Computing Spatial Distance Histograms for Large Scientific Datasets On-the-Fly

Anand Kumar, * *Student Member, IEEE,* Vladimir Grupcev, * *Student Member, IEEE,*
Yongke Yuan, Jin Huang, *Student Member, IEEE,* Yi-Cheng Tu, † *Member, IEEE,* Gang Shen, *Member, IEEE*

***Abstract*—This paper focuses on an important query in scientific simulation data analysis: the Spatial Distance Histogram (SDH). The computation time of an SDH query using brute force method is quadratic. Often, such queries are executed continuously over certain time periods, increasing the computation time. We propose highly efficient approximate algorithm to compute SDH over consecutive time periods with provable error bounds. The key idea of our algorithm is to derive statistical distribution of distances from the spatial and temporal characteristics of particles. Upon organizing the data into a Quad-tree based structure, the spatiotemporal characteristics of particles in each node of the tree are acquired to determine the particles' spatial distribution as well as their temporal locality in consecutive time periods. We report our efforts in implementing and optimizing the above algorithm in Graphics Processing Units (GPUs) as means to further improve the efficiency. The accuracy and efficiency of the proposed algorithm is backed by mathematical analysis and results of extensive experiments using data generated from real simulation studies.**

***Index Terms*—Scientific databases, spatial distance histogram, quad-tree, density map, spatiotemporal locality, GPU**

I. Introduction

The advancement of computer simulation systems and experimental devices has yielded large volume of scientific data. This imposes great strain on the data management software, in spite of effort made to deal with such large amount of data using database management systems (DBMS) [1]–[3]. But the traditional DBMSs are built with business applications in mind and are not suitable for managing scientific data. Therefore, there is a need to have another look at the design of the data management systems. Data in scientific databases is generally accessed through high-level analytical queries, which are much more complex to compute in comparison to simple aggregates. Many of these queries are composed of few frequently used analytical routines which usually take super-linear time to compute using brute-force methods. Hence, the scientific database systems need to be able to efficiently handle the computation of such analytical queries. This paper presents our work related to such type of a query that is very important for the analysis of *molecular simulation* (MS) data.

Molecular (or particle) simulations are simulations of complex physical, chemical or biological structures done on computers. They are extensively used as a basic research tool for analyzing the behavior of natural systems under experimental framework [4], [5]. The number of particles involved in MSs is large, oftentimes counting millions. In addition, simulation datasets may consist of multiple snapshots (*frames*) of the system's state at different time points.

In order to analyze the MS data, scientists compute complex quantities through which statistical properties of the data is shown. Often times, queries used in such analysis count more than one particle as basic unit: such a function involving all m-tuple subsets of the data is called an m-body correlation function. One such analytical query discussed in this paper, is the so called *spatial distance histogram* (SDH) [6]. An SDH is the histogram of distances between all pairs of particles in the system and it represents a discrete approximation of the continuous probability distribution of distances named Radial Distribution Function (RDF). Being one of the basic building blocks for a series of critical quantities (e.g., total pressure and energy) required to describe the physical systems, this type of query is very important in MS databases [4].

Objectives: Our goal with this work is to perform SDH computation on a high level of efficiency and accuracy. Specifically, our approach fundamentally improves over existing solutions by achieving on-the-fly query processing. This is accomplished via a number of techniques that take advantage of spatiotemporal locality within the data and multi-core parallel processing architecture of modern Graphical Processing Units (GPUs). We provide theoretical proof for guaranteed error bound that is validated with experimental results.

A. Problem Statement

The SDH problem can be formally described as follows: given the coordinates of N particles and a user-defined distance w, we need to compute the number of particle-to-particle distances falling into a series of ranges (named buckets) of width w: $[0, w), [w, 2w), \ldots, [(l-1)w, lw]$. Essentially, the SDH provides an ordered list of non-negative integers $H = (h_0, h_1, \ldots, h_{l-1})$, where each $h_i (0 \leq i < l)$ is the number of distances falling into the bucket $[iw, (i+1)w)$. We also use $H[i]$ to denote h_i in this paper. Clearly, the bucket width w is the only parameter of this type of problem.

* These authors contributed equally to this work.
† Author to whom all correspondence should be sent.

Anand Kumar, Vladimir Grupcev and Yi-Cheng Tu are with the Department of Computer Science and Engineering, University of South Florida, 4202 E. Fowler Ave., ENB 118, Tampa, FL 33620, U.S.A. Emails: akumar8@mail.usf.edu, vgrupcev@mail.usf.edu, ytu@cse.usf.edu

Yongke Yuan is with the School of Economics and Management, Beijing University of Technology, 100 Pingleyuan, Chaoyang District, Beijing 100124, China Email: yyuan@gmail.com

Jin Huang is with the Department of Computer Science, University of Texas at Arlington, 500 UTA Boulevard, Room 640, ERB Buildings, Arlington, TX 76019, U.S.A. Email: jin.huang@mavs.uta.edu

Gang Shen is with the Department of Statistics, North Dakota State University, 201H Waldron Hall, Fargo, ND 58108, U.S.A. Email: gang.shen@ndsu.edu

To capture the variations of system states over time, there is a need to compute SDH for a large number of consecutive frames. We denote the count in bucket i at frame j as $H_j[i]$.

B. Overview of Our Approach.

This paper presents a *highly efficient and practical* algorithm for processing SDH of large-scale MS data with improved efficiency and accuracy over existing solutions. To achieve this, the algorithm takes advantage of the two types of uniformity widely present in MS data. To further improve the running time of the algorithm, we utilize Graphics Processing Unites (GPUs).

The first type of data uniformity used by the algorithm refers to the *spatial distribution of data points* (e.g., atoms) in MS datasets. It is well known that parts of natural systems tend to spread out evenly in space due to the existence of inter-particle forces and/or chemical bonds [7], [8]. Because of this, there are many localized regions (we call uniform regions) in the simulation space in which the particles are uniformly distributed.[1] We treat such regions as single entities when computing SDH. Once we identify these uniform regions (using the χ^2 test), we derive the Probability Distribution Functions (PDFs) of the distances between all pairs of these regions by either:

- Mathematical analysis towards a closed-form, or
- Monte Carlo simulations;

Exploiting this property makes algorithm running time independent of the SDH bucket width w – such dependency (as discussed in Section II) is the main drawback of existing algorithms. On the other hand, working with the PDFs of distance distribution guarantees very little error will be made, as shown by our rigorous analysis of the algorithm (Section VI).

The second type of uniformity is about the significant *similarity of the spatial distributions among consecutive frames*. We have observed that such similarity is reflected in the final results of the SDH obtained for neighboring frames. So, given two frames f_0 and f_1, if we have already computed the SDH of f_0, we can obtain the SDH of f_1 by dealing only with the regions that do not exhibit similarity between the two frames while ignoring regions that are similar. To take advantage of such similarities among frames, we design an *incremental algorithm* that can quickly compute SDH of a frame from the SDH of a base frame obtained using traditional single-frame algorithms.

Finally, our algorithm takes advantage of the multi-core parallel processing feature of GPUs. They provide a low-cost and low-power platform to improve efficiency as compared to computer clusters. However, the GPU architecture imposes challenges in developing software that takes full advantage of their computing capability. To address such challenges we develop several techniques that are very different from those used in optimizing CPU-based systems. The techniques generate significant boosts in performance (and energy efficiency) as compared to straightforward GPU implementations.

[1]This does not make the data system-wise uniform. Otherwise, SDH computation becomes a trivial task.

C. Contributions and Paper Organization

We have implemented a composite algorithm combining the above ideas and tested it on real MS datasets. The experimental results clearly show the superiority of the proposed algorithm over previous solutions in both efficiency and accuracy. For example, with the proposed algorithm, we are able to compute 11 frames of a 8-million-atom dataset in less than a second! In addition to a highly efficient and practical algorithm for SDH processing, we also believe that our success will open up new directions in the molecular simulation paradigm. Our work builds a solid foundation for solving the more general and difficult problem of multi-body (m-body) correlation function computation [9]. With a $O\big(N^m\big)$ complexity in nature, such problems can be addressed using the methodologies proposed in this paper.

The major technical contributions presented here are:

1) Techniques to identify spatial uniformity within a frame and temporal uniformity among consecutive frames;
2) An approximate algorithm to compute the SDH of large number of data frames by utilizing the above properties;
3) Analytical and empirical evaluation of the above algorithm, especially rigorous analysis of the tradeoff between performance and guaranteed accuracy; and
4) Implementation of the above algorithms in modern GPUs to boost performance, with a focus on the optimization of such implementations in a GPU environment.

Preliminary results addressing the problem of computing approximate SDH using spatial and temporal uniformities were first reported in [10] (contributions 1 and 2). This work extends the idea of [10] by providing rigorous analysis, empirical evaluation of the algorithm, and implementation in modern GPUs to boost performance.

The remainder of this paper is organized as follows: in Section II we give an overview of the work done in the field related to the SDH problem. Then, in Section III we introduce the main concepts and techniques utilized in our work. Sections IV and V discuss the utilization of the spatio-temporal properties of the data to enhance the algorithm. Then, in Section VI the performance (running time and errors) of the proposed technique in utilizing the spatio-temporal property of the data is analyzed. In Section VII we briefly look at the basic architecture of the GPUs and their programming paradigms and we modify our algorithm to map onto the GPU. Section VIII presents the results obtained through extensive experiments. Finally, we conclude this paper with Section IX in which we also discuss our future work. Due to space limitation, certain details are presented in respective appendices which are part of the supplementary materials.

II. Comparison to Related Work

The brute-force method for SDH computation calculates the distances between all pairs of particles and updates the relevant buckets of the histogram. This method requires quadratic time. Some of the popular software for analyzing the MS data, like GROMACS [11], still utilizes the brute-force method. But, the current state-of-the-art models for SDH computation involve methods that treat a cluster of particles as a single processing unit [6], [12]. Space-partitioning trees (like kd-trees [12]) are often used to represent the system, each node of the tree representing one cluster. The main idea in such approach is to process all the particles in each node of the tree as a whole. This is an obvious improvement in terms of time over the brute-force method which builds the histogram by computing particle-to-particle distances separately. A Density-Map based SDH algorithm (DM-SDH) using a quad-tree data structure is presented in our previous work [6]. It has been proven that the running time for DM-SDH is $\Theta(N^{\frac{3}{2}})$ for 2D data and $\Theta(N^{\frac{5}{3}})$ for 3D data. We will go over the main idea of DM-SDH in more detail later in this paper. Although the DM-SDH algorithm is an improvement over the brute-force method for SDH computation, it is still not a practical and efficient solution for the following reasons:

(1) The running time of DM-SDH increases dramatically as input size N increases and the bucket width w decreases. Therefore, the running time can be greater than that of the brute-force method [6]!
(2) DM-SDH only addresses SDH computation of a single frame whereas MS data analysis of any system requires computation over multiple consecutive frames. To achieve this, DM-SDH needs to be run for every frame. This is not quite acceptable, since usually the number of frames is of the order of tens of thousands.

An approximate SDH algorithm (ADM-SDH), with running time not related to the data size N was introduced in [6]. But its running time is influenced by a guaranteed error bound as well as by the bucket size w. Like the DM-SDH, it also can only be applied to a single frame of the MS system. A thorough analysis of the performance of ADM-SDH is presented in a recent paper [13]. Under some assumptions, that paper also derives an error bound of ADM-SDH that is tighter than the one presented in [6]. We will briefly mention such findings in Section VI-A2. To remedy the cons of the aforementioned algorithms, we direct our current work in designing a new, improved algorithm with higher efficiency and accuracy. Furthermore, we are able to substantially decrease the running time of the algorithm by implementing and optimizing the code in a GPU programming environment. The result is a solution that is both practical and efficient, delivering very accurate results in (almost) real-time manner. A shorter version of this paper can be found in [10]: this paper extends [10] by introducing a rigorous analysis of the key distribution function in Section IV, a GPU version of the algorithm in Section VII, and enhanced performance analysis in Section VI.

It is important to note that the SDH problem is often confused with the force/potentional fields computation in the MS process [14], [15]. In the latter, the physical properties of a particle are determined by the forces applied to it by all other particles in the system. Although the force/potentional fields computation has similar definition to the SDH problem, the algorithms used to solve such problem are not useful in computing the SDH. There is a detailed comparison of the two problems in [16]. Here, we will just note that the force field computation is for simulation of a system, while the SDH computation is for system analysis.

The problem of SDH computation over multiple consecutive frames is related to persistent data structures [17], [18], which allow for various versions of the computation results to be maintained and updated over time for quick query processing. Building persistent index schemes on complex spatio-temporal data allows for a time efficient retrieval [19]. There has been a detailed survey of applications, made by Kaplan [20], in which persistent data structure has been used to improve efficiency. Such structures are designed to resolve the I/O bottleneck problem. But, the multi-frame SDH problem involves heavy computation at each instance, overshadowing the I/O time. Thus, the techniques developed for persistent data can hardly be used for efficient multi-frame SDH computation.

In recent years, there has been a growing interest in improving the performance of computationally intensive tasks using special hardware, such as GPUs [21], [22]. These devices were originally designed for processing graphics, but their parallel computing capability can be utilized for general purpose computing via software frameworks such as CUDA [23] and OpenCL [24]. A number of database operators are implemented on GPUs: relational join [25]; relational operators and aggregations [26]; and sorting. An overview of the GPU techniques is presented in a survey by Owens *et al.* [27]. In this work, we leverage the computing power of the GPUs to achieve the goal of on-the-fly SDH computation with guaranteed accuracy.

III. BACKGROUND

In this section, we introduce the main concepts of our existing work [6] that will serve as a foundation for the proposed algorithm. To represent the simulation data space we use a conceptual data structure that we call *density map* (DM). A DM divides the simulation space into a grid of equal sized cells (or regions). A cell is a square in 2D and a cube in 3D.[2] To generate a density map of higher resolution, we divide each cell of the grid into four equally sized cells. We use a region quad-tree [28] to organize different density maps of the same data. Each cell of the DM is represented by a tree node, so a density map is essentially the collection of all nodes on *one level* of the tree. Each node of the tree contains the cell location (i.e., coordinates of corner points) as well as the number of particles in it. One thing to note here is that we stop building the tree when the number of particles in a node drops below 5 because, otherwise the cost of resolving nodes could be higher than directly retrieving the particles and calculating point-to-point distances [6]. We refer to such a tree as the Density-Map Tree (DM-tree).

The fundamental part of the DM-SDH algorithm is a procedure we call RESOLVETWOCELLS. This procedure takes two cells (e.g., A and B in Fig. 1) from a density map as an input and computes the minimum and maximum distance (denoted as u and v) between them in constant time. The main task of this procedure is to determine whether the two

[2]In this paper, we focus on 2D data to elaborate and illustrate the proposed ideas.

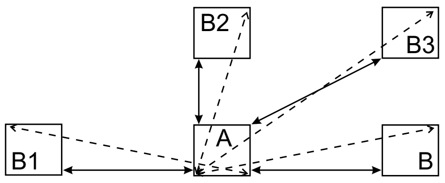


Fig. 1. Computing minimum (i.e., length of solid lines) and maximum distance (i.e., length of dashed lines) range between two cells

cells are resolvable or not. We call a pair of cells *resolvable* if both u and v fall into the same SDH bucket i. In such case we increment the distance count of that bucket by $n_A n_B$, where n_A and n_B are the number of particles in cell A and B, respectively. In the case of non-resolvable cells, we either:

(1) Go to the next density map with higher resolution and resolve all children of A with those of B, or
(2) Compute every distance between particles of A and B and update the histogram accordingly, if it is the leaf-level density map.

To get the complete SDH of the MS system, the algorithm executes the RESOLVETWOCELLS procedure for all pairs of cells on a given density map DM_k (the DM where the diagonal of a cell is smaller than or equal to the bucket width w). So, basically, the algorithm calls RESOLVETWOCELLS recursively (i.e., action (1) above) till it reaches the leaf level of the tree (i.e., action (2) above).

The idea behind the approximate algorithm (ADM-SDH) is to recursively call RESOLVETWOCELLS only for a predetermined number (m) of levels in the tree. If after visiting the m levels, there are unresolved pairs of cells, heuristics is being used to greedily distribute distances into relevant SDH buckets. We will study the heuristics for distance distribution in Section IV. The main benefit of this algorithm is: given a user specified error bound ϵ, our analytical model can tell what value of m to choose [6]. Although ADM-SDH is fast in regard to the data size N, its running time is very sensitive to the bucket width w. The main reason for this is: when w decreases by half, we have to start the algorithm from the next level of the tree. As a result, the number of pairs of cells I increases by a factor of 2^{2d} (d is number of dimension). Since the SDH is a discrete approximation of a continuous distribution of the distances in the MS system, more information is lost with the increase of w. Scientists prefer smaller values of w so that there are a few hundred buckets in the SDH. Here, we present an efficient and accurate *multi-frame SDH computing algorithm whose performance is insensitive to both N and w*. This new algorithm uses the same region quad-tree for data organization as in the DM-SDH and ADM-SDH algorithms.

IV. SDH Computation Based on Spatial Uniformity

A. Algorithm Design

A DM-based algorithm depends heavily on resolving cells to achieve the desired accuracy. It applies heuristics to distribute the distances into relevant buckets after visiting m levels of the tree or after reaching the leaf nodes. That is the main reason for the long running time. Our idea to remedy

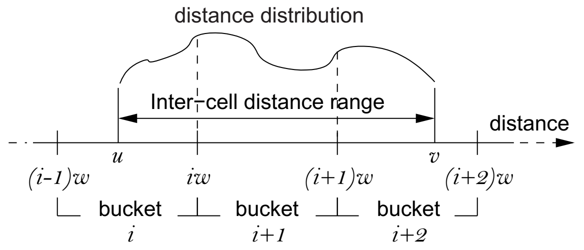


Fig. 2. Distance range of non-resolvable cells overlaps with more than one bucket of the SDH

that problem is to greedily distribute distances between very large regions of the simulation space, even when no pairs of such regions are resolvable. In other words, we *use heuristics for distance distribution as early as possible.* However, the distribution of distances between two large regions may yield arbitrarily large errors. Therefore, the key challenge is to design a heuristic with high accuracy even under large regions.

Our first idea to address the aforementioned challenge is to take advantage of the spatial distribution of data points in the cells. As illustrated in Fig. 2: two cells have a distance range $[u, v]$ which overlaps with three SDH buckets (i.e., from bucket i to $i + 2$). A critical observation here is: if we knew the probability distribution function (PDF) of the point-to-point distances between cells A and B, we can effectively distribute the actual number of distances $n_A n_B$ into the three overlapping SDH buckets. Specifically, the total number of $n_A n_B$ distances will be assigned to the buckets based on the probability of a distance falling into each bucket according to the PDF. For the case in Fig. 2, the relevant SDH buckets and the number of distances assigned to them are as follows:

$$H[i], \qquad n_A n_B \int_{u}^{iw} g(t)dt \tag{1}$$

$$H[i+1], \qquad n_A n_B \int_{iw}^{(i+1)w} g(t)dt \tag{2}$$

$$H[i+2], \qquad n_A n_B \int_{(i+1)w}^{v} g(t)dt \tag{3}$$

where g is the PDF. The biggest advantage of the above approach is that the errors generated in each distance count assignment operation can be very low, and the errors will not be affected by the bucket width w, as long as the PDF is an accurate description of the underlying distance distribution [29]. This is because each integration in the right column is actually the probability of a distance falling into the corresponding bucket shown on the left. Therefore, the main task of the proposed approach is to derive the PDF.

Methods for deriving the PDF: Note that in the work presented in [6], the distances are proportionally distributed into the three buckets based on the overlaps between range $[u, v]$ and the individual buckets. Such a primitive heuristic, which is named PROP (short for "proportional"), implicitly assumes that the distance distribution is uniform within $[u, v]$. However, our experiments show that a typical distance distribution in MS data is far from being uniform. Hence, our proposed solution will naturally introduce less errors than the PROP heuristics adopted by ADM-SDH.

In general, the PDF of interest can be obtained by the spatial distribution of particles in the two relevant cells. The coordinates of any two particles - one from A and the other from B - can be modeled as two random vectors $\vec{v}_A$ and $\vec{v}_B$, respectively. The distance between these two particles can also be modeled as a random variable D, and we have

$$D = ||\vec{v}_A - \vec{v}_B||. \tag{4}$$

Given that, if we know the PDFs of both $\vec{v}_A$ and $\vec{v}_B$, the PDF of D can be derived by one of the following strategies:

(1) generation of a closed-form via analyzing the PDFs of $\vec{v}_A$ and $\vec{v}_B$ as well as Eq. 4; or
(2) Monte Carlo simulations using the PDFs of $\vec{v}_A$ and $\vec{v}_B$ as data generation functions.

In practice, it is difficult to get a closed-form PDF for D even when the particle spatial distributions follow a simple form. In Section IV-B, we present the results of our efforts in obtaining such a closed-form PDF for D under uniformly distributed $\vec{v}_A$ and $\vec{v}_B$.

Monte Carlo simulations can help us obtain a discrete form of the PDF of the distance distribution, given the PDFs of the particle spatial distributions [5]. One important note here is that *the method works no matter what forms the spatial distributions follow*. However, to generate the particle spatial distributions, it is infeasible to test the MS dataset for all possible data distributions. Instead, we focus on testing if the data follows the most popular distribution in MS - *spatial uniform distribution*. We should note here that the particle spatial distribution is different from the distribution of distances between particles.

Pseudocode of the algorithm is shown in Algorithm 1. The algorithm identifies uniform regions (Appendix B) and chooses a density map DM_k (Section VI) before computing SDH. Physical study of molecular systems have shown that it is normal to see a small number of large uniform regions covering most of the particles, leaving only a small fraction of particles in non-uniform regions [7], [8]. This is also verified by our experiments using real MS datasets (Section VIII). This translates into high efficiency of the proposed algorithm. Furthermore, the time complexity is unrelated to the bucket size w.

One detail skipped in the algorithm is that we also need to assign intra-cell distances to the first few buckets of the SDH. In particular, given a cell A with diagonal length of q, the distances between any two particles in A fall into the range $[0, q]$, and can be modeled as the following random variable:

$$D' = ||\vec{v}_A - \vec{v'}_A|| \tag{5}$$

where $\vec{v'}_A$ is an independent and identically distributed variable to $\vec{v}_A$. Let us further assume the range $[0, q]$ overlaps with buckets 0 to j. Then we can follow the same idea shown in Eqs. 1-3 to assign the distance counts of cell A into the relevant buckets (Appendix G).

B. Analysis of the PDF

In practice, it is difficult to get a closed-form PDF for D even when the particle spatial distributions follow a simple form. There has been some work done in [30] that addresses one special case: tackling the distribution of distance between points within a unit square – this can be seen as a case of variable D' shown in Eq. (5). The distribution of random variable D is also studied in [31] under the special case that $\vec{v}_A$ and$\vec{v}_B$ are from two *adjacent* unit squares. Both work show closed-form formulae that can be used in our algorithm (we list their results in Appendix C). To the best of our knowledge, there has not been any work that achieved derivation of a closed-form for the general cases.

Algorithm 1 Computing SDH using uniform regions

```
1:  procedure UNIFORMREGIONSDH(root $Q$, X%)
2:      Identify uniform regions (cells) in tree rooted at $Q$
3:      Choose level $k$ in $Q$ if % of uniform cells $\geq$ X
4:      for each cell $A$ in $DM_k$ do
5:          if $A$ is uniform region then
6:              Derive distance distribution PDF $g_{D'}(t)$
7:              Update $H[0 \dots j]$ using $g_{D'}(t)$
8:          else
9:              Update $H[0 \dots j]$ using PROP heuristic
10:     for each pair $A, B$ ($A \neq B$) of cells in $DM_k$ do
11:         if both $A$ and $B$ are uniform regions then
12:             Derive distance distribution PDF $g(t)$
13:             Update $H[i \dots j]$ using $g(t)$
14:         else
15:             Update $H[i \dots j]$ using PROP heuristic
```

In this part, we show the results of our efforts in obtaining an approximate closed-form for the general case: finding distance distribution between points in any two cells. The main claim is: *if the data points in cells A and B are uniformly distributed, then the square of the distance between the two cells' points can be approximated by a Noncentral chi-square distribution and the distribution is not related to the number of points in cell A or B*.

To shed more light on the claim, we take a look at two randomly chosen cells A and B of same size (i.e., from the same level of the DM tree). We start by assuming that the particles in the cells are uniformly distributed. Our goal here is to give a representation of the PDF of the distance between points in such two cells. Let us choose two random points P_A and P_B, from cell A and B, and denote the coordinates of P_A and P_B as (X_{PA}, Y_{PA}) and (X_{PB}, Y_{PB}), respectively. The square of the distance D between these two points can be expressed with the following equation:

$$D^2 = |X_{PA} - X_{PB}|^2 + |Y_{PA} - Y_{PB}|^2.$$

Since the points are chosen randomly, their coordinates can be regarded as random variables. Furthermore, $|X_{PA} - X_{PB}|$ and $|Y_{PA} - Y_{PB}|$ can be viewed as random variables that follow a triangular distribution. But using triangular distribution would make the result and the analysis really hard (if not impossible) to achieve. In order to ease the analysis process we will approximate the triangular distribution with a normal distribution. So, naturally, we continue by first figuring out how much error will be introduced by such approximation.

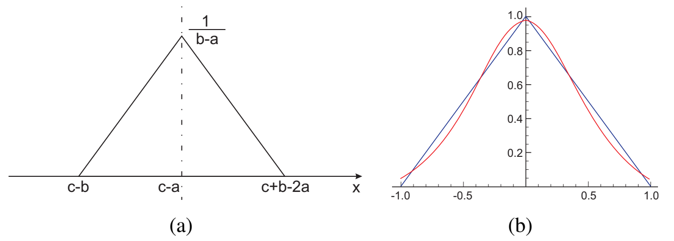


Fig. 3. (a) The distribution of Y-X. (b) The difference between normal and triangular distributions

The following subsection shows that the introduced error is only 10%.

1) Approximating Triangular with Normal distribution:

Lemma 1: If X and Y are independent random variables uniformly distributed on (a, b) and $(c, c+b-a)$, and $c \geq a$, then $Y-X$ is a triangular random variable and can be regarded as a normal random variable with total variation distance 0.1.

Proof:

The probability density of X is

$$f(x) = \frac{1}{b-a}, \quad a < x < b \tag{6}$$

and the probability density of Y is

$$g(y) = \frac{1}{b-a}, \quad c < y < c+b-a \tag{7}$$

There are two cases to be considered: (1) when c is equal to a; and (2) when c is greater than a.

Case 1 ($c = a$): The probability density of $Y - X$ (shown in Figure 3(a)) can be calculated as follows

$$f_{Y-X}(z) = \int_{c}^{c+b-a} f(y-z)g(y)dy \tag{8}$$

When $0 > z > a-b$, the probability density of $Y - X$ can be computed as follows

$$f_{Y-X}(z) = \frac{b-a+z}{(b-a)^2} \tag{9}$$

When $0 < z < b-a$, the probability density of $Y - X$ can be computed as follows

$$f_{Y-X}(z) = \frac{b-a-z}{(b-a)^2} \tag{10}$$

Case 2 ($c > a$):

When $c-b < z < c-a$, the probability density of $Y - X$ can be computed as follows

$$f_{Y-X}(z) = \frac{b-c+z}{(b-a)^2} \tag{11}$$

When $c-a < z < c+b-a$, the probability density of $Y - X$ can be computed as follows

$$f_{Y-X}(z) = \frac{c+b-2a-z}{(b-a)^2} \tag{12}$$

Now, let us take a look at a different random variable Q. Assuming Q is a normal random variable with parameters $(c-a, \frac{(b-a)^2}{6})$, the probability density of Q can be written as follows:

$$f_Q(x) = \frac{\sqrt{6}e^{\frac{-6(x-c+a)^2}{2(b-a)^2}}}{\sqrt{2\pi}(b-a)} \tag{13}$$

Let $u = \frac{x-(c-a)}{b-a}$, then the probability density of Q can be rewritten as follows

$$f_Q(u) = \frac{\sqrt{6}e^{-3u^2}}{\sqrt{2\pi}} \tag{14}$$

Let $v = \frac{z-(c-a)}{b-a}$.Then, the probability density of $Y - X$ can be rewritten as follows

$$f_{Y-X}(v) = Min\{1+v, 1-v\}, \quad -1 < v < 1 \tag{15}$$

Now let us study how well the normal distribution approximates the triangular distribution. Let P be the triangular distribution with PDF $p(x)$ given by Eq. (15) and Q is the normal distribution $N(0, \frac{1}{6})$, the PDF of which is $q(x) = \frac{\sqrt{6}\exp\{-3x^2\}}{\sqrt{2\pi}}$, $x \in (-\infty, +\infty)$. Note that the first two moments of the two distributions are exactly the same: $E[P] = E[Q] = 0$, and $\text{var}[P] = \text{var}[Q] = \frac{1}{6}$. Although these indicate the similarity between the two distribution, we still want to quantify how close the two probability measures are.

A natural measure of the difference between two probability measures P and Q is the *total variation distance* defined as:

$$V(P, Q) \triangleq \sup_{A \in \mathscr{F}} |P(A) - Q(A)| = \frac{1}{2}\int_R |p(x) - q(x)|dx,$$

where $\mathscr{F}$ is the $\sigma-$field upon which the probability space is defined. Note that since P is absolutely continuous with respect to Q in our case, then we have

$$V(P, Q) = \frac{1}{2}\int_R \left|\frac{p(x)}{q(x)} - 1\right| q(x)dx$$

Numerical computation shows that $V(P, Q) = 0.1012$.

Following the above reasoning, we conclude that we can use normal distribution instead of triangular distribution, introducing an error of about 10%. ■

Using the aforementioned findings, we now regard the differences $(X_{PA} - X_{PB})$ and $(Y_{PA} - Y_{PB})$ as random variables with normal distribution. In the following subsection, we continue with the proof of our main claim. Here we show that the square of the distance can be viewed as random variable with non-central chi-squared distribution.

2) Distance distribution of particles in two cells A and B: As we know, if X_{PA} and X_{PB} are independent random variables uniformly distributed on (a, b) and $(c, c+b-a)$ respectively, $(c \geq a)$ then $X_{PB} - X_{PA}$ follows a triangular distribution that we saw can be approximated with normal, introducing an error of not more than 10%. Knowing this, $X_{PB} - X_{PA}$ can be regarded as a normal random variable with parameters $(c-a, (b-a)^2/6)$. Similarly, $Y_{PB} - Y_{PA}$ can be regarded as a normal random variable with parameters $(c'-a', (b-a)^2/6)$. Since $(b-a)^2/6$ is a constant, which is noted as σ^2, we can write the following equation for the distance D between the two points P_A and P_B:

$$\frac{D^2}{\sigma^2} = \frac{(X_{PB} - X_{PA})^2}{\sigma^2} + \frac{(Y_{PB} - Y_{PA})^2}{\sigma^2} \tag{16}$$

As it is known, the right hand side of the above equation is a Noncentral chi-square distribution. This means that the distances between the two cells' points can be described as a Noncentral chi-square distribution with the parameters $(2, \lambda)$, where λ can be defined as follows:

$$\lambda = \frac{(c-a)^2}{\sigma^2} + \frac{(c'-a')^2}{\sigma^2} \tag{17}$$

where $c-a$ and $c'-a'$ are the means of the two normal distributions.

Note that, since our discussions started with the only assumption that points in A and B are uniformly distributed, the parameters of above PDF have no relationship with the actual number of points in cell A or cell B.

So, our conclusion is that the square of the distance between any two points from two cells follows (can be approximated to) a Noncentral chi-square distribution. Since the PDF of a Noncentral chi-square distribution has a closed form [32], the PDF of D (i.e., the square root of the Noncentral chi-square) can be obtained through Jacobian transformation. However, we stop here after obtaining the (approximated) PDF of D^2 since it can already be used to guide distance distributions in our algorithm with minor tweaks. Recall the scenario in Figure 2: the share of distance counts that should go into bucket i is now $\int_{u^2}^{i^2 w^2} h(t)dt$ where $h(t)$ is the PDF of the Noncentral chi-square. The other buckets can be treated in a similar way.

It is our belief, based on the work we have done on this matter, that to get an explicit and more accurate closed form for the distribution of the distances between points of the cells is a really challenging, if not impossible to solve, problem.

C. Monte Carlo Simulations

The distribution of distances between a pair of cells, say A and B, can be determined based on their spatial distribution of particles, by running Monte Carlo simulations. Monte Carlo simulation is a way to model a phenomenon that has inherent uncertainty [5]. If the spatial distributions of particles in A and B are known to be uniform, the simulations can be done by sampling (say n_s) points independently at random from uniform distributions within the spatial ranges of A and B. Then, the distance distribution is computed from the points sampled in both cells. A *temporary* distance histogram can be built for this purpose. All n_s^2 distances are computed (brute-force method), and put into buckets of the temporary histogram (e.g., those overlapping with $[u, v]$ in Fig. 2) accordingly. This temporary histogram is used to obtain the PDF of point-to-point distances between cells A and B (the $g(t)$ of $Eqs.1-3$), which is then used to update the SDH buckets.

Sufficient number of points are needed to get reasonably high accuracy of the SDH generated [29]. The cost of running such simulations can be high if we were to perform one simulation for each pair of uniform regions. This, fortunately, is not the case. First, let us emphasize that the simulations are not related to the number of particles (e.g., n_A and n_B) in the cells of interest - the purpose is to approximate the PDF of distance distribution. Second, and most importantly, the same simulation can be used for multiple pairs of cells in the same density map, as long as the two cells in such pairs have the same relative position in space. A simple example is shown in Fig. 1: cell pairs (A, B) and (A, B_1) will map to the same range $[u, v]$ and can definitely use the same PDF. A systematic analysis of such sharing is presented in following theorem.

Theorem 1: The number of distinct Monte Carlo simulations performed in a density map of M cells, is $O(M)$.

Proof: See Appendix D. ■

Theorem 1 says that, for the possible $O(M^2)$ pairs of uniform regions on a density map, there are only a linear number of simulations needed to be run. Furthermore, as we will see in Section V, the same cells exist in all frames of the dataset, thus a simulation run for one frame can be shared among all frames. Given the above facts, we can create a lookup table (e.g., hash-based) to store the simulation results to be shared among different operations when a PDF is required.

Remark 1: If we were given the PDF of the random variable D and use the integration of the PDF to guide distance distribution in step (2) of our algorithm, the number of distinct integrations is also $O(M)$.

V. SDH Computation Based on Temporal Locality

Another inherent property of the MS is that the particles often exhibit temporal locality which can be utilized to compute the SDH of consecutive frames even faster. The existence of temporal locality is mainly due to the physical properties of the particles in most of the simulation systems [7]. More specifically, such properties can be observed at the following two levels:

(1) Particles often interact with each other in groups and move randomly in a very small subregion of the system;
(2) With particles moving in and out of a cell, the number of particles in that cell does not change much over time.

A. Basic Algorithm Design

We discuss the algorithm in terms of only two frames f_0 and f_1, although the idea can be extended to an arbitrary number of frames. Suppose DM-trees T_0 and T_1 are built for the two frames f_0 and f_1, respectively. Since the DM-trees are built independently from the data they hold, the number of levels and cells, as well as the dimensions of corresponding cells in both DM-trees will be the same. First, an existing algorithm (e.g., DM-SDH or ADM-SDH) is used to compute the SDH H_0 for the *base frame* f_0. Then we copy the SDH of frame f_0 to that of f_1, i.e., $H_1 = H_0$. The idea is to modify the initial value of H_1 to reach its correct form by *only processing cells that do not show temporal locality.*

Let DM_k^0 and DM_k^1 be the density maps, at level k, in their respective DM-trees T_0 and T_1. We augment each cell in DM_k^1 with the *ratio of particle count of that cell in DM_k^1 to the particle count of the same cell in DM_k^0*. A density map that has such ratios is called a *ratio density map* (RDM). The next step is to update the histogram H_1 according to the ratios in the RDM. Let r_A and r_B $(A \neq B)$ be the density ratios of any two cells A and B in the RDM, we have two scenarios:

Case 1: $r_A \times r_B = 1$. In this case, we do not make any changes to H_1. It indicates that the two cells A and

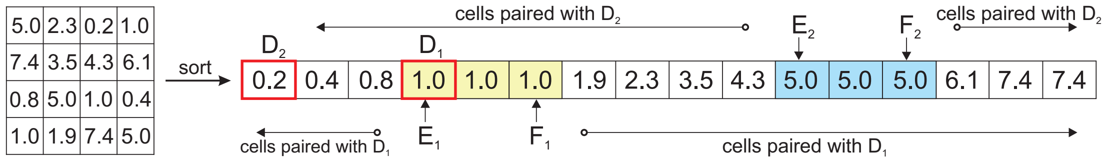


Fig. 4. Grouping cells with equal density ratios by sorting the cell ratios in the RDM

B contributed the same (or similar) distance counts to the corresponding buckets in both histograms H_0 and H_1.

Case 2: $r_A \times r_B \neq 1$, which indicates that some changes have to be made to H_1. Specifically, we follow the PROP heuristic, as in ADM-SDH, to proportionally update the buckets which overlap with the distance range $[u, v]$. For example, as shown in Fig. 2, consider the distance range $[u, v]$ overlapping three buckets $i, i+1$, and $i+2$. The buckets and their corresponding count updates are given in Eqs. 18– 20.

$$H_1[i], \qquad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{iw-u}{v-u} \tag{18}$$

$$H_1[i+1], \qquad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{w}{v-u} \tag{19}$$

$$H_1[i+2], \qquad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{v-(i+1)w}{v-u} \tag{20}$$

where n_A^0 and n_B^0 are counts of particles in cells A and B, respectively, in density map DM_k^0 of frame f_0. Similarly, n_A^1 and n_B^1 are counts of particles in corresponding cells of density map DM_k^1 in frame f_1. Note that we have $n_A^1 = r_A \cdot n_A^0$ and $n_B^1 = r_B \cdot n_B^0$. The total number of distances to be updated in the buckets is $n_A^1 \times n_B^1$ - $n_A^0 \times n_B^0$. This actually gives us the number of distances changed between cells A and B of density map DM_k, going from frame f_0 to frame f_1. There are also intra-cell distances to be processed here, details of which can be found in Appendix A.

Algorithm 2 Computing SDH using temporal locality

```
1:  procedure TEMPORALSDH($DM_k^0$, $DM_k^1$, $\epsilon$)
2:      Compute ratio density map $r$
3:      for each cell $A$ in $r$ do
4:          if $r_A \neq 1 \pm \epsilon$ then
5:              Find bucket range $[0, j]$ where distances fall
6:              Update $H[0 \ldots j]$
7:          else
8:              Do nothing. Cell $A$ does not affect $H$
9:      for each pair $A, B$ $(A \neq B)$ of cells in $r$ do
10:         if $r_A \times r_B \neq 1.0 \pm \epsilon$ then
11:             Find bucket range $[i, j]$ where distances fall
12:             Update histogram $H[i \ldots j]$
13:         else
14:             Do nothing. $A$, and $B$ do not affect $H$
```

B. Algorithmic Details

Pseudocode in Algorithm 2 shows the algorithm using temporal locality. An efficient implementation of this idea requires all pairs of cells that satisfy the *Case 1* condition to be skipped. In other words, our algorithm should *only process the Case 2 pairs, without even checking whether the product of two cells is* 1.0 (explained later). The histogram updates can be made efficiently if cells with equal or similar density ratios are grouped together. Our idea here is to store all the ratios in the RDM in a sorted array (Fig. 4). The advantage in sorting is that the sorted list can be used to efficiently find all pairs of cells with ratio product of 1.0. In other words, for any cell D with density ratio r_D, find the first cell E and the last cell F in the sorted list with ratios $1/r_D$, using binary search. Then, pair cell D with all other cells except the cells between E and F in the sorted list. Fig. 4 shows an example of a cell (D_1) with ratio 1.0 – we mark the first cell E_1 and the last cell F_1 with ratio of 1.0. Then we pair D_1 with rest of the cells in the list. Take another example of cell (D_2) with ratio 0.2 : we will effectively skip all the cells (E_2 to F_2) with ratio 5.0 (as $1/0.2 = 5.0$), and start pairing D_2 with those cells that do not have ratio 5.0 (to the left of E_2 and right of F_2).

In practice, a tolerance factor ϵ can be introduced to the *Case 1* condition such that the cells with ratio product within the range of $1.0 \pm \epsilon$ are skipped from the computations. While saving more time by allowing more cell pairs untouched, the factor ϵ can also introduce extra errors. However, our analysis in Section VI shows that such errors are negligible. Our experimental results in [10] show that there are a large number of pairs of cells whose density ratio products are around 1.0, thus providing sufficient savings of computation.

The proposed techniques are based on temporal and spatial uniformity of data set. Such cell wise uniformity is not only observed in MS, but also in many traditional spatiotemporal database applications [33]. Hence, it can be applied to very different data sets such as crowd of people and stars in astronomical studies.

VI. PERFORMANCE ANALYSIS

A. Analysis of Spatial Uniformity Impact

1) Time analysis: The running time of the algorithm utilizing only the spatial uniformity property is contributed by the following factors:

(1) Quad-tree construction time $O(N \log N)$ where N is the number of particles in simulation;
(2) Identification of uniform regions. This can also be bounded by $O(N \log N)$, as the count in each leaf node is used for at most $\log N$ chi-square tests;
(3) Distribution of distances into buckets; For this, all pairs of cells on a DM need to be computed - in a DM with M cells, the time is $O(M^2)$.
(4) Monte-Carlo simulations that require $O(MT_s)$ time according to Theorem 1. Here T_s is the time of each individual simulation.

Theoretically, the first two costs will dominate as their complexity is related to system size N. In practice, the $O(M^2)$ time for factor (3) can dwarf others if we choose a density map on the lower levels of the quad-tree - M approaches N when the level gets lower (this happens to the ADM-SDH algorithm when the bucket width w gets smaller). However, evaluation of our experimental results shows that M is orders of magnitude smaller than N.

Factor (4) is also worth a special note. Although the simulation time T_s can be regarded as a constant (as it is unrelated to N and w), a larger number of points in the simulation is preferred for better accuracy. Thus, it is crucial to study how many data points we have to simulate to reach desired accuracy. Such analysis is shown in Section VI-A2.

2) Error analysis: Based on the sources, two types of errors are introduced by utilizing the spatial uniformity feature:

I. error (e_u) by pairs of cells that are both uniform, and
II. error (e_a) by those with at least one non-uniform cell.

Type I error is basically the simulation error, i.e., the expected percentage of distances put into the wrong buckets when both cells have uniformly distributed data points. According to the Law of Iterated Logarithm (LIL) [34], such error is up to the order of $\left(\frac{S_m}{\log\log S_m}\right)^{-1/2}$, where S_m is simulation size. Since we compute the Euclidean distance between two randomly selected points which are uniformly distributed in the two cells, we have $S_m = n_s^2$, where n_s is the number of points simulated in each cell. Clearly, the error drops dramatically with the increases of n_s. Considering a scenario where n_A and n_B are of the order of 10^2, the simulation error is slightly smaller than the order of 10^{-2}. In other words, we can effectively control the Type I error without suffering from a heavy simulation overhead.

The Type II error is obviously no greater than the error achieved by the PROP heuristic. It is hard to get a tight error bound when the distribution of points in a cell is not uniform. But it is easy to see that the error for one single distribution using PROP can be arbitrarily large. Unlike the Type I error, error in this category cannot be controlled. At this point, we can at least conclude that, due to the small Type I error, our algorithm will be more accurate than existing solutions based on PROP, such as ADM-SDH [6].

An important note here is that our analysis has so far concentrated on the errors introduced in an individual distribution operation (i.e., between one pair of cells). However, our work [6] has revealed the fact that errors generated by different pairs of cells can cancel out, and reduce the error in the whole SDH to a great extent. We call such a phenomenon *error compensation*. In particular, our qualitative study shows that the error (at the entire SDH level) caused by PROP can be loosely bounded by 10%. Since this is not a tight bound, we expect to see much smaller errors in practice, as shown in our experimental results for the ADM-SDH algorithm (Section VIII-B). For the same reason, the effects of Type I error can also be reduced by error compensation, making the Type I error a negligible quantity.

3) Error/performance tradeoff: Given the above analysis, we show our algorithm is *tunable* in that the user can choose a level of DM-tree to get a desired error guarantee. Suppose p_u is the fraction of pairs of cells that are uniform on a given level, the total error ξ produced by our algorithm based on spatial uniformity is

$$\xi \leq e_u p_u + e_a(1-p_u) \tag{21}$$

A remark here is: as compared to ADM-SDH that is based on PROP heuristics, our algorithm shows an advantage in accuracy: error will be lower by $(e_a - e_u)p_u$.

From Eq. 21, we can solve p_u to obtain a guideline on the level of the DM tree from which we run the algorithm:

$$p_u \geq \frac{e_a - \xi}{e_a - e_u} \tag{22}$$

In other words, a user will choose to work on a DM where the fraction of uniform cells is at least $\sqrt{p_u}$, in order to get an error lower than ξ. More details about the percentage of the cells marked as uniform can be found at the end of Appendix H.

B. Analysis of Temporal Locality Impact

1) Time analysis: The running time is determined by the number of cell pairs that do not satisfy the temporal locality condition, i.e., ratio products are not in the range of $1.0 \pm \epsilon$. Due to the sorted list of ratios in the RDM, all cell pairs satisfying the above condition are skipped by the algorithm. Suppose p_r is the fraction of such cell pairs, only $(1-p_r)$ pairs of cells need to be processed by the algorithm. The sorting and searching of the cells can be performed in $O(M \log M)$ time. Hence, the running time of the algorithm is bound by $(1-p_r)T + O(M \log M)$ where T is the time for processing the base frame. In other words, by utilizing the temporal locality, we achieve a $(1-p_r)$-factor improvement in running time.

2) Error analysis: We tackle this by studying the *extra errors* our algorithm generates for a frame f_1 on top of those in the base frame f_0. The error introduced when utilizing the temporal locality can be categorized based on two cases:

1. temporal locality property is satisfied, and
2. temporal locality property is *not* satisfied.

Case 1: Error is produced by temporal locality property only when the cell pairs satisfy the condition $r_A \times r_B = 1.0 \pm \epsilon$. A small error equal to the fraction ϵ is introduced. When the fraction $\epsilon = 0$, there is no change in the number of distances between the two cells. In both situations, a negligible error, very hard to compute, is produced due to small change in position of the points. The fraction ϵ is negligible when the pairs of cells have uniformly distributed points in both the frames f_0 and f_1. Actually, the small movement of particles has minimal effects on the distance distribution.

Case 2: This case will not cause any additional errors. When the temporal locality condition is not satisfied for a pair of cells in f_1, we update the histograms as if we are running the algorithm for the base frame. Therefore the error will be on the same level as in the base frame. On the other hand, we do not save any processing time in such cases.

From the above analysis, we conclude that the error in the derived frame is on the same level as that of the base frame.

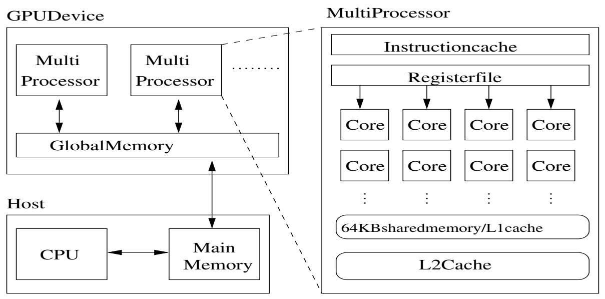


Fig. 5. The basic architecture of modern graphics processors (GPUs)

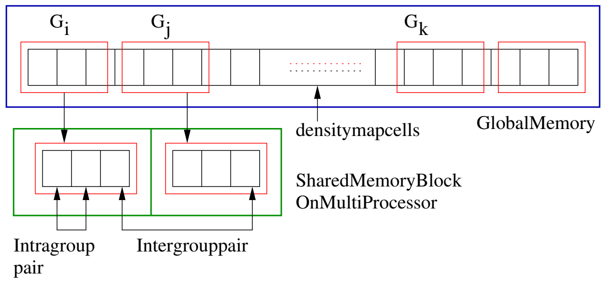


Fig. 6. Grouping cells in global memory and loading into shared memory for improving performance

VII. SDH Computation on Graphics Processors

In this section, we look at the basic architecture of the GPUs and their programming paradigms. Then we modify our algorithm of utilizing spatiotemporal uniformity to map onto the GPU programming environment. Our discussions, however, will focus on *how to optimize our algorithm in a typical GPU architecture rather than a straightforward implementation*. This is because the GPU architecture is very different from that of CPUs thus, code optimization requires special (and sometimes unintuitive) techniques. For example, the GPU hardware provides a hierarchy of programmable memories with heterogeneous capacity and performance. For that, the data can be organized, on these memories, in such a way that the access latency is minimized.

A. GPU Architecture

The basic GPU architecture, for both NVIDIA [23] and AMD [35] products, is illustrated in Fig. 5. The GPU consists of many *multiprocessors* that execute instructions on a number of GPU cores in SIMD (Single Instruction Multiple Data) manner at any given clock cycle. The GPU devices have a considerable amount of *global memory* with high bandwidth. For example, the NVIDIA GTX 570 we used has 15 multiprocessors, each of which encapsulates 32 GPU cores. It also has about 1.2 GB of global memory with a bandwidth of 152 GB/s.[3] Apart from the global memory, the GPUs have programmable, very fast cache memory (called *shared memory*). This type of memory is on-chip and shared by all GPU cores in a single multiprocessor. Since it is on-chip the access latency is very low. In contrast to that, the global memory has high access latency (400 to 800 clock cycles [23]). Therefore, the access pattern should be optimized to reduce the overall latency caused by global memory.

A large number of threads can be executed in SIMD fashion on the GPUs. The major difference between CPU and GPU threads is that the GPU threads have low creation and context-switch time. We follow the terminology of NVIDIA's compute unified device architecture (CUDA) [23] to describe the operation of GPU multiprocessors. A group of threads executing on a multiprocessor is called *block*. The blocks are scheduled dynamically on different multiprocessors. Threads within a block share all the resources, such as registers, L1 cache etc., available on the multiprocessor. 32 consecutive threads make a *warp*. Threads within a warp execute in lock-step. Any divergence in instuctions causes them to execute in sequence (determined by the scheduler). The multiprocessor views a block of threads as group of warps, and is responsible for scheduling them. An interesting feature of the memory in GPUs is that different threads in a block can read different memory locations simultaneously. This is achieved only when threads read consecutive memory locations. The underlying hardware groups the consecutive memory access requests into one access. This process is called *coalesced access*.

[3]In high-end cards such as Tesla C2075, global memory can reach 6GB.

B. Optimization Through Coalesced Access

The information related to each cell in the density map is placed in GPU memory such that coalesced access is possible. We create arrays of cell properties in the memory. For example, a contiguous block of memory is allocated to store the number of atoms present in cells of a given density map. When all threads need atom count from the cells, that they are responsible for to process, coalesced access is made from the GPU memory. Therefore, we create contiguous array of cells' properties instead of array of cells with their properties scattered in the global memory. Other properties like coordinates of cells in the simulation space are also stored in contiguous arrays. Details of different properties of the cells in density map are discussed in [6].

C. GPU Memory Optimization

The speed of memory access can be improved by placing the cells of the density map in shared memory. Each thread can access distinct pairs of cells from the shared memory. Let M be the number of cells in a density map and shared memory can hold $2M_S$ cells. We divide the shared memory into two sections, each holding up to M_S cells. With M_S as the size for group of cells, we have $G_c = M/M_S$ number of groups out of M cells of the density map. Each CUDA block can process two groups of cells in shared memory. Fig. 6 shows the mechanism of processing these groups. First, the cells belonging to groups G_i and G_j are loaded into shared memory. One cell is chosen from each group to form an *inter-group pair* that is processed further. Inter-group pairing is repeated for all cells in G_i and G_j. Cells within each group are processed by forming *intra-group pairs*. Intra-group pairing is required to account distances that are not covered by inter-group pairing process. Next, the second group G_j is evicted

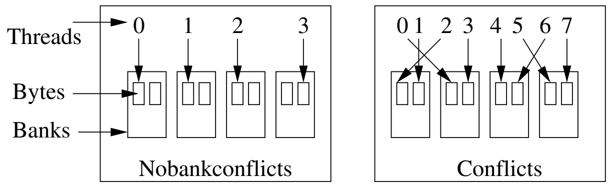


Fig. 7. Illustrating bank conflicts in shared memory access on GPU

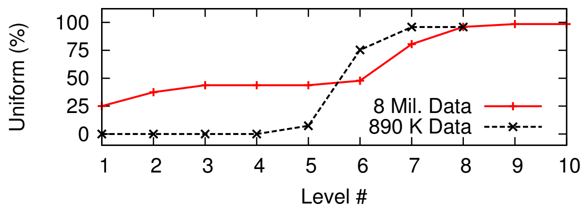


Fig. 8. Percentage area of uniform regions at different levels of the DM tree

and a new group G_k is brought into the shared memory. This is repeated for all the groups of the density map until all the cells are processed. We can easily see that such a cell grouping strategy can significantly reduce the number of global memory accesses.

Bank conflicts: The shared memory is organized as banks in the hardware such that the threads read different banks in parallel. If threads read different addresses in the same bank, it gives rise to an access conflict called *bank conflict*. Fig. 7 shows an example of bank conflicts. The contiguous array of properties technique used for coalesced access helps us in eliminating the bank conflicts. Memory banks can be accessed in parallel when every thread requests 4 bytes of data from different bank [36]. The cell properties, like coordinate or atom count, are actually of 4 bytes. Contiguous palcement of these properties in the shared memory places them in different banks. When threads within a CUDA warp access these banks in parallel, there are no bank conflicts.

Memory access latency: The operations of our algorithm are computation intensive rather than memory access. Once, the information about cells is accessed into shared memory, a large number of operations are performed. Moreover, the coalesced memory access pattern reduces number of read requests issued to global memory. The NVIDIA GPUs used in our experiments can access up to 128 bytes of memory in single request [23]. Thus the combination of computation intensive property of the algorithm and special features of GPU shadows the latency involved in global memory accesses.

D. Efficient Simulation

We utilize the shared memory to optimize the Monte-Carlo simulations on GPU. Given two cells, a set of random numbers are generated between range 0.0 to 1.0, for each cell, in the shared memory. These random numbers are mapped to the boundaries of the cells. The numbers are organized in the shared memory such that all the accesses belong to different banks. Then we perform the simulations and compute the distance distribution. The distributions are stored in a hash table that is created on global memory (as shared memory contains simulated points). The hash table is then used by the algorithm, eliminating the factors that would affect GPU performance in performing all need-based simulations.

VIII. Experimental Evaluations

A. Experimental Setup

We tested the following algorithms to evaluate the performance of our approach.

A1: The ADM-SDH algorithm [6] to process individual frames using Prop heuristic;
A2: The algorithm utilizing only temporal locality to compute SDH continuously over multiple frames;
A3: The algorithm utilizing only spatial uniformity to compute SDH frame by frame;
A4: The algorithm utilizing both temporal locality and spatial uniformity to compute SDH continuously.

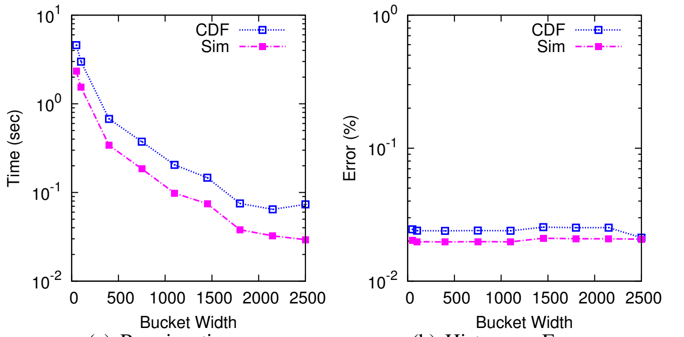
Implementation details of the last technique and thorough comparison of all of these techniques are discussed in Appendix E and F, respectively. Errors in the algorithms are computed by comparing the approximate SDH results with the correct SDH of each frame. The error (in percentage) of each frame is calculated as

$$P_{error} = 100 \times \frac{\sum_{i=0}^{l-1} \left| H[i] - H'[i] \right|}{\sum_{i=0}^{l-1} H[i]}$$

where $H[i]$ and $H'[i]$ are the correct and approximated distance counts in bucket i of the histogram, respectively.

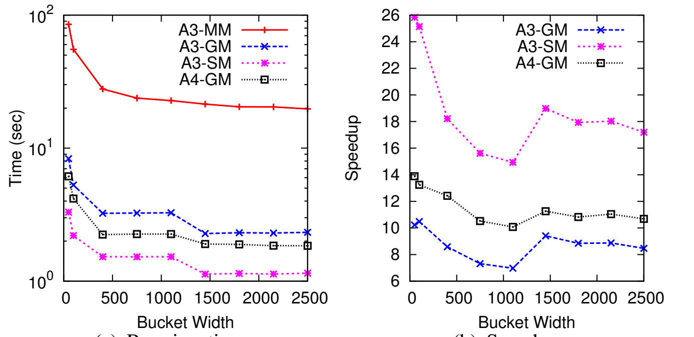
Data Sets: Two datasets from different simulation systems were used for experiments. The first dataset consists of $10,000$ frames captured from a collagen fiber simulation system made of $890,000$ atoms. The second dataset is collected from a cross membrane protein system with about $8,000,000$ atoms and $10,000$ frames. We randomly selected a chunk of 100 consecutive frames from the first dataset and 11 frames from the second dataset for our experiments. The main bottleneck in testing the algorithms is computing the correct histogram of the frames, needed to compute the error. Obtaining correct histogram is basically running the naive or DM-SDH algorithm, which is computationally expensive. Therefore, we could only get the correct histograms of 11 frames from the 8 million dataset (by brute-force in 27 days!).

The percentage of cells with uniform data distribution (i.e., uniform regions) at different levels of the density map tree is shown in Fig. 8. The leaf level of the tree is not used to determine the uniformity, as very few particles fall into small cells. For both datasets, we started to see considerable amount of uniform regions at level 6 of the tree. Note that level 6 is still at the higher end of the tree (total number of levels is 9 for the smaller dataset and 11 for the larger one) and the total number of cells is only $4^6 = 4,096$. At level 8, the percentage of uniform regions is over 90%. This confirmed the potential of using spatial uniformity to save time in SDH processing.



(a) Running time (b) Histogram Error

Fig. 9. Histogram errors and computation time using non-central χ^2 distribution function (CDF) and Monte Carlo simulation (Sim)



(a) Running time (b) Speedup

Fig. 10. Comparing running time and speedup on GPU using different memories. MM: host main memory; GM: GPU global memory; SM: GPU shared memory

B. Results of CPU Experiments

A comparison of average errors and running times of all the algorithms are presented in Appendix F, which clearly shows method A3 stands clear winner in accuracy and performance of the results. In this section, we focus on results related to new techniques that are not presented in [10].

Using noncentral χ^2 distribution: The noncentral χ^2 distribution approximation of the distances between two cells is applied to compare with the Monte-Carlo simulations. Specifically, for each pair of cells, we distribute the distance counts into the relevant buckets based on the values obtained from the Cumulative Distribution Function (CDF) of the noncentral χ^2 distribution. Such values are computed by calling a MATLAB library [37] and cached into a hash table to avoid repeated computations (exactly the same as what we did for the Monte Carlo simulation results). Fig. 9 shows the comparison of errors in the SDH obtained and the running time. The errors generated by using the CDF of noncentral χ^2 are slightly higher than those by the Monte Carlo simulation. This is expected as we know there is a systematic error in using the CDF (Lemma 1) while the Monte Carlo simulations are shown to be very accurate (Section VI-A2). The simulation-based method also beats the CDF-based method in efficiency. This is because the CDF of noncentral χ^2 distribution has a very complex form [38] therefore the time used for numerical computations in Matlab is non-trivial.

Summary: Computation of SDH based on spatial uniformity delivers the significant performance boost over existing algorithm while generating more accurate results. The idea of utilizing the temporal locality can work on top of the spatial uniformity idea to achieve higher performance and also better performance/accuracy tradeoffs. This idea by itself did not show clear advantage, as demonstrated by the bad performance of $A2$ under small bucket width. Monte Carlo simulation should be the choice in making distance distribution decisions, although the approach based on the CDF of noncentral χ^2 is only marginally worse. The simulation-based approach generates very little error even when the simulation size is small, making it a winner over the CDF-based approach. The advantages of the new algorithm over ADM-SDH become small under large bucket width, but this does not generate a concern since the target of the new algorithm is the smaller bucket width, which is preferred in scientific data analysis.

C. Results of GPU Experiments

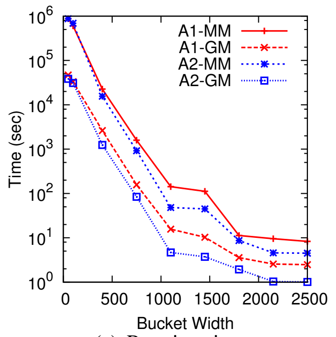
The GPU versions of the proposed algorithm were implemented under CUDA, v.4.0 [23]. The performance of the algorithms was evaluated on NVIDIA GeForce GTX 570. We report results for processing the 8-million-atom dataset.

Main results: A comparison of results of different implementations of the proposed algorithms are shown in Fig. 10 and Fig. 11, in which we show the performance of processing the first frame only using. The running time on GPU shows the trend similar to that on CPU, but much faster. The speedup of varies with the use of different types of memory. When only the global memory is used, the speedup achieved by $A3$ ranges from $7X$ to $10.4X$. The use of shared memory pushes the speedup further by a factor of 2 (i.e., actual speedup ranges from $14.9X$ to $25.8X$). The speedup is limited by the random memory access patterns emerging due to divergence in the thread computation. The thread divergence also serializes the execution of some of the threads. The size of the DM cells that are stored in the memory also affect the access patterns due to bank conflicts in shared memory.

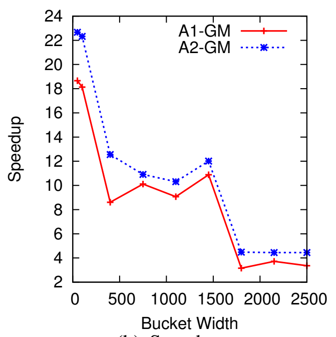
The huge speedup under small w values is due to two factors: (1) All cells of the density map are processed in parallel (2) Reduced divergence in the threads of each GPU block. Even though the computations diverge in processing some pairs of cells, the speedup is achieved by processing on different multiprocessors. Each multiprocessor has its own dedicated shared memory and does not interfere with other multiprocessors' execution.

The separation of simulation and other computations made the algorithm running time almost constant for consecutive frames, for fixed bucket width. Fig. 12 shows the processing time of all 10 frames using the $A3$ algorithm implemented in both CPU and GPU. Employing the GPUs reduces the computation time of first frame significantly. Also, all the simulations can be done within $100ms$, significantly reducing their contribution to the algorithm's running time. Hence, the SDH can be computed efficiently in real time.

In order to compare the other algorithms on GPUs, we implemented their global memory versions. Algorithm $A4$



(a) Running time



(b) Speedup

Fig. 11. Comparing running time and speedup on GPU using different memories. MM: host main memory; GM: GPU global memory; SM: GPU shared memory

achieves a small improvement in the running time (and speedup) from the temporal locality of atoms. Similarly, the performance of algorithms $A1$ and $A2$ are compared with their CPU implementations, as shown in Fig. 11. We observed speedup in the range $3X$–$18.5X$ for approxmate algorithm $A1$. In obtaining this result, we restricted the tree traversal up to two levels. Further traversal causes thread divergence and un-coalesced memory accesses, killing the performance gain, making it worst than CPU implementation.

GPU implementation of algorithm $A2$ showed speedup from $4X$ to $23X$ (again, due to temporal locality). The speedup numbers give an impression that $A1$, $A2$ are much faster than $A3$ and $A4$. But, actual running times are much higher than algorithm $A3$ (compare Fig. 10(a) and Fig. 11(a)). Use of shared memory for other algorithms would not improve the performance due to following reasons: (1) multiple tree levels can't be loaded into (limited size) shared memory for $A1$; (2) advantages of temporal locality in $A2$ and $A4$ are shadowed by time required to load into, and access from, shared memory. Also, the temporal locality property in $A2$ and $A4$ increases histogram erros [10].

Energy efficiency: Energy consumption has become a major concern in database system design [39]. The product of computation time and active power[4] consumed for SDH processing define the energy efficiency of the algorithms. Fig. 13 plots the energy consumed by both CPU and GPU versions of the $A3$ algorithm. Although the active power consumption of a GPU is a couple of times higher than that of the CPU (46 watts vs. 17 watts as we recorded), the efficiency of the GPU algorithms makes it an energy efficient device for SDH computation - active energy consumption is 5.39 to 9.13 times lower for the GPU code using shared memory. Even for the one that uses only global memory, energy efficiency is 2.51 to 3.81 times higher. To calculate the total energy consumption for the whole machine, we have to add an idle power of 114.5 watts to the active power readings and that will translate into even larger energy savings for the GPU implementations.

Summary: The GPU versions of our algorithm demonstrate the great potential of GPUs in large-scale data analytics.

[4]Active power: Power measured for the entire database server less the system idle power. It can be viewed as the power used for processing the workload. We used WattsUp power meter in our experiments.

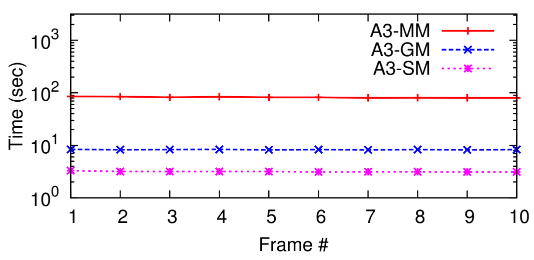


Fig. 12. Processing time of consecutive frames on GPU with $w = 50$

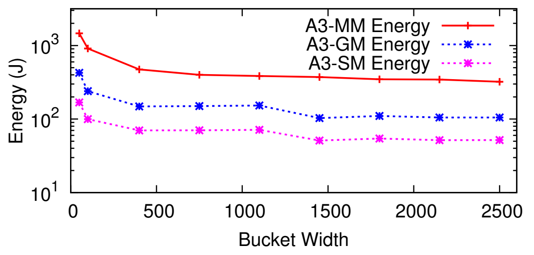


Fig. 13. Active energy consumption of CPU and GPU implementations of $A3$ algorithm under different bucket width

For the SDH problem we tested, speedup over the single-CPU implementation reaches $25X$ - that is a significant improvement of performance. The speedup decreases under larger bucket width, but it is always the cases of smaller bucket width that make the SDH problem difficult. Such diminish of speedup, as well as the different optimization strategies, however, indicate that GPU programming is a non-trivial task. Finally, the combination of multi-core GPU's and efficient algorithm to utilize the spatio-temporal uniformity, delivers very high performance. As a result, we are able to analyze scientific simulation data in a real time manner.

IX. Conclusions and Future Work

An efficient approximate solution to the spatial distance histogram query is provided in this paper. The algorithm presented in this work achieves higher efficiency and accuracy by taking advantage of the data locality and statistical data distribution properties. It makes it practically feasible to perform SDH analysis on data with large number of frames continuously. The efficiency and accuracy claims are supported by mathematical analysis and extensive experimental results. We have also shown that, through experiments, utilizing power of modern GPUs gives very significant improvement in the performance. The scientific data analysis can be performed in real time by using such modern hardware systems.

An important direction of research would be to study computation of general m-body correlation functions in scientific databases. Such functions, despite the high scientific value they carry, have not been used for MS system analysis due to their computational complexity. We strongly believe the idea based on spatial uniformity as well as GPU programming can be extended to m-body correlation function computation. Another direction of our future work might be the extension of spatiotemporal idea in 3D space and the integration of our algorithm into simulation software so that effective tuning of the simulation process becomes feasible.

Acknowledgements: The project was supported by Award R01GM086707 from NIH, USA. Part of the reported work is supported by a grant (IIS-1117699) from the NSF, USA.

REFERENCES

[1] M. Eltabakh *et. al.*, "BDBMS: A Database management system for biological data," in *CIDR*, 2007.

[2] J. Gray *et. al.*, "Scientific data management in the coming decade," *In SIGMOD Record*, vol. 34, 2005.

[3] M. Ng *et. al.*, "In BioSimGrid: grid-enabled biomolecular simulation data storage and analysis," *Future Gen. Comput. Syst.*, vol. 22, 2006.

[4] D. Frenkel *et. al.*, *Understanding Molecular Simulation: From Algorithms to Applications*, 2nd ed. Academic Press, Inc., 2001, vol. 1.

[5] D. Landau *et. al.*, *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2005.

[6] Y. Tu *et. al.*, "Computing distance histograms efficiently in scientific databases," in *ICDE*, 2009.

[7] M. Allen, *Introduction to Molecular Dynamics Simulation*. John von Neumann Institute of Computing, NIC Seris, 2003, vol. 23.

[8] A. Omeltchenko *et. al.*, "Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm," *Computer physics communications*, vol. 131, no. 1–2, 2000.

[9] I. Szapudi, *Introduction to Higher Order Spatial Statistics in Cosmology*. Lecture Notes in Physics, Springer Verlag, 2009, vol. 665.

[10] A. Kumar *et. al.*, "Distance histogram computation based on spatiotemporal uniformity in scientific data." in *EDBT*, March 2012.

[11] B. Hess *et. al.*, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *Journal of Chemical Theory and Computation*, vol. 4, no. 3, March 2008.

[12] A. Gray *et. al.*, "N-body problems in statistical learning," in *Advances in Neural Info. Processing Systems*, 2001.

[13] V. Grupcev *et. al.*, "Approximate algorithms for computing spatial distance histograms with accuracy guarantees." *TKDE*, 2012.

[14] J. Barnes *et. al.*, "A Hierarchical O(N log N) Force-Calculation Algorithm," *Nature*, vol. 324, no. 4, 1986.

[15] L. Greengard *et. al.*, "A Fast Algorithm for Particle Simulations ," *Journal of Computational Physics*, vol. 135, no. 12, 1987.

[16] S. Chen *et. al.*, "Performance analysis of a dual-tree algorithm for computing spatial distance histograms," *VLDBJ*, vol. 20, no. 4, 2011.

[17] P. Dietz *et. al.*, "Persistence, amortization and randomization," in *Proc. of the ACM-SIAM symposium on Discrete algorithms*, 1991.

[18] T. Teraoka *et. al.*, "The MP-tree: A data structure for spatio-temporal data," in *Phoenix Conf. on Computers and Communications*, 1995.

[19] G. Lagogiannis *et. al.*, "A time efficient indexing scheme for complex spatiotemporal retrieval," *SIGMOD Record*, vol. 38, 2010.

[20] H. Kaplan, "Persistent data structures," in *Handbook on Data Structures and Applications*. CRC Press, 2001.

[21] W. Hwu, *GPU Computing Gems Jade Edition*, 1st ed. Morgan Kaufmann Publishers Inc., 2011.

[22] D. Kirk *et. al.*, *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed. Morgan Kaufmann Publishers Inc., 2010.

[23] NVIDIA, "CUDA C Programming Guide, Version 4.0." [Online]. Available: http://developer.nvidia.com/object/cuda.html

[24] Khronos Group, "OpenCL - The open standard for parallel programming of heterogeneous systems." [Online]. Available: http://www.khronos.org/opencl/

[25] B. He *et. al.*, "Relational joins on graphics processors," in *SIGMOD*, 2008.

[26] N. Govindaraju *et. al.*, "Fast computation of database operations using graphics processors," in *SIGMOD*, 2004.

[27] J. Owens *et. al.*, "A survey of general-purpose computation on graphics hardware," in *Eurographics, State of the Art Reports*, 2005.

[28] J. Orenstein, "Multidimensional tries used for associative searching," *Information Processing Letters*, vol. 14, no. 4, 1982.

[29] L. Breiman, *Probability (Classics in Applied Math.)*. SIAM, 1992.

[30] E. Weisstein, "Square line picking." [Online]. Available: http://mathworld.wolfram.com/SquareLinePicking.html

[31] V. Alagar, "The distribution of distance between random points," *Journal of Applied Probability*, vol. 13, no. 3, 1976.

[32] E. Weisstein, "Noncentral χ^2 distribution, from MathWorld – A Wolfram Web Resource." [Online]. Available: http://mathworld.wolfram.com/NoncentralChi-SquaredDistribution.html

[33] Y. Tao *et. al.*, "Analysis of predictive spatio-temporal queries," *ACM Trans. Database Syst.*, vol. 28, no. 4, 2003.

[34] R. Bhattacharya *et. al.*, "Comparisons of chisquares, edgeworth expansions and bootstrap approximations to the distribution of the frequency chisquare," *Indian J. of Statistics*, vol. 58, no. 1, 1996.

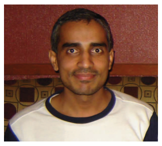
[35] AMD, "Close to Metal (CTM) Technology." [Online]. Available: http://ati.amd.com/products/streamprocessor/

[36] NVIDIA, "CUDA C Best Practices Guide, Version 4.0," 2011. [Online]. Available: http://developer.nvidia.com/object/cuda.html

[37] MATLAB, *version 7.14.0 (R2012a)*. The MathWorks Inc., 2012.

[38] N. Johnson *et. al.*, *Continuous Univariate Distributions*, 2nd ed. John Willey and Sons, 1994, vol. 1.

[39] R. Agrawal *et al.*, "The claremont report on database research," *SIGMOD Record*, vol. 37, no. 3, 2008.



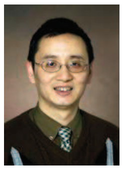
Anand Kumar received BE degree in Computer Science & Engineering from Visvesvaraya Technological University, India and MS degree in Computer Science from IIIT Hyderabad, India. He is a PhD student in the department of Computer Science & Engineering at the University of South Florida. His interests are in management of big data, data compression, GPU computing and privacy in queries.

Vladimir Grupcev received a BS degree in Applied Mathematics & Computer Science from University of Ss. Cyril and Methodius, Macedonia; MS degree in Mathematics from University of South Florida in 2007. He is a PhD student in the department of Computer Science & Engineering at the University of South Florida. His interests includes scientific data management and high performance computing.

Yongke Yuan is associate professor in the Department of Economics and Management at Beijing University of Technology in Beijing, China. He received his PhD in Management Engineering from Peking University, China. His current focus is in coupling natural and social system science with engineering to forecast the development of Chinese industries.

Yi-Cheng Tu received a Bachelor's degree in horticulture from Beijing Agricultural University, China, and MS and PhD degrees in computer science from Purdue University. He is an assistant professor in the department of Computer Science & Engineering at the University of South Florida. His research is in energy-efficient database systems, scientific data management and data stream management systems.

Jin Huang received the BS degree in mathematics from Dalian University of Technology, China. He is PhD student in the department of Computer Science at the University Texas at Arlington. His research includes data mining and medical informatics.



Gang Shen is an assistant professor of statistics in the Department of Statistics at the North Dakota State University. He received his BE degree from the Fudan University, China; MS degree in applied statistics from Worcester Polytechnic Institute, MS and PhD degrees in Statistics from Purdue University. His research includes: Statistical modeling and estimation, Asymptotic theory, Change-point problem, etc.

Appendix A
Intra Cell Distances

We assume the cell of interest has diagonal length q, and the distance range $[0, q]$ overlaps with buckets $0, 1, \ldots, j$. If an individual cell is with an RDM of 1.0, nothing needs to be done. For those cells whose RDM is not 1.0, the following rules are used to update the counts.

$$H_1[0], \quad \left[\frac{n_A^1(n_A^1-1)}{2} - \frac{n_A^0(n_A^0-1)}{2}\right]\frac{w}{q} \tag{23}$$

$$\ldots \qquad \ldots$$

$$H_1[j-1], \quad \left[\frac{n_A^1(n_A^1-1)}{2} - \frac{n_A^0(n_A^0-1)}{2}\right]\frac{w}{q} \tag{24}$$

$$H_1[j], \quad \left[\frac{n_A^1(n_A^1-1)}{2} - \frac{n_A^0(n_A^0-1)}{2}\right]\frac{(j-1)w}{q} \tag{25}$$

Appendix B
Identification of Uniform Regions

Given a spatial region (represented as a quad-tree node), how do we test if it is a uniform region? We take advantage of the chi-square (χ^2) goodness-of-fit test to solve this problem. Here we show how the χ^2 test is formulated and implemented in our model.

Definition 1: Given a cell Q (i.e., a tree node) in the DM-tree, we say Q is uniform if its probability value p in the chi-square goodness-of-fit test against uniform distribution is greater than a predefined bound α.

To obtain the p-value of a cell, we first need to compute two values: the χ^2 value and the degree of freedom (df) of that particular cell. Suppose cell Q resides in level k of the DM-tree (see Fig. 14). We go down the DM-tree from Q till we reach the leaf level, and define each leaf-level descendant of Q as a separate *category*. The intuition behind the test here is: *Q is uniform if each category contains roughly the same number of particles*. The number of such leaf-level descendants of cell Q is 4^{t-k}, where t is the leaf level number. Therefore, the df becomes $4^{t-k}-1$. The observed value, O_j, of a category j is the actual particle count in that leaf cell. The expected value, E_j, of a category is computed as follows:

$$E_j = \frac{Total\ Particle\ Count\ in\ Cell\ Q}{\#\ of\ leaf\ level\ descendants\ of\ Q} = \frac{n_Q}{4^{t-k}} \tag{26}$$

Having computed the observed and expected values of all categories related to Q, we obtain the χ^2 test score of cell Q through the following equation:

$$\chi^2 = \sum_{j=1}^{4^{t-k}} \frac{(O_j - E_j)^2}{E_j} \tag{27}$$

Next, we feed these two values, the χ^2 and the df, to the R statistical library, which computes the p-value. We then compare the p-value to a predefined probability bound α (e.g., 0.05). If $p > \alpha$, we mark the cell Q as uniform, otherwise we mark it as non-uniform. Note that the χ^2 test performs poorly when the particle counts in the cells drop bellow 5. But, we already had similar constraint in our algorithm while building the DM-tree, essentially making the cells in the leaf level contain more than 4 particles. Hence, we choose leaf level nodes as the categories in the test.

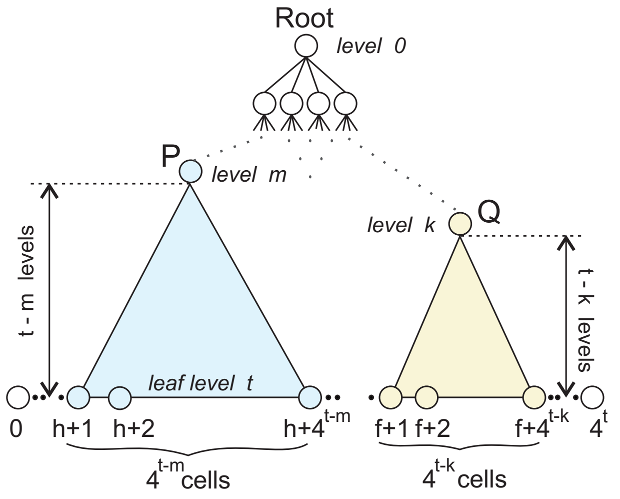


Fig. 14. Sub-trees of nodes P and Q with their leaf nodes

To find all the uniform regions, we traverse the DM-tree starting from the root and perform the above χ^2 test for each node we visit. However, once a node is marked uniform, there is no need to visit its subtree. The pseudo code shown in Algorithm 3 represents this idea – to find all uniform regions, we only need to call procedure MarkTree with the root node of the DM-tree as input.

Algorithm 3 Marking uniform regions

```
1:  procedure MarkTree(node $Q$, level $a$)
2:      checkUniform($Q, a$)
3:      if $Q$ is NOT uniform then
4:          for each child $B_i$ of cell $Q$: $i := 1 \ldots 4$ do
5:              MarkTree ($B_i$, $a+1$)
6:
7:  procedure checkUniform(node $Q$, level $a$)
8:      Go to leftmost leaf level $(t)$ descendent of $Q$
9:      for $k = 1$ to $4^{t-a}$ do
10:         $\chi^2 := \chi^2 + \frac{(O_k - E_k)^2}{E_k}$
11:     Get $pval(\chi^2)$ using $R$ library
12:     if $pval >$ significance value $\alpha$ then
13:         mark $Q$ as uniform
14:     else
15:         mark $Q$ as not uniform
```

Appendix C
Distance Distribution Within and Between Two Unit Squares

If two points are randomly and uniformly taken from the same unit square (i.e., one with lateral length 1), the distribution of the distance between such two points has the following probability density function:

$$f(x)=\begin{cases}2x(x^2-4x+\pi) & 0\le x\le 1\\ 2x\big[4\sqrt{x^2-1}-(x^2+2-\pi) & \\ \quad -4\tan^{-1}\sqrt{x^2-1}\big] & 1\le x\le\sqrt{2}\end{cases}$$

For two points uniformly sampled from two adjacent unit squares, the distance has the following distribution function:

$$F(x)=\begin{cases}\frac{2x^3}{3}-\frac{x^4}{4} & 0\le x\le 1\\ \frac{3x^2}{2}-\frac{4x^3}{3}+\frac{x^4}{2}+2x^2\arccos\left(1/x\right) & \\ \quad -\frac{1}{4}-\frac{2(1+2x^2)(x^2-1)^{\frac{1}{2}}}{3} & 1\le x\le\sqrt{2}\\ 2x^2\arcsin\left(1/x\right)-\frac{11}{12}-\frac{x^2}{2}-\frac{4x^3}{3} & \\ \quad +\frac{2(1+2x^2)(x^2-1)^{\frac{1}{2}}}{3} & \sqrt{2}\le x\le 2\\ \frac{2(1+2x^2)(x^2-1)^{\frac{1}{2}}}{3}-\frac{75}{12}-\frac{9x^2}{2} & \\ \quad +\frac{2(2+x^2)(x^2-4)^{\frac{1}{2}}}{3}+\frac{5x^4}{12} & \\ \quad +2x^2\arcsin\left(1/x\right) & \\ \quad +2x^2\arccos\left(2/x\right)\Big[-1 & \\ \quad +\frac{5-x^2}{(x^2-4)^{\frac{1}{2}}}\Big] & 2\le x\le\sqrt{5}\\ 1 & x\ge\sqrt{5}\end{cases}$$

Appendix D
Total Number of Simulations Performed

The density map is organized as a grid of $M = n \times n$ cells. We represent the position of each cell by an ordered pair (x, y), where x and y are the horizontal and vertical displacements respectively, of the cell from the top-left corner of the density map. A cell C with displacements i, j is represented by $C(i, j)$. The width or side of each cell is denoted by t (see Fig. 15). We discuss the number of Monte Carlo simulations performed in a density map through a special feature called L-shape (Definition. 2). The number of simulations performed is directly related to the number of distinct L-shapes found in the density map.

Definition 2: L-shape of two cells A and B, $L(A, B)$, is a subset of the density map that includes the two end cells $A(x_A, y_A)$ and $B(x_B, y_B)$ and all the cells with positions

$$(x_A+1, y_A), (x_A+2, y_A), \ldots, (x_B, y_A) \textit{ and}$$
$$(x_B, y_A+1), (x_B, y_A+2), \ldots, (x_B, y_B-1)$$

or the positions

$$(x_A, y_A+1), (x_A, y_A+2), \ldots, (x_A, y_B) \textit{ and}$$
$$(x_A+1, y_B), (x_A+2, y_B), \ldots, (x_B-1, y_B)$$

Without loss of generality we assume $x_A < x_B$ and $y_A < y_B$ in rest of the discussion. It is to be noted that both cells, A and B, have only one neighbor cell in the $L(A, B)$-shape.

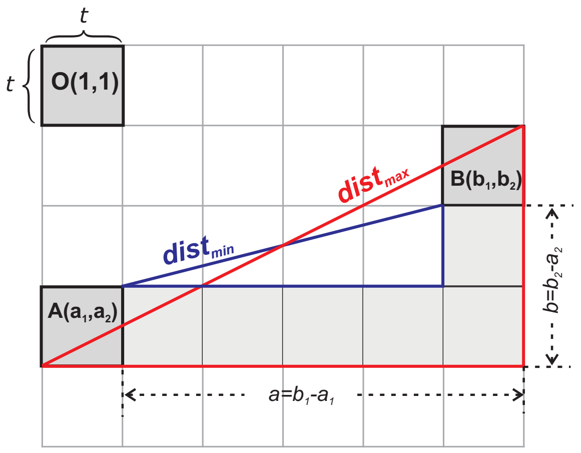


Fig. 15. Illustration of L-shape $L(A, B)$ of size $d(L(A, B)) = (a, b)$ in a density map

Definition 3: The size of an $L(A, B)$ shape, which is denoted as $d(L(A, B))$, is the ordered pair (a, b) where a is the horizontal distance (counted in number of cells) and b is the vertical distance between the cells A and B.

Definition 4: Equivalent L-shapes: Let $L(A, B)$ and $L(C, D)$ be two L-shapes with sizes $d(L(A, B)) = (a, b)$ and $d(L(C, D)) = (c, d)$. Then $L(A, B)$ is equivalent to $L(C, D)$ (i.e,. $L(A, B) \equiv L(C, D)$) iff ($a = c$ and $b = d$) or ($a = d$ and $b = c$).

Lemma 2: $L(A, B) \equiv L(C, D)$ iff the minimum and maximum distances between A, B and between C, D are equal. In other words, $L(A, B) \equiv L(C, D)$ iff $dist_{min,max}(A, B) = dist_{min,max}(C, D)$.

Proof: Consider two L-shapes, $L(A, B)$ and $L(C, D)$ with sizes $d(L(A, B)) = (a, b)$ and $d(L(C, D)) = (c, d)$.

If $L(A, B) \equiv L(C, D)$ then, by the definition 4, $d(L(A, B)) = d(L(C, D))$. Thus, $a = c$ and $b = d$ or $a = d$ and $b = c$.

Fig. 15 shows maximum distance between cells A and B.

$$\begin{aligned} dist_{max}(A, B) &= \sqrt{((a+1)*t)^2 + ((b+1)*t)^2} \\ &= \sqrt{((c+1)*t)^2 + ((d+1)*t)^2} \\ &= dist_{max}(C, D) \end{aligned}$$

Similarly for the minimum distance between cells A and B,

$$\begin{aligned} dist_{min}(A, B) &= \sqrt{((a-1)*t)^2 + ((b-1)*t)^2} \\ &= \sqrt{((c-1)*t)^2 + ((d-1)*t)^2} \\ &= dist_{min}(C, D). \end{aligned}$$

Let two pairs of cell (A, B) and (C, D) have same minimum and maximum distance between them i.e.,

$$dist_{min,max}(A, B) = dist_{min,max}(C, D)$$

or in an equivalent form:

$$\sqrt{((a-1)*t)^2 + ((b-1)*t)^2} = \\ \sqrt{((c-1)*t)^2 + ((d-1)*t)^2}$$

The equation holds only if ($a = c$ and $b = d$) or ($a = d$ and $b = c$). Thus, $d(L(A,B)) \equiv d(L(C,D))$. By definition, if two L-shapes have same size, they are equivalent. ■

Theorem 2: The number of distinct L-shapes (regardless of position) in a density map with $M = n^2$ cells is $\frac{n(n+1)}{2} - 1$.

Proof: The form of each L-shape $L(A,B)$ is defined by its size $d(L(A,B)) = (a,b)$, where $0 \leq a \leq n-1$ and $0 \leq b \leq n-1$. But, since the L shapes with size (a,b) are equivalent to the L-shapes with size (b,a) we need only to count the L-shapes with size (a,b) where $b \geq a$ and $b \neq 0$. The number of such L-shapes for given values of $a = 1, 2, \ldots n-1$ are $n-1, n-2 \ldots, 1$ respectively. For $a = 0$ there are $n-1$ L-shapes. Obviously, the total number of all distinct L-shapes of size (a,b) is $\frac{n*(n+1)}{2} - 1$. ■

As the number of distinct Monte Carlo simulations performed in an RDM is equal to the number of distinct L-shapes, the total number of simulation performed to compute SDH is bound by $O(M)$.

Appendix E
Putting Both Ideas Together

The continuous histogram processing is sped up by utilizing both spatial uniformity and temporal locality properties. An overview of the technique is shown in the flow diagram of Fig. 16. The left branch (decision $A \equiv B$) is to compute the intra-cell distances. In the right branch we check the locality property of every pair of cells before checking for uniform distribution of the particles. Any pair that satisfies the locality property is skipped from further computations. The pairs that fail the locality property check are tested for the uniformity property. Based on the results of the check, subsequent steps are taken and the histogram buckets are updated.

The Monte Carlo simulation step introduced in our algorithm is expensive when computing SDH of a sequence of frames. As mentioned in Section IV, the cost can actually spread over when we are processing a sequence of frames. It is an interesting fact that the tree building process is such that a cell in the DMs of same level in all frames is of same dimensions. Therefore, a simulation done once can be reused in all other frames. Given a pair of cells A and B and their respective distance range $[u,v]$, we compute the proportions of distances that map to each bucket covered by $[u,v]$ through Monte Carlo simulation. For each distinct $[u,v]$ range, we store such (and only such) proportions of distance distributions in a universal hash table.

For every pair of uniform cells that do not resolve and have distance range $[u,v]$, we look into the hash table to get the proportions to distribute the distances into buckets. If an entry is available in the hash table, we use it directly. Otherwise, a new simulation is done and proportions are calculated. This new simulation information is stored in the hash table. The hash table is universal and is used for computing the histogram of all the frames for a given bucket width.

The same strategy can be followed if we were to use closed-form PDFs (if available) to determine the proportions of distances.

To simplify the implementation, one decision we made was to choose a level k in the DM-tree and process cells on that level only (instead of working on uniform regions on different levels). We need a level that balances both SDH computation time and the error – choosing a level close to the leaves may increase the time, while a level close to the root will introduce higher errors in the SDH. An important feature of our algorithm is that *the user can choose a level to run the algorithm according to her tolerance of the errors.* Such choices can be made beforehand by analysis as discussed in Section VI-A3. Note that all the cells in the DM-tree that are uniform are marked before the continuous SDH processing begins.

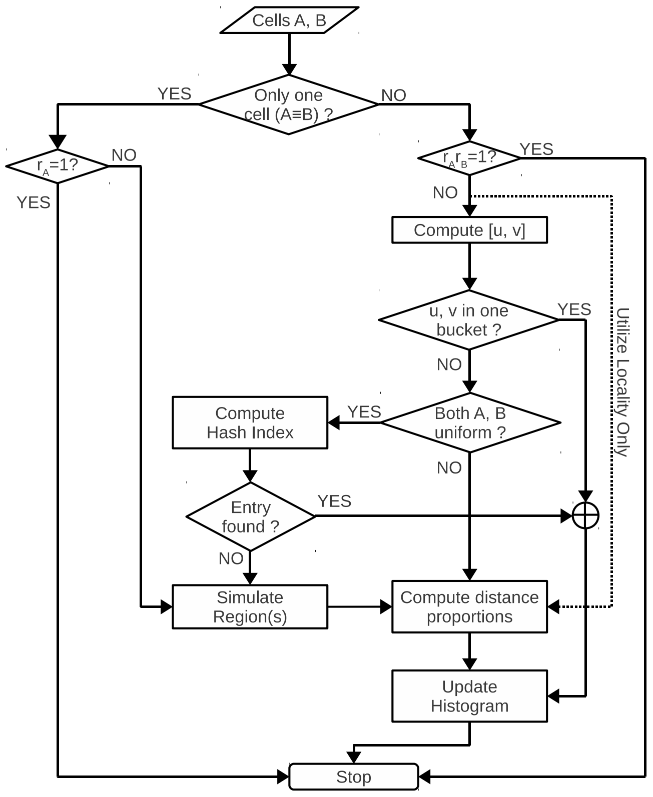


Fig. 16. Steps in dealing with two cells of the composite algorithm for computing SDH

Appendix F
Experimental Results on CPU

The proposed continuous SDH computation algorithm was implemented in C++ programming language and tested on real MS data sets. The experiments were conducted on an Apple Xserve server with two Intel quad-core processors and 24 GB of physical memory. The Xserve was running OS X 10.6 Snow Leopard operating system.

The running time of the algorithms on different data sets are measured for comparison, along with the errors introduced due to approximation. The errors are computed by comparing the approximate SDH results with the correct SDH of each frame.

We also observed significant temporal similarity a-mong frames of both datasets. The success of utilizing the temporal similarity property depends on the total fraction of cells that

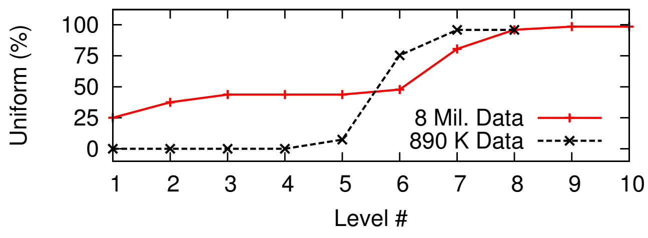
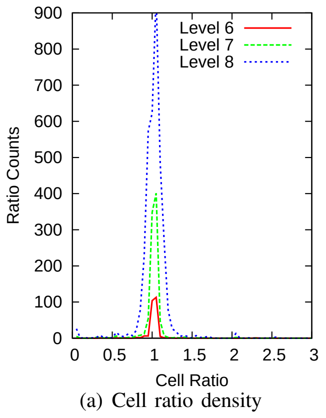
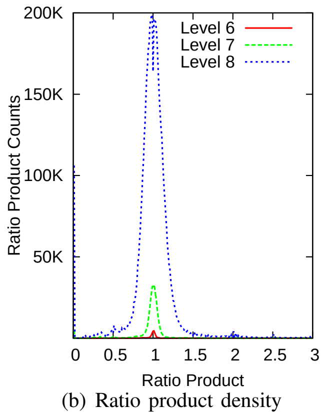


Fig. 17. Percentage of the area of uniform regions at different levels of the DM tree



(a) Cell ratio density



(b) Ratio product density

Fig. 18. Temporal similarity between two consecutive frames chosen randomly from the dataset of $890K$ atoms

exhibit such property. In fact, the running time of the technique is affected by the number of cell pairs (A, B) for which $r_A \times r_B = 1 \pm \epsilon$. Figs. 18(a) and 18(b) show the density of ratios and ratio products at each level of the DM-tree in two consecutive frames, chosen randomly from the dataset of $890,000$ atoms. For all levels we tested, majority of the cells (cell pairs) show ratio (ratio product) that is close to 1.0. The number of cell pairs with ratio product of 1.0 increases as we descend down the tree.

Main results: The average running time of all the algorithms for different bucket widths is shown in Fig. 19(a) and 19(c). It can be noted that the running time of $A1$ can be orders of magnitude longer than our proposed algorithms. The important observation to be made about algorithm $A1$ is that the running time increases dramatically with the decrease of w (note the logarithmic scale). Method $A2$ is similar to $A1$ but, utilizes temporal locality while working on only one level. When the bucket width is small, both methods work on lower tree levels, with small number of atoms in the cells. The utilization of locality gives scope to save some running time in $A2$. Unlike the first two methods, the time spent by methods $A3$ and $A4$ does not change much with the change of bucket width w. The data size however, limits the levels at which the algorithms work. Changing levels would affect the running time. The algorithms run at tree levels 6 and 7 for $890K$ and 8 million data set, respectively.[5] Such levels are chosen to ensure that 80% of the area is covered by uniform regions (see Fig. 17). We generate 75 points from each of the two cells in Monte Carlo simulations - this number is chosen based on our empirical results about sufficient simulation size (Fig. 22). Note that the average running time presented here have amortized all “start-up” costs including that for running Monte Carlo simulations and spatial uniformity test. The running time for larger bucket width is close to algorithm $A1$ and $A2$. This is because, in $A1$ and $A2$, the processing level is closer to root than the (fixed) level of tree used in algorithms $A3$ and $A4$. When we choose to have $A3$ and $A4$ run on a higher level, their time will clearly beat $A1$ and $A2$, as we have shown in [10]. The performance of $A2$ under smaller bucket width is bad because it works for lower levels of the tree, and the temporal locality is weak due to the small number of particles in each cell. For example, if there are 4 atoms in a cell in the base frame and one atom moves out of it in the derived frame, the density ratio is as low as $3/4 = 0.75$.

The average errors (in percentage) of each method are shown in Fig. 19(b) and 19(d) for different values of w. The errors rendered by $A3$ and $A4$ are always lower than those by method $A1$. However, the errors of $A2$ are slightly higher than $A1$ for small bucket width. The number of distances to be distributed between two cells is very small, as the algorithm works close to leaf level. Therefore, by utilizing the temporal locality property the small errors are added on top of the PROP method applied for other cell pairs. Although method $A4$ is faster than $A3$, the price for that is an error rate that is slightly higher, as we expected based on our analytical results (Section VI-B). However, it provides a good tradeoff as the improvement of performance is of larger magnitude than the loss of accuracy. The method $A3$ stands clear winner in accuracy of the results. The distance distribution curve computed by Monte Carlo simulations diminishes the error that would have been introduced by heuristically distributing distances as in $A1$. The errors in method $A1$ stay low (still equal to or higher than other methods) for smaller bucket width but goes higher under larger w values. The reason being, proportions for small buckets are almost similar in all the algorithms. Number of distances that are in the range of very small buckets are few and therefore their proportional distribution are not much different. Hence, the error is low. With the increase of bucket width, $A1$ would end up distributing the distances equally in all the buckets while our methods accurately compute the proportions of distances that should go into each bucket. For both datasets, $A2$ has the same level of errors with those of $A1$, although the error fluctuates in the spectrum of different bucket width and tends to be larger under smaller bucket width. The reason for this, again, is because $A2$ works for lower levels of the tree and the number of particles is small.

Deeper insights on the performance/error tradeoff of different algorithms can help users make justifiable choices. One way to quantify the performance/error tradeoff is the *product of time and error* - an algorithm with lower *time–error product* (TEP) is obviously preferred. We calculated the TEPs of all tested algorithms and found that, among all settings and algorithms, $A4$ stands the winner by producing the smallest TEPs under all bucket widths (Fig. 20), although its advantage over $A3$ is very small in the 8-million atom dataset. Algorithm $A3$ is only second to $A4$ with slightly higher TEPs, beating $A1$

[5]In [10], we run experiments on levels 5 and 6 of these two datasets, respectively, and very similar results are reported.

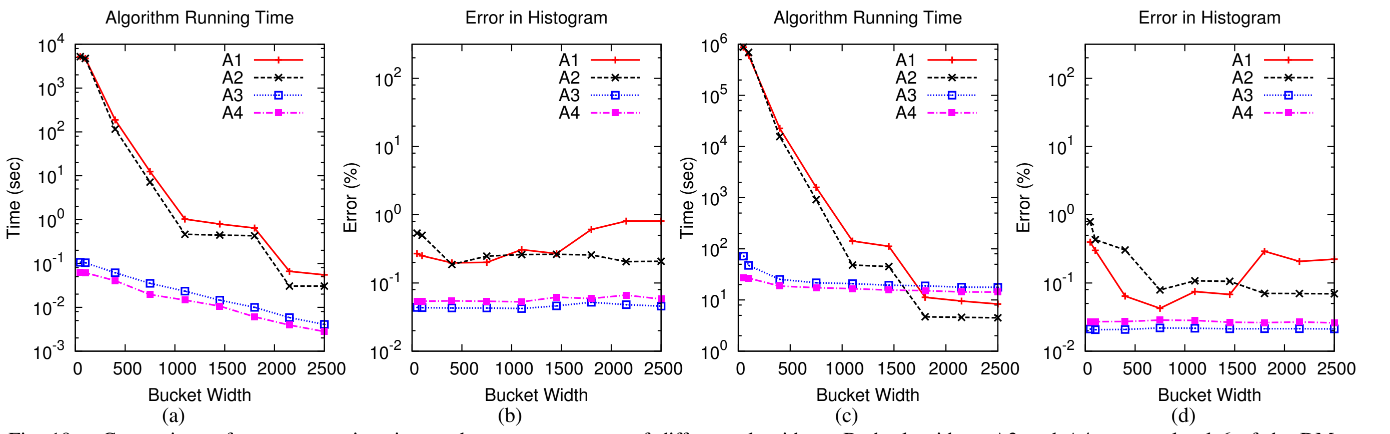


Fig. 19. Comparison of average running time and percentage errors of different algorithms. Both algorithms A3 and A4 process level 6 of the DM tree. (a)–(b) The results from 890, 000 atom dataset. (c)–(d) Results from 8 million atom dataset

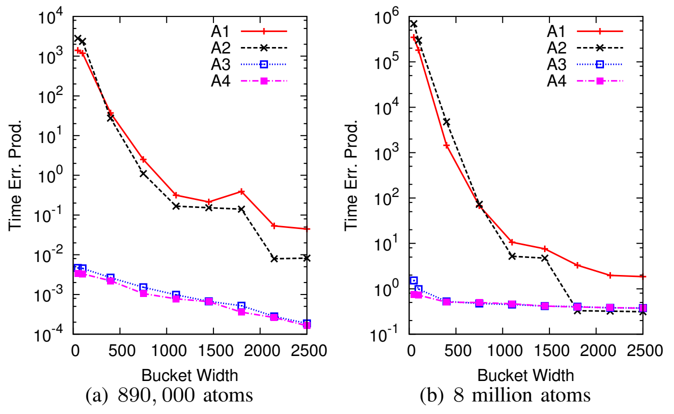


Fig. 20. Time–Error Product (TEP) of different SDH computation algorithms

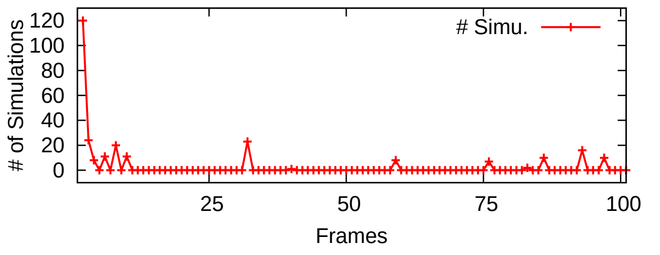


Fig. 21. Number of simulations performed per frame to process 100 frames together under bucket width of 1450

analysis of the Type I error in Section VI-A2 only gives a loose error bound whereas the actual errors are much lower.

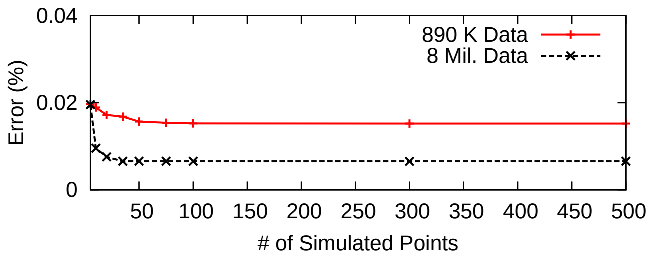


Fig. 22. Effects of simulation size on SDH error

and $A2$. This clearly shows that $A4$, although carries a larger error than $A3$, can still be a viable choice – its performance gain overshadows the loss of accuracy as compared to $A3$. The gain or loss in time and error may compensate each other in some cases, producing similar TEPs. It is user's choice to pick either $A3$ or $A4$. Again, $A2$ only shows its advantage over $A1$ under larger w values, indicating that using temporal locality alone is not a viable choice.

Number of simulations: Much time in computation of the first few frames is spent in performing the simulations to update the hash table entries. In our experiments on the dataset of $890K$ atoms, the number of simulations performed for each frame dropped quickly. In total, 100 frames were processed to compute SDH using algorithm $A3$. Fig. 21 shows the distribution of simulations performed over 100 frames. We can see that the first frame peaks at 120 simulations. In most of the other frames, no simulations are performed except for few frames for which less than 25 simulations are performed. This clearly states that the hash table utilized in $A3$ saves running time by reusing the simulations performed in previous frames.

Simulation size: The number of points used in every Monte Carlo simulation does not affect the SDH results, as long as sufficient number of points are generated. The error shown in Fig. 22 does not change when the number of points in the Monte Carlo simulations goes beyond 50. Thus, our

Appendix G
Assigning Distance Counts from a Single Cell

$$H[0], \qquad \frac{n_A(n_A-1)}{2}\int_0^w g_{D'}(t)dt \tag{28}$$

$$H[1], \qquad \frac{n_A(n_A-1)}{2}\int_w^{2w} g_{D'}(t)dt \tag{29}$$

$$\dots \qquad \dots$$

$$H[j], \qquad \frac{n_A(n_A-1)}{2}\int_{(j-1)w}^{q} g_{D'}(t)dt \tag{30}$$

where $g_{D'}(t)$ is the PDF for random variable D', and can also be generated by mathematical analysis or approximated by Monte Carlo simulations.

Appendix H
Percentage of Uniform Cells

One special note about p_u is: defined as the fraction of *actual* uniform cell pairs, p_u is smaller than the percentage of cell pairs marked as uniform by our algorithm. This is because it is not a deterministic decision to mark a cell uniform, and cases of false positive can happen. In marking the cells, the chance of getting a false positive consists of the approximation error of the Pearson's χ^2 test statistic [34] and the probability bound α used in the test. The test statistic error is up to the order of $O_t^{\frac{1-\nu}{\nu}}$, where ν is degree of freedom and O_t is the number of observations in χ^2 test. In our environment, O_t tends to be a large number, as we often see large uniform regions. The α value is user tunable and usually set around 5%. When ν is sufficiently large, the error in marking a cell uniform is $\gamma = \alpha + 1/O_t \approx \alpha$. Thus, if the percentage of pairs of cells marked uniform by our algorithm is p'_u, we have

$$p_u = (1-\gamma)^2 p'_u \approx (1-\alpha)^2 p'_u$$