CHARACTERIZATION OF COARSE GRAIN MOLECULAR DYNAMIC SIMULATION PERFORMANCE ON GRAPHIC PROCESSING UNIT ARCHITECTURES

Ardita Shkurti[1], Andrea Acquaviva[1], Elisa Ficarra[1], Mario Orsi[2] and Enrico Macii[1]
[1]*Corso Duca degli Abruzzi 24, Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy*
[2]*School of Chemistry, University of Southampton, SO17 1BJ, Southampton, U.K.*

Keywords: CUDA, GPU acceleration, Molecular dynamics, Coarse grain, Membrane modeling, Lipid modeling.

Abstract: *Coarse grain* (CG) molecular models have been proposed to simulate complex systems with lower computational overhead and longer timescales with respect to atomistic level timescales. However, their acceleration on parallel architectures such as *Graphic Processing Units* (GPU) presents original challenges that must be carefully evaluated. The objective of this work is to characterize the impact of CG model features on parallel simulation performance. To achieve this target, we implemented a GPU-accelerated version of a CG biomembrane simulator called BRAHMS, to which we apply specific optimizations for CG models, such as dedicated data structures to handle different bead type interactions. Moreover, we explore different GPU architectures to characterize the behavior of the optimized CG model.

1 INTRODUCTION

Most aspects of cell activity such as the conformation of embedded proteins, the biomembrane permeability, interaction with drugs and signaling may be perceived observing the dynamics of cell membranes which are composed of patterns of lipid bilayers (Orsi, 2010).

The value of such biological processes brings to an increasing appeal for hardware and software optimized realistic models which may provide *in silico access* to biological cell mechanisms otherwise impossible to achieve through real experiments.

Atomic level (AL) models introduced (Wohlert, 2006; MacCallum, 2006) require huge quantities of computational resources due to the need of interaction calculations among system atoms.

CG modeling techniques have been proposed for handling shortcomings of atomic level models. In the CG representation specific atom groups are organized in clusters and each of the clusters is considered as a unique system particle called a *bead*. Hence, by diminishing the system particles number, the computational requirements are reduced as well. Figure 1 illustrates an example of AL and CG modeling for lipid bilayers. Typically, a lipid of about 100 particles in AL approaches is modeled by approximately 10 beads in CG representations (Orsi, 2010; Marrink, 2007). While CG models are promising and may allow simulations at longer time scale, their acceleration must be carefully characterized. In this work, we accomplish this characterization dealing with a CG lipid bilayer simulator.

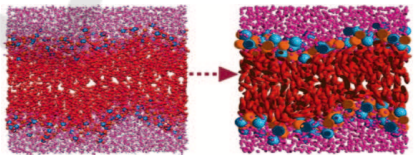

Figure 1: Coarse-grain mapping of DMPC lipids from atomistic representation.

While several papers have addressed atomistic model acceleration on GPUs (Anderson, 2008; Shkurti, 2010; Stone, 2010; van Meel, 2008; Rapaport, 2011; Bauer, 2011), the specific features of CG models have only recently gained attention (Shkurti, 2010; Zhmurov, 2010).

In this paper, we characterize specific bottlenecks of CG models proposing specific strategies for handling them, through the implementation of an accelerated CG model. We identify the impact of characteristics and data structures on the optimization of the CG application. Moreover, we explore the speed-up achieved across various architectures and quantify the overhead of the additional structures required by the CG nature simulation methodology to take into account interaction among heterogeneous bead types.

Shkurti A., Acquaviva A., Ficarra E., Orsi M. and Macii E..
CHARACTERIZATION OF COARSE GRAIN MOLECULAR DYNAMIC SIMULATION PERFORMANCE ON GRAPHIC PROCESSING UNIT ARCHITECTURES.
DOI: 10.5220/0003790703390342
In *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms* (BIOINFORMATICS-2012), pages 339-342
ISBN: 978-989-8425-90-4

The rest of this paper is organized as follows. Section 2 provides background on CUDA programming and the simulated model. Section 3 covers the optimization methods, while Sections 4 and 5 present and discuss the achieved results and conclude the work.

2 BACKGROUND

2.1 CUDA Environment

The CUDA environment represents a parallel programming model oriented on highly parallel computing, while making use of a large number of computational cores. With respect of general purpose CPUs, much more transistors are assigned to data processing rather than to flow control or data caching. However, memory latency is hidden with compute-intensive calculations. The GPU is divided into a set of Streaming Multiprocessors (SM) in which hundreds of *threads* reside concurrently. A large number of CUDA threads, can be launched in parallel by means of *kernels*, which are C functions executed in parallel from all CUDA threads launched (CUDA, 2011).

2.2 Coarse Grain Simulator

BRAHMS (Biomembrane Reduced ApproacH Molecular Simulator) is a CG lipid bilayer simulator we optimized and accelerated for CUDA environment. It describes a method for CG modeling of lipid bilayers of biomembranes and contains several original features such as specific original representations for water and charges (Orsi, 2010). It simulates the displacement in time of the particles of a system, calculating some of its macro properties such as temperature, lateral pressure, potential and kinetic energy (Orsi, 2010). It employs Molecular Dynamics and in particular Newton Leapfrog Equations to move the positions of beads, which in CG models represent clusters of atoms, and their velocities one time step forward (Rapaport, 2004).

3 GPU OPTIMIZATION OF COARSE GRAIN MODELS

Code profiling of the CG simulator has been performed to identify the most computation demanding parts of the code which resulted to be: *(i)* Non-bonded forces computation; *(ii)* Integration timestep; *(iii)* Neighbor structure generation, accounting for about 95.2%, 2% and 1% of the total execution time of the sequential simulator, respectively.

Accordingly to the bottlenecks identified, we have implemented three *kernels* to be executed on the device (GPU), one for each of the most onerous parts of code. Concerning data structures, a comprehensive one is required containing for each bead, the information on its position, velocity, force, torque, orientation, angular momentum in the body-fixed frame, rotation matrix, bead type, mechanical type, lipid identifier and lipid unit identifier.

The integration algorithm exploits Leapfrog equations (Rapaport, 2004). This algorithm is implemented by the *integration* kernel on the GPU, to avoid additional data transfers between host and device.

For the calculation of the non-bonded *forces*, a neighbor structure is needed to avoid considering a contribution for each pair of beads, which would lead to a *quadratic* time complexity. Therefore the system is divided into cells of beads, where each cell has a cubic shape and the edge size equal to the maximum cut-off distance among cut-offs established for different bead types. Only bead pairs with a distance value under the cut-off are considered as neighbors. Hence, the search for neighbors is applied to bead pairs of the same cell or the eventual 26 adjacent cells.

In the CPU version, the cell, neighbor and interaction type structures cannot be efficiently mapped on a GPU. Therefore we have created new structures for the cells, neighbor beads and relative interaction type storage, which provide coalesced[1] accesses for CUDA threads.

Implementation and update of neighbor and interaction type structures within the GPU is needed to obtain maximum performance from GPU by avoiding data transfer overheads between CPU and GPU. To this purpose we have implemented a kernel called *cudaneigh*, in which we have used an algorithm similar to the algorithm proposed in (Anderson, 2008) for AL models, where the texture type used was one-dimensional. Since the texture cache is optimized for $2D$ spatial locality (CUDA, 2011), we have used two dimensional textures in our code even for the other two kernels and employ them when the accesses in device memory are not coalesced.

The non-bonded forces computation is implemented by the kernel *forces*, that exploits the optimized access to neighbor and interaction type structures. Compared to what has been done for AL models in (Anderson, 2008), we use an additional interaction type structure with the same size and organization

[1]When different threads need some particular piece of information, they find it in the same relative address, calculated as $base\ address + absolute\ thread\ identifier$.

as the neighbor structure and we use two dimensional textures for random accesses in memory, to retrieve the needed information.

In addition, we created separate structures for all beads positions, all beads velocities, all beads orientations and all beads types, as these data are the most frequently accessed by the GPU. In this way, we could enable it to access the information contained in our new structures in a *coalesced* way.

4 RESULTS

In this Section we present results about the speed-up of the GPU version of BRAHMS with respect to the CPU, considering: *(i)* Biological systems of different dimensions and complexity (i.e. lipid, water); *(ii)* Different versions of BRAHMS GPU code (i.e. customized or not for water systems simulation); *(iii)* Three different GPU architectures: GeForce GTX295, GeForce GTX480 and Tesla C2050 that have respectively 240, 448 and 480 cores and an Intel® Core™i7-920 Processor CPU architecture.

In the charts presented, we consider systems having from 840 to 218904 number of beads[2]. As for complexity, we consider systems composed by heterogeneous types of beads: Either water systems uniquely containing water molecules or lipid systems containing lipid molecules in water solution. We develop the GPU *water only version* customized to water systems, thus treating all the beads as a unique type (water beads), to evaluate the impact of additional structures used for handling interactions among the lipids. The GPU *complete version* handles also the beads of lipid molecules. In Figure 2, we report

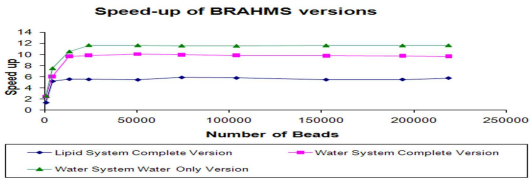

Figure 2: Comparison among the speed-up of different BRAHMS versions with respect to the CPU achieved on the GTX295 architecture.

the comparison among the speed-ups (related to the CPU version) of two biological systems, lipid and water only, characterized by different complexity simulated on the *GTX295* architecture. For water systems, we distinguish between water only and complete version. Single precision floating point arithmetic is used for these simulations. For a number of beads lower than $23K$ the speed-up is lower than the maximum achievable because the parallelism of the GPU is not completely exploited. The *water only version* is the fastest version with speed-ups up to *12x*, due to the absence of structures to handle different bead types. On the other hand, the lower performance of the complete version in the case of lipid systems with respect to the case of water systems is due to the fact that for lipid systems there are six different types of beads involved instead of the single type considered in water systems. This leads to divergent execution flows among the threads, thus leading to a performance hit.

From Figure 2, we can also observe the effect of lipid data structures, highlighted by the gap between the water-only and complete version for water systems. The maximum difference in the total speed-up between the water-only and complete version for water systems is 36.4% when single precision arithmetic is used for floating point operations and 15.71% when double precision arithmetic is employed.

The larger impact of lipid structures for single precision arithmetic computation is due to the constant overhead for memory accesses and additional control statements associated to beads types and interactions, which has a larger relative contribution to a shorter execution (i.e. the single precision arithmetic simulation).

In Figures 3 and 4 we report respectively for lipid systems and water systems, the speed-ups[3] achieved by the three considered architectures for an increasing system size. The double precision arithmetic has been used for floating point operations.

For lipid systems we achieve speed-ups up to $6.35x$, $5.82x$ and $2.24x$ for respectively GTX480, Tesla C2050 and GTX295 architectures, while for water systems the respective speed-ups achieved are $12,83x$, $12.05x$ and $4.94x$.

The GTX295 architecture is the slowest architecture, as we expected, due to its lower number of cores and hence of parallelism. GTX480 has a higher parallelism with respect to Tesla C2050, having one additional SM (32 additional cores), which causes the better performance of GTX480 versus Tesla C2050.

When evaluating speed-up results, we have to take into account that: *(i)* The CPU used for comparison represents a single core but *high performance architecture*; *(ii)* The characteristics of the CG model, such as different force field for pairs potentials that in-

[2]The largest system simulated for the reported experiments using the slowest version of the simulator, takes about 200 minutes.

[3]The speed-ups presented in Figures 3 and 4 are relative only to the GPU implemented part of the application which stands for 98% of the total execution time of the application. We decided to present only this part, because the servers where the GPU cards are situated have different CPUs.

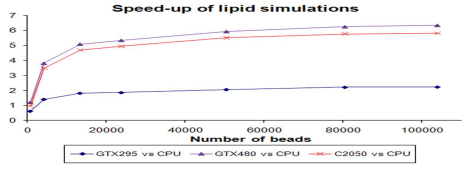

Figure 3: Speed-up of lipid simulations achieved on different architectures with respect to the CPU execution.

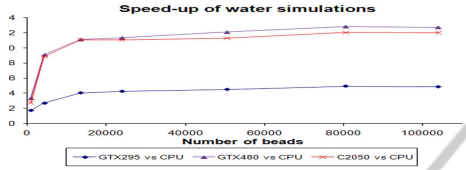

Figure 4: Speed-up of water simulation achieved on different architectures with respect to the CPU execution.

cludes particular representation for water and charges (Orsi, 2010), which depends on the type of interaction, lead to frequent divergent branches which impact the overall speed-up; *(iii)* The complexity of the force fields considered requires a large amount of local physical resources (i.e. registers), which have to be distributed among all threads on an SM impacting on performance; *(iv)* Furthermore, the pair interaction potentials computation causes a relatively large amount of scattered memory accesses, leading to a considerable texture cache miss rate. That is because neighbors of a bead are not spatially clustered. This issue could be addressed through additional optimizations obtained by adapting techniques developed for AL simulations (Anderson, 2008) that cannot be applied as is for CG models. These optimizations will be the object of future work.

Although the three architectures we evaluated are compliant with the *IEEE 754* standard, we observe a difference in the results of CPU versus GPU, after thousands of simulation steps. Anyway, this is expected and acceptable as long as the ensemble average, as for instance the average temperature or the pressure, is coherent after a reasonably large number of steps which is the case of our simulations. These simulations on the different architectures are coherent from an ensemble viewpoint: Mean values related to energy, pressure and temperature, both of GPU and CPU simulations, are the same. This is what matters in this application due to the chaoticity of molecular dynamics simulation systems.

5 CONCLUSIONS

In this work, we characterized the speed-ups of coarse grain simulations, considering biological systems with different number of beads and among three different GPU architectures. To this purpose, we implemented an accelerated version of a CG lipid bilayers simulator. We introduced specific optimizations and data structures required by CG models for handling different types of beads and interactions and calculated their overhead in the performance. We characterized the impact of biological system complexity in terms of beads type, observing that lipid systems achieve a speed-up almost 2.5 times slower than water systems. We finally determined the impact of GPU architectures on the acceleration performance.

REFERENCES

Anderson, J. A. (2008). General purpose molecular dynamics simulations fully implemented on graphics processing units. In *volume 227*. Journal of Computational Physics.

Bauer, B. A. (2011). Molecular dynamics simulations of aqueous ions at the liquid vapor interface accelerated using graphics processors. In *volume 32*. Journal of Computational Chemistry.

CUDA (2011). Nvidia cuda c programming guide. In *http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf*. NVIDIA.

MacCallum, J. L. (2006). Computer simulation of the distribution of hexane in a lipid bilayer:spatially resolved free energy, entropy and enthalpy profiles. In *volume 128*. Journal of the American Chemical Society.

Marrink, S. J. (2007). The martini force field: coarse grained model for biomolecular simulations. In *volume 111*. Journal of Physical Chemistry B.

Orsi, M. (2010). A quantitative coarse-grain model for lipid bilayers. In *volume 112*. IEEE CS.

Rapaport, D. C. (2004). *The Art of Molecular Dynamics Simulation*. Cambridge University Press, New York, 2nd edition.

Rapaport, D. C. (2011). Enhanced molecular dynamics performance with a programmable graphics processor. In *volume 182*. Computer Physics Communications.

Shkurti, A. (2010). Gpu acceleration of simulation tool for lipid-bilayers. In *IEEE International Conference on Bioinformatics and Biomedicine Workshops*. IEEE CS.

Stone, J. E. (2010). Gpu-accelerated molecular modeling coming of age. In *volume 29*. Journal of Molecular Graphics and Modelling.

van Meel, J. A. (2008). Harvesting graphics power for md simulations. In *volume 34*. Molecular Simulation.

Wohlert, J. (2006). Dynamics in atomistic simulations of phospholipid membranes: Nuclear magnetic resonance relaxation rates and lateral diffusion. In *volume 125*. Journal Of Chemical Physics.

Zhmurov, A. (2010). Sop-gpu: Accelerating biomolecular simulations in the centisecond timescale using graphics processors. In *volume 78*. Proteins.