Simulations of Large Membrane Regions using GPU-enabled Computations - Preliminary Results

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

Sandeep Patel
Dept. of Chemistry and Biochemistry
University of Delaware
Email: patel@udel.edu

***Abstract*—In this short paper we present a GPU code for MD simulations of large membrane regions in the NVT and NVE ensembles with explicit solvent. We give an overview of the code and present preliminary performance results.**

***Keywords*-Algorithms; Performance; Ewald Summation; PME; Reaction Force Field Method**

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. When studying membrane-bound protein receptors, it is necessary to move beyond the current state-of-the-art simulations that only consider small regions (or patches) of physiological membranes since the heterogeneity of the membrane spans length scales much larger than included in these smaller model systems. Towards this end, our work proposes to apply large-scale GPU-enabled computations of extended phospholipid bilayer membranes, in this case dimyristoylphosphatidylcholine (PC) based lipid bilayers.

In this paper we present a GPU code for MD simulations that enables the fast simulations of whole membrane regions in the NVT and NVE ensembles. The code uses a modified version of the CHARMM force field (not the CHARMM program) [1] and includes different methods for the representation of the electrostatic interactions, i.e., reaction force field (RF) and Ewald summation methods. It reads standard files, e.g., pdb and psf files, as input files. The code implementation supports both global neighbor list structure and cell-based list structure calculations for the non-bonded interactions. The solvent in which the membrane is immersed is treated explicitly. This paper gives an overview of the code and preliminary performance results.

II. Code Description

Our MD code uses a modified version of the CHARMM force field in terms of force field functional forms and measurement units [1]. The intramolecular potential, which includes bonds, angles, and dihedral, is as follows:

$$V_{intra} = \frac{k_b}{2}\left[\left(r-r^0\right)^2+\left(r-r^0\right)^2\right]+ \frac{k_a}{2}\left(\theta_\angle-\theta^0_\angle\right)^2 \qquad (1)$$

where k_b is the bond force constant, r^0 is the equilibrium bond distance, k_a is the angle force constant, and $\theta^0_\angle$ is the equilibrium bond angle.

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

$$V_{LJ} = \sum_{i,j}^{pairs}\left(4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12}-\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]\right) \qquad (2)$$

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

Electrostatic interactions can be computed either by using the reaction force field (RF) or the Ewald summation method. Recent studies by Hummer *et al.* indicate the adequacy of reaction field approaches with diffuse charge representations for treating molecular and ionic systems, with structural properties in agreement with those obtained using lattice based methods [2]. Reaction field formulations have also been recently applied to studies of hydration and salt-bridge effects related to proteins [3]. We propose the RF method as a faster, still accurate, alternative to the Ewald summation method in membrane simulations. When using RF, electrostatic interactions are represented by a site-site based RF potential [4], [5] to represent the usual Coulomb interaction. Between two charged sites, i and j, the RF potential used for the present study is:

$$V_{ij}^{RF} = q_i\ q_j\ \left(\frac{1}{r_{ij}}+\frac{(\epsilon_{RF}-1)}{2\epsilon_{RF}+1}\frac{r_{ij}^2}{r_c^3}\right) \qquad (3)$$

where q_i is the charge on atom i, and likewise for site j. The value of r_c is the spherical cavity radius beyond which the RF formalism treats the environment as a continuum (whose perturbation by the charge distribution within the spherical cavity gives rise to the RF). This potential is shifted smoothly to zero at the cutoff and pairs separated by a distance greater than the cutoff are neglected.

In the Ewald summation method the electrostatic interactions are divided into the direct space (E_{dir}), the reciprocal space (E_{rec}), and the self energy (E_{self}) contributions to the total energy, depending on the distance of the interaction [6]. The three contributions are explicitly given by:

$$E_{dir} = \sum_{i=1}^{N-1} \sum_{j>i}^{N} \frac{q_i q_j \mathrm{erfc}(\beta r_{ij})}{r_{ij}} \tag{4}$$

$$E_{rec} = \frac{1}{2\pi V} \sum_{\vec{m}\neq 0} \frac{\exp(-\pi^2 \vec{m}^2/\beta^2)}{\vec{m}^2} S(\vec{m}) S(-\vec{m}) \tag{5}$$

$$E_{self} = -\frac{\beta}{\sqrt{\pi}} \sum_{i=1} N q_i^2 \tag{6}$$

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,

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

The solvent is treated explicitly. Water bond, angle, and charge parameters are transferred directly from the SPC/Fw model of Wu *et al.* [7];

III. Code Implementation on GPU

We use CUDA for our code implementation. Bond, angle, and dihedral interactions are each handled by separate kernels (i.e., `Dihed`, `Angle`, and `Bonds` respectively). This reduces the size of each kernel and minimizes the amount of data required for an individual kernel call. Bond, angle, and dihedral kernels evaluate the potentials by using a list approach in which each thread iterates through 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 beginning of the simulation and then copied to the GPU.

Non-bonded interactions (Lennard-Jones and electrostatic) are calculated by a single kernel (`NonBondForce`), when the Ewald summation method is not used. When a global neighbor list structure is used, each thread iterates through the neighbor list for a single atom i and accumulates the interactions between i and all its neighbor list entries in kernel `NBListBuild`. 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. Shifted force forms are used for the electrostatic and Lennard-Jones potentials to ensure that both energies and forces go smoothly to zero at the cutoff r_{cut}. The global neighbor list is constructed using the Verlet list approach [8], in which a list is constructed for each atom containing all atoms within a cutoff $r_{list} > r_{cut}$. This way, the list only needs to be updated when an atom has moved more than $\frac{1}{2}(r_{list} - r_{cut})$. To construct this list on the GPU, each thread checks the distance between an atom i and all other atoms, adding to i's neighbor list those atoms that are within r_{list} of i. This process is accelerated by having each block take advantage of the on-chip shared memory using a previously described tiling approach [9].

When a cell-based neighbor list structure is used, the list of all the neighboring atoms within the cutoff r_{cut} is build in a similar fashion in kernel `NBListBuild`, but instead of searching the global list of all the atoms in the system, only atoms in the neighboring cells are searched. The entire domain of atoms is decomposed into regular cubic cells of edge length equal to r_{cut}. Therefore 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.

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 [6].

Of these five steps, the charge spreading is one of the most compute intensive. In [10], 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 [10] 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. Each lattice point is managed by a single thread which iterates only through a local list of charges, exclusive to that point, rather than the global list. The charge spreading is performed in the `ChargeSpread` kernel and the updating is done in the `LatticeUpdate`. 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`.

IV. Performance

Performance was measured on a commonly used membrane benchmark, the lipid bilayer membranes (DMPC) with two different sizes, one four times larger than the other with 93.6 X 93.6 X 152.0 Å, 68484 atoms and 46.8 X 46.8 X 76.0 Å, 17004 atoms respectively. 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.

To measure the performance, we ran 10,000 steps of a NVE MD simulation (constant energy) using a r_{cutoff} of 8Å with a buffer cutoff r_{list} for the list updates of 9.5Å. We studied the code scalability and we profiled the different force-fields methods and data structures with the two DMPC systems. Note that one has exactly the same conformation but is four times larger than the other. More in particular, we measured average, max, and, min time per MD step in milliseconds when the step does and does not include a list update as well as the ratio of steps including a list update over the total number of steps (ratio). Finally, we computed the overall average time per step over the whole MD simulation of 10,000 steps. Table I presents the results for the smaller DMPC membrane (17004 atoms) when using a simple global neighbor list structure for the non-bond interactions with RF (I), a cell-based list structure with RF (II), a global neighbor list structure with PME (III), and a cell-based list structure with PME (IV). Table II presents the same results but for the larger membrane system (68 484 atoms).

Table I
PERFORMANCE SPECIFICATIONS OF DMPC WITH 17K WITH NEIGHBOR LIST AND RF (I), WITH CELL-BASED LIST AND RF (II), WITH NEIGHBOR LIST AND PME (III), AND WITH CELL-BASED NEIGHBOR LIST AND PME (IV).

	Steps w/o Update (msec/step)			Steps w Update (msec/step)				Tot.
	avg	max	min	avg	max	min	ratio	avg
I	7.5	7.7	7.4	78.8	80.4	77.8	6.3	12.0
II	8.2	8.4	8.2	82.0	85.9	76.8	6.3	12.9
III	15.7	16.3	15.4	90.0	90.9	88.9	5.3	19.6
IV	15.9	16.5	15.6	103.7	108.7	98.9	5.3	20.5

From the data structure point of view, we observe that for small systems, smaller than 20K, a global neighbor list implementation is more efficient that a cell-based list implementation since the time to update the non-bond list is smaller (see time for `NBListBuild`). For large molecular systems such as the large membrane of 67K atoms, a cell-based list structure is more efficient. When considering the small membrane, our code currently enables simulations of 7.2ns per day with RF and 4.4ns per day with PME with 1fs step size. When considering the large membrane, our code enables simulations of 1.6ns per day with RF and 1.1ns per day with PME. Again a 1fs step size is used.

Table II
PERFORMANCE SPECIFICATIONS OF DMPC WITH 68K WITH NEIGHBOR LIST AND RF (I), WITH CELL-BASED LIST AND RF (II), WITH NEIGHBOR LIST AND PME (III), AND WITH CELL-BASED NEIGHBOR LIST AND PME (IV).

	Steps w/o Update (msec/step)			Steps w Update (msec/step)				Tot.
	avg	max	min	avg	max	min	ratio	avg
I	30.5	34.6	29.7	739.1	756.7	719.2	6.6	77.4
II	35.0	39.5	34.1	278.3	320.5	265.0	6.7	51.2
III	62.9	78.4	60.8	790.4	820.6	779.2	5.6	103.9
IV	64.8	82.4	61.8	292.7	365.2	251.1	5.6	77.7

V. Conclusion

In this paper we present a flexible MD code for GPUs integrating both Ewald summation and reaction force field methods. The code supports explicit solvent representations and enables fast simulation of large membrane regions. Work in progress includes the optimization of the GPU code and the analysis of the membrane simulations.

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.

References

[1] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus, *J. Comp. Chem.*, 4, 187–217, 1983.

[2] G. Hummer, D. Soumpasis, and M. Neumann, *J. Phys.: Condens. Matter*, 6, A141–A144, 1994.

[3] T. Ghosh, S. Garde, and A. Garcia, *Biophys. J.*, 85, 3187–3193, 2003.

[4] M. Neumann, *J. Chem. Phys.*, 82, 5663–5672, 1985.

[5] J. A. Barker and R. O. Watts, *Mol. Phys.*, 23, 789, 1973.

[6] U. Essmann, L. Perera, and M. Berkowitz, *J. Chem. Phys.*, 103(19), 8577–8593, 1995.

[7] Y. Wu, H. L. Tepper, and G. Voth, *J. Chem. Phys.*, 124, 024503, 2006.

[8] M. P. Allen and D. J. Tildesley. Oxford: Clarendon Press, 1987.

[9] L. Nyland, M. Harris, and J. Prins, in *GPU Gems 3*. Addison-Wesley, 31, 677–695, 2008.

[10] M. Harvey, G. Giupponi, and G. De Fabritiis, *J. Chem. Theory Comput.*, 5, 16321639, 2009.