existence of an $u_1$ value that equals $u - p$ for every single cell pair. Besides this, we can still deduce, from Lemma 3.2, that the error is greatly related to the minimum distance quantity $u$. So, in the following text we analyze the mechanisms of partial error compensation by adjacent pairs of cells in the DM grid.

Let us consider two cells on the same level of the density map, with a horizontal distance of $x$ cells and a vertical distance of $y$ cells in between them. Cells $A$ and $B$ in Figure 3.7 is one such pair of cells.

Figure 3.7: Pairs of cells that lead to total or partial error compensation. Copyright © 2012, IEEE.
Without loss of generality and for the ease of presentation, we set the SDH bucket’s width \( p \) to be a unit value (\( p = 1 \) unit). Following this, the length of a cell’s side will be \( \sqrt{2} \). Similarly, the vertical and horizontal distances between cell \( A \) and cell \( B \) become \( u_{vertical} = \frac{\sqrt{2}}{2} y \) and \( u_{horizontal} = \frac{\sqrt{2}}{2} x \) respectively as demonstrated in Figure 3.7. If all of the above is taken in consideration, the minimum distance, \( u \), between cells \( A \) and \( B \) can be computed as follows:

\[
u = \sqrt{u_{horizontal}^2 + u_{vertical}^2} = \sqrt{\frac{x^2}{2} + \frac{y^2}{2}} \quad (3.15)
\]

Crucial information that can significantly help our analysis is attained by examining another cell, i.e., cell \( B' \) in Figure 3.7. The minimum distance from \( A \) to this new cell \( B' \) is \( u_1 = \sqrt{(x-1)^2 + (y-1)^2} \). For the convenience of the presentation, we will use \( \Delta \) in place of the quantity \( u - u_1 \). We have

\[
\Delta = u - u_1 = \frac{(u - u_1)(u + u_1)}{u + u_1} = \frac{u^2 - u_1^2}{u + u_1} = \frac{x^2 + y^2 - \frac{(x-1)^2}{2} - \frac{(y-1)^2}{2}}{u + u_1} \approx \frac{x + y - 1}{2u} \quad (3.16)
\]

Assuming \( x \geq y \) and \( z = y/x \), Equation (3.16) becomes:

\[
\Delta \approx \frac{x + y - 1}{2u} = \frac{x + y - 1}{2\sqrt{\frac{x^2}{x} + \frac{y^2}{y}}} = \frac{1 + \frac{y}{x} - \frac{1}{x}}{\sqrt{4\left(\frac{y^2}{x^2} + \frac{y^2}{x^2}\right)}} = \frac{1 + z - \frac{1}{x}}{\sqrt{2 + 2z^2}} \approx \frac{1 + z}{\sqrt{2 + 2z^2}} \quad (3.17)
\]

Even though \( x \) and \( y \) are integers, \( z \) can still be viewed as a continuous variable. The reason behind this is that there is big number of possible \( x \) and \( y \) values in a density map with many
cells. We can conclude that $\Delta$ grows with $z$, given that $d\Delta/dz > 0$. There are two border cases here: 1) when $y = 0$, then $z = 0$ and $\Delta = \sqrt{2}$; and 2) when $y = x$, then $z = 1$ and $\Delta = 1$.

In accordance with lemma 3.2, we can approximate the error generated by the Skew heuristic, $e_{\text{skew}}$, as the difference of one and $\Delta$, i.e., $e_{\text{skew}} \approx 1 - \Delta$. Since $\Delta$ varies from $\sqrt{2}/2$ to 1, the ‘skew’ error, $e_{\text{skew}}$, varies from 0 to $1 - \sqrt{2}/2$. We can observe such compensating process in Figure 3.7. In the same figure, we also see that the difference between minimum distances of $AC$ and $AC'$ is one, and that the difference between minimum distances of $AB$ and $AB'$ is different than one (smaller than one to be more precise).

![Figure 3.8: Distance compensation between two pairs of cells. Copyright © 2012, IEEE.](image)

Let us consider two pairs of cells, $(A, B)$ and $(A, B')$ with their minimum distances $u$ and $u_1$, respectively (as in Figure 3.7). By now, we are aware that every such pair of minimum distances $(u, u_1)$ will generate an error if their difference is not equal to one (i.e., if $u - u_1 \neq 1$). In the following text we are going to do a quantitative approximation of the error produced by such pair of minimum distances. We will also demonstrate that the summation of all such errors (i.e., the total error) is qualitatively constrainable and furthermore, that is fairly low.

In order to analytically describe a quantity $(1 - \Delta \text{ or } 1 - (u - u_1))$, we would have to take advantage of its underlying distribution. We can view the distribution of distances between particles from $A$ and $B$ ($B'$) as noncentral chi-squared. With no loss of any generality, we
designate one point from cell A to be the base point with coordinates \((0, 0)\). Now, we can regard the distribution of distances between points from cell \(B\) (or \(B'\)) and this base point, as triangular.

We will use Figure 3.8 in order to geometrically analyze the mechanism of error compensation that led to low error rates in our algorithm. The three triangles in Figure 3.8 depict the triangular density distributions of distances between the base point and the points of three cells. Each of the triangles’ bases represents the \([\text{min, max}]\) range of distances between points of a particular cell and the base point. We define the following identities we will use throughout the analysis: \(G = [u]\), \(H = G + 1\), and \(I = H + 1\).

The triangles \(STU\), \(AEW\) and \(BCF\) represent the density distributions of \(u\), \(u_1\) and \(u - 1\), respectively. The vertical line passing through the point \(D\) represents the line of symmetry for lines \(H'S'D'\) and \(WFH\).

As we saw earlier, for the case of \(u_1 = u - 1\), the error generated from the range between \(H\) and \(I\) may be totally compensated by the error generated from the range between \(G\) and \(H\). Or, following the SKEW heuristic, when the minimum distance between two cells is \(u\), all of the distance counts from the range \([H, I]\) will be added to a certain bucket and that will lead to a positive error (at least in that bucket). However, when the minimum distance between two other cells is \(u - 1\), all of the distance counts from the range \([G, H]\) that should have fallen in the same bucket, will be missed and that will lead to a negative error (in that same bucket). Assuming they have same distribution, the area of \(GBCFH\) will be equal as the area of \(HSTUI\), thus no error is generated.

In the case when \(u_1 \neq u - 1\), the area of \(GAEW\) will not be equal to the area of \(HSTUI\). Thus, there will be a difference between the areas of \(GAEW\) and \(GBCFH\). We can calculate the said difference through the area of \(ABD'S'\). At the same time, that area also represents the
error imposed by the difference of \( u_1 \) and \( u - 1 \). We denote that error as \( e_{u_1, u-1} \). It is clear that \( CE = u_1 - u + 1 \) and \( GH' < u - u_1 = \Delta \). We know (by theory) that each of the three triangles in Figure 3.8 has an area of one. Because of this, and since the base of each triangle is \( v - u \), their height is given by \( \frac{2}{v-u} \). Thus, the ratio \( \frac{AB}{CE} \) can be computed as (more details can be found at the end of this section under Similar Triangles Ratio):

\[
\frac{AB}{CE} = \frac{\frac{2}{v-u}}{\frac{v-u}{2}} = \frac{4}{(v-u)^2}
\]

(3.18)

Moreover, the length of \( AB \) can be calculated as follows:

\[
AB = \frac{4}{(v-u)^2} \ast CE = \frac{4(u_1 - u + 1)}{(v-u)^2} = \frac{4(1 - \Delta)}{(v-u)^2}
\]

(3.19)

Considering the previous equations, the area of \( ABD'S' \), and therefore the error \( e_{u_1, u-1} \), can be calculated as:

\[
e_{u_1, u-1} = AB \ast GH' < \frac{4(1 - \Delta) \ast \Delta}{(v-u)^2}
\]

(3.20)

\[
e_{u_1, u-1} < \frac{(1 - \Delta)}{\Delta} = 1 - 1
\]

(3.21)

So the total (accumulated) error, \( e_{z=0,1} \) over the range of \( z (z \in [0, 1]) \) and considering Equation 3.17 for \( \Delta \), can be calculated as follows:

\[
e_{z=0,1} = \sum_{z=0}^{1} e_{u_1, u-1} = \sum_{z=0}^{1} \left( \frac{1}{\Delta} - 1 \right)
\]

\[
\approx \sum_{z=0}^{1} \left( \frac{\sqrt{2 + 2z^2}}{1 + z} - 1 \right)
\]

(3.22)
When $0 \leq z \leq 1$, we can take advantage of some traditional mathematical tools (like Matlab, used in Figure 3.9) to approximate $e_{z=0,1}$ and get:

$$e_{z=0,1} \leq \sum_{z=0}^{1} (0.211 \times (1-z)^5 + 0.211 \times (1-z)^2) \quad (3.23)$$

As mentioned earlier, we can treat $z$ as continuous variable and thus, we get:

$$e_{z=0,1} \leq \int_{0}^{1} (0.211 \times (1-z)^5 + 0.211 \times (1-z)^2) \, dz = 0.1055 \quad (3.24)$$

Equation (3.24) serves as a qualitative proof showing that the total error generated by our approximate algorithm utilizing the SKEW heuristics and following the assumptions stated at the opening part of Section 3.5, is lower than 10.55%. Naturally, because of the assumptions made to get to this result, we do not claim it to be a very rigorous bound of the error. Nevertheless, it sheds some insight of the mechanisms of ADM-SDH, showing it is capable of delivering an efficient, yet pretty accurate solution to the SDH problem by only visiting a single level of the Quad-tree (the Density Map). This conclusion leads us to the possibility of an improvement to the ADM-SDH algorithm and will serve as a foundation upon which we will build our new and improved approximate algorithm (as seen in the next section, Section 3.6).

It is worth mentioning that Equation 3.24 does not include the cases when the minimum distance $u$ is smaller than the bucket width $p$ (i.e., $u$ falls in the first bucket of the histogram). But, further analysis presented below shows that such types of cases will not considerably affect the result presented in Equation 3.24.
Let us consider the boundary situation in the error compensation analysis. When the minimum distance \( u \) is within the first bucket, i.e. \( u \leq p \), the error produced by SKEW cannot be compensated in the way described in Section 3.5. The reason being, there are no buckets to the left of the first bucket, therefore the positive errors in it cannot be compensated. The average error caused by this, denoted as \( e_{u \leq 1} \), can be computed as follows

\[
e_{u \leq 1} = \frac{0.3N^2}{2N(N - 1) + N^2} \approx 0.1
\]  

(3.25)

Therefore, by considering the Equations (3.25) and (3.24), the average error over all buckets of our algorithm based on SKEW can be computed as follows:

\[
e_{final} = \lim_{n \to \infty} \frac{e_{u \leq 1} \cdot \tau + e_{z = 0, T} \cdot (n - \tau)}{n} = \lim_{n \to \infty} \frac{0.1 \cdot \tau + 0.1055 \cdot (n - \tau)}{n} = 0.1055
\]

Figure 3.9: Approximation of \( \frac{1}{3} - 1 \) obtained in Matlab. Copyright © 2012, IEEE.
where \( n \) is the total number of distances and \( \tau \) is the number of the distances in the first bucket.

In the following text we describe the similar triangles ratio used in Equations 3.18 and 3.19. Figure 3.10 represents the two leftmost triangles from Figure 3.8. These two triangles, triangles \( MNC \) and \( KPE \), are equivalent, with an area of one. Their sides \( MC \) and \( KE \) are parallel to each other. This imposes that \( MK = CE \) and \( KK' = AB \). Following the values from Figure 3.8, we note the following:

\[
MK = u_1 - u + 1 = 1 - \Delta, \quad MN = v - u, \quad MC' = \frac{MN}{2} = \frac{v-u}{2}, \quad CC' = \frac{2}{MN} = \frac{2}{v-u} \quad \text{(from the formula of the area of the triangle: 1 = \( \frac{1}{2}MN \ast CC' \)).}
\]

![Figure 3.10: Similar triangles ratio.](image)

Now, let us look at triangles \( MKK' \) (the red triangle) and triangle \( MC'C \) (the green one). These two triangles are similar right triangles. Following the properties of similar triangles we get the following:

\[
\frac{K'K}{MK} = \frac{CC'}{MC'} = \frac{2}{\frac{v-u}{2}} = \frac{4}{(v-u)^2}
\]

\[
AB = K'K = \frac{4}{(v-u)^2} \ast MK
\]

\[
= \frac{4}{(v-u)^2} \ast (1 - \Delta) \quad (3.26)
\]
3.6 Single Level Approximate Algorithm

As shown in the previous section through the analysis of the ADM-SDH algorithm, and concluded by the error bound shown in Equation 3.6, we see that the ADM-SDH algorithm delivers very efficient solution with low error rates. Even in the case when resolution of cells is not possible (i.e., $\epsilon' = 100\%$), the algorithm still produces results with controllable and low error rate. Encouraged by such observation, we have designed and implemented a new and improved approximate algorithm, named single level SDH algorithm, or SL-SDH. We point out two critical differences that set apart these two algorithms: 1) the number of visited levels of the quad-tree (or density maps): ADM-SDH goes through $m + 1$ levels, whereas the SL-SDH visits only one tree level, thus the name Single Level SDH Algorithm. The level that is inspected by the SL-SDH is defined by the user and it can be any tree level. 2) the starting level: ADM-SDH has a fixed, predetermined starting level, defined by two values, the maximal distance between any two points in the system and the width of the histogram bucket $p$. On the other hand, the starting (and the only visited) level of the SL-SDH can be any level of the quad-tree and it is the same, user defined value as in 1). Based on these important differences, we can note two essential aspects for which the SL-SDH is an improvement over the ADM-SDH. First, we don’t need the whole Quad-tree in order to run the SL-SDH. We only need a single level of it (or a single density map). This single density map can be built in $O(N)$ time. Unlike that, the ADM-SDH needs the whole Quad-tree, which takes $O(N \log N)$ to build. And second, since SL-SDH only calls the RESOLVETWOTREES procedure for cells on a single density map (single tree level), it lowers the running time after tree construction without (significant) introduction of extra error.
One crucial feature that the SL-SDH has, is the fact that its running time is unrelated to $p$, the width of the histogram bucket. Unlike it, we saw earlier that the ADM-SDH algorithm, by its design, starts at a density map $DM_0$ at which the cells have diagonals smaller than or equal to the bucket width $p$. This leads to a lot more calls that ADM-SDH has to make to the RESOLVETWOCELLS procedure, because for small $p$ values, there will be a large number of cells in the starting density map $DM_0$.

Our remedy for this issue, is that SL-SDH is allowed to run on any density map, even one that has lower resolution than $DM_0$ (i.e., one with bigger, thus fewer cells). We propose the aforementioned remedy based on the results of the performance analysis of ADM-SDH found in Section 3.5: we suspect that the error compensation mechanism of the ADM-SDH will work even for density maps above $DM_0$. We saw that the errors induced by RESOLVETWO TREES procedure ran on cells from $DM_0$, is low. So, applying the same mechanism to a higher-level density map should still produce reasonable (even though probably bigger) error. Having said that, we know that an analytical examination of such error would be extremely challenging. However, in the next section, we do experimentally evaluate the error and efficiency/accuracy trade offs of the new, SL-SDH algorithm.

3.6.1 Experimental Results

The proposed SL-SDH algorithm was implemented using the C programming language and was tested on various real and generated MS data sets. The experiments were run on an Apple Mac Pro workstation with 8 GB of physical memory and two Intel Xeon dual-core processors running at 2.66GHz. The Mac Pro was running OS X 10.5 Leopard operating system. In the experiments, we set the algorithm to only visit a single level of the Quad-tree. And we record
few runs of the algorithm with different (starting) levels of the DM. After resolving cells in the starting/only level the SL-SDH visits, the algorithm distributes the remaining distance counts based on the three heuristics we described in Section 3.3.1. We then compare the histogram obtained by our single level approximate algorithm to the exact SDH obtained by the regular DM-SDH. By doing this we observe the error imposed by the approximate approach in the SDH computation.

Figure 3.11: Accuracy of SL-SDH using different bucket width. Generated (uniform and skewed) and real data considered. Copyright © 2012, IEEE.

Based on the experiments run on the ADM-SDH (Section 3.4), we know that the best heuristic among the three (Prop, Skew, Even) for distributing the distance counts from unresolved
Figure 3.12: Accuracy of SL-SDH for synthetic data. Different bucket width and different atom counts for synthetic data (uniform and skewed) considered. Copyright © 2012, IEEE.

pairs of cells, is the PROP heuristic. So, in the experiments of our improved approximate SL-SDH, we have only incorporated the PROP heuristic (since anyways it produces results with the smallest error). The SL-SDH was run on two generated MS datasets, one with uniform and one with skewed spatial distribution of particles and in both cases with five different data sizes \( N \), i.e., we tested 1, 3, 5, 7.5, and 12 million particles. Furthermore, we have run the SL-SDH on one set of real MS data, containing 891,272 particles.

The top and middle part of Figure 3.11 represents the experimental results obtained when SL-SDH is run on a generated dataset with 7.5 million particles with uniform (top part) and skewed (middle part) spatial particle distribution. The bottom part of Figure 3.11 shows the results when SL-SDH is run on the real data set with 891,272 particles. It is obvious, from this
figure, that the error rate is related to the level of the density map in which SL-SDH operates. Specifically, the error rate decreases when a higher level of density map is selected. We also note that the error rate is inversely proportional to the bucket width.

It is worth pointing out that both experiments, on generated as well as real data, follow the same trend, showing that the error rate is independent of the number of particles in the dataset and that it decreases when the level in the density map increases.

Figure 3.12 sheds some light on the effects the data size $N$, as well as the selected level can have on the accuracy of the SL-SDH algorithm. Each line in Figure 3.12 represents the error induced by the algorithm when it is run on a dataset of a particular size as well as on a particular level of the quad-tree (density map). It is noticeable that the results (lines) of the five different size datasets obtained from the same density map are very close to each other. Such five lines of different density maps will essentially create clusters (each of five lines), each cluster representing the results from a particular density map (or a quad-tree level). So, we can deduce from Figure 3.12, that the accuracy (error rate) of the SL-SDH is closely related to the level of density map the algorithm uses, and is totally unrelated to the size of the datasets, $N$.

Furthermore, it is very important to note that, as shown in Figure 3.13, the running time of the SL-SDH algorithm is also unrelated to the dataset size $N$. There are two exceptions to this observation, however: namely, for levels 3 and 4, the running time seems related to $N$. But that is because the overall running time of the SL-SDH algorithm for these levels is dominated by the tree construction time.

One way to compare the efficiency/accuracy of the SL-SDH to that of the ADM-SDH algorithm, is to look and compare the figures representing their respective result, namely Figure 3.13 with Figure 3.4, and Figure 3.12 with Figure 3.5. But in order to better understand
their efficiency/accuracy tradeoffs, we have designed a new metric named Error Delay Product (EDP). We define the EDP as the product of the error rate and running time of the algorithm. It is clear that the lower the EDP of an algorithm the better performance/accuracy tradeoffs that algorithm has. Figure 3.14 represents the EDPs for ADM-SDH as well as SL-SDH executed with four different values for the SDH bucket width (i.e., $p = 100, 500, 1000, 2000$). Figure 3.14 clearly shows that the EDP of the SL-SDH is better (lower in value) than the EDP of the ADM-SDH (which is higher in value). This is particularly true for small values of the bucket width $p$: in such cases, the EDPs of SL-SDH are significantly smaller than the EDPs of the ADM-SDH. The logic behind this is that the ADM-SDH, by its design, starts at a predetermined level that is (very much) defined by the width of the bucket $p$. SL-SDH, on the other hand, can start at any user defined/desired level. This is the reason behind the different number of lines (representing different visited levels) plotted in in the ADM-SDH part in Figure 3.14 for different bucket widths (for instance, there is only one line (one level visited) when the width of the bucket is 100; but there are 3, 4 or 5 lines plotted when the bucket width is 500, 1000 or 2000

![Figure 3.13: Efficiency of SL-SDH.](image-url)

Figure 3.13: Efficiency of SL-SDH. Copyright © 2012, IEEE.
respectively). On the contrary to that, the number of lines plotted in the SL-SDH part in Figure 3.14 do not change with different bucket widths. To elaborate on this: in the case of small $p$, ADM-SDH has to start at a lower level of the quad-tree (higher in numeration, actually) \(^1\) and this level has more pairs of cells needed to be resolved (as compared to any other higher level). This increases the number of calls ADM-SDH makes to the \texttt{RESOLVETWOCELLS} procedure, which leads to increased running time, and consequently increased EDP.

As mentioned earlier and as shown in Figure 3.14, when the width of the bucket is 100, the starting level of ADM-SDH is level 9. As a matter of fact, that is the only level it visits, due to the fact the quad-tree (with so many particles in it) has only 9 levels (because of the tree building constrain that each leaf node has to have more than four particles in it). But the running time when working on level 9 is very long, given the huge number of pairs of cells that need to be processed. On the other side, SL-SDH can process any one (single) level that the user picks and usually this means any level that produces results with satisfying accuracy yet acceptable running time. It is evident, by looking at the higher bucket width parts in Figure 3.14, that the EDPs of the two algorithms get ever so closer as the width of the bucket increases. Even so, we can always find an EDP of the SL-SDH algorithm that is better (lower in value) than that of the ADM-SDH.

Looking at the bigger picture, we note that the EDPs of both ADM-SDH and SL-SDH decline when the density map level decreases (with level 9 being the worst of all levels). Nevertheless, SL-SDH has one big advantage over the ADM-SDH: it can start at any level selected by the user and, as shown by our experiments, even when it works on a coarse density map its accuracy is remarkably high.

\(^1\)Just to recall: ADM-SDH starts at the level at which the diagonal of the cell is smaller then or equal to the bucket width
Figure 3.14: Accuracy and performance tradeoffs of ADM-SDH and SL-SDH. Different SDH bucket widths considered. Copyright © 2012, IEEE.

To recap, the results of our experiments bring three important observations in view: 1) the error rate as well as the running time of the SL-SDH algorithm are unrelated to the data size $N$. 2) based only on the user-desired accuracy, the user selects the (single) level which SL-SDH processes. And probably the most notable observation is 3) the SL-SDH algorithm considerably boosts the performance/accuracy tradeoffs of the ADM-SDH, which is especially the case when the width of the SDH bucket is small. This is very significant because the smaller
the bucket width the more desirable the SDH result is, as it holds bigger knowledge of the underlying spatial distribution of the distances.
Chapter 4: Push-based MS Data Analysis System

Much of the MS data analysis in today’s scientific research is done using one of the mainstream systems like Gromacs, MDAnalysis, Charmm, etc. Many scientists in different fields are using these systems to get as much insight as possible of the system under study. But, as mentioned in Chapters 1 and 2, to the best of our knowledge, all of these systems are based on the pull-based idea. This means that the queries, when executed, are requesting the data they need and the system provides them with it. This approach, however, introduces I/O traffic overhead and cpu/data latency that, in turn, reduces the efficiency of such systems. We believe that using a different approach, one that incorporates the idea of a push-based design, can remedy the aforementioned issues and provide a much more efficient way of data analysis.

4.1 Motivation and the Shortcomings of a Pull-based Design

The motivation for the work presented in this chapter comes from two different observations. First, the features that are shared among many of the data-intensive scientific applications. As it has been reported in some previous projects [37, 41, 57, 59], and through our own research as well as collaboration with scientist working in the field of MS data analysis, we believe that the following list represents such features:
(1) The amount of data is very large, and data is usually stored in legacy file format,

(2) Large portion of data is queried as well as returned as a result of the querying. Many analytical queries demand scans of the whole dataset on which complex function are being evaluated,

(3) Data, once stored, remains static (unchanged), with new data appended to the existing dataset,

(4) The number of prominent and most commonly used queries executed against the scientific data is fairly limited.

The first two features are linked to high demand on both CPU and I/O capacity. They also reflect the challenge in achieving high data throughput in systems where multiple queries need to be executed simultaneously. The last two features imply mostly "stationary" system with static data and reasonably small pool of queries. We specifically want to stress out that feature (4) is a routine one for many of the scientific domains. For example, in the fields of bio-sequences [57] and molecular simulations [44], most of the data analysis is done through relatively small number of low-level analytical queries. Also, many of the functions most commonly used in the field of data mining are built on a handful of statistical kernels as described in [16]. Furthermore, in astronomy, a methodical examination [59] on the Sloan Digital Sky Survey (SSDS) database traffic reveals highly skewed user queries towards a very small query subset (out of the whole query space) and 50 of these queries are presented on the SSDS website [1].

The second observation that motivated us to do this work is the design of the data management and/or data analysis systems used for scientific data analysis. Since their inception, almost 50 years ago, the database systems have been designed as "compute-centric" systems.
This means that when a query needs certain data for execution, it sends a request to the system for that data. The system, in turn, finds the chunk of data requested and delivers it to the query (the CPU, specifically). This design is also known as pull-based. Such a pull-based design is also being used by the mainstream data analysis systems (like Gromacs). This approach, when applied to read-mostly, analytical processing frameworks such as the scientific simulations, we believe imposes two specific types of problems:

(1) No efficient way exists for data request coordination when multiple queries are being executed in a pull-based environment. So, every query that requests chunk of the data, introduces its own data access latency, as the data is being pulled through the system’s memory hierarchy. Additionally, each such request will eat up extra memory bandwidth, used every time a data record is being transferred from memory to the CPU. Therefore, the I/O resources used by the system grow linearly at best with the amount of the executed queries.

(2) A system that uses the pull-based design will exhibit loss of CPU cycles even when a single query is being executed. The reason: memory access latency. By the fundamentals of a pull-based design, every time a query needs data, it requests it and, if the data is not already in the CPU, the query/CPU remains idle until the data becomes available.

So, when these two observations, or aspects of the scientific data analysis are put together, they magnify the aforementioned issues imposed by their respective reasons. For instance, knowing the fact that, in a pull-based design, each query request its own data which is then pulled through the system, and adding on top the fact that in scientific data analysis, the data requested by such query is of very large volume, the introduced data access latency will be
much bigger (amplified by combining the scientific data analysis feature with the feature of the pull-based design).

4.2 The Remedy: Push-based Design

The problems described in the above text, gave rise to the fundamental principle behind the design of our system. Namely, that principle is to maximize the resource sharing (mostly I/O and memory bandwidth) between queries running concurrently in the system. The fact that usually a handful of queries are repeatedly used, implies that the data accessed by the "different" queries will largely overlap. This, in turn, indicates that the I/O sharing is indeed feasible and will be very effective. Furthermore, we can take advantage of the existence of commonly used analytical kernels, by pre-computing them so that their result may then be used in the computation of higher-level analytical queries. We achieve the aforementioned principle through a push-based data analysis system that we describe, design, implement, and test in the following few sections.

The push-based design, essentially envelops a scan-based I/O framework. The data is sequentially read and fed to the CPU. This kind of table/file sequential read, we believe, can really improve the data throughput of a data analysis system. The following observations back up our beliefs:

1. Using this approach we can greatly avoid random I/Os, which are known to depreciate the overall data throughput. As per [35], index-based scans are better than sequential scans only in the cases when the data being accessed falls below 0.08% of the whole database (in a classic storage system). Often, large segments of data are being accessed/processed by queries in scientific databases. Instances of analytical queries and non-index search
queries having wide query range present the most notable examples of such queries. The popularity of the mentioned search queries in vast spectrum of scientific domains is presented in [52, 72].

(2) The push-based approach will allow the query processing cost to be more or less (evenly) distributed among many queries. A number of, otherwise concurrent, queries can be run on the same data. This is especially true if the data is “new”, i.e., just pushed onto the queries (CPU). This also can benefit the fact that there are many commonly computed analytical kernels which can be processed on the same data stream way before the user queries come to the system. The results of such pre-computed kernels can then be used for the computation of the more involved analytics. And since, many of the high-level analytical queries require the read of the whole data to be able to be processed, it further backs up the idea of the kind of preemptive execution on as many queries as possible while the data scanning is taking place,

(3) One peculiar characteristic of many scientific databases is that the system does an initial scan in order to upload the data from some sort of a remote source, like simulation programs, to the database. We believe this is something we can take an advantage of, and, in a way get a “free” scan of the data and process as many queries as possible on it.

So, in summary, we believe that a push-based design will be a big improvement over the commonly used pull-based design that is almost exclusively incorporated in today’s mainstream software for scientific data analysis.

Motivated by the above observations, we have designed and implemented a push-based MS data analysis system that we present in this chapter. We have exhaustively tested our system
on various data sets and numerous query loads. The results of our experiments concur with our believe that this approach offers better, more efficient method for MS data analysis.

4.3 Network of Queries

In order to study some important statistical features of an MS system, scientists need to “extract” various statistical quantities out of the data produced by the simulation. To achieve this, queries are executed against the data. Most of the queries used in the analysis of MS systems are analytical in nature. Essentially, these analytical queries are mathematical functions that translate a selection of atoms (atoms’ measurements) to a scalar, vector, a matrix, or a data cube [21]. Once the simulation is done, the analysis carried out will depend on the structure being studied as well as the features of the system that need exploring. In other words, not all system’s quantities need to be computed every time the system is being analyzed. Some of the more popular queries, including density (atom counts), first-order statistics (mean), second-order statistics (variance), and histograms among others, can be seen in Table 4.1. The queries shown in this table are the ones that we have also incorporated in our system. Just to clarify some of the notation in Table 4.1: we assume that the MS system comprises of $n$ particles and $r_i$, $m_i$, $c_i$ and $q_i$ denote coordinates (vector form), mass, charge, and number of electrons of a particle $i$, respectively.

There are two types of queries/functions among the ones used to analyze an MS system. The first type is one-body functions. Such functions usually are algebraic functions [63] and only involve quantities (attributes) from a single atom at any given time in the process of computation. Each atom (atom’s attributes) is being processed a constant number of times, thus the total running time of such functions/queries is $O(n)$. This type of functions is very
suitable for the idea of push-based system, or an online system in which the data is being read once and acted upon. In other words, in a single run of the incoming data, all such queries will produce useful final results. Except the SDH and the RDF (i.e., the last two in the table), all other queries shown in Table 4.1 fall into this category of one-body functions. Most of these functions are defined on a single frame of the MS data. Only the autocorrelation functions are defined on two distinct frames.

Table 4.1: Popular analytical queries in MS.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Equation/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moment of Inertia</td>
<td>( I = \sum_{i=1}^{n} m_i r_i )</td>
</tr>
<tr>
<td>Moment of Inertia on z</td>
<td>axis ( I_z = \sum_{i=1}^{n} m_i r_{i} )</td>
</tr>
<tr>
<td>Sum of masses</td>
<td>( M = \sum_{i=1}^{n} m_i )</td>
</tr>
<tr>
<td>Center of mass</td>
<td>( CoM = \frac{I}{M} )</td>
</tr>
<tr>
<td>Radius of Gyration</td>
<td>( RG = \sqrt{\frac{I_z}{M}} )</td>
</tr>
<tr>
<td>Dipole Moment</td>
<td>( D = \sum_{i=1}^{n} q_i r_i )</td>
</tr>
<tr>
<td>Dipole Histogram</td>
<td>( D_z = \sum_{i=1}^{n} \frac{D}{z} )</td>
</tr>
<tr>
<td>Electron Density</td>
<td>( ED = \frac{\sum_{i=1}^{n} (e_i - q_i)}{d_{x,y}} )</td>
</tr>
<tr>
<td>Heat Capacity</td>
<td>( HC = \frac{3000 \sqrt{T \cdot \text{boltz}}}{2 \sqrt{T - n \cdot d \cdot VarT}} )</td>
</tr>
<tr>
<td>Mean Square Displacement</td>
<td>( msd = \langle (r_{t+\Delta} - r_t)^2 \rangle )</td>
</tr>
<tr>
<td>Diffusion Constant</td>
<td>( D_t = \frac{6 \cdot msd(t)}{t} )</td>
</tr>
<tr>
<td>Velocity Autocorrelation</td>
<td>( V_{\text{acor}} = \langle (V_{t+\Delta} \cdot V_t) \rangle )</td>
</tr>
<tr>
<td>Force Autocorrelation</td>
<td>( F_{\text{acor}} = \langle (F_{t+\Delta} \cdot F_t) \rangle )</td>
</tr>
<tr>
<td>Density Function</td>
<td>Histogram of atom counts</td>
</tr>
<tr>
<td>SDH</td>
<td>Histogram of all distances</td>
</tr>
<tr>
<td>RDF</td>
<td>( rd f(r) = \frac{SDH(r)}{4\pi r^2 \cdot \sigma_r \cdot \rho} )</td>
</tr>
</tbody>
</table>
The second type of functions is multi-body functions that are holistic in nature. The computation of such functions involves more than one atom’s attributes and cannot produce final result in a single run of the MS data (i.e., if traditional methods are used for their computation). Such queries include the Radial Distribution Function (RDF) [25, 45, 61] as well as some quantities associated with chemical shifts [69]. Generally, such functions are computed through histograms. For instance, the RDF is obtained from a histogram of all pairwise atom distances (this is the Spatial Distance Histogram or SDH). The traditional, straightforward (often the brute-force) way of computing these holistic functions is a very time consuming process. On top of that, these methods cannot produce the final result in a single run of the MS data, making such functions unsuitable for our idea of a push-based system. However, as seen earlier in Chapter 3, we have designed a data structure together with an algorithm that opens up the possibility for such queries to be executed in a push-based type environment. Further details on this are given in Section 4.4.
Fig. 4.1 represents an idea to show how such query processing system can be improved. The idea behind it is that some of the queries share same sub-routines. Having all the queries made as separate modules, these sub-routines can be computed once the data is being pushed through the system and then be used anytime a more complex query needs them. Having in mind the amount of data in a single frame that the queries (sub-routines as well) need to go through, and the fact that in a single MS there are thousands of frames, we believe this can be immense improvement in terms of total running time.

4.4 Building the System

4.4.1 MS Data Retrieval and in Memory Organization

A typical MS system generates and stores the data in a number of trajectory files, usually including multiple frames (snapshots of the simulated system taken at certain time intervals). Such MS generated data oftentimes goes through a simple lossless compression and, depending on the simulation software it may be stored in a binary format. Such trajectory file format is one of most often used MS file format (e.g., GROMACS, PDB). But such format is unrecognizable to our system. So our system has to do three things before it starts executing the queries: 1) read the MS data from a trajectory file, 2) translate the MS data to a form recognizable to our system, and 3) load the data to memory.

As mentioned above, in order for our system to be able to read the MS generated data, the data needs to be transformed. The reason for this is the following: the MS data is stored in multiple files and possibly in different formats as well. One such file holds the global data (identifying the system and the simulation). Another file holds each frame’s data. The frame’s data contains general information about the frame, but the main part is a sequential list of each
atom’s info, including atom’s mass, position, charge, number of electrons, velocities, forces, etc. Another file (topology file) holds the molecule/residue info that identifies what atom belongs to which molecule.

So, in order to extract the data from these files, we have created a sort of “extractor” of the attributes needed for the execution of the queries in our system. This extractor, essentially, is a separate piece of code that does three things: 1) it reads the MS generated data that the MS system stores in one of the often used MS file format (e.g., GROMACS); 2) it translates the data into a format that our system can read (taking only the information our system needs); and 3) it stores the data in a file that has basic structure to it. So, in the end, the data transformer produces a data file that our system takes as input. This code serves as a connection between an MS system like GROMACS, and our system. With this, our system can essentially be used as an add-on to GROMACS or other simulation systems and help improve the efficiency of the data analysis.

Once the data is in a format our system can read, the data is being loaded into the main memory one frame at a time. The in-memory organization of the particle’s data\textsuperscript{1} is in the form of a simple, two-dimensional array where a single row represents an atom in the system with all its attributes (e.g., coordinates, mass, charge, residue info, etc.). We also keep (in a one dimensional array) crucial system’s information for each frame, like temperature, energy, pressure, etc. We have used such structures because they are very suitable for simulating a push-based type of system: a simple sequential read of the array gives that on-line type of data stream. So, as the system reads the array, it pushes the data onto the query-modules. The one-body (algebraic) queries will produce a final result at the end of the first sequential read.

\textsuperscript{1}This paragraph talks only about the data organization used by one-body queries. For two-body queries (e.g., SDH), the data organization is discussed later on.
However, the two-body (holistic) functions, like SDH and RDF, cannot do this in a single read of the data. Therefore, each frame’s data array can be continuously read (in a loop manner) as many times as a query needs it.

4.4.2 Query Modules

As mentioned earlier, there are two types of functions/queries used for analysis of MS systems: algebraic or one-body, and holistic or two-body queries (these in general can be multi-body, but in this dissertation we only deal with a two-body functions).

4.4.2.1 One-body Queries

Most of the query modules in Table 4.1 (except the SDH and RDF) are not that involved. They only contain computations of fairly simple, one-body functions. These queries were coded as separate modules in our system. Each of these modules takes few attributes as input (e.g., atom selection, frames selection (for the autocorrelation functions), number of atoms, etc.). The system pushes the data as it becomes available onto these modules. The queries are being executed on the selection and are put in a “ready” mode, awaiting the next frame’s data. First, the more basic queries, like total mass, are being computed. The results of such queries are temporary stored (in main memory) and are available for use anytime a more complex query needs them.

---

However, earlier in Chapter 3, we have designed and created a data structure and an algorithm that can take the advantage of a single data read and produce final results for SDH computation. We have incorporated this into our system presented in this dissertation.


4.4.2.2 Two-body Queries

In general, queries involving two-body functions are a bit more complex and cannot provide the final result in a single data read if a straightforward method is used for their computation. However, in the proposed system we have incorporated a data structure and an algorithm for the SDH (also RDF) that is suitable for push-based type of system. In this subsection we give a brief overview of the data structure (DM) and the basic algorithm (DM-SDH) described in details in section 3.2 that we have implemented in the system proposed in this dissertation.

The simulation data space is represented by a conceptual data structure we named Density Map (DM). The density map splits the simulation space into a grid of equal size regions (or cells). The cells are cubes in 3D and squares in 2D\(^3\). Resolution of a density map is the reciprocal of the cell size in that density map. In order to generate higher resolution density map, we split each cell of the current resolution’s grid into four smaller cells of equal size. This design allows us to use a region quad-tree [54] to organize density maps of the same data but with different resolutions. So, essentially, a node in the quad-tree represents a single cell from the DM. Therefore, a density map of a certain resolution basically is the set of all nodes of one level of the tree. Each of the tree nodes records the cell’s location in the density map (coordinates of corner points) as well as the number of particles in each cell. We call such tree Density-Map tree (DM-tree).

The essential part of the DM-SDH algorithm is a procedure named RESOLVETWOCELLS. The input to this procedure is two cells from the density map (e.g., \(A\) and \(B\) in Fig. 4.2). It computes, in constant time, the minimum and maximum distance between the two cells. A pair of cells is *resolvable* if both the min and max distance between them fall into the same

\(^3\text{In this dissertation, we focus only on the 2D data to elaborate and illustrate the proposed ideas.}
SDH bucket $i$. If that is the case, the distance count of that bucket is being increase by $n_A n_B$ ($n_A$ and $n_B$ are the number of particles in cell $A$ and $B$, respectively). Otherwise, the cells are non-resolvable and we either:

1. Go to the next density map with higher resolution and resolve all children of $A$ with those of $B$, or
2. If leaf-level has been reached: compute every distance between particles of $A$ and $B$ and update the histogram accordingly.

In order to generate the complete SDH, the RESOLVETWOCELLS procedure is executed for all pairs of cells for a given density map $DM_k$ and the algorithm would recursively call the procedure (action (1) above) until leaf-level has been reached (action (2) above).

We have used this basic idea of the aforementioned DM-SDH algorithm, and have also designed two approximate SDH algorithms, ADM-SDH and Single-level SDH, introduced and thoroughly described and evaluated in sections 3.3 and 3.6, respectively. These approximate algorithms are substantially faster than the brute-force algorithm and also than the DM-SDH algorithm as they take advantage of some heuristic. For more details on these algorithms please refer to Chapter 3.
4.4.3 Working of the System

In this subsection we give an overview of how the system works at runtime. Please note that the first, preliminary part is only executed once, i.e., the data transformation from MS data files to a file that our system can read.

Here are the steps taken throughout the analysis:

1. Execute the data transformer
   a. Read the MS data from trajectory files
   b. Extract the info needed for our system
   c. Save the read data to a file recognizable to the system

2. Load the data into main memory (one frame at a time)
   a. Load data into a double array (for one-body queries)
   b. Load data into the quad-tree structure (for two-body queries: SDH, RDF)

3. Push the data to all queries

4. A query, if available, acts upon the pushed data (first executing the lower level, sub-queries)

5. Store intermediate results (results of sub-queries)

6. Repeat steps 3-5 if needed.

7. Output results

8. Go to step 2 and load the next frame (if needed).
Figure 4.3 depicts the flow of the system.

Step 2, loading the data into main memory, is different for SDH query compared to the one for the one-body queries. The reason for that, as mentioned earlier, is the different data structure used to store the data in memory. While we use double array to store the data for the one-body queries, we use quad-tree like data structure to store the data needed to compute the SDH. The loading to the double array is straightforward. However, to load the quad-tree structure, we need to use some of the info from the data itself. Namely, the coordinates of the atoms are used to determine in which tree node an atom belongs. That way we build the so-called density map (DM), i.e., the different regions with a certain number of atoms in them (including all the atom’s attributes). So basically, to solve the SDH problem in a push-based manner, we convert the problem into populating a data structure in push-based manner. This data structure will then be used as an input to our approximate algorithm that, although not completely in "on-the-fly"
Such workload set ups are being executed on 6 different size data sets: 50K, 200K, 800K, 2.5M, 4.5M, and 8.8M atoms.

Figure 4.4: Workload setup.

way, is great improvement over the naive methods used in much of today’s MS analysis systems (i.e., GROMACS, PDB, CHARMM, etc.).

4.5 Experimental Results

The system was implemented in C++ programming language and tested on real molecular simulation data sets. The experiments were carried out on an Apple MacPro machine with 8GB of physical memory and two Quad-Core Intel Xeon 3GHz processors. The MacPro was running OS X Mavericks 10.9.3 operating system. We have compared the results obtained by our system to those obtained by running the analysis through the GROMACS system (v. 4.5.7). Both systems were analyzing the same data sets with same number of time frames, as well as the same workload.
In our experiments, six data sets from different simulations were used. All simulations were done on a POPC lipid bilayer, but were all set to produce data of different sizes (i.e., different number of particles in the simulation). Namely, we have tested the system on simulations with 52,400; 209,600; 838,400; 2.5M; 4.4M; and 8.8M atoms. Also, since the simulations were run separately, they produced six different MS systems with distinct characteristics (distinct structure, atom’s positioning, etc.). From all of the generated data sets we have randomly selected sets of 10, 100, and 1000 consecutive frames for the purpose of our experiments. This gave us 18 different datasets on which we tested our system.

Two types of query workload were considered: 1) one involving one-body queries only, and 2) one including two-body queries (SDH and RDF) as well. The reason for this is that GROMACS, the system we used to compare our system to, uses the naive method of solving the RDF (SDH) problem (like almost all MS analysis systems). On the other hand, in our system we have incorporated SDH (RDF) algorithms that are far more superior to the naive method. So, because the differences in the running times of the two methods (brute-force and our approximate) are so big (days vs. second), we decided not to present the results/comparison of the systems based on the running times of the RDF (SDH) queries, since it would have been impossible to put them in prospective on the same graph.

The following set of one-body queries were included in the test workload: mean square displacement (msd), radius of gyration, dipole moment, center of mass, velocity autocorrelation, electron density, mass density, and charge density. This set of queries was pointed to us by a group in the physics field with extensive MS background, as one of the most commonly used in the field of collagen bilayer MS system analysis. A workload group contains all 8 queries.

\[^{4}\text{POPC is a chemical compound composed of a diacylglycerol and phospholipid. Its full name is 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine and it is one of the most important lipids in bio-physical molecular simulation.}\]
that are executed on one of the 12 selections. This makes 12 groups in total. Such groups are executed on six different size data sets, with 10, 100 and 1000 frames. This workload is then repeated 5 more times, by executing each of the queries in the groups 5, 10, 25, 50, and 100 times, essentially just magnifying the workload intensity. In total, we have $12 \times 6 \times 3 \times 6 = 1,296$ different workload setups to test the system on. Figure 4.4 shows the organization of the workload setup.

4.5.1 Benchmark

Through extensive collaboration with a research group from the Physics department at the University of South Florida, we have come up with a benchmark that can be used for testing the efficiency of an analysis system for molecular simulations. The benchmark consists of three essential parts: 1) simulation data produced by MS, 2) queries that are to be executed onto that data in order to produce some information of interest, and 3) benchmark parameters that control the size of the benchmark.

4.5.1.1 Benchmark Data

The data used in the benchmark was real molecular simulation data, produced through the GROMACS MS system. The initial, pre-simulation data file consisted of 200 POPC and 12000 solvent molecules, or 12200 molecules in total. This type of system was used because it is sufficiently diverse, containing enough distinct POPC and solvent molecules (e.g., each POPC molecule includes approximately 52 different atoms) and yet simple enough to be easily transformed into another system of different size. By using the `genconf` function in GROMACS, we produced pre-simulation files of different sizes (essentially by changing the system’s size
Six different sized pre-simulation files were created. A molecular simulation was then run on each of these 6 files, each producing an MS system of certain size (volume/number of particles). All of the simulations were set up to produce 1000 frames (snapshots in time of the systems), each frame containing the same number of particles as the base one. In our experimental results, we present the outcomes for 10, 100 and 1000 frames. The produced files contained: $52,400; 209,600; 838,400; 2.5M; 4.4M; \text{ and } 8.8M$ atoms per frame. So, for example, the file with 52,000 atoms holds 52,000,000 records in total (for 1000 frames, each containing 52,000 records). As mentioned earlier, this simulation data comes mostly in binary formats and in trajectory files having a lot of unneeded overhead. Therefore, it was transformed to a data arrays files containing only crucial information of the particles and the system. The size of the files ranged from $716MB$ for 52,000 atoms to $120GB$ for 8.8 million atoms (this is for data with 100 frames).

### 4.5.1.2 Benchmark Queries

The queries selected to be included in this benchmark were derived through a thorough observation of the way an MS system is being analyzed. They were found to be the base of the analysis of many MS systems. In other words, no matter how small or big the analysis was, these queries were included in that analysis. As mentioned earlier, they are of two types: one-body (and algebraic) and two-boy (and holistic). Table 4.1 shows these queries.

### 4.5.1.3 Benchmark Parameters

There are several parameters that can be used to control the overall size of the system. We divide the parameters into two groups:
Figure 4.5: Speed up over different levels of atom selection for 838K atoms.

(1) Data size parameters:

(a) Select different sized dataset

(b) Number of frames

(c) Data selection (within the selected dataset) onto which the queries are being executed

(2) Workload size parameters:

(a) Number of queries to be executed

(b) Number of times each query is executed

By changing these parameters, we can produce a versatile testing benchmark for MS analysis systems.

4.5.2 Results

We have run extensive experiments over all the different setups of workload mentioned earlier in this section. However, in this dissertation we present only the workload setups of four
Figure 4.6: Speed up over different levels of atom selection for 2.5M atoms.

Figure 4.7: Speed up over different levels of atom selections for 4.2M atoms.

Figure 4.8: Speed up over different levels of atom selections for 8.8M atoms.
data size sets: 838,400, 2.5M, 4.2M, and 8.8M atoms because we believe they convey enough information about the efficiency of our system compared to that of the Gromacs system. The running times of our push-based system were compared to those of the Gromacs system. The first set of figures, namely Figures 4.5-4.8, represents the speedup that our system obtains over the Gromacs system with various atoms selection levels. We define the selection levels based on the number of comparisons we have to make in order to extract the needed group (selection) of atoms. For example, if we want to do analysis on all molecules containing oxygen, or hydrogen, or carbon we would go over each molecule and compare its components to the selection list. The bigger the selection list, the higher the select level in our system. For better visualization, we note three different selection levels: high (at least 10 comparisons made), medium (between 1 and 10 comparisons made), and low select level (with one or less comparisons made). As seen in the figures, for high selection level, the speedup is smaller compared to that achieved in low select levels. The reason for this, we believe is in that the amount of time our system spends extracting the atoms group increases with the level of selection. Even though our system still shows considerable speedup over Gromacs in high level selections, we do believe there is room for improvement in our system and that is our immediate future work we are planning on doing. These figures also show the relation of the speedup to the workload intensity, i.e., the higher the workload intensity the higher the speedup.

The connection between the workload intensity and the speedup is better represented in the next set of figures, Figures 4.9-4.12. They show the speedup our system achieves over the Gromacs system on varying workload intensity. Each of those figures show the speedup with different dataset sizes (e.g., 838,000, 2,567,600 atoms, etc.), including 10, 100, and 1000 data frames. The speedup is calculated simply as a ratio between the running time of our system on
Figure 4.9: Showing speedup for different workload intensity for 838K atoms.

Figure 4.10: Showing speedup for different workload intensity for 2.5M atoms.

Figure 4.11: Showing speedup for different workload intensity for 4.2M atoms.
Figure 4.12: Showing speedup for different workload intensity for 8.8M atoms.

Figure 4.13: Showing speedup for different data size over all workload.
a certain set of workload and that of the Gromacs system on the same workload. These figures show that the speedup over varying workload intensity achieved by our system ranges anywhere from 2 to 230 times, depending on the size of the dataset, number of frames and the selection of the atoms.

Figure 4.13 shows the speedup our system achieves over all workload intensity (average workload intensity) with varying dataset sizes. It is clear that, again our system has better performance than the Gromacs system. The speedup presented in this set of figures ranges anywhere from 3.3 to 130 times.

The last figure, Figure 4.14, shows the speedup our system achieves over all workload intensity and all select levels with varying dataset sizes. This figure, in a way, summarizes the previous two sets of figures, bringing together the workload and the different selections.
through the average. It is clear that, again our system has better performance than the Gromacs system. The speedup ranges anywhere from 11 to 55.

All four sets of figures show that such push-based design has clear advantages over the pull-based type of design incorporated in the Gromacs system.
Chapter 5: Conclusions and Future Work

5.1 Conclusion

The main objective of our work is twofold. First, we aim to accomplish efficient computation of SDH, a popular quantity in particle simulations, with guaranteed accuracy. In this dissertation, we introduce approximate algorithm for SDH query processing based on our previous work developed around a quad tree-like data structure named *density map*. The experimental results show that our approximate algorithm has very high performance (short running time) while delivering results with astonishingly low error rates. Aside from the experimental results, we also analytically evaluate the performance/accuracy tradeoffs of the algorithm. Such analysis showed that the running time of our algorithm is completely independent of the input size $N$, and derived a provable error bound under desired running time. We further developed another mathematical model to perform in-depth study of the mechanism that leads to low error rates of the algorithm. Aside from administering tighter bounds (under some assumptions) on the error of the basic approximate algorithm, our model also gives insights on how the basic algorithm can be improved. Following these insights, a new single level approximate algorithm with improved time/accuracy tradeoff was proposed. Our experimental results supported our analysis. Having these experimental results on hand, one aspect of our future work will be to establish a provable error bound for the new algorithm.
And second, we strived to design and implement improved data analysis system that can be used in the field of molecular simulation system’s analysis. In this work, we introduce the idea for such system. We build our system on a push-based type design, where data from data arrays is being pushed onto available queries in the system. These queries are being executed on the pushed data and produce intermediate / final result that would be used as part of the data analysis. We are able to achieve an improvement over existing, pull-based type designs because of the I/O overhead such designs introduce when dealing with large volumes of scientific data. Also, our queries can be executed on the same stream of data, making it suitable solution for streaming circumstances. We designed a benchmark that can be used to test data analysis systems. This benchmark comprises of three parts: 1) benchmark data, 2) benchmark queries, and 3) benchmark parameters. We use this benchmark to compare our system to Gromacs, which is one of the most frequently used MS analysis systems. The efficiency and speedup achieved by our system is supported by extensive experiments and their results. The results show that our push-based design achieves up to 230 times speedup in comparison to a pull-based design, i.e., Gromacs.

5.2 Future Work

At times, the molecular simulation systems are analyzed through queries involving more than two particles in their computation. Therefore, one future work of interest would be to expand our research to the computation of $m$-body correlation functions with $m > 2$. This is a more general form of spatial statistics that involves counting all possible $m$-particle tuples. Such $m$-body correlation functions, although carrying great importance in scientific value, have
not been comprehensively examined, the reason being the absence of efficient processing algo-

rithms.

In the case of the proposed system, one general direction of our future work will be to
further improve our push-based design. Through the extensive experiments we have learned
that our design can be improved when the atom selection clause involves many conditions. This
improvement may be in the direction of improving the algorithmic part, but it can also be in the
direction of improving the data presentation/organization we have used in the system.
List of References


Appendices
Appendix A: Copyright Permissions

A.1 IEEE, License for Material in Chapter 3

Title: Approximate Algorithms for Computing Spatial Distance Histograms with Accuracy Guarantees

Author: Grupcev, V.; Yongke Yuan; Yi-Cheng Tu; Jin Huang; Shaoping Chen; Pandit, S.; Weng, M.

Publication: Knowledge and Data Engineering, IEEE Transactions on

Publisher: IEEE

Date: Sept. 2013

Copyright © 2013, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.

2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.

3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author’s approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]

2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.

3) In placing the thesis on the author’s university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity’s name goes here]’s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright © 2015 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Comments? We would like to hear from you. E-mail us at customercare@copyright.com