Parallelization of Push-based System for Molecular Simulation Data Analysis with GPU

Iliiazbek Akhmedov, Yi-Cheng Tu †, Vladimir Grupcev, Joseph Fogarty, Sagar Pandit

***Abstract*—Modern simulation systems generate big amount of data, which consequently has to be analyzed in a timely fashion. Traditional database management systems follow principle of pulling the needed data, processing it, and then returning the results. This approach is then optimized by means of caching, storing in different structures, or doing some sacrifices on precision of the results to make it faster. When it comes to the point of doing various queries that require analysis of the whole data, this design has the following disadvantages: considerable overhead on random I/O while reading from the simulation output files and low throughput of the data that consequently results in long latency, and, if there was any indexing to optimize selections, overhead of storing those becomes too big, too.**

There is a new approach to this problem presented in the previous paper – Push-based System for Molecular Simulation Data Analysis for processing network of queries proposed in the previous paper and its primary steps are: i) it uses traditional scan-based I/O framework to load the data from files to the main memory and then ii) the data is pushed through a network of queries which consequently filter the data and collect all the needed information which increases efficiency and data throughput. It has a considerable advantage in analysis of molecular simulation data, because it normally involves all the data sets to be processed by the queries.

In this paper, we propose improved version of Push-based System for Molecular Simulation Data Analysis. Its major difference with the previous design is usage of GPU for the actual processing part of the data flow. Using the same scan-based I/O framework the data is pushed through the network of queries which are processed by GPU, and due to the nature of science simulation data, this gives a big advantage for processing it faster and easier. In the old approach there were some custom data structures such as quad-tree for calculation of histograms to make the processing faster and those involved loss of data and some expectations from the data nature, too. In the new approach due to high performance of GPU processing and its nature, custom data structures were not even needed much though it didn't bear any loss in precision and performance.

***Index Terms*—Push-based system, molecular simulation, scientific databases, spatial distance histogram, GPU, parallel processing, CUDA.**

I. Introduction

In various sciences simulation systems take big place and often times they may be the clue for results. One of such sciences, which is primarily related to this paper, is physics. In this case, from computer engineering point of view, simulations have the following flow:

1) initial physical properties are given to simulation software as arguments which are interpreted and provided in a language defined by the simulation software
2) then it runs for given amount of time that can generally last from milliseconds to months or even more
3) finally, the file generated during the simulation is analyzed to extract useful information

The purpose of this paper is primarily focused on the third step, which basically involves the entire simulation data to be processed by the analysis software.

Big data processing is becoming one of the key issues with the amount of data being generated by modern systems. Eventually, this data is required to be processed on the fly, thus, analysis software should be able to handle massive data in a very short period of time. When working with huge volume of data, there are non-trivial issues arise. For example, it can take days and weeks to analyze big enough data sets, because the data cannot be simply loaded into memory, since it can get up to terabytes, thus, it has extra overhead because of widely used random access disk I/O framework to read from disk chunks by chunks. Besides it, analysis of the data can get even more complicated, since going through certain parts of the data once would not be enough, which leads to low throughput and efficiency of loaded data. One such example, the data analysis approach has polynomial complexity just for reading the data in order to come up with result, and since the data can't simply be saved in memory, it raises the overhead of disk I/O, too. Pull-based architectures in data processing engines are inefficient, since having a set of specific queries, in order to compute them all, it is needed to fetch data, filter it, and apply needed formulas. It is inefficient, since for every query the same chunk of data needs to be pulled into the memory at least the number of queries times or more in case of more complex queries.

As it has already been mentioned in the previous paper [1], one of the modern issues of analyzing massive data on the fly is social networks. . "In order for a system to be able to perform analytical examination of the data produced in such streaming media, the system should have the capability of fast data access. The reason, the millions of data records(tweets) produced every second. Moreover, these tweets may have different geographical origin, introducing different languages and forms and often times containing unsolicited messages, errors, malicious content, etc. Therefore, some low level data uniformity and cleaning on top of the data access and man-

† Author to whom all correspondence should be sent.

Iliiazbek Akhmedov, 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: akhmedovi@mail.usf.edu, ytu@cse.usf.edu

Joseph Fogarty and Sagar Pandit are with the Department of Physics, University of South Florida, 4202 E. Fowler Ave., ISA 2019, Tampa, FL 33620, U.S.A. Emails: jcfogart@mail.usf.edu, pandit@usf.edu

agement issues should be considered and possibly incorporated in the process of analytical investigation in order to achieve relevant result. ” [1]–[5]

The primary focus and problem in this paper is scientific data analysis. Particles simulation is one of the most popular methods of analyzing certain chemical reactions, physical processes, or other behavior of different materials. Molecular simulations (Molecular Dynamics) are applied in different fields and represent a method of analyzing physical movements of particles, atoms, and molecules in a fixed space with a given period of time, apparently with a possibility of giving initial state for each item that is involved in the process and can affect the system. This system is an N-body simulation. The number of atoms in simulations vary in hundred of thousands, particularly, we may observe two simulation systems of a collagen fiber structure and dipalmitoylphosphatidylcholine (DPPC) bi-layer lipid system consisting of 890,000 and 402,400 atoms respectively on Figure 1. Simulation data represents number of records of physical properties such as mass, charge, velocity, coordinates, and forces for each item aggregated as frames, where each frame represents a snapshot of time, placed with a fixed time interval which may also vary depending on the simulation itself and simulation precision requirement. ”Quantities measured during the simulations are analyzed to test the theoretical model [6], [7]. In short, the MS is proven and powerful tool for understanding the inner-workings of a biological system, by supplying a model description of the biophysical and biochemical processes that are being unfold at a nanoscopic scale.” [1]

Scientist gives the properties to simulation software (for example, Gromacs), runs the simulations, and finally get the output file. The output file must then be analyzed to produce certain results which may help him come up with certain consensus on original theoretical model that resulted in the molecular simulation system [6]. Gromacs is simulation software tool that helps scientist to run the actual simulation. It is a molecular dynamics package primarily designed for biomolecular systems such as proteins and lipids. [8]. Besides the fact that it helps to generate the output files for the simulations, apparently it also helps to analyze the data itself, but the original problem is that it is not as optimized as it can be in order to analyze the data. Gromacs follows approach of pull-based design, which means that for any given query (e.g. total mass or total charge, which are very similar type of 1-body queries without sophisticated selection) it will pull data separately and generate addition overhead wasting disk read I/O in order to come up with the result just for a single query. As it has been proposed in the previous paper, in order to remedy such issues, the push-based design does exactly the opposite, where instead of loading the data on demand for each query, the queries are batched into a network, then the entire dataset is loaded chunk by chunk pushing it through the network which has its internal relationships and dependencies amonth the queries. In this case, since scientific simulation data is run once with specific physical properties, it is never modified, thus, on continuation, it will only append, which means that the processing can also be run on appended frames. [1] This type of approach has already been revised by other systems [9]–[11] as of reusing loaded data through queries produced [12]–[14].

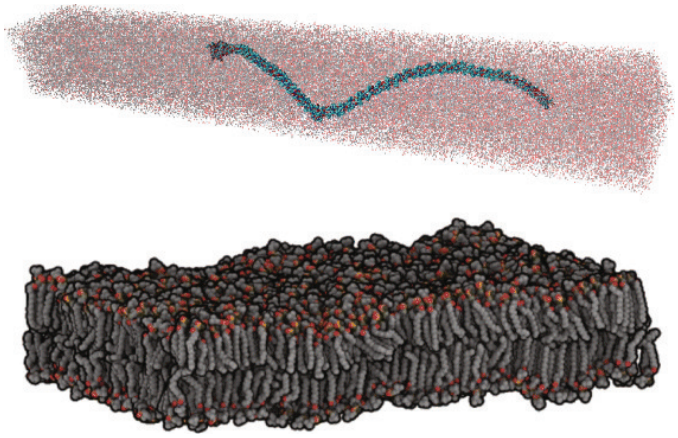

Fig. 1: Snapshots of two MS systems: a collagen fiber structure with 890,000 atoms (top) and a dipalmitoylphosphatidylcholine (DPPC) bi-layer lipid system with 402,400 atoms (bottom) [15]

In this paper, we are incorporating parallelization into the processing part of the proposed design by means of CUDA programming language for GPU. Since storage of the simulation data is very expensive, it might come to the point of analyzing the data on the fly (meaning running the simulation and analyzing it at the same time in a streaming manner), which leads to a problem of optimizing the processing part of the design proposed in the previous paper, because time spent on generating data should be tried to conceal the time spent on the processing part by means of overlapping or simply running it in a quick manner.

A. Problem Statement

Simulation software systems, in general, follow the same methodology of running and storing simulation data. The simulation software system examples are: Gromacs [16], VMD [17], MDAnalysis [18], Wordom [19], MD-TRACKS [20], SimulaidOne [21], Charmm [22]. In the type of simulations brought up as examples above, the flow of the data is the following. Once the simulation is run, the output files are contained as trajectory files with descriptors (they contain information about space dimensions, number of atoms and frames, etc.) that can be easily transposed into simple flat files containing the physical properties atom by atom, frame by frame, which are consequently read and processed by the proposed push-based system. Since we have certain amount of queries needed to be run on given simulation data, generating high I/O traffic followed by design of pull-based system is not considerable. The approach proposed in the previous paper is very good in terms of performance in comparison with the original pull-based system [1]. The problem is still that some of the queries processed by those means are still improvable, especially taking into consideration the fact that in the used previous works for calculation of 2-body functions, which take the biggest time for processing [23], [24], we might have some error bounds, which might be unacceptable

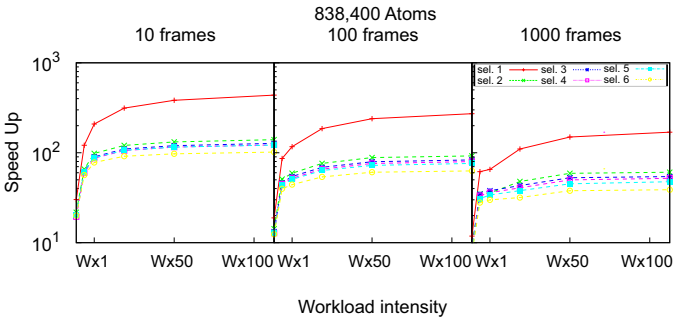

Fig. 2: Speed up over different levels of atom selection for 838,400 atoms [1]

for certain simulation analysis where sacrifice on loss of data is unbearable. Besides it, although we might have huge performance boost on specific data structure as density map, we still have certain expectation from data nature (such as its uniformity), thus, it makes sense to have desire to simplify the processing and still having pure computation based on any values, considering that the queries should be available to be pre-programmed by user in a separate module of the code.

B. Our approach with improvement

Reading the data from files is a huge overhead, thus, speedup of push-based system in general is significant over pull-based system in our given case. It's been demonstrated in the previous paper with different amount of atoms, frames, and workloads. Since we have to load the data every time there is a different query versus push-based system loads a chunk of data once, it is quite noticeable how the framework contributes to efficiency of analysis of simulation data. You can observe some estimation samples in Figure 2. Even though the memory loading and pushing steps are the key features of the proposed design, nevertheless, having general knowledge about the network of queries, in this paper, we will try to focus on optimizing the actual execution of them.

Some of the queries may be very slow due to their nature on a sequential type of computation, especially, 2-body functions. Since the nature of the data that comes with simulation is basically physical properties of atoms, there is a lot of computation that involves independent primitive mathematical operations, which makes it a perfect problem for parallelism.

Although the original idea of push-based system for molecular simulation data analysis was focused on optimizing throughput and usage of loaded data, there were some extra approaches specifically for 2-body functions. For example, Density Map for Spacial Distance Histogram was used in order to avoid additional memory allocation and latency reduction with proven error bounds. SDH is a quite intensive and computational problem, especially with increasing amount of atoms.

We believe that having this nature of computational problems, the proposed GPU improved version of push-based system will significantly change in terms of performance by incorporating parallelism with CUDA.

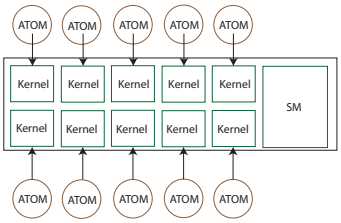

Fig. 3: Example of atom mapping with kernels

C. Contribution and roadmap of the paper

The proposed improved version of push-based system for molecular simulation data analysis, we believe, gives an opportunity for scientists to run their analysis even faster now. Taking an advantage of GPU devices nature and incorporating parallelism and streaming for different types of queries, we have come up with a good speed up over sequential processing, which already had a good speed up in terms of performance in comparison with original pull-based approach. Since in this paper we don't expect certain form of data, we have come up with a tool that generates mock data for any number of atoms and frames, which can be used for performance tests that were done consequently, too. This mock data has exactly the same information that would come with Gromacs simulation files, and it has exactly the same format as in the previous paper [1] right before loading it to memory. This improved version has been developed on Amazon Elastic GPU [25] nodes that consequently can be replicated and scaled up becoming more of a streaming distributed processing engine (which can be very effective on costs, since it is always easy to scale them up, shut them down and spin them back up), as well as on a simple desktop computer with a GPU device. In general, major contribution of this paper is the framework of push-based system with an incorporated parallelism based on CUDA programming language. Besides it, there was developed a tool which generates mockup data of the same format as of the real MS simulations and represent a python script with arguments given as number of frames and atoms.

The structure of the framework which will be described later is developed in such a way that one can easily add new queries based on their complexity or modify existing ones adding appropriate selections and filters.

II. Related Work

Push-based system for MS data analysis is primarily proposing an idea of pushing chunks of data through a network of queries. Essentially, this has also been proposed, as mentioned already, by other frameworks [9]–[11]. In other words, the major point is to use common data loaded into memory for execution of concurrent queries, which is supposed to eliminate I/O overhead predominantly. The architecture and explanation of DSMS (Data Stream Management System) versus DBMS is well explained in one of the lectures of Morgan and Claypool [26]. It is very important to capture the differences and specialties of DSMS in network architecture, stream models and windows, scheduling, load balancing, approximation, data

expiration, etc, because of the fact that data flow design is the key that required all of those changes to common designs.

Since dealing with data is very delicate in this kind of engines, there are couple of features and requirements that this architecture is applicable to, in order to avoid inappropriate understanding or application of this approach in systems that were covered in the lectures [26]. The data is volatile and not persistent anymore because of obvious sizes of data. Since we are pulling the data as little times as possible taking into consideration that MS data is only appended and not modified, the access is sequential rather than random. There are concepts of framing and windowing, and memory is limited in streaming engine, while in DBMS secondary storage is considered to be unlimited. Obviously, because of all of these features or requirements to data, the streaming engine frameworks have some limitations like going back into the history of data, since it is volatile. In fact, this has also been tried by Borealis Streaming Engine [27]. On top of the fact that it is a streaming engine, as it was built based on Medusa [28] and Aurora [29], it proposes distributed processing at this point having some features to handle errors and look back in the history. In our case, it is primarily a single data source, since it is based on the simulation, though in the future the simulation software systems might introduce distributed computing, too. It is also important to mention such projects as PeriScope [30], SciDB [31], and other [32], [33], since they target scientific data, which is a lot more different, than usual industry customer oriented data.

This version of Push-based System of MS data analysis is primarily about taking an advantage of GPU processing of the data. Hadoop is a well-known and well-used framework by many companies nowadays [34]. Its purpose is an ability to store and serve large scale data across clusters in a timely fashion. Since usage of GPU of massive data (not only related to graphics processing) is a relatively new concept, there is also an improved version of Hadoop MapReduce Framework with GPU [35]. Having an ability to scale it up in a distributed computing environment is probably one of the best approaches, but again it might be a bigger overhead in terms of costs, since keeping all nodes up and still being able to solve specific problem sets with hardware stack of a lower capacity raise a will to explore more. Having a specific software for simulation like Gromacs, apparently there is also a possibility of taking an advantage of GPUs. For example, Gromacs allows to optimize simulation and query tools by adding configurations based on GPU device located on the processing computer. Unfortunately, even with this feature, Gromacs doesn't follow push-based approach which means that improvement with GPU with pull data per query makes it negligible.

Gromacs generate output files with physical properties description for each atom in each frame. Modern CPUs do great job in terms of caching and computing, but obviously in this particular problem massive monolithic computation is needed, thus, GPU devices are perfect candidates for such a problem following SIMD type of computation. To be more exact, for example, a single kernel in a GPU device could be dedicated to a single atom (this relationship might change depending on the query, but this is to understand the scale and relationship of multiprocessors), and since we have scientific data of MS, the data is appended, which means that we will only move forward and not take an advantage of caching on chunks high level (though we might take an advantage of GPU caching features particularly in processing phase) having multiple workloads per chunk. Just to make it more clear, workloads in this particular case represent transactions based on number of clients. For example, if we were to serve it all in a streaming fashion, 100 clients might demand concurrent queries with different slight selection properties and expect their results, each workload might be dedicated to each client if not aggregated. Thus, the performance of GPU is certainly increasing based on workload too, which might benefit considerably.

III. Network of Queries

Having generated big amount of data in large files, now scientists need to get useful information out of it by means of analysis utilities. Often times, the queries that need to be run over the data look like selection and some kind of accumulation, if it is not more complicated. To be more exact, in this section we will try to summarize common set of queries we will be improving.

In previous work, there has already been developed a network of queries widely used by scientists for MS systems. In Table I you may observe the common set of queries that has been developed. Basically, these are the queries that need to be improved with our new approach of parallelism. Just to mention, n, r_i, m_i, c_i and q_i denote number of particles, coordinates (vector form), mass, charge, and number of electrons of a particle i.

In general, with the given queries we have divided them into two categories: one body functions and multiple body functions. The first ones apparently have linear complexity of $O(n)$, while the other ones have bigger complexity.

One body functions such as sum of masses, center of mass, or simple counting with selection are of a complexity $O(n)$, thus, in order to get the results for them going through every atom (essentially over the entire data) only once is enough, and taking into consideration the fact that we are going with push-based system, it is done at the same time for all of them.

Multiple body functions such as SDH and RDF (Spacial Distance Histogram and Radial Distribution function) require computation of distances pairwise across all the particles. This is generally combination of two across N data: $C\binom{N}{2}$. These are the most expensive queries, thus, it is important to focus on their implementation taking advantage of GPU nature and its processing power. SDH is a very expensive computation, and we have used some of the existing work of ours [36] to incorporate in this design.

The might be some dependencies and preliminary computation possibilities. For example, some queries need total mass, and we could compute it before we push it further, but this can be easily done by user based on the complexity and dependency. It is explained in more details in further sections.

Function Name	Equation/Description
Moment of Inertia	$I = \sum_{i=1}^{n} m_i r_i$
Moment of Inertia on z axis	$I_z = \sum_{i=1}^{n} m_i r_{zi}$
Sum of masses	$M = \sum_{i=1}^{n} m_i$
Center of mass	$CoM = \frac{I}{M}$
Radius of Gyration	$RG = \sqrt{\frac{I_z}{M}}$
Dipole Moment	$D = \sum_{i=1}^{n} q_i r_i$
Dipole Histogram	$D_z = \sum_{i=1}^{n} \frac{D}{z}$
Electron Density	$ED = \frac{\sum_{i=1}^{n}(e_i - q_i)}{dz \cdot x \cdot y}$
Heat Capacity	$HC = \frac{3000 \cdot \sqrt{T} \cdot boltz}{2 \cdot \sqrt{T} - n \cdot df \cdot VarT}$
Isothermal Compressibility	$I = \frac{VarV}{V_{avg} \cdot boltz \cdot T \cdot PresFac}$
Mean Square Displacement	$msd = \langle (r_{t+\Delta_t} - r_t)^2 \rangle$
Diffusion Constant	$D_t = \frac{6 \cdot msd(t)}{t}$
Velocity Autocorrelation	$V_{acor} = \langle (V_{t+\Delta_t} \cdot V_t) \rangle$
Force Autocorrelation	$F_{acor} = \langle (F_{t+\Delta_t} \cdot F_t) \rangle$
Density Function	Histogram of atom counts
SDH	Histogram of all distances
RDF	$rdf(r) = \frac{SDH(r)}{4 \cdot \pi \cdot r^2 \cdot \sigma_r \cdot \rho}$

TABLE I: Popular analytical queries in MS [1]

IV. The system design

The proposed design has a pretty straightforward structure which is going to be explained in this section. Shortly, the system starts with original input files (or data) and gets consumed in a sequential manner by simple disk IO framework into random access memory by chunks, then, chunk gets uploaded to GPU device and processed, at some point, the results that need to be shown to the user are transported from GPU (where they initially get accumulated) to the main memory, and then might get written as well to some other output files.

A. Data flow

Since the size of the data in the problem is huge, organization of the data flow is crucial in our case. As mentioned before, original data is retrieved from simulation software output files and in our case it is Gromacs Software's formatted trajectory files. Tool for decompressing those files to a flat format has been developed in the previous work [1].

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

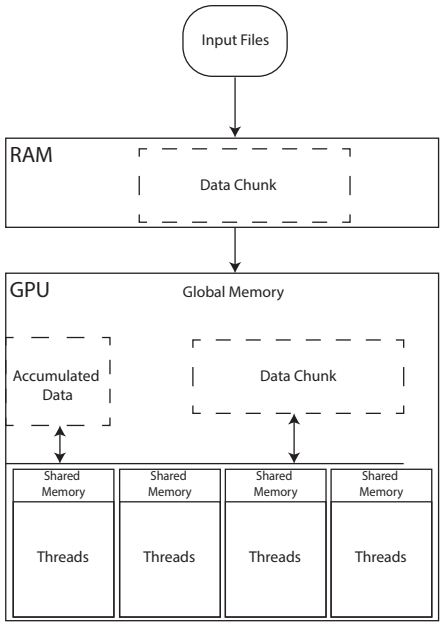

Fig. 4: Data flow

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

Data organization in main memory.[1] Once the data is in a format our system can read, the data is being loaded into the main memory one frame at a time. The organization of the in-memory particle's data is in the form of a simple, two dimensional array where a single row represents an atom in the system with all its attributes (e.g., coordinates, mass, charge, residue info, etc.). We also keep (in a one dimensional array)

[1]This paragraph talks only about the data organization used by one-body queries. For two-body queries (e.g., SDH), the data organization is discussed later on.

crucial system's information for each frame, like temperature, energy, pressure, etc. We have used such structures because they are very suitable for simulating a push-based type of system: a simple sequential read of the array gives that online type of data stream. So, as the system reads the array, it pushes the data onto the query-modules. As mentioned earlier, the one-body (algebraic) queries will produce a final result at the end of the first sequential read. However, the two-body (holistic) functions, like SDH and RDF, cannot do this in a single read of the data.[2] Therefore, each frame's data array can be continuously read (in a loop manner) as many times as a query needs it.

B. Query modules

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

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

2) Two-body queries: In general, queries involving two-body functions are a bit more complex and cannot provide the final result in a single data read if a straightforward method is used for their computation. However, in the proposed system we have incorporated a data structure and an algorithm for the SDH (also RDF) from our previous work that is suitable for push-based type of system. In this subsection we give a brief description of the data structure (DM) and the algorithm (DM-SDH) designed in [23], [24] and implemented in the system proposed in this paper. For more detailed information, please refer to our previously published work on this topic [23], [24]

The data structure. The simulation data space is represented by a conceptual data structure we named Density Map (DM). The density map splits the simulation space into a grid of equal size regions (or cells). The cells are cubes in 3D and squares in 2D[3] Resolution of a density map is the reciprocal of the cell size in that density map. In order to generate higher resolution density map, we split each cell of the current resolution's grid into four smaller cells of equal size. This design allows us to use a region quad-tree [37] to organize density maps of the same data but with different resolutions. So, essentially, a node in the quad-tree represents a single cell from the DM. Therefore, a density map of a certain resolution basically is the set of all nodes of one level of the tree. Each of the tree nodes records the cell's location in the density map (coordinates of corner points) as well as the number of particles in each cell. We name the afore describe tree the Density-Map tree (DM-tree).

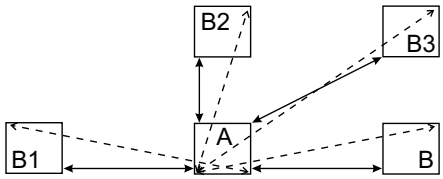

Fig. 5: Computing minimum (i.e., length of solid lines) and maximum distance (i.e., length of dashed lines) range between two cells

The algorithm. The essential part of the DM-SDH algorithm is a procedure named RESOLVETWOCELLS. The input to this procedure are two cells from the density map (e.g., A and B in Fig. 5). It computes, in constant time, the minimum and maximum distance between the two cells. A pair of cells is *resolvable* if both the min and max distance between them fall into the same SDH bucket i. If that is the case, the distance count of that bucket is being increase by $n_A n_B$ (n_A and n_B are the number of particles in cell A and B, respectively). Otherwise, the cells are non-resolvable and we either:

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

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

On top of the aforementioned DM-SDH algorithm, we have also incorporated two approximate SDH algorithms (ADM-SDH), introduced and described in [24], and [38]. These approximate algorithms are substantially faster than the the brute-force algorithm and also than the DM-SDH algorithm as they take advantage of some heuristic. For more details on the ADM-SDH algorithms please see the aforementioned work.

C. Working of the system

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

Here are the steps taken through out the analysis:

(1) Execute the data transformer
  (a) Read the MS data from trajectory files
  (b) Extract the info needed for our system

[2]However, we have previously deigned and created a data structure and an algorithm that can take the advantage of a single data read and produce final results for SDH computation [23], [24]. We have incorporate this into our system presented in this paper.

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

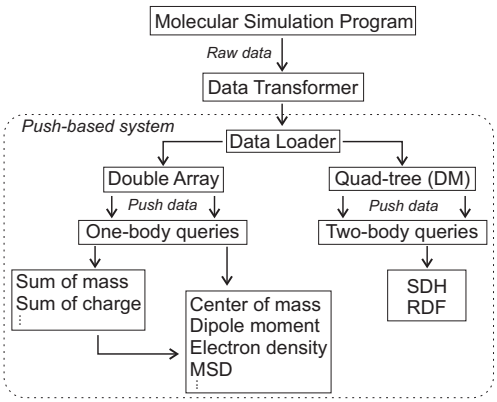

Fig. 6: Push-based system flow

(c) Save the read data to a file recognizable to the system

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

(3) Push the data to all queries
(4) A query, if available, acts upon the pushed data (first executing the lower level, sub-queries)
(5) Store intermediate results (results of sub-queries)
(6) Repeat steps 3-5 if needed.
(7) Output results
(8) Go to step 2 and load the next frame (if needed).

Fig. 6 depicts the flow of the system.

Step 2, loading the data into main memory is different for SDH query compared to the one for the one-body queries. The reason for that, as mentioned earlier, is the different data structure used to store the data in memory. While we use double array to store the data for the one-body queries, we use quad-tree like data structure to store the data needed to compute the SDH. The loading to the double array is straightforward. However, to load the quad-tree structure, we need to use some of the info from the data itself. Namely, the coordinates of the atoms are used to determine in which tree node an atom belongs. That way we build the so called density map (DM), i.e., the different regions with a certain number of atoms in them (including all the atom's attributes). So basically, to solve the SDH problem in a push-based manner, we convert the problem into populating a data structure in push-based manner. This data structure will then be used as an input to our DM-SDH algorithm that, although not completely in "on-the-fly" way, is a great improvement over the naive methods used in much of today's MS analysis systems (i.e., GROMACS, PDB, CHARMM, etc.).

V. Experimental results

The system was implemented in C++ programming language and tested on real molecular simulation data sets. The experiments were carried out on an Apple MacPro machine with 8GB of physical memory and two Quad-Core Intel Xeon 3GHz processors. The MacPro was running OS X Mavericks 10.9.3 operating system. We have compared the

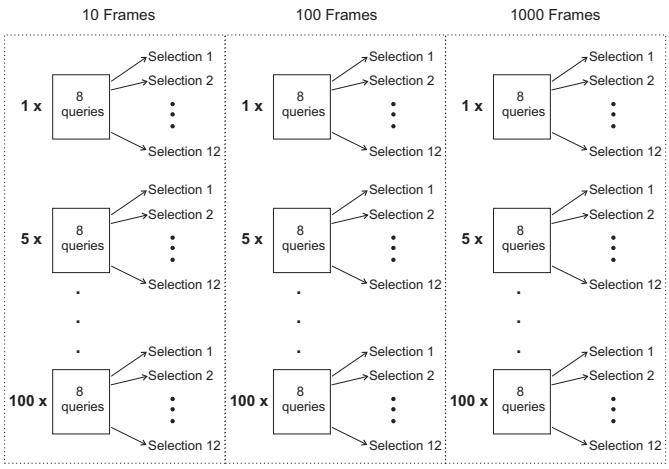

Fig. 7: Workload setup

results obtained by our system to those obtained by running the analysis through GROMACS (v. 4.5.7). Both systems were analyzing the same data sets.

Data sets. In our experiments, six data sets from different simulations were used. All simulations were done on a POPC [4] lipid bilayer, but were all set to produce data of different sizes (i.e., different number of particles in the simulation). Namely, we have tested the system on simulations with $52,400$; $209,600$; $838,400$; $2.5M$; $4.4M$; and $8.8M$ atoms. Also, since the simulations were run separately, they produced six different MS systems with distinct characteristics (distinct structure, atom's positioning, etc.). From all of the generated data sets we have randomly selected sets of 10, 100, and 1000 consecutive frames for the purpose of our experiments. This gave us 18 different datasets on which we tested our system.

Query work load. Two types of query workload were used: 1) one involving one-body queries only 2) one including two-body queries (SDH and RDF) as well. The reason for this is that GROMACS, the system we used to compare our system to, only has a naive method of solving the RDF (SDH) problem (like almost all MS analysis systems). In our system we have incorporated SDH (RDF) algorithms that are far more superior to the naive method, and comparing the systems like that would not have been fair (we believe).

One-body queries only. The following set of one-body queries were included in the test workload: mean square displacement (msd), radius of gyration, dipole moment, center of mass, velocity autocorrelation, electron density, mass density, and charge density. This set of queries were pointed to us, by a group in the physics field with extensive MS background, as one of the most commonly used in the field of collagen bilayer MS system analysis. A workload group contains all 8 queries executed on one of the 12 selections, making 12 groups. Such groups are executed on six different size data sets, with 10, 100 and 1000 frames. This workload is then

[4]POPC is a chemical compound composed of a diacylglycerol and phospholipid. Its full name is 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine and it is one of the most important lipids in bio-physical molecular simulation.

repeated 5 more times, by executing each of the queries in the groups 5, 10, 25, 50, and 100 times, essentially just magnifying the workload intensity. In total, we have $12x6x3x6 = 1,296$ different workload setups to test the system on. Fig. 7 shows the organization of the workload setup.

A. Benchmark

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

1) Benchmark Data: The data used in the benchmark was real molecular simulation data, produced through the GROMACS MS system. The initial, pre-simulation data file consisted of 200 POPC and 12000 solvent molecules, or 12200 molecules in total. This type of system was used because it is sufficiently diverse, containing enough distinct POPC and solvent molecules (e.g., each POPC molecule includes approximately 52 different atoms) and yet simple enough to be easily transformed into another system of different size. By using the $genconf$ function in GROMACS, we produced pre-simulation files of different sizes (essentially by changing the system's size (box)). Six different sized pre-simulation files were created. A molecular simulation was then run on these 6 files, each producing an MS system of certain size (volume/number of particles). All of the simulations were set up to produce 1000 frames (snapshots in time of the systems), each frame containing the same number of particles as the base one. The produced files contained: $52,400$; $209,600$; $838,400$; $2.5M$; $4.4M$; and $8.8M$ atoms per frame. So, for example, the file with $52,000$ atoms holds $52,000,000$ records in total (1000 frames, each containing $52,000$ records). As mentioned earlier, this simulation data comes mostly in binary formats and in trajectory files having a lot of unneeded overhead. Therefore, it was transformed to a data arrays files containing only crucial information of the particles and the system. The size of the files ranged from $135MB$ for $52,000$ atoms to $24GB$ for 8.8 million atoms (this is for data with 100 frames).

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

3) Benchmark Parameters: There are several parameters that can be used to control the overall size of the system. We divide the parameters into two groups:
Data size parameters:

- Select different sized dataset
- Number of frames
- Data selection (within the selected dataset) onto which the queries are being executed

Workload size parameters:

- Number of queries to be executed
- Number of times each query is executed

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

B. Results

We have run extensive experiments over all the different setups of workload mentioned previously in this section. However, in this paper we present only the workload setups of 4 data size sets: $838,400$, $2.5M$, $4.2M$, and $8.8M$ atoms because we believe they convey enough information about the efficiency of our system compared to that of the Gromacs system. The running times of our push-based system were compared to those of the Gromacs system. The first set of figures, namely Figures **??**-**??**, represent the speedup that our system obtains over the Gromacs system with various atoms selection levels. We define the selection levels based on the number of comparisons we have to make in order to extract the needed group(selection) of atoms. For example, if we want to do analysis on all molecules containing oxygen, or hydrogen, or carbon we would go over each molecule and compare its components to the selection list. The bigger the selection list, the higher the select level in our system. For better visualization, we note three different selection levels: high (at least 10 comparisons made), medium (between 1 and 10 comparisons made), and low select level (with one or less comparisons made). As seen in the figures, for high selection level, the speedup is smaller compared to that achieved in low select levels. The reason for this, we believe is in that the amount of time our system spends extracting the atoms group increases with the level of selection. Even though our system still shows considerable speedup over Gromacs in high level selections, we do believe there is room for improvement in our system and that is our immediate future work we are planning on doing. These figures also show the relation of the speedup to the workload intensity, i.e., the higher the workload intensity the higher the speedup.

The connection between the workload intensity and the speedup is better represented in the next set of figures, Figures **??**-**??**. They show the speedup our system achieves over the Gromacs system on a varying workload intensity. Each of those figures show the speedup with different dataset sizes (e.g., $838,000$, $2,567,600$ atoms, etc.), including 10, 100, and 1000 data frames. The speedup is calculated simply as a ratio between the running time of our system on a certain set of workload and that of the Gromacs system on the same workload. These figures show that the speedup over varying workload intensity achieved by our system ranges anywhere from about 10 to 1000 times, depending on the size of the dataset, number of frames and the selection of the atoms.

Figure **??**, shows the speedup our system achieves over all workload intensity (average workload intensity) with varying dataset sizes. It is clear that, again our system has better performance than the Gromacs system. The speedup presented in this set of figures ranges anywhere from about 15 to 650 times.

The last figure, Figure **??**, shows the speedup our system achieves over all workload intensity and all select levels with varying dataset sizes. This figure, in a way, summarizes the previous two sets of figures, bringing together the workload and the different selections through the average. It is clear that, again our system has better performance than the Gromacs system. The speedup ranges anywhere from 50 to 250.

All four sets of figures show that such push-based design has clear advantages over the pull-based type of design incorporated in the Gromacs system.

VI. Conclusions and Future Work

The objective of our work is to design and implement improved data analysis system that can be used in the field of molecular simulation system's analysis. In this paper, we introduce the idea for such system. We build our system on a push-based type design, where data from data arrays is being pushed onto available queries in the system. These queries are being executed on the pushed data and produce intermediate / final result that would be used as part of the data analysis. We are able to achieve an improvement over existing, pull-based type designs because of the I/O overhead such designs introduce when dealing with large volumes of scientific data. Also, our queries are able to be executed on the same stream of data, making it suitable solution for streaming circumstances. We designed a benchmark that can be used to test data analysis systems. This benchmark comprises of three parts: 1)benchmark data, 2)benchmark queries, and 3) benchmark parameters. We use this benchmark to compare our system to one of the most frequently used MS analysis systems, Gromacs. The efficiency and speedup achieved by our system is supported by extensive experiments and their results. The results show that our push-based design achieves up to about 1000 times speedup in comparison to a pull-based design, i.e., Gromacs.

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

Acknowledgements: The project described was supported by an Award (R01GM086707) from the National Institute Of General Medical Sciences (NIGMS) at the National Institutes of Health (NIH). The authors would like to thank Anand Kumar who has contributed his time and knowledge toward completion of the work.

References

[1] Vladimir Grupcev, Yicheng Tu *et. al.*, "Push-based system for molecular simulation data analysis." 2015.

[2] D. H. et al., "Big data: The future of biocuration," *Nature*, vol. 455, pp. 47–50, 2008.

[3] B. Huberman, "Sociology of science: Big data deserve a bigger audience," *Nature*, vol. 482, p. 308, 2012.

[4] D. Centola, "The spread of behavior in an online social network experiment," *Science*, vol. 329, pp. 1194–1197, 2010.

[5] J. Bollen, H. Mao, and X.-J. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, pp. 1–8, 2011.

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

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

[8] Gromacs group, "GROMACS - Online Reference." [Online]. Available: http://gromacs.org/

[9] Subi Arumugam and Alin Dobra and Christopher Jermaine and Niketan Pansare and Luis Perez, "The datapath system: A data-centric analytical procesing engine for large data warehouses," *SIGMOD*, vol. 1, pp. 519–530, 2010.

[10] Goetz Graefe, "Volcano - an extensible and parallel query evaluation system," *TKDE*, vol. 6, pp. 120–135, 1994.

[11] Stavros Harizopoulos and Vladislav Shkapenyuk and Anastassia Ailamaki, "Qpipe: A simultaneously pipelined relational query engine," *SIGMOD*, pp. 383–394, 2005.

[12] Gorge Candea and Neoklis Polyzotis and Radek Vingralek, "A scalable, predictable join operator fo highly concurrent data warehouses," *VLDB*, pp. 277–288, 2009.

[13] P Unterbrunner and G Giannikis and G Alonso and D Fauser and D Kossmann, "Predictable performance for unpredictable workloads," *VLDB*, vol. 2, pp. 706–717, 2009.

[14] Marcin Zukowski and Sandor Heman and Niels Nes and Peter Boncz, "Cooperative scans: Dynamic bandwidth sharing in dbms," *VLDB*, pp. 723–734, 2007.

[15] Vladimir Grupcev, Yicheng Tu, Meryem Berrada *et. al.*, "Dcms: A data analytics and management system for molecular simulation," 2014.

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

[17] W. Humphrey, A. Dalke, and K. Schulten, "Vmd: visual molecular dynamics," *Journal of Molecular Graphics*, vol. 14, pp. 33–38, 1996.

[18] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, "Mdanalysis: A toolkit for the analysis of molecular dynamics simulations," *Journal of Computational Chemistry*.

[19] M. Seeber, M. Cecchini, F. Rao, G. Settanni, and A. Caflisch, "Wordom: a program for efficient analysis of molecular dynamics simulations," *Bioinformatics*, vol. 31, pp. 2658–2668, 2010.

[20] T. Verstraelen, M. V. Houteghem, V. V. Speybroeck, and M. Waroquier, "Md-tracks: a productive solution for the advanced analysis of molecular dynamics and monte carlo simulations," *Journal of Chemical Information and Modeling*, vol. 48, pp. 2414–2424, 2008.

[21] M. Mezei, "Simulaid: a simulation facilitator and analysis program," *Journal of Computational Chemistry*, vol. 23, pp. 2625–2627, 2007.

[22] B. R. Brooks *et. al.*, "Charmm: the biomolecular simulation program," *Journal of Computational Chemistry*, vol. 30, pp. 1545–1614, 2009.

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

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

[25]

[26] L. Golab and T. Ozsu, *Data Stream Management*. Morgan And Claypool, 2010.

[27] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska *et. al.*, "The design of the borealis stream processing engine," 2005.

[28] B. L. *et. al.*. Moo-Ryong Ra, "Medusa: A Programming Framework for Crowd-Sensing Applications," 2012.

[29] Apache, "Aurora Framework." [Online]. Available: http://aurora.apache.org/

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

[31] P. Cudre-Mauroux *et. al.*, "A demonstration of scidb: A science-oriented dbms," *VLDB*, vol. 2, pp. 1534–1537, 2009.

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

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

[34] S. R. R. C. Konstantin Shvachko, Hairong Kuang, "The Hadoop Distributed File System," 2010.

[35] A. S. *et. al.*. RadhaKishan Yadav, Robin Singh Bhadoria, "Gpu-accelerated large scale analytics using mapreduce model," 2015.

[36]
[37] J. Orenstein, “Multidimensional tries used for associative searching,” *Information Processing Letters*, vol. 14, no. 4, 1982.
[38] Vladimir Grupcev *et. al.*, “Approximate algorithms for computing spatial distance histograms with accuracy guarantees.” *TKDE*, 2012.