2011 IEEE International Parallel & Distributed Processing Symposium

FENZI: GPU-enabled Molecular Dynamics Simulations of Large Membrane Regions based on the CHARMM force field and PME

Narayan Ganesan, Michela Taufer
Dept. of Computer & Inf. Sciences
University of Delaware
Email: {ganesan, taufer}@udel.edu

Brad Bauer, Sandeep Patel
Dept. of Chemistry and Biochemistry
University of Delaware
Email: {babauer, patel}@udel.edu

***Abstract*—When studying membrane-bound protein receptors, it is necessary to move beyond the current state-of-the-art simulations that only consider small membrane patches and implicit solvent. Limits of traditional computer platforms negatively impact the model's level of realism and the computational scales achievable. On the other hand, multi-core platforms such as GPUs offer the possibility to span length scales in membrane simulations much larger and with higher resolutions than before.**

To this end, this paper presents the design and implementation of an advanced GPU algorithm for Molecular Dynamics (MD) simulations of large membrane regions in the NVT, NVE, and NPT ensembles using explicit solvent and Particle Mesh Ewald (PME) method for treating the conditionally-convergent electrostatic component of the classical force field. A key component of our algorithm is the redesign of the traditional PME method to better fit on the multithreading GPU architecture. This has been considered a fundamentally hard problem in the molecular dynamics community working on massively multithreaded architecture. Our algorithm is integrated in the code FENZI (*yun dong de FEN ZI* in Mandarin or *moving molecules* in English). The paper analyzes both the performance and accuracy of large-scale GPU-enabled simulations of membranes using FENZI, showing how our code can enable multi-nanosecond MD simulations per day, even when using PME.

***Keywords*-Biomolecular Systems Simulation; Molecular Dynamics; DMPC Membranes; Performance; Accuracy; Ewald Summation; Particle Mesh Ewald.**

I. Introduction

Roughly one-third of the human genome is composed of membrane-bound proteins that are only now becoming structurally resolved due to heroic experimental efforts. Furthermore, pharmaceuticals target membrane-bound protein receptors (such as G-protein coupled receptors, GPCR's), thus emphasizing the importance of such systems to human health and understanding of dysfunction. Computing and communication bottlenecks limit membrane simulations to the simulation of small regions (or patches) of physiological membranes using implicit solvent representations. Heterogeneity of membranes spans length scales much larger than included in these smaller model systems. Thus when studying membrane-bound protein receptors, it is necessary to move beyond the current state-of-the-art simulations that only consider small patches and implicit solvent. While the biological models are mature for this step, still limits of traditional computer platforms negatively impact the model's level of realism and the computational scales achievable. With the advent of multi-core platforms such as GPUs, scientists have now new computing tools to overcome these barriers. At the same time the migration of the biological models to the new platforms include new challenges, i.e., algorithm redesign and implementation to meet the new platform configurations.

To this end, in this paper we present the design and implementation of an efficient algorithm for Molecular Dynamics (MD) simulations for GPUs. The algorithm is integrated in the code FENZI (*yun dong de FEN ZI* in Mandarin or *moving molecules* in English) that enables the fast simulation of membrane regions in the NVT (i.e., constant atom number, constant volume, and constant temperature) NVE (i.e., constant atom number, constant volume, and constant energy), and NPT (i.e., constant atom number, constant pressure, and constant temperature) ensembles with explicit solvent. A key contribution of this paper is the presentation of an efficient GPU algorithm for the Ewald summation for treating the conditionally-convergent electrostatic component of the classical force field and the Particle Mesh Ewald (PME) method to compute the corresponding reciprocal space terms. While adding accuracy to the simulation results, the use of the PME method also adds an additional layer of complexity to the efficient implementation of FENZI on GPUs, since requiring to deal with charge location and spread as well as Fast Fourier Transform (FFT) calculations on a multi-threading platform. We address this challenge in FENZI by rethinking the way changes are spread and handled from a charge-centric representation (as it is traditionally done in CPU algorithms) to a lattice-centric representation.

We use FENZI to analyze the performance and accuracy of large-scale, GPU-enabled computations of extended phospholipid bilayer membranes (DMPC), and show how our code can enable multi-nanosecond MD simulations per day, even when using PME for large membranes, with 273,936 atoms in explicit solvent. We use CHARMM as the reference code to validate the FENZI accuracy and

IEEE computer society

performance by comparing results of FENZI versus results of the same DMPC simulations with CHARMM on multi-core systems. Even though the motivation of this work was the study of large membrane regions, FENZI is a full-fledged general purpose MD package based on canonical force fields and PME for GPU enabled simulation of a broad class of molecular systems. Thus, for the sake of comparison with other existing MD codes for GPU including PME, we measure FENZI performance when running the DHFR benchmark.

The rest of the paper is organized as follows: Section II presents the modeling and implementation of our MD code, including the PME algorithm for GPUs; Section III shows the performance and accuracy of the code from the scientists' and computer scientists' prospectives; Section IV discusses relevant related work; and Section V concludes the paper and lists future work.

II. Modeling and Implementation

A. MD Modeling

FENZI (*yun dong de FEN ZI* in Mandarin or *moving MOLECULES* in English) enables GPU-based MD simulations in constant energy (also called NVE) or constant temperature (also called NVT) ensembles using a modified version of the CHARMM force field in terms of force field functional forms and measurement units [1]. The entire MD simulations (i.e., intra-molecular and long range potentials) are computed on GPU.

The intra-molecular potential, which includes bonds, angles including the Urey-Bradley summation, and dihedral, is as follows:

$$V_{intra} = \sum_{\text{bonds}} K_b (r - r_0)^2 + \sum_{\text{angles}} K_a (\theta - \theta_0)^2 + \sum_{\text{Urey-Bradley}} K_{UB} (S - S_0)^2 + \sum_{\text{dihedrals}} K_\phi (1 + \cos(n\phi - \delta)) + \sum_{\text{impropers}} K_\omega (\omega - \omega_0)^2$$

where $K_b, K_a, K_{UB}, K_\phi, K_\omega$ are the bond, angle, Urey-Bradley, dihedral and improper force constants, and $r_0, \theta_0, S_0, \delta, \omega_0$ are their corresponding equilibrium distances.

Non-bonded Lennard-Jones interactions are modeled using a standard 6-12 dispersion-repulsion potential:

$$V_{LJ} = \sum_{i,j}^{pairs} \left(4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \right)$$

where ϵ_{ij} and σ_{ij} are the potential well depth and the van der Waals radius, respectively, used in the Lennard-Jones potential.

Long range electrostatic interactions can be computed either by using the reaction force field (RF) or the Ewald summation method. In this work we employ Ewald summation for electrostatic interactions and the PME method to compute the corresponding reciprocal space terms. In the Ewald summation method the electrostatic interactions are divided into the direct space energy (E_{dir}), the reciprocal space energy (E_{rec}), and the self energy (E_{corr}) contributions to the total energy, depending on the distance of the interaction [2]. The three contributions are:

$$E_{dir} = \sum_{i=1}^{N-1} \sum_{j>i}^{N} \frac{q_i q_j \operatorname{erfc}(\beta r_{ij})}{r_{ij}}$$

$$E_{rec} = \frac{1}{2\pi V} \sum_{\vec{m} \neq 0} \frac{\exp(-\pi^2 \vec{m}^2 / \beta^2)}{\vec{m}^2} S(\vec{m}) S(-\vec{m})$$

$$E_{corr} = -\frac{1}{2} \sum_{(i,j) \in \text{Excl}} \frac{q_i q_j \operatorname{erf}(\beta |r_i - r_j|)}{|r_i - r_j|} - \frac{\beta}{\sqrt{\pi}} \sum_{i=1} N q_i^2$$

where β is the Ewald parameters and $\vec{m} = (m_1, m_2, m_3)$ are reciprocal space lattice vectors, V is the volume of the unit cell in the reciprocal space, and $S(\vec{m})$ is the lattice structure factor given by:

$$S(\vec{m}) = \sum_j q_j \exp(\vec{m}\ _j)$$

The solvent is treated explicitly. Water bond, angle, and charge parameters are transferred directly from the SPC/Fw model of Wu *et al.* [3].

B. Code Implementation on GPU

We use CUDA for our code implementation. Bond-, angle-, and dihedral interactions are each handled by a unified kernel that evaluates the potentials by iterating through the list of all atoms bonded to or involved in an angle with an atom i and accumulates the appropriate forces. Unlike non-bonded lists, the bond-, angle-, and dihedral lists never require updating, so they are constructed once on the CPU at the start of the simulation and then copied to the GPU.

Non-bonded interactions (i.e., Lennard-Jones and direct space electrostatic terms) are handled by two kernels, `NBBuild` and `NonBondForce`.

In `NBBuild`, FENZI uses the cell based approach for building the nonbond neighbor list. The entire domain of atoms is decomposed into regular cubic cells of edge length equal to r_{cut}. The list of all the neighboring atoms within the cutoff r_{cut} is build so that only atoms in the neighboring cells are searched. The neighbor list is constructed using the Verlet list approach [4], in which a list is constructed for each atom containing all atoms within a cutoff $r_{list} > r_{cut}$ and not in the exclusion. During the construction of the neighbor list, the exclusion list is rapidly searched by constructing a bitmask of exclusion atoms [5] for each particle, which enables bitwise operations. The nonbond neighbor list only needs to be updated when an atom has moved beyond a user defined buffer size. While building the neighbor list for each atom, only the atoms within the current cell and 26 neighboring cells in 3-dimensional space need to be searched for atoms within the cutoff distance. The cells themselves are updated efficiently by carefully keeping track of movement atoms within the cells and employing atomic intrinsics where possible.

In `NonBondForce`, each thread iterates through the list of neighbors for a single atom i and accumulates the interactions between i and all its neighbor list entries. The texture cache, which speeds up reading from global memory locations that are not contiguous, is used for reading the coordinates of the neighbor atoms. Switched force forms are used for the Lennard-Jones potential to ensure that both energies and forces go smoothly to zero at the cutoff r_{cut}.

C. Smooth Particle Mesh Ewald

The multithreaded GPU architecture requires us to re-design the Smooth Particle Mesh Ewald (PME) algorithm, used to compute the corresponding reciprocal space terms in the Ewald summation. More in particular, we need to rethink the way charges are spread, handled, and updated during the MD simulation. This problem has been considered fundamentally hard to be ported onto massively multithreaded platforms [6] due to the atomic nature of computations involved. Here we propose to redesign the PME computation by moving away from a charge-centric representation (as it is traditionally done in CPU algorithms) to a lattice-centric representation for GPUs. This enables us to keep track of the movement of charges to different lattice points efficiently and optimally.

The Smooth Particle Mesh Ewald (PME) component of the Ewald summation method (E_{rec}) requires us to deal with location of charges. The entire procedure involves: (1) spreading the charges within a neighboring volume of 4x4x4 cells for each charge in order to obtain a 3-dimensional charge matrix, (2) computing the inverse 3D FFT of the matrix, (3) multiplying the charge matrix by pre-computed structure constants, (4) computing the forward FFT of the product, and (5) summing the entries of the matrix to obtain energy and forces [2]. The charge spreading is performed in the `ChargeSpread` kernel and the list updating is done in the `LatticeUpdate` kernel. The FFT computations, i.e., inverse and forward FFTs, are performed using the CUDA FFT library. The kernel `CUFFTExec` performs both the FFTs. The multiplication of the charge matrix by pre-computed structure constants is performed by the kernel `BCMultiply`. Last but not least, the summation of the entries of the matrix to obtain energy and forces is performed by the kernel `PMEForce`.

Of these five steps, the charge spreading is one of the most compute intensive. In [7], where another example of MD code with PME for GPUs is presented, this step is implemented by a three-step process which involves placing the charges on lattice points with utmost one charge per point (with extra charges considered separately), accumulating the impact of charges for each points, and finally accumulating the effects of the extra charges. Our implementation differs from the implementation in [7] since we accumulate impact of charges based on a local list of charges for each lattice point. Our charge spreading on GPU is performed by maintaining a local neighbor list for each lattice point, followed by an efficient update of list caused due to the movement of particles. In general, the charge contained in each atom is spread within a neighborhood of 4x4x4 lattice points around the particle. The spreading of charges is done with the help of the cardinal B-spline according to the summation:

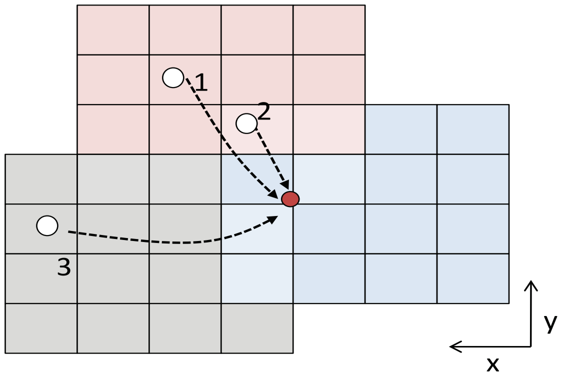

Figure 1. Example of influence region of three charges. The combined effect of the multiple charges impacts the volume under intersection of neighborhoods.

$$\begin{aligned} Q(k_x, k_y, k_z) = & \sum_{i=1}^{N} \sum_{n_1,n_2,n_3} q_i M_4(u_{xi} - k_x - n_1 K_x) \\ & \times M_4(u_{yi} - k_y - n_1 K_y) \\ & \times M_4(u_{zi} - k_z - n_1 K_z) \end{aligned}$$

where k_x, k_y, k_z index the 3-D charge matrix of size, K_x, K_y, K_z and u_{xi}, u_{yi}, u_{zi} are the relative x, y, z coordinates of the ith particle w.r.t. the charge matrix. The cardinal B-spline M_4 is continuously differentiable and has a compact support. As shown in Figure 1, each charge is spread around a neighborhood of 4x4x4 lattice points and multiple charges can contribute to the total charge at each point.

When a charge moves due to dynamics of the system, as shown in Figure 2 where a charge moves from Point 2 to Point 2', we can identify three regions of the lattice: (1) a region from which the charge is removed from the neighbor lists (light gray region in Figure 2), (2) a region whose neighbor lists gain the charge (dark grey region in Figure 2), and (3) a region whose neighbor lists are not impacted (white region in Figure 2). The solid arrow indicates the

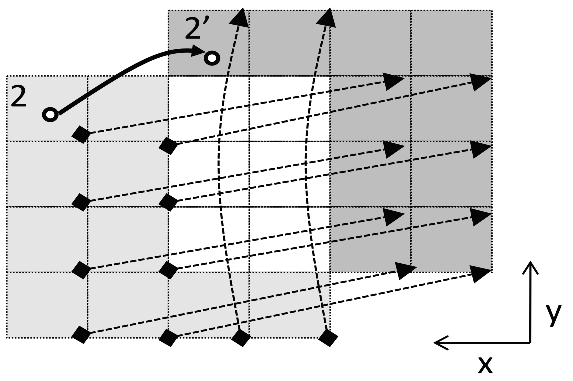

Figure 2. Example of charge update for which only the affected lattice points are updated.

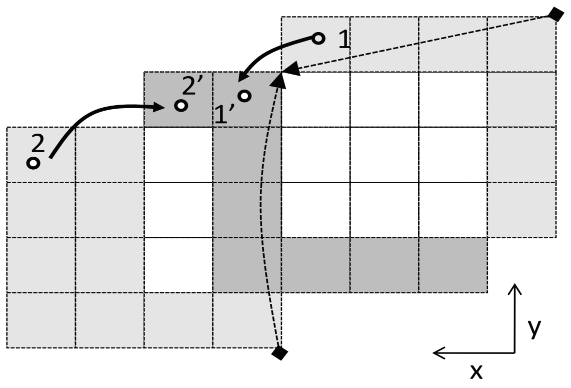

Figure 3. When a single cell is affected due to displacement of multiple charges, it is updated with the help of integer atomic intrinsics.

charge movement; dash arrows indicate mapping between lattice points. With reference to the mapping, the lattice points gaining the charge impact are initially not aware of the charge movement and thus need to search through the global list of charges for the information when an update takes place. This search is time demanding. At the same time, we observe that the movement of a charge produces an equal number of lists gaining the charge and lists losing the charge. The latter lists are associated to lattice points that are aware of the destination points in a one-to-one mapping schema showed in Figure 2. We can benefit from this one-to-one mapping to reduce the time for the update of the lists by having the threads of the points losing the charge updating the lists of the points gaining the charge. This will prevent the lattice points gaining the charge impact to explore the whole global list of charges. In case multiple charges from different lattice points move into the neighborhood of the same lattice point, then the neighbor list of that point is updated by multiple threads. In Figure 3 the local neighbor list of a point point is updated by two different threads due to the movement of two different charges (Charges 2 and 1). Since this update requires the addition of two charges by different threads to the same neighbor list, this is performed with the help of thread safe integer atomic intrinsics.

To minimize the number of floating point calculations, we pre-compute several functions (i.e., B-Spline values, the structure constants, and the value of error function(erfc)) and store them in the constant memory.

III. Performance and Accuracy

A. Benchmark and Set-up

To demonstrate both FENZI scalability and performance across multiple scales of system sizes, we considered a commonly used membrane system, the lipid bilayer membrane (DMPC), with three different sizes, one four times larger than the previous, as shown in Figure 4:

- Small membrane (DMPC 1x1): 46.8 X 46.8 X 76.0 Å, 17,004 atoms (i.e., 14,096 bonds, 19,108 angles, and 22,536 dihedrals)
- Medium membrane (DMPC 2x2): 93.6 X 93.6 X 76.0 Å, 68,484 atoms (i.e., 56,696 bonds, 76,588 angles, and 90,144 dihedrals)
- Large membrane (DMPC (4x4): 187.2Å X 187.2Å X 76.0Å, 273,936 atoms (i.e., 226,784 bonds, 306,352 angles, 360,576 dihedrals)

In the small membrane (DMPC 1x1) we deal with 2,836 explicit water molecules; in the medium membrane we deal with 11,500 explicit water molecules; in the large membrane we deal with 46,863 explicit water molecules. Normally

Lipid bilayer membranes are an important class of biological components, and fundamental study of their structure, dynamics, and interactions with peptides, proteins, and medicinally-relevant small molecules is important. Current state-of-the-art simulations employ traditional supercomputers and study regions of the size of the small DMPC, because of time and resource constrains.

We ran the same DMPC simulations (with the same input files - i.e., coordinates, topology, and input parameters) on a cluster node using CHARMM [1] as a reference for assessing both FENZI accuracy and performance. CHARMM simulations were run on 1, 2, 4, and 8 cores of an Intel Xeon with speed 2.6GHz and 8GB of memory. We did not scale CHARMM beyond 8 cores of a single node because the cluster used was not provided with InfiniBand and thus the performance was very poor across nodes. For our GPU simulations, we used both a GTX 480 GPU (Fermi) with 480 cores and 1.5 GB memory and a C2050 GPU (Fermi) with 448 cores and 3 GB memory. The performance values presented are the average values computed over three repeated simulations. The standard deviation is not reported because it is close to zero.

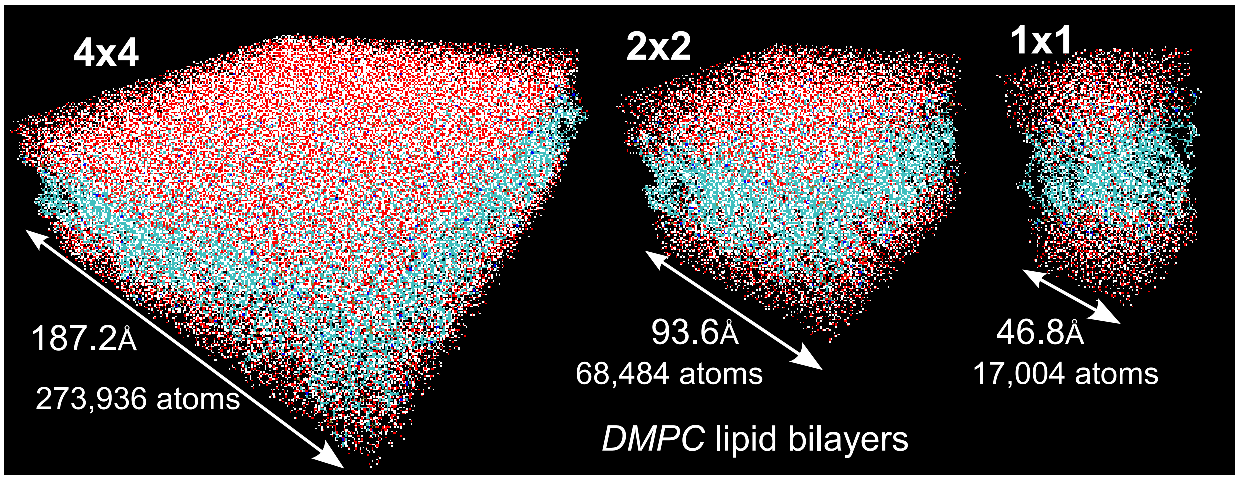

Figure 4. The three DMPC membranes considered in this paper - each membrane is four times larger than the previous membrane.

B. Performance

When addressing performance, scientists want to know how much simulated time (e.g., in nanoseconds or microseconds) can be performed in one day. Flops and MIPS have little meaning for the scientists. Thus, in this paper we measure FENZI performance and compare it against other codes in terms of nanoseconds per day.

First of all, we compared FENZI performance versus CHARMM performance, since CHARMM is the code we normally use for our membrane studies. For our comparison, we ran 10,000 steps of a constant energy MD simulation (NVE) with the smaller membrane (DMPC 1x1) using a r_{cutoff} of 8Å with a buffer cutoff r_{list} for the list updates of 9.5Å. Each step was 1 fs long. For the GPU platform, we considered both a GTX 480 and a C2050. Figure 5 shows the number of nanoseconds per day for the double-precision parallel CHARMM code optimized for 1, 2, 4, and 8 CPU cores versus the number of nanoseconds per day of FENZI on one single GPU. CHARMM uses MPI for distributing the membrane domain across cores and was optimized for the specific node. The figure shows how the speedup of one single-precision GPU is up to 10X faster than a 8-core, double-precision CPU node.

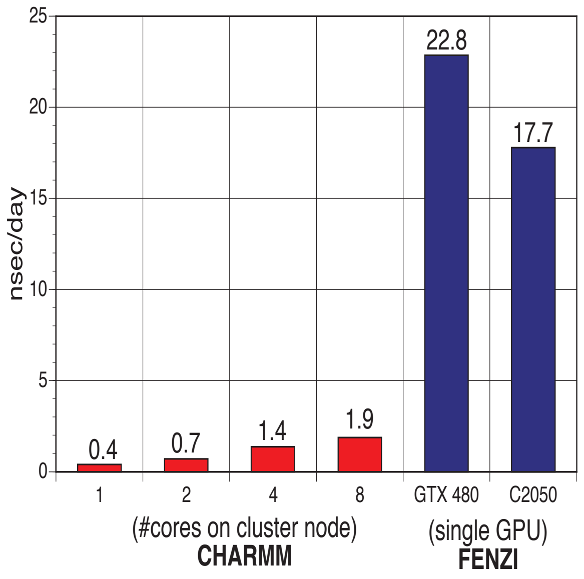

Figure 5. Comparison of performance in terms of ns/day for CHARMM on 1, 2, 4, and 8 CPU cores versus FENZI on GTX 480 (Fermi chip).

To study the scalability of our code, we compared its performance in terms of ns/day for the three lipid bilayer membranes (DMPC) systems with different sizes. Each DMPC membrane is four times larger than the previous. FENZI simulations were performed on the GTX 480 and C2050 GPU. Figure 6 shows how FENZI enables multi-nanosecond MD simulations per day with the considered membranes. More in particular, for the small membrane and the faster GTX 480 GPU, our MD code running on GPUs achieves simulation rates of up to 22.86 million MD steps with 3.78 ms per step (22.86 nanoseconds per day with a MD step size of 1 fs), up to 6.79 million MD steps per day with 12.7 ms per step (6.79 nanoseconds per day with a MD step size of 1 fs) for the medium membrane, and up to 1.63 million MD steps per day with 52.71 ms per step (1.63 nanoseconds per day with a MD step size of 1 fs) for the large membrane. The performance on C2050 GPUs is slightly slower but still very competitive compared with traditional and more expensive supercomputers.

Overall, FENZI allows us to simulate larger membranes, larger than simple regions, over a longer simulated time interval in a significantly shorter turnaround time. Work in progress includes the study of the membrane properties - i.e., structural (densities, electron density profiles), order pa-

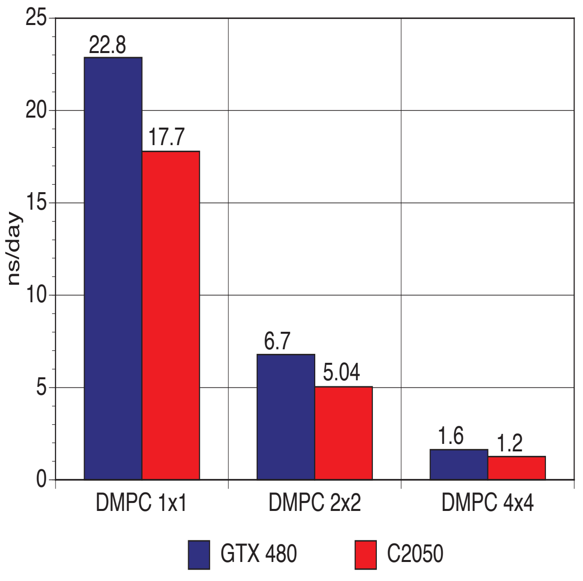

Figure 6. Simulations of three lipid bilayer membranes (DMPC) with three different sizes, each four times larger than the previous.

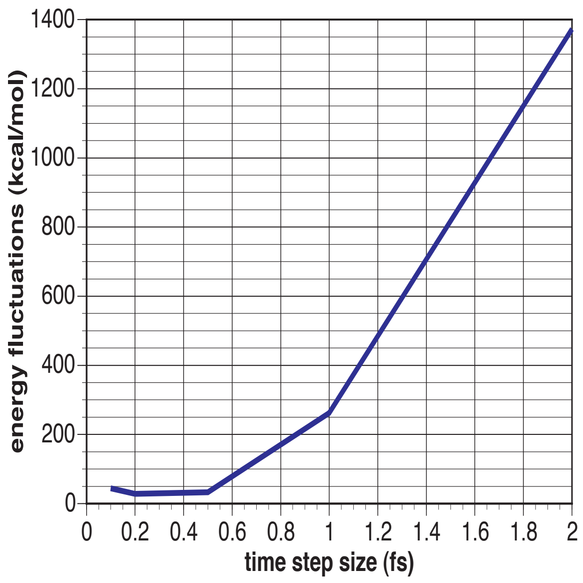

Figure 7. Plot of the total energy fluctuations of DMPC 1x1 as a function of time step size for different time step size - i.e., 0.1 fs, 0.20 fs, 0.5 fs, 1 fs, and 2 fs and single precision.

rameters (SCD) and electrostatic properties (dipole potential, water dipole moments), as well as orientational properties of water - over a time scale of tens of hundreds nanoseconds, to assess whether properties observed in small membranes with simulations on traditional CPU clusters are still true as the membranes significantly grow in size.

C. Accuracy

Clearly, even the fastest MD code has little value if the results it produces are not accurate. To ensure that our simulation results are accurate, we first ran the same 100 ps MD simulation for the smaller membrane system in equilibrium with different time step size - i.e., 0.1 fs, 0.20 fs, 0.5 fs, 1 fs, and 2 fs - and single precision. We plot the fluctuations in the total energy as a function of the time step size in Figure 7. According to Allen and Tildesley [4], a plot of the energy fluctuations versus time step size should follow an approximately logarithmic trend. In Figure 7 we observe that, for large time step size (larger than 0.5 fs), the fluctuations in total energy of MD simulations are proportional to the time step size. On the other hand, we observe a different behavior for step size less than 0.5 fs. This is consistent with results previously presented and discussed in [8].

Next, we ran simulations with the smaller membrane (DMPC 1x1) system in equilibrium on CPU using the double-precision CHARMM code and on GPU using our single precision code. The simulations are performed in constant temperature. As pointed out in [9], energy drift can be observed in long constant energy simulations and the drift can be easily overcome by maintaining constant temperature (e.g., using velocity reassignment which effectively changes the energy state of the system at specified intervals, thereby eliminating the drift for production simulations)

Figure 8 plots the temperature (a function of kinetic energy), bond energies (i.e., bond-, angle, Urey-Bradley-, improper-, and dihedral energies), and non-bond energies (i.e., VDW, electrostatic, and PME) over a 3 ns simulation time for the small membrane. The several quantities (i.e., temperature and energy) fluctuate, as expected, around the same average value for both the double precision CPU simulation and single precision GPU simulation. We observed that FENZI initial energy (step 0) matches the CHARMM initial energy. During the initial phase of the simulation, before the system reaches the equilibrium, we also observed some energy drifting due to the different thermostats used by the two codes (i.e., CHARMM uses Langevin thermostat while FENZI uses velocity reassignment). Eventually the two simulations converge, indicating that FENZI does in fact produce results consistent with CHARMM and the use of double precision is not needed on GPUs for this type of simulations. A detailed analysis of the energy profiles from the science point of view is work in progress.

IV. Related Work

Other scientists have been targeting the design and implementation of MD codes including the PME computation entirely on GPUs. One of the most relevant published work is ACEMD [7]. As outlined in Section II-C, FENZI is different from ACEMD in terms of how it deals with the charges and charge spreading for PME. Since a profiling of

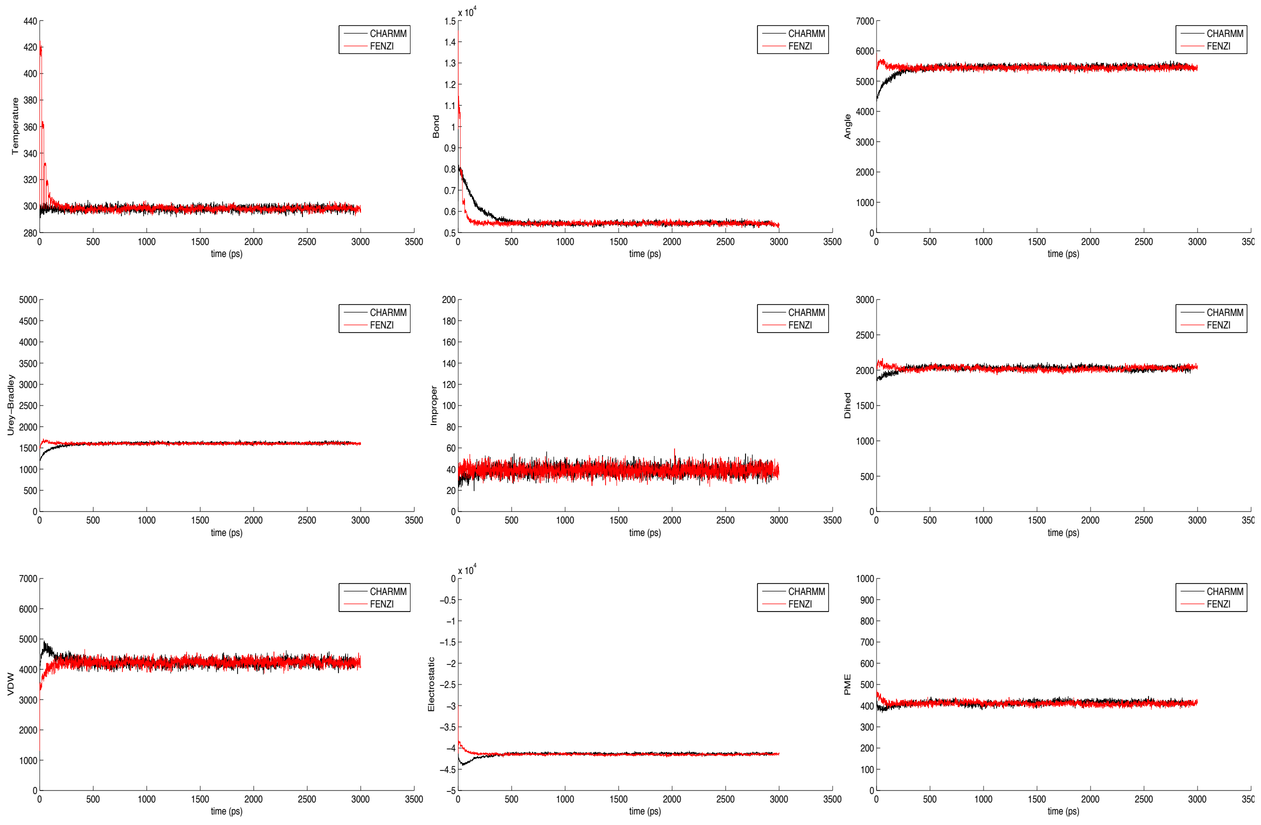

Figure 8. Comparison of the temperature and energy profiles for 3 ns of NVT MD simulation with 1fs step size for CHARMM on single core, 64 bits and FENZI on GTX 480, 32 bits.

the individual kernels for ACEMD is not feasible (the code is not open-source) and this information is not available (data in [7] is not readable for comparison), we cannot quantify the impact of our kernel performing the charge spreading versus the equivalent ACEMD kernel. As also outlined in [10], ACEMD has been designed and optimized for single GPUs and small molecular systems. ACEMD has been parallelized by distributing the kernels across three GPUs in a task-based parallelism. This approach can limit the size of a molecular system that can be ultimately simulated with ACEMD to approximately 100K atoms, due to global memory limitations. On the contrary, while designing FENZI as described in Section II, we have been using data structures and algorithms that can be easily parallelized in a data parallelism way. We are currently working on a parallel version of FENZI that uses domain decomposition to assure scalability of the simulation for very large molecular systems supported by multiple GPUs.

Walker and co-workers are also currently working on a version of AMBER for GPUs including PME. Preliminary performance results for the DHFR (23K atoms) and ApoA1 (92K atoms) benchmarks are presented in their webpage [11] but a manuscript reporting their algorithm is still work in progress.

GROMACS [6] also supports Particle-Mesh-Ewald (PME). As FENZI, GROMACS uses the mathematical models in [2] for the implementation of the PME method, however we were not able to find a manuscript presenting the algorithmic implementation of its PME code on GPU.

To compare FENZI performance with other codes performing PME entirely on GPU (i.e., ACEMD and AMBER), we ran the MD simulation of a well-known benchmark such as the DHFR system (23K atoms with 16,569 bonds, 11,584 angles, and 6,701 dihedrals) using initial parameters that are similar, when possible, to the parameters used by the other codes in their webpages, i.e., 8 Å nonbond cutoff, a buffer cutoff of 9.5 Å, 10,000 MD steps in an NVE ensemble, and with both 1 fs and 2 fs time step. Figure 9 shows the performance of DHFR MD simulations with FENZI on GTX 480 and C2050 (Fermi chip). Performance results on ACEMD and AMBER are presented in [12] and in [11] respectively. While comparing these results with our results, we observe that FENZI gives better performance than AMBER (i.e., according to [11], AMBER reaches 21.11 ns/day on a C2050 with a MD step of 2 fs) and

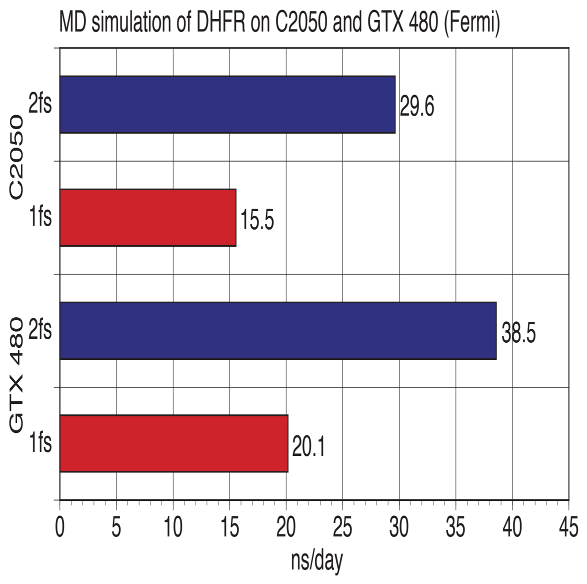

Figure 9. FENZI performance in ns/day of DHFR simulations using different time steps and GPUs.

similar performance as ACEMD (i.e., from our testing of ACEMD on GTX 480, the code reaches 37.24 ns/day with a MD step of 2 fs). Note that our MD algorithm is different than the ACEMD algorithm. As outlined above, we deal with the PME charge spreading in a significantly different way that ACEMD. Moreover, the nonbond interactions in FENZI still use exclusion and non-bond lists as in codes such as CHARMM, while ACEMD does no longer refer to these data structures in [7]. GROMACS performance in [13] indicates values below 20 ns/day for the DHFR system on a C2050 GPU, however the webpage does not include important information such as cutoff values and other parameter settings. Thus we cannot fairly compare these values with FENZI performance.

Coarse-grained MD simulations of large membranes for long time intervals (of the order of 100 ns) are also presented in [14]. The coarse-grained nature of these simulations results in significant loss of information and a lower resolution than our code. Up to a few microseconds of MD simulations of small membrane can be reached on special purpose architectures such as Anton [15]. Such performance is feasible on Anton because of multi-step integration, less frequent long distance force calculations, fixed precision calculations, and small FFT grid size (32x32x32). At the same time these factors lead to loss of resolution and accuracy. Moreover, the size of the hardware defines the size of molecular systems being simulated. The size of the molecular system has a significant impact on Anton's performance: the simulation of larger systems can lead to a loss in performance. The application specific nature of the architecture does not allow for flexibility in terms of FFT grid size, choice of crystal lattice (cubic dimensions), and spline functions. This level of flexibility is possible on general purpose GPUs.

Last but not least, we do not compare FENZI with the GPU-based NAMD [16] since the current version of NAMD for GPUs does not perform PME on GPUs, thus using different algorithm for the electrostatic interactions.

V. Conclusion

In this paper we presented the design and implementation of FENZI, a MD code for the simulation of large membrane regions in both NVT and NVE ensembles. A key contribution of the paper is the PME algorithm that we redesigned from a charge-centric to a lattice-of-points-centric prospective to better fit the multithreading GPU architecture. FENZI enables multi-nanosecond MD simulations per day, even when using PME for large membranes. FENZI achieves simulation rates of up to 22.8 nanoseconds per day with a MD step size of 1 fs for a small DMPC membrane of 17,004 atoms, up to 6.7 nanoseconds per day with a MD step size of 1 fs for a medium DMPC membrane of 68,484 atoms, and up to 1.6 nanoseconds per day with a MD step size of 1 fs for a large DMPC membrane of 273,936 atoms; all membranes were in explicit solvent. Because of our general design and implementation approach, FENZI is full-fledged general purpose MD package based on canonical force fields and PME for GPU enabled simulation of a broad class of molecular systems.

Work in progress includes the analysis of properties such as structural (densities, electron density profiles), order parameters (SCD) and electrostatic properties (dipole potential, water dipole moments), as well as orientational properties of water over a time scale of tens of hundreds nanoseconds. Our final goal is to assess whether properties observed in small membranes with simulations on traditional CPU clusters are still true as the membranes significantly grow in size. By speeding up the MD simulation of large membranes, FENZI allows us to answer this question accurately in a short turnaround time.

Acknowledgment

This work was supported by the National Science Foundation, grant #0941318 and grant #0922657, by the U.S. Army, grant #ARO 54723-CS, and by the NVIDIA University Professor Partnership Program. The authors want to thank Peng Wang (NVIDIA) and Shucai Xiao (UVA) for their feedback on optimization techniques in CUDA.

References

[1] B. Brooks, *et al.*,*CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations.* J. Comp. Chem., 4, 187-217, 1983.

[2] U. Essman, *et al.*, *A Smooth Particle Mesh Ewald Potential.* J. Chem. Phys.. 103, 8577-8592, 1995.

[3] Y. Wu, H. L. Tepper, and G. Voth, *A Flexible Simple Point Charge Water Model with Improved Liquid State Properties.* J. Chem. Phys., 124, 024503, 2006.

[4] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids.* Oxford: Clarendon Press, 1987.

[5] J. E. Stone, *et al.*, *Accelerating Molecular Modeling Applications with Graphics Processors*. J of Comp. Chemistry, 28(16), 26182640, 2007.

[6] D. van der Spoel, *et al.*, *GROMACS: Fast, Flexible, and Free*. J. Comput. Chem. 26, 1701-1718, 2005.

[7] M. Harvey, G. Giupponi and G. De Fabritiis, *ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale.* J. Chem. Theory and Comput. 5, 1632 - 1639, 2009.

[8] B. A. Bauer, J. E. Davis, M. Taufer, and S. Patel, *Molecular Dynamics Simulations of Aqueous Ions at the Liquid-Vapor Interface Accelerated Using Graphics Processors*, J. Comp. Chem., 32(3), 375385, 2011.

[9] M. S. Friedrichs, *et al.*, *Accelerating Molecular Dynamic Simulation on Graphics Processor Units*, J. Comput. Chem., 30, 864-872, 2009.

[10] J. E. Stone, D. J. Hardya, I. S. Ufimtsevb, K. Schulten *GPU-accelerated Molecular Modeling Coming of Age.* J. Molecular Graphics and Modeling, 29, 116-125, 2010.

[11] S. Le Grand, A. W. Goetz, D. Xu, D. Poole and R. C. Walker, *URL: http://ambermd.org/gpus/*, 2010.

[12] M. Harvey, G. Giupponi and G. De Fabritiis, *URL: http://www.acellera.com/acemd/performance*, 2010.

[13] GROMACS Performance on GPU, *URL: http://www.gromacs.org*, 2010.

[14] B. A. Hall and M. S. P. Sansom, *Coarse-Grained MD Simulations and Protein-Protein Interactions: The Cohesin-Dockerin System.* J. Chem. Theory Comput., 5(9), 2465-2471, 2009.

[15] D E. Shaw, *et al.*, *Millisecond-scale Molecular Dynamics Simulations on Anton.* In Proceedings of the ACM/IEEE Conference on Supercomputing (SC09), November 14-20, 2009.

[16] J. C. Phillips, *et al.*, *Scalable Molecular Dynamics with NAMD*. J. of Comp. Chemistry, 26, 1781-1802, 2005.