Proceedings of the 2012 IEEE 12th International Conference on Bioinformatics
& Bioengineering (BIBE), Larnaca, Cyprus, 11-13 November 2012

GPU-SD and DPD Parallelization for Gromacs tools for molecular dynamics simulations

Nicolae Goga[1,2], Siewert Marrink
[1]*Molecular Dynamics Group*
University of Groningen
Groningen, The Netherlands
n.goga, s.j.marrink@rug.nl

Ruxandra Cioromela, Florica Moldoveanu
[2]*Politehnica University of Bucharest*
Bucharest, Romania
Ruxandra.cioromela12@gmail.com, fm@cs.pub.ro

***Abstract*—This article presents the GPU parallelization of new algorithms SD and DPD types for molecular dynamics systems developed by the Molecular Dynamics Group, University of Groningen, the Netherlands. One should note that molecular dynamics simulations are time-consuming simulations of systems, running time ranging from days to weeks and months. Therefore parallelization is a key issue for the well-running and use of MD software. The paper presents the main ingredients of GPU parallelization of the new algorithms and simulation results. It can be concluded that the parallelization through the use of graphical cards improves the performances of the runs as compared to the serial version of the code.**

Keywords-molecular dynamic systems, graphic cards systems, CUDA programing

I. Introduction

Molecular simulation is a useful tool for studies of chemical and biomolecular systems, ranging from simple liquids to proteins and DNA in realistic solvent environments. Such simulations allow functional observation of proteins, nucleic acids, membranes and other building blocks. Molecular Dynamics uses Newtonian equations of motions for describing atomistic system movements. For preserving the temperature of the particle systems, stochastic dynamics (SD), and, more recently, dissipative particle dynamics (DPD) integrators are used in the simulations.

This article presents the GPU parallelization of new algorithms, SD and DPD types, developed by the Molecular Dynamics Group, University of Groningen, the Netherlands. One should note that molecular dynamics simulations are time-consuming simulations of systems, running time ranging from days to weeks and months. Therefore parallelization is a key issue for the well-running and use of MD software. A recent trend is to use the power of graphical cards for the parallelization of MD code. As choice of implementation was the tool Gromacs [3, 8] started in the MD group of Groningen - an open-source tool used world-wide by academia and industry.

The new algorithms were developed inside the OpenMM library [9], provided by Stanford University, that Gromacs interfaces in order to be able to run simulations also on GPU. The OpenMM library has support both for CUDA architecture provided by NVIDIA and OpenCL, the open standard defined by Khronos Group. It has quite a wide range of algorithms implemented that run on GPU, but still it’s not a valid equivalent of Gromacs: its functionality is lower as compared to Gromacs. This is explained by the fact that OpenMM has a recent development history while Gromacs has a much longer development history.

This paper is organized as fellow. The next section outlines the new SD and DPD theory while Section III presents briefly Gromacs and OpenMM engines of molecular simulations. Section IV deals with the design and implementation and Section V discusses the performance issues. At the end, we draw the conclusions and some future directions of development in Section VI.

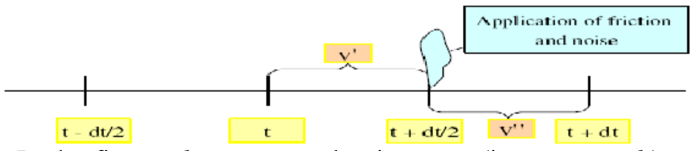
II. SD and DPD New Algorithms

The movement of the particles in the traditional case of stochastic dynamics for molecular systems starts from the following basic equation:

$$m_i \dot{v}_i = F_i - m_i Y_i v_i + R(t)$$

where m_i is the mass of the particle, v_i' is the acceleration, F_i is the force due to Newtonian particle interactions, $m_i\, Y_i\, v_i$ is the friction force – that depends on a friction coeficient Y_i and is directly proportional with the velocity of the particle v_i and *R(t)* is a random term computed in such a way to maintain the desired temperature of the system. In the DPD case the friction and the random terms are considered between pairs of two particles.

The new developed algorithms take a new approach that reduces the complexity of computations, in the sense of applying the friction and the random terms at the levels of velocities in place of forces. To get a better idea let's look to the following figure.



In the figure, *dt* represents the time step (in *picoseconds*) at which the simulation is performed. Every computational step, the velocities are computed at the half of the *dt* range and the positions are computed at the extremes of the range. The application of friction and noise happens at the *dt* half-range. The computation for the new SD algorithm proceed as follows:

1. ***Update velocity due to Newtonian force:***
   $v' = v(t+dt/2) + F/m * dt$
2. ***Apply friction and noise***

$$v'' = fv' + \sqrt{(1 - f^2)(k_B T_{\mathrm{ref}}/m)}\,\xi$$

where f is a friction coefficient (constant given by user) that fixes the coupling strength, K_b is Boltzman constant, T_{ref} is the desired temperature, and ξ is white noise (random term).

3. ***Update positions and velocities***
   $x(t+dt) = x(t) + (v'+v'')/2$
   $v(t+dt/2) = v''$

The algorithm described above is referred further in the paper as *SDNew*. For DPD, the difference is in the fact that the friction and random terms are applied between pairs of particles to the velocity difference between the two particles. There are three DPD variants that will be referred further in the paper as follows: 1) *ISO:* when the friction is applied between the velocity difference of two particles; 2) *DPDPar:* when the friction is applied to the velocity difference of two particles projected on the line that unites the coordinates of the two particles; 3) *DPDPerp:* when the friction is applied to the component of the velocity difference of two particles perpendicular on the line that unites the coordinates of the two particles.
After introducing the new algorithms, in the next section we will shortly present Gromacs and OpenMM tools.

III. Gromacs and OpenMM

GROMACS (GROningen MAchine for Chemical Simulation) [3] is an engine to perform molecular dynamics simulations and energy minimization. It was first developed by the Department of Biophysical Chemistry of the University of Groningen and is still under ongoing development. Also, GROMACS adoption in the *Folding@Home* project [4] is the result of the continuous development and wide support of this software.

The package itself is composed not only from an MD simulator, but also has a variety of tools for analyzing and visualizing the output of the simulations [5]. Its functionality is enabled by many lines of code and is dependent on the mathematical models implemented in it. It also employs a multitude of scientific algorithms and several dozen functions (called "non-bonded kernels") for the short-range non-bonded interactions, each offering a different combination of methods for electrostatic and van der Waals forces.

OpenMM library [9] is developed by a team from Stanford University, that Gromacs interfaces in order to be able to run simulations also on GPU. The OpenMM library has support both for CUDA architecture provided by NVIDIA and OpenCL, the open standard defined by Khronos Group. It has a quite wide range of algorithms implemented that run on GPU, but still it's not a valid equivalent of Gromacs: its functionality is lower as compared to Gromacs. This is explained by the fact that OpenMM has a recent development history while Gromacs has a much longer development history. In the combination of Gromacs/OpenMM our new algorithms were developed.

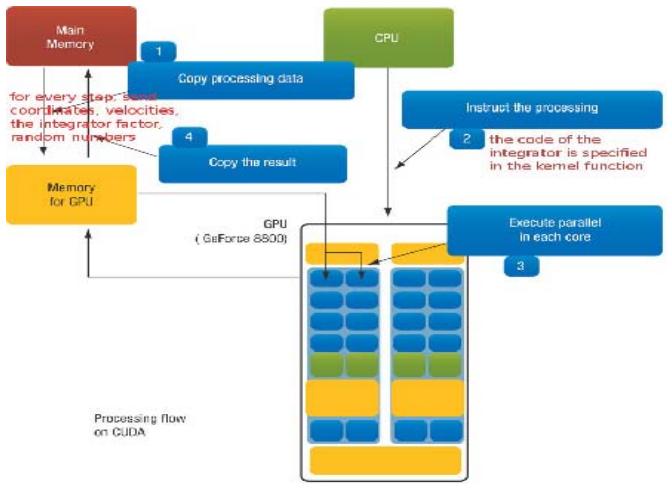
IV. Algorithms implementation

The way of executing the code for the 4 integrators is based on the execution of CUDA code. The main computational flow is represented in Figure 1 while in Figure 2 we represent the GPU parallelization information flow.

First the integrator object is created. After creating it, the afferent function will be executed. The functions that use the integrators are written in plain C, but they call methods and functions that use CUDA, implicitly the kernel functions. Once a kernel is launched, it is executed on GPU.

```
I. For n simulations steps do
    I.1 Compute forces and energies due to particle interactions in GPU fashion
    I.2 Update velocities and positions and velocities in GPU fashion (detailed bellow)
        I.2.a load on gpu data;
        I.2.b instantiate a number of threads with the kernel code for the integrator
        I.2. c. load in the thread the data needed from the main memory
            (velocities, positions, factor, random numbers)
        I.2.d. For each particle do specific computations according to SD/DPD formulas
end
```

In consequence thousands of threads are created, prepared to execute the same piece of code that resides in the kernel. Once the kernel is prepared to be executed, data from structures that are called inside the kernel code are brought in the memory of the GPU and also the relevant data, in the memory of each multiple processors.


Main Memory
CPU
1
Copy processing data
for every step: send coordinates, velocities, the integrator factor, random numbers
Instruct the processing
2
the code of the integrator is specified in the kernel function
4
Copy the result
Memory for GPU
GPU (GeForce 8800)
Execute parallel in each core
3
Processing flow on CUDA

This way, the threads have direct access to the data relevant to them. Once all the threads are finished and the kernel code has been executed, the results are sent to the processor and RAM, and the execution is continued on the CPU.

In order to use this integrator, the friction factor must be specified in the *.mdp file, also the reference temperature. In the openmm_wrapper.cpp file of Gromacs, when this integrator is called, a new object from the SDNew Integrator is instantiated. This way, the specific methods and kernels are called. The main algorithm is located in the kSDNewUpdate.h. The kernel methods are called in the file CudaKernels.cpp, where the SDNew integrator is created and used.

V.Results

The integrators were tested on a system of coarse grained water, parameterized with Martini [2]. The number of particles varied from small systems to bigger systems. All the systems were tested on a quadcore machine with processor type Intel (R) Core (TM) i7 with 2.67 Mhz frequency and 1.5 Gb internal memory per core and a NVIDIA GEForce 9600 GT graphical card with 512 Mb memory and 64 GPU processors.

Integrator	**Water (3200 particles)**	**Water (6400 particles)**	**Water(102400 particles)**
SDNew	**1 processor:** 259.01 ns/day **2 processors:** 576.05 ns/day **4 processors:** 1080.1 ns/day **NVIDIA:** 425.33 ns/day **Temperature:** 314.97 K	**1 processor:** 117.851 ns/day **2 processors:** 246.88 ns/day **NVIDIA:** 184.898 ns/day **Temperature:** 315.37 K	**1 processor:** 7.606 ns/day **2 processors:** **16.318 ns/day** **NVIDIA:** 7.877 ns/day **Temperature:** 314.916
ISO	**1 processor:** 238.16 ns/day **2 processors:** 520.34 ns/day **NVIDIA:** 410.75 ns/day **Temperature:** 314.31 K	**1 processor:** 108.91 ns/day **2 processors:** 220.68 ns/day **NVIDIA:** 181.122 ns/day **Temperature:** 314.88 K	**1 processor:** 6.57 ns/day **2 processors:** 14.13 ns/day **NVIDIA:** 7.778 ns/day **Temperature:** 314.889 K
DPDPar	**1 processor:** 234.27 ns/day **2 processors:** 516.55 ns/day **NVIDIA:** 409.682 ns/day **Temperature:** 314.332 K	**1 processor:** 103.67 ns/day **2 processors:** 215.85 ns/day **NVIDIA:** 181.484 ns/day **Temperature:** 314.679 K	**1 processor:** 5.87 ns/day **2 processors:** 12.24 ns/day **NVIDIA:** 7.772 ns/day **Temperature:** 314.646 K
DPDPer	**1 processor:** 224.15 ns/day **2 processors:** 527.67 ns/day**NVIDIA:** 408.804 ns/day **Temperature:**	**1 processor:** 101.13 ns/day **2 processors:** 213.17 ns/day **NVIDIA:** 181.006 ns/day **Temperature:**	**1 processor:** 5.34 ns/day **2 processors:** 11.45 ns/day **NVIDIA:** 7.790 ns/day **Temperature:**

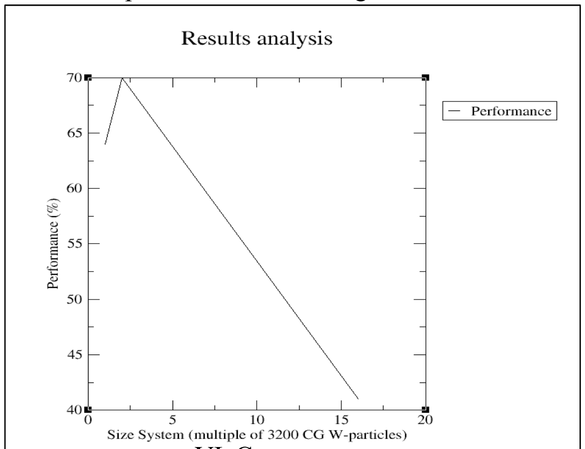
For performing the simulations, the integrators with the names "SDNew", "ISO", "DPDPar", "DPDPer" should be specified in the *.mdp file (a file containing run parameters for a simulation in Gromacs), at the integrator section. The friction factor can take different values and the temperature is also specified in this file.

The number of steps must be described in the input file. For our system the reference temperature was *315 K*.

Above, there is the table with the measures and performances on 1, 2, 4 processors and on the NVIDIA graphical board for the same algorithm, with implementations according to the hardware architecture. We did the measurements as an average of 8 runs. In table we put the mean. The standard deviation was bellow 5%.

Analyzing the results, it can be noticed that the temperature mainly converges to the reference temperature for all integrators. Also, it can be observed that the performance obtained on the GPU is greater than on a single processor, but not equivalent with two processors. Generally, the performances on the graphical board should be more increased but due to the current implementation in OpenMM, this thing is not possible (we performed similar simulations on the same molecular systems with OpenMM/Gromacs without our special algorithms and the trends are similar).

Comparing the ns/day obtained for one processor, for two processors and on NVIDIA, it can be noticed that the performance on NVIDIA is increased with almost 70% comparing to one processor. But this happens on systems, with less than 80000 particles. For larger systems the performances on NVIDIA card start to be comparable to just one processor. Because of the reduced memory of the GPU, a big number of particles cannot be loaded on the board. Also it means that the interchange of data between the GPU shared memory of each microprocessor and the main slow memory of the GPU occurs very often, that increases the latency. For larger systems their is need for more data transfers between the GPU memory and the ram memory, fact that explains the decrease in the performance of the algorithm.



VI. Conclussions

In this paper we describe the GPU parallelization of novel SD and DPD algorithms of molecular systems simulations. The new algorithms were developed by the Molecular Dynamics Group of the University of Groningen.

The algorithms were parallelized on the Gromacs/OpenMM software for molecular dynamics.

The performances of the algorithm have been compared with the performances of the code on one, two and four processors. It can be observed that the performance obtained on the GPU is greater than on a single processor, but not equivalent with two processors. Comparing the *ns/day* obtained for one processor, for two processors and on NVIDIA, it can be noticed that the performance on NVIDIA is increased with almost 70% comparing to one processor. This happens on systems with less than 80000 particles. For larger systems the performances on NVIDIA card start to be comparable to just one processor. Similar conclusions for larger atomistic systems were drawn for other MD algorithms on GPU in literature (see for example [1]). It can be concluded that the parallelization through the use of graphical cards improves the performances of the runs as compared to the serial version of the code. More improvements are possible for the case of larger atomistic systems. Second, although the GPU performances were slower than running on two cores, the described work opens the way for getting the new algorithms on GPU, which for other molecular systems is expected to give better results as compared with more cores runs.

As future work, the new code, representing the implementation of the integrators, will become a plug-in that the developing research group of OpenMM from Stanford wants to integrate into the library, in order to be used world wide.

REFERENCES

[1] S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. LeGrand, A. L. Beberg, D. L. Ensign, C. M. Bruns, V. S. Pande. Accelerating Molecular Dynamic Simulation on Graphics Processing Units.” *J. Comp. Chem.*, 30(6):864-872 (2009)

[2] Siewert J. Marrink,՚H. Jelger Risselada,Serge Yefimov,D. Peter Tieleman, and and ,Alex H. de Vries, The MARTINI Force Field:□ Coarse Grained Model for Biomolecular Simulations, *The Journal of Physical Chemistry B* **2007** *111* (27), 7812-7824.

[3] D. van der Spoel, E. Lindhal, B. Hess and et. al, Gromacs: fast, flexible and free. J. Comput. Chem., 2005.

[4] Folding@home. http://folding.stanford.edu.

[5] D. van der Spoel, E. Lindhal, B. Hess, and et. al, Gromacs user manual version 3.3. [online], 2006.

[6] B. Hess, C. Kutzner, D. van der Spoel, and et. Al, Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. J. Chem. Theory Comput., 2008.

[7] K. J. Bowers, R. O. Dror, and D. E. Shs aw. Zonal methods fo the parallel execution of range-limited N-body simulations. J. Comput. Phys., 2007.

[8] Gromacs, www.gromacs.org

[9] OpenMM, https://simtk.org/home/openmm