Computing Distance Histograms Efficiently in Scientific Databases

Yi-Cheng Tu [1], Shaoping Chen [2], and Sagar Pandit [3]

[1]*Department of Computer Science and Engineering, University of South Florida*
4202 E. Fowler Ave., ENB118, Tampa, FL 33620, U.S.A.
ytu@cse.usf.edu

[2]*Department of Mathematics, Wuhan University of Technology*
122 Luosi Road, Wuhan, Hubei, 430070, P. R. China
chensp@whut.edu.cn

[3]*Department of Physics, University of South Florida*
4202 E. Fowler Ave., PHY114, Tampa, FL 33620, U.S.A.
pandit@cas.usf.edu

***Abstract*— Particle simulation has become an important research tool in many scientific and engineering fields. Data generated by such simulations impose great challenges to database storage and query processing. One of the queries against particle simulation data, the spatial distance histogram (SDH) query, is the building block of many high-level analytics, and requires quadratic time to compute using a straightforward algorithm. In this paper, we propose a novel algorithm to compute SDH based on a data structure called density map, which can be easily implemented by augmenting a Quad-tree index. We also show the results of rigorous mathematical analysis of the time complexity of the proposed algorithm: our algorithm runs on $\Theta(N^{\frac{3}{2}})$ for two-dimensional data and $\Theta(N^{\frac{5}{3}})$ for three-dimensional data, respectively. We also propose an approximate SDH processing algorithm whose running time is unrelated to the input size N. Experimental results confirm our analysis and show that the approximate SDH algorithm achieves very high accuracy.**

I. Introduction

Many scientific fields have undergone a transition to data and computation intensive science, as the result of automated experimental equipments and computer simulations. Recent years have witnessed significant efforts in building data management tools suitable for processing scientific data [1], [2], [3], [4], [5]. Scientific data imposes great challenges to the design of database management systems that are traditionally optimized toward handling business applications. First, scientific data often come in large volumes, thus requires us to rethink the storage, retrieval, and replication techniques in current DBMSs. Second, user accesses to scientific databases are focused on complex high-level analytics and reasoning that go beyond simple aggregate queries. While many types of domain-specific analytical queries are seen in scientific databases, the DBMS should be able to support those that are frequently used as building blocks for more complex analysis. However, many of such basic analytical queries require superlinear processing time if handled in a straightforward way, as they are in current scientific databases. In this paper, we report our efforts to design efficient algorithms for a type of query that is extremely important in the analysis of particle simulation data.

Particle simulations are computer simulations in which the basic components of large systems (e.g., atoms, molecules, stars, galaxies ...) are treated as classical entities (i.e., particles) that interact for certain duration under postulated empirical forces. For example, molecular simulations (MS) explore relationship between molecular structure, movement and function. These techniques are primarily applicable in the modeling of complex chemical and biological systems that are beyond the scope of theoretical models. MS are most frequently used in material sciences, biomedical sciences, and biophysics, motivated by a wide range of applications. In astrophysics, the N–body simulations are predominantly used to describe large scale celestial structure formation [6], [7], [8], [9]. Similar to MS in applicability and simulation techniques, the N–body simulation comes with even larger scales in terms of total number of particles simulated.

Results of particle simulations form large datasets of particle configurations. Typically, these configurations store information about the particle types, their coordinates and velocities - the same type of data we have seen in spatial-temporal databases [10]. While snapshots of configurations are interesting, quantitative structural analysis of inter-atomic structures are the mainstream tasks in data analysis. This requires the calculation of statistical properties or functions of particle coordinates [6]. Of special interest to scientists are those quantities that require coordinates of two particles simultaneously. In their brute force form these quantities require $O(N^2)$ computations for N particles [7]. In this paper, we focus on one such analytical query: the **Spatial Distance Histogram (SDH)** query, which asks for a histogram of the distances of all pairs of particles in the simulated system.

S. Chen is currently a visiting professor in the Department of Computer Science and Engineering at the University of South Florida (USF). His email at USF is: schen11@cse.usf.edu

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.30

A. 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) [6], [11], [12]. The RDF is defined as

$$g(r) = \frac{N(r)}{4\pi r^2 \delta r \rho} \quad (1)$$

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

The RDF is of great importance in computation of thermodynamic characteristics of the system. Some of the important quantities like total pressure, and energy cannot be calculated without $g(r)$. For mono–atomic systems, the RDF can also be directly related to the structure factor of the system [13]. In current solutions, 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 [14], [12]. 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 [15]. MS systems also hold up to millions of atoms. Such scale of the simulated systems 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.

B. Contributions and roadmap

We claim the following contributions via this work:

1. We propose an innovative algorithm to solve the SDH problem based on a Quadtree-like data structure we call *density map*;
2. We accomplish rigorous performance analysis of our algorithm and prove its time complexity to be $\Theta\big(N^{\frac{3}{2}}\big)$ and $\Theta\big(N^{\frac{5}{3}}\big)$ for 2D and 3D data, respectively;
3. Our analytical results on the algorithm give rise to an approximate SDH solution whose time complexity is independent to the size of the dataset. In practice, this algorithm computes SDH with very low error rates.

We continue this paper by formally defining the SDH problem and listing important notations in Section II; we introduce our SDH processing algorithm in Section III; performance analysis of our algorithm is sketched in Section IV; we discuss an approximate SDH solution in Section V; Section VI is dedicated to experimental evaluation of our algorithms; we provide a brief survey on related work in Section VII, and conclude this paper by Section VIII.

II. Problem statement and list of notations

The SDH 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: $[r_0, r_1), [r_1, r_2), [r_2, r_3), \cdots, [r_{l-1}, r_l]$. 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 paper, we focus our discussions on the case of *standard SDH query* where all buckets have the same width p and $r_0 = 0$, which gives the following series of buckets: $[0, p), [p, 2p), \cdots, [(l-1)p, lp]$. Generally, the boundary of the last bucket lp is set to be the maximum distance of any pair of points. 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. The only complication of non-uniform bucket width is that, given a distance value, we need $O\big(\log l\big)$ time to locate the corresponding bucket instead of constant time in processing SDH with equal bucket width. The SDH is basically a series of non-negative integers $\mathbf{h} = (h_1, h_2, \cdots, h_l)$ where h_i $(0 < i \leq l)$ is the number of pairs of points whose distances are within the bucket $[(i-1)p, ip)$.

In Table I, we list the notations that are used throughout this paper. Note that symbols defined and referenced in a local context are not listed here.

TABLE I
SYMBOLS AND NOTATIONS.

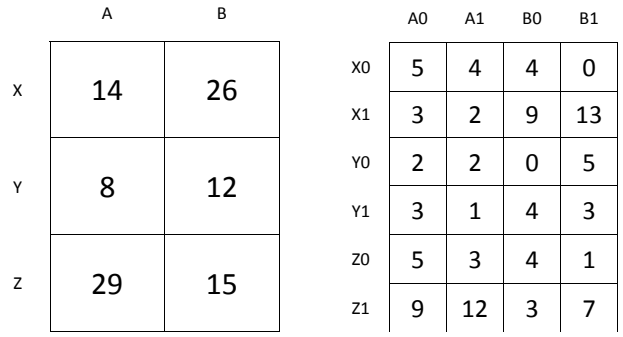
Symbol	Definition
p	width of histogram buckets
l	total number of histogram buckets
$\mathbf{h}$	the histogram with elements h_i $(0 < i \leq l)$
N	total number of particles in data
i	an index symbol for any series
DM_i	the i-th level density map
d	number of dimensions of data
δ	side length of a cell
S	area of a region in 2D space
ϵ	error bound for the approximate algorithm
H	total level of density maps, i.e., tree height

III. Our Approach

A. Overview

In processing SDH using the naïve approach, the difficulty comes from the fact that the distance of any pair of points is calculated to determine which bucket a pair belongs to. An important observation here is: a histogram bucket always has a non-zero width. Given a pair of points, their bucket membership could be determined if we only know a range that the distance belongs to and this range is contained in a histogram bucket. With the bucket width p increases (i.e., user sends in a coarser SDH query), the chance that any such range with a fixed span will fall into a bucket also increases. In other words, we need to save time in our algorithm by calculating point-to-point distances approximately.

The central idea of our approach is a conceptual data structure called *density map*. For a 3D space, a density map



a. low resolution map b. high resolution map

Fig. 1. Two density maps of different resolutions.

is essentially a 3D grid that divides the simulated space into cubes of equal volumes. For a 2D space, it consists of squares of equal size. From now on, we use 2D data and grids to elaborate and illustrate our ideas unless specified otherwise. Note that extending our discussions to 3D data/space would be straightforward. In every cell of the grid, we record the number of particles that are located in the space represented by that cell as well as the four coordinates that determine the exact boundary of the cell. The reciprocal of the cell size is called the *resolution* of the density map. In order to process SDH, we build a series of density maps with different resolutions. We organize the array of density maps in a way such that the resolution of a density map is always doubled as compared to the previous one in the series. Consequently, any cell in a density map is divided into exactly four (eight for a 3D space) disjoint cells in the next density map. In Figure 1, we illustrate two density maps with different resolutions built for the same dataset. For example, the simulated space is divided into six cells in Fig. 1a, each with side length 2, and cell XA has 14 particles in it. The next density map is shown in Fig. 1b: cells are of side length 1 and XA is divided into 4 cells on this map: X0A0, X0A1, X1A0, and X1A1. A natural way to organize the density maps is to connect all cells in a quad-tree. We will elaborate more on the implementation of the density maps in Section III-C.

B. The Density-Map-based SDH (DM-SDH) algorithm

In this section, we describe how to use the density maps to process the SDH query. The details of the algorithm are shown in Fig. 2. The core of the algorithm is a procedure named RESOLVETWOCELLS, which is given as inputs a pair of cells m_1 and m_2 on the same density map.

In RESOLVETWOCELLS, we first compute the minimum and maximum distances between any particle from m_1 and any one from m_2 (line 1). This can be accomplished in constant time given the corner coordinates of two cells stored in the density map (only three cases are possible, as shown in Fig. 3). When the minimum and maximum distances between m_1 and m_2 fall into the same histogram bucket i, we say these two cells are *resolvable* on this density map, and they *resolve*

Algorithm DM-SDH
Inputs: all data points, density maps built beforehand, and bucket width p
Output: an array of counts $\mathbf{h}$

```
1  initialize all elements in h to 0
2  find the first density map DM_i whose cells have
     diagonal length k ≤ p
3  for all cells in DM_i
4  do n ← number of particles in the cell
5     h_1 ← h_1 + (1/2)n(n − 1)
6  for any two cells m_j and m_k in DM_i
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 ← index of the bucket m_1 and m_2 resolve into
3       n_1 ← number of particles in m_1
4       n_2 ← number of particles in m_2
5       h_i ← h_i + n_1 n_2
6  else if m_1 and m_2 are on the last density
        map (i.e., the one with highest resolution)
7       for each particle A in m_1
8           for each particle B in m_2
9           do f ← distance between A and B
10              i ← the bucket f falls into
11              h_i ← h_i + 1
12 else
13      DM' ← 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)
```

Fig. 2. The density-map-based SDH algorithm.

into bucket i. If this happens, the histogram is updated (lines 2 - 5) by incrementing the count of the specific bucket i by $n_1 n_2$ where n_1, n_2 are the particle counts in cells m_1 and m_2, respectively. If the two cells do not resolve on the current density map, we move to a density map with higher (doubled) resolution and repeat the previous step. However, on this new density map, we try resolving all four partitions of m_1 with all those of m_2 (lines 12 - 16). In other words, there are $4 \times 4 = 16$ recursive calls to RESOLVETWOCELLS if m_1 and m_2 are not resolvable on the current density map. In another scenario where m_1 and m_2 are not resolvable yet no more density maps are available, we have to calculate the distances of all particles in the non-resolvable cells (lines 6 - 11). The DM-SDH algorithm starts (line 2) at the first density map DM_i whose cell diagonal length is smaller than the bucket width p (i.e., cell side length $\delta \leq \frac{p}{\sqrt{2}}$). It is easy to see that no pairs of cells are resolvable in density maps with resolution

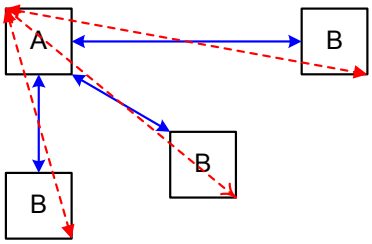


Fig. 3. Three scenarios to consider when computing the minimum and maximum distance between two cells A and B, with solid (dotted) line representing minimum (maximum) distance in each case.

lower than that of DM_i. Within each cell on M_i, we are sure that any intra-cell point-to-point distance is smaller than p thus all such distances are counted into the first bucket with range $[0, p)$ (lines 3 - 5). The algorithm proceeds by resolving inter-cell distances (i.e., calling RESOLVETWOCELLS) for all pairs of cells in M (lines 6 - 7).

Clearly, the main idea behind our algorithm is to avoid computing any point-to-point distances. By only considering atom counts in the density map cells, we are able to process multiple point-to-point distances between two cells in one shot. This translates into significant improvements over the brute-force approach.

A case study. Let us study an example by revisiting Fig. 1. Suppose the query asks for SDH with a bucket width of 3 (i.e., histogram buckets are $[0, 3), [3, 6), [6, 9), \cdots$) and we start on the low-resolution map in Fig. 1a. First, since all particles in XA are within a distance $2\sqrt{2} < 3$, we can safely increase the count of the first bucket (with range 0-3) by $14 \times (14 - 1)/2 = 91$, and we do this for all other cells in Fig. 1a. Then we try to resolve XA with each and every other cell in the same density map, e.g., cell ZB. However, we cannot draw any conclusions as the distances between a particle in XA and one in ZB are within the range $[2, \sqrt{52}] \approx [2, 7.2111]$, which overlaps with the first and second buckets. In this case, we turn to the next density map in Fig. 1b, in which a cell in Fig. 1a is divided into four smaller cells. We start comparing counts of all XA cells (i.e., X0A0, X0A1, X1A0, and X1A1) with all ZB cells (i.e., Z0B0, Z0B1, Z1B0, and Z1B1). Out of the 16 pairs of cells, six can be resolved (Table II). For example, since the distances between any particle in X0A0 and any one in Z0B0 are within $[\sqrt{10}, \sqrt{34}] \approx [3.162, 5.831]$, we increment the count of the second bucket (with range $[3, 6)$) in the histogram by $5 \times 4 = 20$. For those the are not resolvable, we need to visit a density map with an even higher resolution, or, calculate all the inter-cell point-to-point distances when no such density maps exist. Note that those cells with a zero particle count (e.g., cell Y0B0) can be ignored in this process.

C. Implementation of density maps

In DM-SDH, we assume that there are a series of density maps built beforehand for the dataset. In this section, we describe relevant details on the implementation and maintenance of the density maps.

TABLE II

INTER-CELL DISTANCE RANGES ON DENSITY MAP SHOWN IN FIG. 1B. RANGES MARKED WITH * ARE RESOLVABLE INTO BUCKETS OF WIDTH 3.

ZB	**XA cells**			
cells	Z0B0	Z0B1	Z1B0	Z1B1
X0A0	$[\sqrt{10}, \sqrt{34}]^*$	$[\sqrt{13}, \sqrt{41}]$	$[\sqrt{4}, \sqrt{45}]$	$[\sqrt{20}, \sqrt{52}]$
X0A1	$[3, \sqrt{29}]^*$	$[\sqrt{10}, \sqrt{34}]^*$	$[\sqrt{4}, \sqrt{40}]$	$[\sqrt{17}, \sqrt{45}]$
X1A0	$[\sqrt{5}, \sqrt{25}]$	$[\sqrt{8}, \sqrt{32}]$	$[\sqrt{10}, \sqrt{34}]^*$	$[\sqrt{13}, \sqrt{41}]$
X1A1	$[2, \sqrt{20}]$	$[\sqrt{5}, \sqrt{24}]$	$[3, \sqrt{29}]^*$	$[\sqrt{10}, \sqrt{34}]^*$

1) Tree structure: As mentioned earlier, we organize the cells on different density maps into a tree structure, much like the point region (PR) Quad-tree presented in [16]. The nodes in the tree hold the following information:

```
(p-count, x1, x2, y1, y2, child, p-list, next)
```

where `p-count` is the number of particles in the cell, `x1` to `y2` are the four coordinates that define the region of the cell (for 3D data, we need two more coordinates for the 3rd dimension), `child` is a pointer to the first child on the next level.[1] The `p-list` element, which is meaningful only for leaf nodes, is the head of a list of data structures that store the real particle data. Unlike a regular Quad-tree, we add a `next` pointer to chain the sibling nodes together (the order of the four siblings in the list can be arbitrarily determined). Furthermore, for the last of the four siblings, its `next` pointer is used to point to its cousin. By this, all nodes on the same level are connected - such a connected list essentially forms a density map with a specific resolution. The head of all listed can be stored in an array for the ease of locating the appropriate density map to start the DM-SDH algorithm (line 2, Fig. 2). From now on, we use the phrases "density map' and "tree level", "cell" and "tree node" interchangeably. For example, the density maps in Fig. 1 can be put into a tree structure as shown in Fig. 4, in which each node is shown with its `p-count` field.

2) Tree height.: To be able to answer SDH queries with different parameters (e.g., bucket width p, subregion of the simulated space), we need to build a series of density maps from the most coarse resolution to the finest. On the coarsest end, we can build a single node map that covers the whole simulated space. The question is from the other end: what should be the highest resolution in the maps? This is a subtle issue: first, given any bucket width p, the percentage of resolvable cells increases with the level of the tree. However, the number of pairs of cells also increases dramatically (i.e., by a factor of 2^d).

Recall that DM-SDH saves our time of processing SDH by

[1] A `parent` pointer could be added to each node to achieve logarithmic data insertion/deletion time. However, we assume the scientific dataset is static (no sporadic insertions, no deletions) therefore a `child` pointer is sufficient for efficient bulk loading.

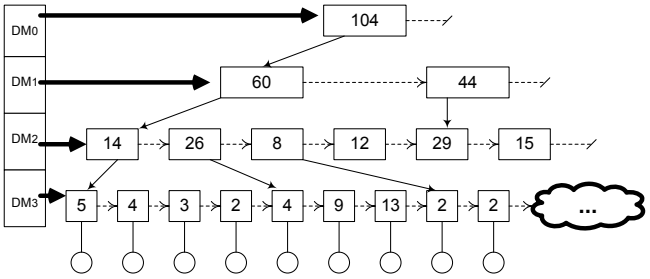


Fig. 4. Tree structure to organize the density maps in Fig. 1. Here we show the `p-count` (number in each node), `next` (dotted lines), `child` (thin solid lines), and `p-list` (lines connecting to a ball).

resolving cells such that we need not calculate the point-to-point distances one by one. However, when the `p-count` of a cell decreases, the time we save by resolving that cell also decreases. Imagine a cell with a `p-count` of 4 or smaller (8 or smaller for 3D data/space), it does not give us any benefit in processing SDH to further partition this cell on the next level: the cost of resolving the partitions could be higher than directly retrieving the particles and calculating distances (lines 7 - 11 in RESOLVETWOCELLS). Based on this observation, the total level of density maps H is set to be

$$H = \left\lceil \log_{2^d} \frac{N}{\beta} \right\rceil + 1 \qquad (2)$$

where d is the number of dimensions and 2^d is essentially the degree of tree nodes, β is the average number of particles we desire in each leaf node. In practice, we set β to be slightly greater than 4 in 2D (8 for 3D data) since the CPU cost of resolving two cells is higher than computing the distance between two points.

3) Other issues: In addition to the bucket width p, user can attach other conditions to a SDH query. Two common varieties of the regular SDH query are: (1) Compute the SDH of a specific region of the whole simulated space; and (2) Compute the SDH of all particles of a specific type (e.g., carbon atoms) in the dataset. The first variety requires modifications to our algorithm: in RESOLVETWOCELLS, we need to check if both cells are contained by the query region and add one more case for the recursive call, that is, if the cells are resolvable but at least one of the cells overlaps with, or locates out of, the query region, we still need to go to the next density map. If both cells are out of the query region, nothing needs to be done. In calculating the distances of particles (lines 7 - 11), again, we only consider those particles that are within the query region. The second variety requires more information be stored in the density map cells: in addition to the `p-count` field, we keep a list of counts, one for each possible type of particles in the data. Fortunately, the number of particle types is not very large in the sciences of interest (e.g., about 10 for molecular simulation).

Another piece of information we can store in the tree nodes is the minimum bounding rectangle (MBR) formed by of all the particles in a node. In RESOLVETWOCELLS, we can use the MBR of the two cells to compute the minimum and maximum point-to-point distances. As compared to the theoretical bounds of the space occupied by a tree node, the MBR will cover a smaller area. Intuitively, the chance of a cell's being resolvable under a given p increases as the cell shrinks. The use of MBR can thus shorten the running time by making more cells resolvable at a higher level on the tree. The MBR can be easily computed when data points are loaded to the tree, with the storage overhead of four extra coordinates in each node.

IV. ANALYSIS OF THE ALGORITHM

The running time of DM-SDH consists of two main parts:

1. the time spent to check if two cells are resolvable (line 0 in RESOLVETWOCELLS, constant time needed for each operation); and
2. distance calculation for data in cells non-resolvable even on the finest density map (lines 7 - 11 in RESOLVETWOCELLS, constant time needed for each distance).

As compared to the brute-force algorithm, we save time by performing operation 1 in hope of handling multiple distances in one shot. However, it is not clear how much overhead this bears. Consider a tree illustrated in Fig. 5 where each level represent a density map but each node represents a pair of cells in the original density map. Given a histogram bucket width p_1, we start from a density map DM_i with c_i cells. Thus, there are $O(c_i^2)$ entries on the corresponding level of the tree shown in Fig. 5. On the next map DM_{i+1}, there are $4 \times 4 = 16$ times of cell pairs to resolve. However, some of the cells in DM_{i+1} do not need to be considered as their parents are resolved on DM_i. Consider a resolvable entry a in DM_i, the whole subtree rooted at a needs no further consideration, leaving a "hole" on the leaf level. Similarly, if b is a resolvable entry on a lower level DM_j, the time needed for resolving everything in the subtree of b is also saved. However, this subtree is smaller than that of a, leading to less savings of time. From this we can easily see that the running time depends on the bucket width: if we are given another query with bucket width $p_2 < p_1$ such that we will start DM-SDH from level DM_j of the tree, more cell comparisons have to be done, giving rise to longer running time.

In analyzing the time complexity of DM-SDH, we are interested in how the running time increases as the total number of particle N increases, with a fixed p as the query parameter. Qualitatively, as N increases, the height of the trees also increases (as we fix β in Equation (2)), thus a higher percentage of particle pairs can be resolved in the cells. However, the total number of entries on the leaf level in Fig. 5 also increases (quadratically). Therefore, a quantitative study on the percentage of resolvable cells on a given level is essential in such analysis.

We have accomplished a quantitative analysis on the running time of our algorithm, which involves non-trivial geometric modeling. Due to space limitations, here we only sketch the main ideas of the analysis. Interested readers can find more details in a longer version of this paper [17].

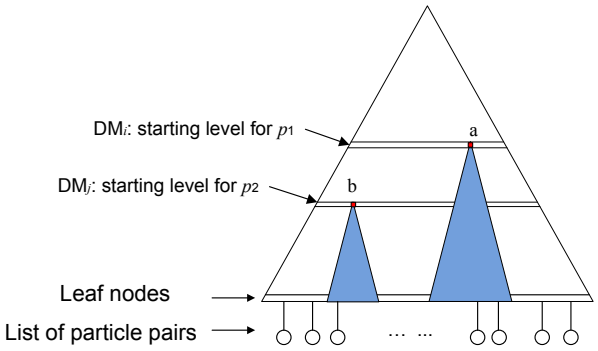


Fig. 5. A conceptual tree structure with each node representing a pair of cells in a density map. Similarly, the data nodes hold pairs of particles.

Given any cell $\mathbf{A}$ on the first level of density map visited, our analysis develops formulae for the area of a theoretical region $\mathbf{A_i}$ containing all particles that can possibly resolve into the ith bucket with any particle in $\mathbf{A}$. For example, the region $\mathbf{A_1}$ in Fig. 6 refers to the one for bucket 1: it consists of four quarter cycles with radius p and four rectangles with side length δ and p (note that $\delta = \frac{p}{\sqrt{2}}$ is the side length of cell $\mathbf{A}$), plus cell $\mathbf{A}$ itself. The cells that are resolvable into bucket i with any sub cells in $\mathbf{A}$ also form a region (whose border shows a zigzag pattern) denoted as $\mathbf{A_i'}$. When the resolution of the density map increases, the boundary of region $\mathbf{A_i'}$ approaches that of $\mathbf{A_i}$. In Fig. 6, the blue lines form one such region related to bucket 1 on the density map four levels down. The shape of region $\mathbf{A_i}$ becomes more complex when $i > 1$: its outer boundary is similar to $\mathbf{A_1}$ except the quarter cycles are with radius ip, but it also contains a hole in it [17]. We define the ratio of $\sum_i \mathbf{A_i'}$ to $\sum_i \mathbf{A_i}$ as the *covering factor*. This is a critical quantity in our analysis as it tells how many of the particle pairs are handled by resolving cells. Obviously, the covering factor increases when we visit more levels of density map. Table III shows the values of the covering factor under different situations. Of special interest to our analysis is the complement to covering factor named the *non-covering factor*, which represents the percentage of areas that are not resolvable. Let us present the most important result in our analysis in the following lemma.

Lemma 1: For any given standard SDH query with bucket width p, let DM_i be the first density map our DM-SDH algorithm visits, and $\alpha(m)$ be the non-covering factor of a density map that lies m levels below DM_i (i.e., map DM_{i+m}). We have

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

Proof: See [17] for details. ■

What Lemma 1 tells us is: the chance that any pair of cells is not resolvable decreases by half with the density map level increases by one. In other words, for a pair of non-resolvable cells on DM_i (or any level lower than DM_i), among the 16 pairs of subcells on the next level, we expect $16 \times 0.5 = 8$ pairs to be resolvable. From Table III, we can also conclude that Lemma 1 not only works well for large l (i.e., smaller p, and more meaningful in simulation data analysis), it **quickly converges even when l is reasonably small**. Furthermore, the above result is also true for **3D data**, although we can only give numerical results due to complex forms of the formulae developed in the 3D analysis (see Section IV-D of [17]).

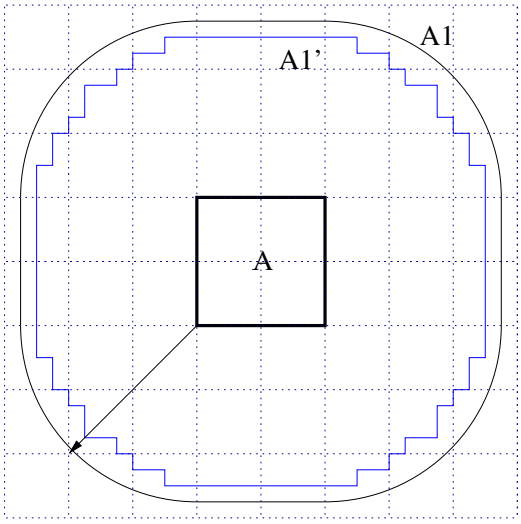


Fig. 6. Boundaries of bucket 1 regions of cell A, with the bucket width p being exactly $\sqrt{2}\delta$. The arrowed line is of length p.

A. Time complexity of DM-SDH

With Lemma 1, we achieve the following analysis of the time complexity of DM-SDH.

Theorem 1: In DM-SDH, the time spent on operation 1 (i.e., resolving two cells) is $\Theta\left(N^{\frac{2d-1}{d}}\right)$ where $d \in \{2, 3\}$ is the number of dimensions of the data.

Proof: Given a SDH query with parameter p, the starting level DM_i is fixed in DM-SDH. Assume there are I pairs of cells to be resolved on DM_i. On the next level DM_{i+1}, total number of cell pairs becomes $I2^{2d}$. According to Lemma 1, half of them will be resolved, leaving only $I2^{2d-1}$ pairs unresolved. On level DM_{i+2}, this number becomes $I2^{2d-1}\frac{1}{2}2^{2d} = I2^{2(2d-1)}$. Therefore, the number of calls to resolve cells on the different density maps form a geometric progression

$$I, I2^{2d-1}, I2^{2(2d-1)}, \ldots, I2^{n(2d-1)}$$

where n is the total number of density maps visited. The time spent on all cell-resolving operations T_c is basically the sum of all items in this progression :

$$T_c(N) = \frac{I\left[2^{(2d-1)(n+1)} - 1\right]}{2^{2d-1} - 1}. \quad (3)$$

We use $T_c(N)$ to denote the time under a given size N of the dataset. According to Equation (2), one more level of

TABLE III

EXPECTED PERCENTAGE OF PAIRS OF CELLS THAT CAN BE RESOLVED UNDER DIFFERENT LEVELS OF DENSITY MAPS AND TOTAL NUMBER OF HISTOGRAM BUCKETS. COMPUTED WITH MATHEMATICA 6.0.

Map	**Total Number of Buckets (l)**							
levels	2	4	8	16	32	64	128	256
m=1	50.6565	52.1591	52.5131	52.5969	52.6167	52.6214	52.6225	52.6227
m=2	74.8985	75.9917	76.2390	76.2951	76.3078	76.3106	76.3112	76.3114
m=3	87.3542	87.9794	88.1171	88.1473	88.1539	88.1553	88.1556	88.1557
m=4	93.6550	93.9863	94.0582	94.0737	94.0770	94.0777	94.0778	94.0778
m=5	96.8222	96.9924	97.0290	97.0369	97.0385	97.0388	97.0389	97.0389
m=6	98.4098	98.4960	98.5145	98.5184	98.5193	98.5194	98.5195	98.5195
m=7	99.2046	99.2480	99.2572	99.2592	99.2596	99.2597	99.2597	99.2597
m=8	99.6022	99.6240	99.6286	99.6296	99.6298	99.6299	99.6299	99.6299
m=9	99.8011	99.8120	99.8143	99.8148	99.8149	99.8149	99.8149	99.8149
m=10	99.9005	99.9060	99.9072	99.9074	99.9075	99.9075	99.9075	99.9075

density map will be built when N increases to 2^dN. Revisiting Equation (3), we have the following recurrence:

$$T_c(2^dN) = \frac{I\left[2^{(2d-1)(n+2)} - 1\right]}{2^{2d-1} - 1} = 2^{2d-1}T_c(N) - o(1) \quad (4)$$

Based on the master theorem [18], the above recurrence gives

$$T_c(N) = \Theta\big(N^{\log_{2^d} 2^{2d-1}}\big) = \Theta\big(N^{\frac{2d-1}{d}}\big).$$

■

Now let us investigate the time complexity for performing operation 2, i.e., calculating distance between particles. We have similar results as in Theorem 1.

Theorem 2: In DM-SDH, when the spatial distribution of particles is *reasonable*, the time spent on operation 2 (i.e., distance calculation) is also $\Theta\big(N^{\frac{2d-1}{d}}\big)$.

Proof: As in the derivation of Equation (4), we consider the situation of increasing the dataset size from N to 2^dN. For any pair of cells on the last density map when dataset is of size N, we have the chance to divide each cell into 2^d smaller cells when another density map is built (as a result of the increase of N). Altogether we have on the new density map $2^d2^d = 2^{2d}$ pairs to resolve, among which half are expected to be resolved (Lemma 1). When the spatial distribution of particles is reasonable (e.g., a uniform distribution), this leaves half of the distances in the unresolved cells and they need to be calculated. By changing the size of the dataset from N to 2^dN, the total number of distances between any pair of cells increases by 2^{2d} times. Since half of the distances need to be calculated, the total number of distance calculations T_d increases by 2^{2d-1}. Therefore, we have the following recurrence:

$$T_d(2^dN) = 2^{2d-1}T_d(N),$$

which is essentially the same as Equation (4), and this concludes the proof. ■

In the proof of Theorem 2, we extended Lemma 1 from percentage of cell pairs to that of distances. This extension is obviously true for uniformly distributed data. Actually, this conclusion is also true as long as the particle distribution is not positively related to non-resolvable cell pairs, which is generally true in MS data. These distributions are what we call *reasonable* distributions. More details can be found in Section IV-F of [17], in which we also present remedies to our algorithm under the bizarre cases of unreasonable particle distributions.

Theorem 3: The time complexity of the DM-SDH algorithm is $\Theta\big(N^{\frac{2d-1}{d}}\big)$.

Proof: Proof is concluded by combining Theorem 1 and Theorem 2. ■

B. Other costs

I/O costs. In the previous analysis, we focus on the CPU time of the algorithm. Depending on the blocking strategy we use for retrieving data from disk, the exact I/O cost of DM-SDH varies. The bottomline, however, is that the I/O complexity will be asymptotically lower than the quadratic I/O cost needed for calculating all distances.[2] A straightforward implementation of DM-SDH will give us an I/O complexity $O\big((\frac{N}{b})^{\frac{2d-1}{d}}\big)$ where b is the number of records in each page. To be specific:

1. the distance calculations will happen between data points organized in data pages of associated density map cells (i.e., no random reading is needed). On average, one data page only needs to be paired with $O\big(\sqrt{N}\big)$ other data pages for distance calculation (Theorem 2) in 2D space;
2. I/O complexity for reading density map cells will be the same as in 1. In practice, it will be much smaller, as the size of the nodes is small.

[2]To be specific, it is $O\big((\frac{N}{b})^2\frac{1}{B}\big)$ where b is the page factor and B is the blocking factor if we use a strategy like in block-based nested-loop join.

It would be interesting to investigate how we can improve our algorithm to take advantage of blocking and prefetching.

Storage overhead. The storage cost of our algorithm is bound by the size of the density map of the highest resolution we store (leaf nodes in the Quad-tree), as map size deceases exponentially with the decrease of resolution. Obviously, the space complexity is $O(N)$. The total storage overhead will be really small if we have a Quadtree index built for the dataset, which is a popular practice in scientific databases [19]. In this case, we only need to add a `p-count` field and `next` pointer to each index node.

V. Approximate SDH query processing

While the DM-SDH algorithm is more efficient than current SDH processing methods, its running time for large datasets is still undesirably long. Actually, there are cases where even a coarse SDH will greatly help the fine-tuning of simulation programs [6]. On the other hand, the main motivation to process SDHs is to study the statistical distribution of point-to-point distances in the simulated system [6]. Since a histogram by itself is an approximation of the underlying distribution $g(r)$ (Equation 1), an inaccurate histogram generated from a given dataset will still be useful in a statistical sense. In this section, we introduce a modified SDH algorithm to give such approximate results to gain better performance in return. Two must-have features for a decent approximate algorithm are :1) provable and controllable error bounds such that the users can have an idea on how close the results are to the fact; and 2) analysis of costs to reach (below) a given error bound, which enables desired performance/correctness tradeoffs. Fortunately, our analytical results shown in Section IV makes the derivation of such error bounds and cost model an easy task.

In the DM-SDH algorithm, we have to : 1) keep resolving cells till we reach the lowest level of the tree; 2) calculate point-to-point distances when we cannot resolve two cells on the leaf level of the tree. Our idea for approximate SDH processing is: stop at a certain tree level and totally skip all distance calculations if we are sure that the number of distances in the unvisited cell pairs fall below some error tolerance threshold.

Recall that, for any given density map DM_{i+m} and total number of buckets l, our analytical model gives the percentage of non-resolvable cell pairs $\alpha(m)$, which can be efficiently computed. We list some values of $1-\alpha(m)$, the percentage of *resolvable* cell pairs, in Table III. Given a user-specified error bound ϵ, we can find the appropriate levels of density maps to visit such that the unvisited cell pairs only contain less than $\epsilon\frac{N(N-1)}{2}$ distances. For example, for a SDH query with 128 buckets and error bound of $\epsilon = 3\%$, we get $m = 5$ by consulting the table. This means, to ensure the 3% error bound, we only need to visit five levels of the tree (excluding the starting level DM_i), and no distance calculation is needed. According to Lemma 1 and Table III: $\alpha(m)$ almost exactly halves itself when m increases by 1. Therefore, a rule-of-thumb for choosing m for given error bound ϵ is

$$m = \lg \frac{1}{\epsilon}.$$

The cost of the approximate algorithm only involves resolving cells on the $m+1$ levels of density maps. Borrowing Equation (4), we obtain the time complexity of the new algorithm

$$T_c(N) \approx I2^{(2d-1)m} = I2^{(2d-1)\lg\frac{1}{\epsilon}} = I\left(\frac{1}{\epsilon}\right)^{2d-1} \quad (5)$$

where I is the number of cell pairs on the starting density map DM_i, and it is solely determined by the query parameter p. Apparently, the running time of this algorithm is not related to the input size N.

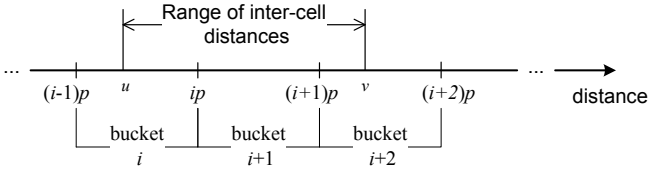


Fig. 7. Distance range of two non-resolvable cells overlap with three buckets.

Now let us discuss how to deal with those non-resolvable cells after visiting $m+1$ levels on the tree. In giving the error bounds in our approximate algorithm, we are conservative in assuming the distances in all the unresolved cells will be placed into the wrong bucket. In fact, this almost will never happen because we can distribute the distance counts in the unvisited cells to the histogram buckets heuristically and some of them will be done correctly. Consider two non-resolvable cells in a density map with particle counts n_1 and n_2 (total number of n_1n_2 distances between them), respectively. We know their minimum and maximum distances u and v (calculated in resolving two cells) fall into multiple buckets. Fig. 7 shows an example that spans three buckets. Using this example, we describe the following heuristics to distributed the n_1n_2 total distance counts into the relevant buckets. These heuristics are ordered in their expected correctness.

1. Put all n_1n_2 distance counts into one bucket (chosen arbitrarily beforehand or randomly at runtime);
2. Evenly distribute the distance counts into the three buckets involved, i.e., each bucket gets $\frac{1}{3}n_1n_2$;
3. Distribute the distance counts based on the overlaps between range $[u, v]$ and the buckets. In Fig. 7, the distances put into buckets i, $i+1$, and $i+2$ are $n_1n_2\frac{ip-u}{v-u}$, $n_1n_2\frac{p}{v-u}$, and $n_1n_2\frac{v-(i+1)p}{v-u}$, respectively. Apparently, by adapting this approach, we assume the (statistical) distribution of the point-to-point distances between the two cells is uniform;
4. Assuming a spatial distribution model (e.g., uniform) of particles within individual cells, we can generate the statistical distribution of the distances either analytically or via simulations, and put the n_1n_2 distances to involved buckets based on this distribution.

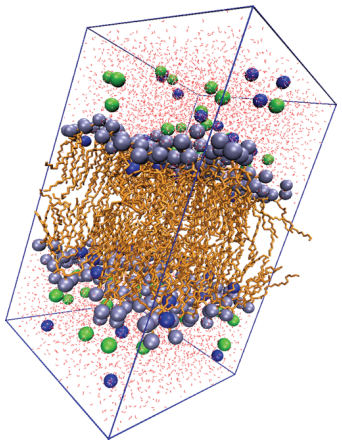


Fig. 10. The simulated hydrated dipalmitoylphosphatidylcholine bilayer system. We can see two layers of hydrophilic head groups (with higher atom density) connected to hydrophobic tails (lower atom density) are surrounded by water molecules (red dots) that are almost uniformly distributed in space.

Note that all four methods need constant time to compute a solution for two cells (In the fourth one, the distribution of the distances can be derived offline). According to our experiments (Section VI-B), they generate much less error than we expect from the theoretical bounds given by Table III.

VI. Experimental Results

We have implemented the algorithms using the C programming language and tested it with various synthetic/real datasets. The experiments are run in an Apple Mac Pro workstation with two dual-core 2.66GHz Intel Xeon CPUs, and 8GB of physical memory. The operating system is OS X 10.5 Leopard.

A. *Exact SDH processing using* DM-SDH

The main purpose of this experiment is to verify the time complexity of DM-SDH. In Fig. 8, the running time of our algorithm are plotted against the size of 2D experimental datasets. Fig. 8a shows the results of using synthetic datasets where the locations of individual atoms are distributed uniformly in the simulated space and Fig. 8b for those following a Zipf distribution with order one, and Fig. 8c for data generated from Pandit's previous work to simulate a bilayer membrane lipid system in NaCl and KCl solutions, as illustrated in Fig. 10. This dataset has 286000 atoms in it, we randomly choose and duplicate atoms in this dataset to reach different total number of atoms to make the experiments comparable to those in Fig. 8a and 8b. Note that both the running time and data size are plotted on logarithmic scales therefore the gradient of the lines reflects the time complexity of the algorithms. In all graphs, we show the results of a series of doubling N values ranging from 100,000 to 6,400,000.[3]

[3]Each point in the graphs shows the result of one single run of DM-SDH as the long running time under large N prohibits having multiple runs. However, we did run multiple experiments with different random seeds for the cases of smaller N and observed very little variances in running time.

For all three datasets, the brute-force approach ('Dist') shows an exact quadratic running time (i.e., the gradient of the line is 2). The other lines (with spots) represent experiments using our algorithm under different bucket numbers. Clearly, the running time of our algorithm grows less dramatically - they all have a gradient of about 1.5. For comparisons, we draw a dotted line in each graph with a slope of exactly 1.5. When bucket size decreases, it takes more time to run our algorithm, although the time complexity is still $\Theta(N^{1.5})$. The cases of large bucket numbers ('$l = 256$') are worth some attention: its running time is similar to that of the brute-force approach when N is small. However, as N increases, the gradient of the line changes to around 1.5. The reason for this is: when N is small, we have a tree with very few levels; when the query comes with a very small bucket size p, we end up starting DM-SDH from the leaf level of the tree (recall Fig. 5) and have to essentially calculate most or all distances. However, the same query will get the chance to resolve more cells when the tree becomes taller, as a result of larger N. Another interesting discovery is that the actual running time for the skewed data (Zipf) is always lower than the uniform data. This can be seen by the relative positions of colored lines to the '$T = O(N^{1.5})$' line. This gain of performance comes from the larger number of empty cells on each density map when the particles are clustered. This also confirms our argument that skewed distribution does not affect the correctness of Theorem 2. The results of the real dataset are almost the same as those for the uniform data.

We have similar results for 3D data (Fig. 9): the corresponding lines for DM-SDH have slopes that are very close to $\frac{5}{3}$, confirming our asymptotic analysis. Again, the cases for large l values are worth more discussions. For '$l = 64$', we started to see the scenarios of '$l = 256$' in the 2D case: the running time grows quadratically till N becomes fairly large (1,600,000) and then the line changes its slope to $\frac{5}{3}$. One thing to notice is the slope of the last segment of '$l = 64$' in Fig. 9b is almost 2. This does not mean the time complexity is going back to quadratic. In fact, it has something to do with the zigzag pattern of running time change in the Zipf data: for three consecutive doubling N values (8-fold increase), the running time increases by 2, 4, and 4 times, which still gives a $2 \times 4 \times 4 = 32$ fold increase in total running time (vs. a 64-fold increase in algorithms with quadratic time).

B. *Approximate histogram processing*

Figure 11 shows the results of running the approximate algorithm. In these experiments, we set the programs to stop after visiting m levels of density maps and distribute the distances using the first three heuristics in Section V. We then compare the approximate histogram with those generated by regular DM-SDH. The error rate is calculated as $\sum_i |h_i - h'_i| / \sum_i h_i$ where for any bucket i, h_i is the accurate count and h'_i the count given by the approximate algorithm.

According to Fig. 11a, the running time does not change with the increase of dataset size for $m = 1, 2, 3$. When m is 4 or 5, the running time increases when N is small and

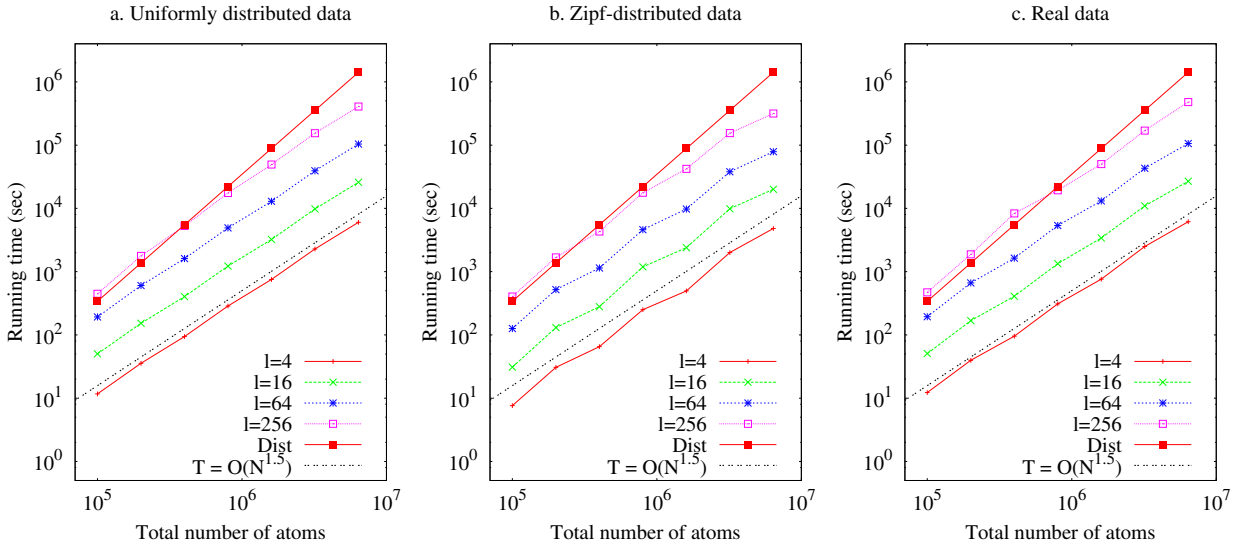


Fig. 8. Running time of the SDH processing algorithms with 2D data.

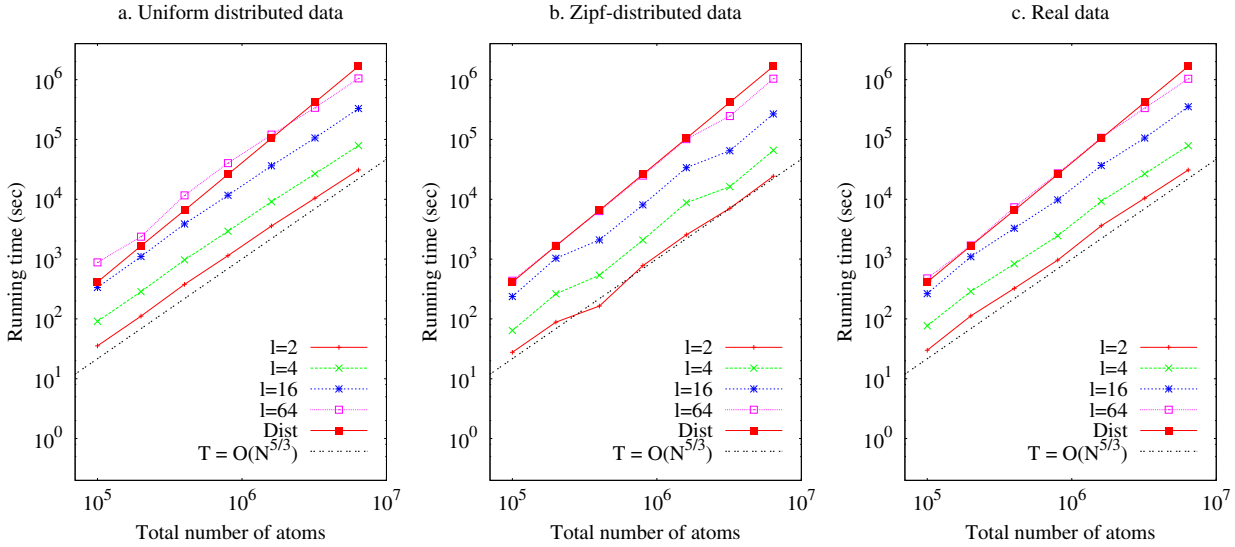


Fig. 9. Running time of the SDH processing algorithms with 3D data.

then stays as a constant afterwards. Note that the 'unlimited' case shows the results of the basic SDH algorithm. Again, this is because the algorithm has less than 4 or 5 levels to visit in a short tree resulted from small N values. In these cases, our algorithm only saves the time to calculate the unresolved distances. When N is large enough, running time no longer changes with the increase of N.

We observe surprising results on the error rates (Fig. 11 c-d): all experiments have error rates under 3%, even for the cases of $m = 1$! These are much lower than the error bounds we get from Table III. The correctness achieved by heuristic 1 is significantly lower than those by heuristic 2 and 3, as expected. The performance of the latter two are very similar except in the case of $m = 1$ where heuristic 2 had error rates around 0.5%. When $m > 2$, the error rate approaches zero with the dataset becomes larger. Heuristic 3 achieves very low error rates even in scenarios with small m values.

C. Discussions

At this point, we can conclude with confidence that the DM-SDH algorithm has running time in conformity with our analytical results. On the other hand, we also see that, in processing exact SDHs, it shows advantages over the brute-force approach only when N is large (especially when l is also big). However, we would not call this a major limitation of the algorithm as the SDH problem is less meaningful when N is small (especially for large l). Its limitation, in our opinion,

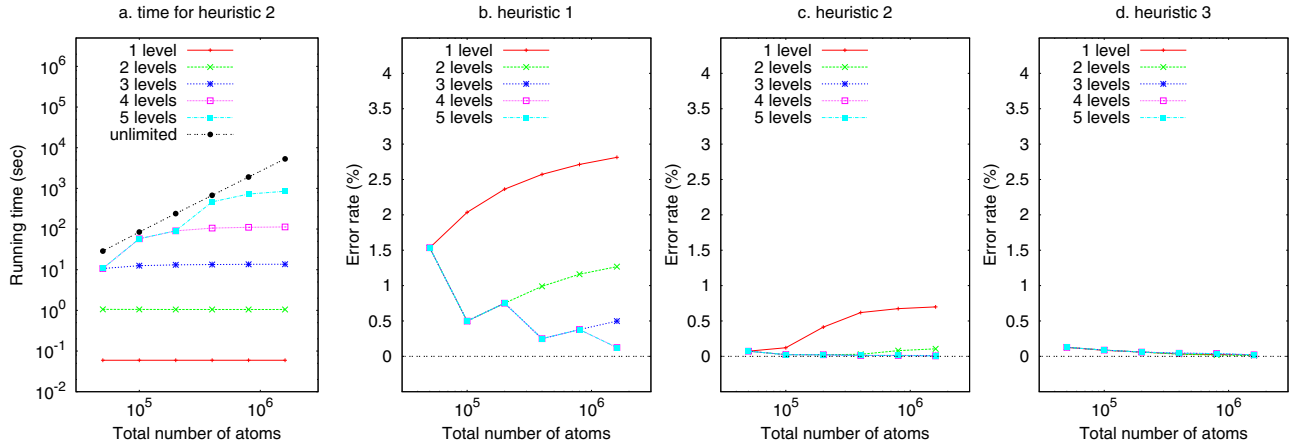


Fig. 11. Running time and correctness of the approximate SDH processing algorithm.

is that the time complexity, although superior to quadratic, is still too high to compute SDH for reasonably large simulation dataset. Fortunately, our approximate algorithm provides an elegant practical solution to the problem. According to our experiments, extremely low error rates can be obtained even we only visit as few as three levels of density maps. Note that for large N, the trees are very likely to have more than three levels to explore even when l is large.

The potential of the approximate algorithm shown by current experiments is very exciting. Our explanation for the surprisingly low error rate is: in an individual operation to distribute the distance counts heuristically, we could have made a significant mistake by putting too many counts into a bucket (e.g., bucket i in Fig. 7) than needed. But the effects of this mistake could be canceled out by a subsequent mistake where too few counts are put into bucket i. The error rate is measured after the binning is done, thus reflecting the net effects of all positive and negative mistakes. While more experiments under different scenarios are obviously needed, investigations from an analytical angle are more urgent. We understand that the bound given by Table III is a loose bound. The real error bound should be described as $\epsilon = \epsilon_1\epsilon_2$ where ϵ_1 is the percentage given by Table III and ϵ_2 is the error rate created by the heuristic binning.

VII. Related Work

Conventional (relational) database systems are designed and optimized towards data and applications from the business world. In recent years, the database community has invested much efforts into constructing database systems that are suitable for handling scientific data. The following are well-known examples of such projects: the GenBank (http://www.ncbi.nlm.nih.gov/Genbank) database for biological sequences; the Sloan Digital Sky Survey [2] to explore over 200 million astronomical objects in the sky; the QBISM project [20] for querying and visualizing 3D medical images; the BDBMS project [3] for handling annotation and provenance of biological sequence data; and the PeriScope [5] project for declarative queries against biological sequences. The main challenges and possible solutions of scientific data management are discussed in [1]. Traditionally, molecular simulation data are stored in large files and queries are implemented in standalone programs, as represented by popular simulation/analytics packages [21]. The scientific community has gradually moved towards using database systems for the storage, retrieval, and analysis of large-scale simulation data, as represented by the BioSimGrid [4] and SimDB [22] projects developed for molecular simulations. However, such systems are still in short of efficient query processing strategies. To the best of our knowledge, the computation of SDH in such software packages is done in a brute-force way.

Although the SDH problem has not been studied in the database community, our work is deeply rooted in the philosophy of using tree-based indexing for pruning the information that is not needed. Quadtree has been a well-studied data structure that has applications spanning databases, image processing, GIS, and computer graphics [23] and has been a topic for research till now: a recent work [24] showed some performance advantages of Quadtree over R-tree in query processing. Various metric trees [25], [26] are closely related to our density map construction: in these structures, space is partitioned into two subsets on each level of the tree. However, the main difference between our work and these tree-based strategies is: we consider the relative distance of cells to group the distance between the data in the cells while they focus more on searching based on a similarity measure.

In particle simulations, the computation of (gravitational/electrostatic) force is of similar flavor to the SDH query. Specifically, the force (or potential) 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 [27] and the Multi-pole algorithm [28] with linear running time. Although both algorithms use a Quad-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; 2). the effects of symmetric interactions cancel out. However, neither features are applicable to SDH computation, in which every pairwise interaction counts and all are equally important.

VIII. Conclusions and Future Work

In this paper, we argue that the SDH query is critical in analyzing particle simulation data. To improve the efficiency of processing this query, we take advantage of the fact that distance calculation can be processed in a batch instead of individually. We build a data structure based on a point region Quadtree to systematically solve this problem. Our analysis shows that the time complexity of our basic algorithm beats current solutions: it runs at $\Theta\left(N^{\frac{2d-1}{d}}\right)$ with d being the data dimension number. An approximate algorithm derived from our approach runs at constant time while giving surprisingly low error rates. We believe our work has provided valuable theoretical and practical insights on the problem such that computing SDH in large simulations has become a reality.

Our work on this topic can be extended in multiple directions. First, the approximate algorithm has shown great potential. Based on the experimental results shown in this paper, we strongly believe there is a tighter bound on the level of errors. Sophisticated (statistical) models should be generated to study this error bound. Second, we should explore more space partitioning plans in building the Quadtree in hope to find one with the "optimal" (or just better) cell resolving percentage. Another topic that we did not pay much attention to is the optimization targeting at the I/O costs. The main issue is how to pack tree nodes into pages and what prefetching strategy we can adopt to improve I/O performance. Finally, our discussions totally ignored another dimension in the data - time. Simulation data are essentially continuous snapshots (called *frames* in simulation terminology) of the simulated system. With large number of frames, processing SDH separately for each frame will take intolerably long time for any meaningful simulation dataset. Incremental solutions need to be developed, taking advantage of the similarity between neighboring frames.

Acknowledgment

The authors would like to thank Prof. Dan Lin of the Department of Computer Science at Missouri University of Science and Technology for sharing insights on this work.

References

[1] J. Gray, D. Liu, M. Nieto-Santisteban, A. Szalay, D. DeWitt, and G. Heber, "Scientific Data Management in the Coming Decade," *SIGMOD Record*, vol. 34, no. 4, pp. 34–41, December 2005.

[2] A. S. Szalay, J. Gray, A. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg, "The SDSS Skyserver: Public Access to the Sloan Digital Sky Server Data," in *Proceedings of International Conference on Management of Data (SIGMOD)*, 2002, pp. 570–581.

[3] M. Y. Eltabakh, M. Ouzzani, and W. G. Aref, "BDBMS - A Database Management System for Biological Data," in *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Resarch (CIDR)*, 2007, pp. 196–206.

[4] M. H. Ng, S. Johnston, B. Wu, S. E. Murdock, K. Tai, H. Fangohr, S. J. Cox, J. W. Essex, M. S. P. Sansom, and P. Jeffreys, "BioSimGrid: Grid-enabled Biomolecular Simulation Data Storage and Analysis," *Future Generation Computer Systems*, vol. 22, no. 6, pp. 657–664, June 2006.

[5] J. M. Patel, "The Role of Declarative Querying in Bioinformatics," *OMICS: A Journal of Integrative Biology*, vol. 7, no. 1, pp. 89–91, 2003.

[6] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithm to Applications*, ser. Computational Science Series. Academic Press, 2002, vol. 1.

[7] M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids*. Clarendon Press, Oxford, 1987.

[8] J. M. Haile, *Molecular Dynamics Simulation: Elementary Methods*. Wiley, New York, 1992.

[9] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulation in Statistical Physics*. Cambridge University Press, Cambridge, 2000.

[10] P. K. Agarwal, L. Arge, and J. Erikson, "Indexing Moving Objects," in *Proceedings of International Conference on Principles of Database Systems (PODS)*, 2000, pp. 175–186.

[11] M. Bamdad, S. Alavi, B. Najafi, and E. Keshavarzi, "A new expression for radial distribution function and infinite shear modulus of lennard-jones fluids," *Chem. Phys.*, vol. 325, pp. 554–562, 2006.

[12] J. L. Stark and F. Murtagh, *Astronomical Image and Data Analysis*. Springer, 2002.

[13] A. Filipponi, "The radial distribution function probed by X–ray absorption spectroscopy," *J. Phys.: Condens. Matter*, vol. 6, pp. 8415–8427, 1994.

[14] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. Mark, and H. Berendsen, "GROMACS: fast, flexible, and free," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1701–1718, December 2005.

[15] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce, "Simulations of the Formation, Evolution and Clustering of Galaxies and Quasars," *Nature*, vol. 435, pp. 629–636, June 2005.

[16] J. A. Orenstein, "Multidimensional Tries used for Associative Searching," *Information Processing Letters*, vol. 14, no. 4, pp. 150–157, 1982.

[17] Y.-C. Tu, S. Chen, and S. Pandit, "Computing Spatial Distance Histograms Efficiently in Scientific Databases," Department of Computer Science and Engineering, University of South Florida, Tech. Rep. http://www.cse.usf.edu/~ytu/pub/tr/pdh.pdf, 2008.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[19] I. Csabai, M. Trencseni, L. Dobos, P. Jozsa, G. Herczegh, N. Purger, T. Budavari, and A. S. Szalay, "Spatial Indexing of Large Multidimensional Databases," in *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Resarch (CIDR)*, 2007, pp. 207–218.

[20] M. Arya, W. F. Cody, C. Faloutsos, J. Richardson, and A. Toya, "QBISM: Extending a DBMS to Support 3D Medical Images," in *ICDE*, 1994, pp. 314–325.

[21] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *Journal of Chemical Theory and Computation*, vol. 4, no. 3, pp. 435–447, March 2008.

[22] M. Feig, M. Abdullah, L. Johnsson, and B. M. Pettitt, "Large Scale Distributed Data Repository: Design of a Molecular Dynamics Trajectory Database," *Future Generation Computer Systems*, vol. 16, no. 1, pp. 101–110, January 1999.

[23] H. Samet, *Foundations of Multidimensional and Metric Data Structure*. Morgan Kaufman, 2006.

[24] Y. J. Kim and J. M. Patel, "Rethinking Choices for Multi-dimensional Point Indexing: Making the Case for the Often Ignored Quadtree," in *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Resarch (CIDR)*, 2007, pp. 281–291.

[25] J. K. Uhlman, "Metric Trees," *Applied Mathematics Letters*, vol. 4, no. 5, pp. 61–62, 1991.

[26] P. N. Yianilos, "Data Structures and Algorithms for Nearest Neighbor Search in Metric Spaces," in *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993, pp. 311–321.

[27] J. Barnes and P. Hut, "A Hierarchical O(N log N) Force-Calculation Algorithm," *Nature*, vol. 324, no. 4, pp. 446–449, 1986.

[28] L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations ," *Journal of Computational Physics*, vol. 135, no. 12, pp. 280–292, 1987.