GP-GPU Processing of Molecular Dynamics Simulations

Emmanuel Millán Kujtiuk[1], Eduardo M. Bringa[2], Andrew Higginbotham[4], Carlos Garcia Garino[1,3]

1- ITU & ITIC, UNCuyo, Mendoza, Argentina, {emillan,cgarcia}@itu.uncu.edu.ar
2- CONICET & ICB, UNCuyo, Mendoza, Argentina, {ebringa@yahoo.com}
3- Facultad de Ingeniería, UNCuyo, Mendoza, Argentina,
4- Department of Physics, Clarendon Laboratory, University of Oxford, Oxford OX1 3PU, United Kingdom, {andrew.higginbotham@physics.ox.ac.uk}

Abstract. Graphics Processing Units (GPUs) is an emergent hardware technology potentially suitable for High Performance Applications. In this work GPU processing of Molecular Dynamics (MD) simulations is discussed. An overview of GPU architecture is provided and its main features are outlined. Some relevant atomistic MD applications are processed using GPU hardware. The same applications are simulated using CPU oriented applications as well in order to obtain comparable results. In this case both serial and MPI codes are considered. Wall-clock timing of GPU results obtained with a relatively low-end card are found to be comparable to timing from multiple core clusters.

1 Introduction

This paper discusses the General-Purpose Graphics Processing Units (GPGPU)[1] processing of Molecular Dynamics (MD) simulations [2,3], as a model for High Performance Applications in general.

In this context Molecular Dynamics (MD) simulations are chosen as a benchmark problem. This is because there are a number of free open source MD codes highly optimized to run in either large CPU clusters [4,5,6] or GPUs [7,8,9], with some codes able to take advantage of both processing options, like LAMMPS [10] or NAMD[11]. Some of these codes include detailed benchmark data [10].

Starting several years ago, graphics cards and graphics oriented devices like Playstation[12] and Nintendo[13] received attention from industry due to the increasing requirements of High Performance Applications for the entertainment sector, including videogames, and powerful graphics cards for PC's. This trend also received some attention from the scientific community, and a Playstation PS3 based Cluster was built a few years ago[14], but the programming style was not simple at the early stages of these devices.

More recently Graphics Processing Units (GPUs) originally devoted to the PC market and manufactured by companies like NVIDIA and ATI have become more attractive for the High Performance Community due to the GPGPU emergent platform. Moreover, several programming kits have been released in

order to facilitate the general purpose usage of graphics devices. Compute Unified Device Architecture (CUDA)[15] released by NVIDIA, Stream[16] developed by AMD/ATI and the OpenCL[17] proposed as an standard by Apple to the Khronos Group provide environments for general purpose programming of these devices.

There are thousands of different application codes available in the literature for GPGPU processing. Besides the cited codes for MD simulations, there are GPGPU implementations ranging from mathematical libraries to design codes. For instance, the BLAS library[18] is included in CUDA SDK and CUFFT as well. As another example, a discussion of Finite Element Codes implementation on GPGPUs can be found in the works of Göddeke et al [19].

In this paper the results of different MD numerical experiments carried out using LAMMPS and HOOMD codes together with a rather low cost GPU card are presented. The GPGPU based results are compared with serial CPU results and MPI processes as well, whenever possible. While discussed results are only a preliminary effort within a joint computing initiative between the ICB and ITIC at the National University of Cuyo, they are promising because they show the potential of using GPU processing even with a low cost card (NVIDIA Ge Force 9500 GT with a current price about 105 US$).

The representation of real variables is discussed in the work as well. In most of the cases MD codes use double precision for CPU cases while GPU software is based on single precision variables. This fact can affect both speedup and quality of results obtained.

The paper is organized as follows: in section 2 a discussion of GPGPU programming and architecture is provided. Molecular Dynamics simulations are introduced in section 3 in order to provide some background material for benchmarks problems discussed in section 4. Finally, concluding remarks are included in section 5.

2 Introduction to General Purpose Computation on GPU

This section presents some background issues. These have been grouped into topics related to hardware and software development tools that are discussed in subsection 2.1. The two main technologies for GPGPU are presented in this section, CUDA of NVidia and Stream of AMD/ATI. Comments on the standard OpenCL are also included in this section.

2.1 GPU Architecture

NVIDIA CUDA is an architecture for general purpose parallel computing capable of resolving complex computational problems in a fraction of time compared to a CPU. This architecture can be programmed in C, Fortran or C++. The development tools for CUDA allow to resolve problems of audio processing, video, simulations of physics problems, product design, etc. CUDA can be executed in Linux 32/64 bits, MS-Windows 32/64 bits and MacOS X.

Some interesting features of CUDA are access to any position in memory; 16 KB of shared memory between threads, with the ability of being used as cache memory; fast read access from and to the GPU; integer support and bit level operators; and fast data transfer between the GPU and the CPU, due to a dedicated CUDA Driver.

Amongst CUDA limitations there are the impossibility of using recursion, function pointers, static variables inside functions or functions with a variable number of parameters. Simple precision does not support denormalized numbers or NaNs. The communication between the CPU and GPU is limited due to the bandwidth of the PCI-Express bus.

The threads must be launched in groups of at least 32, with thousands of threads in total. The threads are organized in blocks, all blocks run the same program and they are split into groups of 32 threads, each group of threads is executed by the stream processors in a SIMD (Single Instruction Multiple Data) form [20].

The Stream technology from AMD/ATI has similar features than CUDA. AMD provides a Software Development Kit (SDK). The performance of both technologies is very similar and the difference is in their hardware setup, ref. [20] explains in details the differences between NVIDIA and ATI GPUs.

Both technologies support the use of more than one GPU per machine. NVIDIA uses the technology SLI (Scalable Link Interface) which gives the possibility of using up to four video boards on the same machine. The requirements for using SLI is having enough PCI-Express 16x slots availables and that the video board has to support such a technology. For example, one possible configuration is use two GeForce GTX 295 with 960 processing units and 3.5 GB of total video memory in one machine.

On the other hand, AMD/ATI has Crossfire, a technology similar to SLI. A possible implementation for Crossfire is to use two Radeon HD 3850 video boards with 640 processing units each, achieving 1280 processing units and 2 GB of total video memory.

The OpenCL standard (Open Computing Language) exists with the purpose of making the GPU development process easy, and consist of an API and a programming language. Together they allow for the creation of applications with parallelism at data and task level which can be executed in GPUs and CPUs. The language is based in C99, with vector operations. OpenCL requires a video board with support for CUDA or Stream. The API is independent of the technology, and the generated code should work without modifications in any of the mentioned technologies (CUDA and Stream). AMD has develop a SDK for OpenCL development over x86, allowing the development and testing over x86 CPUs without the need of having a GPU with CUDA or Stream.

2.2 Available Video Cards in the Market

NVIDIA offers a medium and high end range of video boards that support CUDA. The GeForce line has between 16 up to 480 processing units per GPU, processing speeds from 550 MHz to 700 MHz, GDDR2/3/5 memory from 256 MB

up to 1.5 GB, support Quad-SLI, 2-way and 3-way SLI, the energy consumption is from 50w up to 250w, the target users of this line are gamers and desktop users. The Tesla line of NVIDIA offers a greater memory capacity (3 to 6 GB), ECC memory and 448 processing units at 1.15 GHz per GPU. In the Argentine market are available, among other boards, the GeForce 9500 GT 1024 MB DDR2 with 32 processing units at a cost of US$ 105, the GeForce GT240 1024 MB DDR3 with 96 processing units at US$ 175 or the GeForce 8400 GS 512MB DDR2 with 16 processing units at US$ 75.

The ATI video boards offers the following features: GPU speed from 668 MHz up to 850 MHz, memory capacity from 256 MB up to 1 GB GDDR3/5, from 320 to 3200 processing units (Stream processors). Some models that support Stream are: Radeon HD 3850 (320 Stream processor), Radeon HD 4890 (800 Stream processors) and the Radeon HD 5970 (3200 Stream processors). The high end lines of ATI are FireStream and FireGL, they have a greater memory capacity and Stream processors. The prices in Argentina for ATI boards are: Radeon HD 4850 1 GB GDDR5 with 800 processing units at a cost of US$ 240, or Radeon HD 4350 512 MB GDDR3 with 80 processing units, at a cost of US$ 73.

3 Molecular Dynamics

3.1 Overview

MD simulations solve the equations of motion for a set of atoms interacting through interatomic potentials [2,3]. MD simulations are a versatile tool to study mechanical properties of materials, and they have often predated experimental results regarding understanding of nanomaterials. One of the advantages of MD is that the strain, stress, temperature, velocity, etc. are known in detail [2]. We use two freeware MD codes: HOOMD [9,21], and LAMMPS [10,22].

MD simulations are as reliable as the interatomic potentials they use. These potentials are typically obtained using a database of experimental data and ab-initio results. For GPU applications typically the simplest of potentials, the Lennard-Jones potential, is the only one implemented. This pair potential can be extremely useful to extract the basic physics behind processes far from equilibrium, like grain-grain collisions[23], cosmic-ray interaction with materials [24], plastic behavior of crystals under high pressure conditions[25], etc. One advantage of many pair potentials, including the LJ potential, is that the strength of the interaction between two particles decays rapidly with distance, and can therefore be neglected beyond certain distance. This cut-off of the interactions allows for simple domain decomposition and parallelization schemes [2]. We note that there are a number of MD-GPU codes focusing on biological materials [8,7], and related codes focusing on N-body simulations for astrophysics [26]. They present a different type of challenge related to the long-range interactions between particles, which goes as the inverse of the distance between particles, due to Coulomb interaction between charged particles in biomaterials and to gravitational forces in N-body simulations.

3.2 Codes

Both LAMMPS and HOOMD run in Windows, Linux and MacOS, include working examples in their distributions, and count with a growing user community. For the results in this paper, we use the distributions from April 7, 2010 for LAMMPS and April 6, 2010 for HOOMD from the SVN repository. Details on the codes can be found on their web sites and online manuals, but a few of their features are mentioned below.

LAMMPS is written in C++, runs on a single processor or in parallel (using MPI and spatial-decomposition of simulation domain for efficient parallelism), open-source license, use of FFTW for long range Coulomb interactions, and it runs from an input script which can be easily tailored to particular problems. LAMMPS supports simulation of 2D and 3D systems, NVE, NVT, NPT, NPH, Parinello/Rahman integrators, and a large number of force fields, but only Lennard-Jones and Gay-Berne force fields can be run on a GPU. LAMMPS doesn't have a GUI, but its results can be visualized using a number of utilities, some of which are included in its distribution.

HOOMD is written in Python (interpreted) and C++ for the CUDA code, which makes it slower compared with LAMMPS which is written in C++ (compiled). HOOMD doesn't support MPI so it cannot be tested in a cluster, but it can run with OpenMP in multiples cores in one machine. The code of HOOMD is optimized for running all the simulation in one GPU but also has support for running in multiples GPUs present on the same machine. HOOMD has the following features: pair potentials (Lennard-Jones, Gaussian, CGCMM), bond Potentials (FENE, Harmonic), angle Potentials (Harmonic, CGCMM), dihedral/Improper Potentials (Harmonic), wall potentials (Lennard-Jones). HOMMD allows a number of integrators (Brownian dynamics NVT, NPT, NVE, NVT)[9].

This kind of codes are presently accepted like de facto tools in MD simulations. Researchers many times have to deal with scripting than code modifications in order to study disciplinary problems of their interest.

4 Benchmark Problems

In this section we study a few sample MD problems comparing GPU and CPU results. For benchmarking with LAMMPS we have used three examples that come already prepared for running in the LAMMPS distribution, only modifying the system size and the length (number of steps) of the run. We have also created a script for one example of interest in high speed deformation of materials. This same script was adapted to run an equivalent simulation in HOOMD.

The examples from LAMMPS are labeled "crack", "melt" and "melt+min". We have changed the size of the system and the number of simulated steps from their original values to study performance scaling. Details on the examples can be found in Table 1.

The example we created consists of a block of crystalline material, which is first relaxed. A slab of material on one end is then given a fixed high velocity to emulate a piston that sends a traveling shock wave into the rest of the

Table 1. Details of the tested examples

Example	Dimension	# Atoms	2D/3D	Steps	Description
melt	10x10x10	4000	3D	250	crystalline box heated above melting
	15x15x15	13500			
	20x20x20	32000			
melt+min	20x20x0.2	800	2D	1000	melting plus conjugate gradient minimization
	60x60x0.2	7200			
	100x100x0.2	20000			
crack	100x40x0.5	8141	2D	5000	crack opening under tension
	100x70x0.5	14171			
	100x100x0.5	20201			
shock	5x5x50	5000	3D	6000	
	6x6x50	7200			
	12.5x12.5x50	31200			
	18x18x47.5	62208			

block. Similar set-ups have been used to study thermodynamics[27] and plastic deformation[25] of solids deformed at high strain rates, and are still in use today but typically employing more complex interaction potentials [28].

4.1 Testing Infrastructure

The GPU and 1 core CPU experiments were performed in the following hardware: AMD Athlon 64 X2 Dual Core Processor 4600+ 2.4 GHz each core, with 3 GiB of RAM DDR-2, hard disk 80 GB SATA, video card NVIDIA GeForce 9500 GT 1GB DDR-2 with 32 processing units (cost US$ 105 dollars), running Slackware Linux 13.0 with CUDA 3.0.

The MPI experiments were executed in two different clusters. The first one, called Storm, has a master node plus 12 slave stations. All the computers have the same hardware: Intel P4 processor at 3.0 Ghz, 1 GiB of DDR-1 RAM, Gigabit Ethernet Card and Rocks Linux 5.3. The present (remaining) value of this hardware is estimated in 200 US$ dollars approximately. The second cluster, called Twister, has 16 nodes with a Intel Core 2 Duo at 3.0 Ghz processor, 4 GiB of DDR-2 RAM, Gigabit Ethernet card and Rocks Linux 5.0. Presently the cost of each node is 630 US$ dollars approximately. Results obtained with these two clusters for the shock experiments listed in Table 1 are compared with the ones computed with a NVIDIA GeForce 9500 GT GPU card.

4.2 LAMMPS Experiments

The results obtained with LAMMPS code for melt, melt+min and crack problems, listed in Table 1, are discussed in this subsection. These experiments are processed with three different number of atoms that determine the size of the

problem to be solved. The hardware used to run this experiments is the workstation described in the section 4.1, the AMD Athlon 64 X2 4600+ CPU and the NVIDIA GeForce 9500 GT GPU.

The graphics of figures 1, 2 and 3 compare the time processing required to compute the melt, melt+min and crack experiments respectively. In the figures GPU processing time for the different sizes of each experiment are shown together the 1 Core CPU processing time. In the cited figures can be observed that GPU processing times are in general smaller than 1 Core CPU time. However for a small number of atoms results are quite similar for CPU and GPU cases, as can be seen in figure 2. Possibly the poor performance of GPU card for small number of atoms can be explained because communication time between CPU and GPU is larger than GPU computation time.

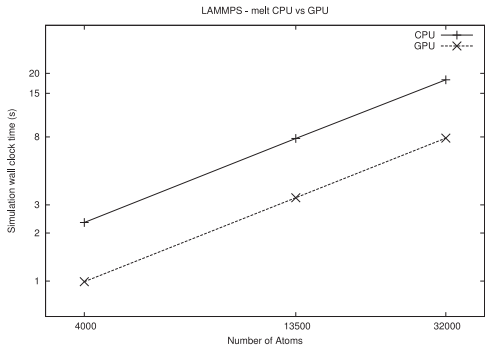

Fig. 1. Code LAMMPS running melt example (crystalline box heated above melting) for CPU versus GPU. The speedup for GPU is between 2.31 to 2.35.

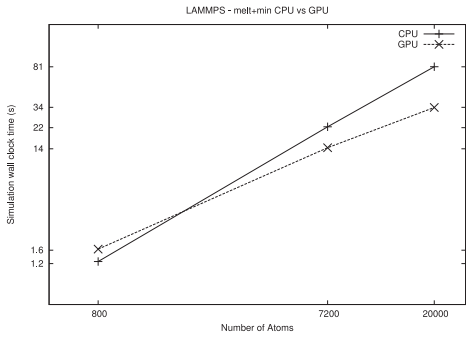

Fig. 2. Code LAMMPS running melt+min example (melting plus conjugate gradient minimization) for CPU versus GPU. For a small amount of atoms (800) the CPU is faster than the GPU, this is probably due to communication times between the CPU and the GPU is larger than the GPU processing time.

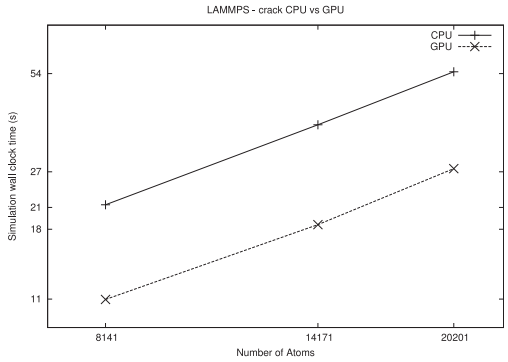

Fig. 3. Code LAMMPS running crack example (crack opening under tension) for CPU and GPU. For this experiment the Speedup in all cases is around 2.

In Table 2 can be observed the speedup obtained for the different experiments carried out.

Table 2. Speedup for LAMMPS examples

Example	# Atoms	CPU	GPU	Speedup
melt	32000	18.22	7.86	2.31
melt+min	20000	81.07	33.88	2.39
crack	20201	54.67	27.61	1.89

LAMMPS CPU compilation uses double precision, as in most MD codes for CPU. LAMMPS GPU compilation does not support double precision for the type of studies carried out in in this paper [29]. Consequently compilation was done for single precision. Single precision codes can show larger numerical instabilities than double precision codes. However, the velocity Verlet integrator which is used by LAMMPS for instance, is extremely stable compared to other integrators, provided that one uses reasonable time steps, as we have done here. This is demonstrated by our results, which even in single precission display excelent energy conservation, without noticeable diffusion of the energy values.

We have checked if this distinction modifies the outcome of the numerical integration and resulting evolution of the system, using the total energy of the system as diagnostic. We do not detect changes larger than $10^{-5}\%$, except for the case where the system is minimized using a conjugate gradient scheme[10] before the dynamics is started. Given the sensitivity of this minimization method to the force evaluation it is not surprising that differences arise.

The use of double precission is sometimes related to check-point files, used to re-start the simulation, so that the re-initiated run would continue in the same energy hypersurface when running an energy conservation algorithm. On the

other side, as has been pointed out in the introduction, it could be interesting to processes GPU codes using double precision, when possible, in order to get information about speedup and quality of results obtained in this case.

4.3 LAMMPS and HOOMD Experiments

In this subsection the results obtained for the shock experiments are discussed. LAMMPS and HOOMD are used in this case. In order to run the shock problem one script for each code was created from scratch .

In figure 4 can be seen the CPU time required for both codes computed with a AMD Athlon 64 X2 4600+ PC described in subsection 4.1. Only one core was used in order to process the experiments. LAMMPS performs better than HOOMD because in all the problems tested. A possible explanation for these results is that while LAMMPS is fully written in C++ (compiled), HOOMD is written in Python (interpreted) and C++ for the GPU code. On the other hand LAMMPS is a highly optimized code.

Figure 4 shows the GPU processing times as well. Again, LAMMPS is faster than HOOMD, but the speedup is smaller than for the CPU cases. It is important to point out that GPU processing time for HOOMD is shorter than the CPU times for both codes. Finally, due to some limitations of LAMMPS, it can't execute the shock problem for 62500 atoms for the GPU experiment, this is because the current implementation of GPU in LAMMPS cannot allocate memory for this amount of atoms, while HOOMD does not suffer this drawback.

No comparison is possible for distributed computing infrastructure in this case because LAMMPS has a MPI version and HOOMD an OpenMP one.

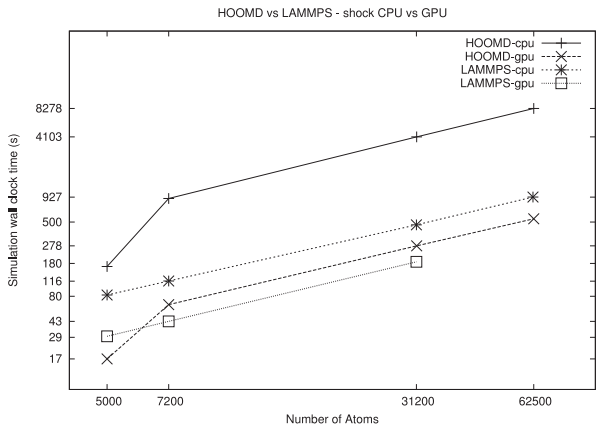

Fig. 4. Shock example for LAMMPS and HOOMD - CPU and GPU times. The difference between HOOMD and LAMMPS in time is due to the fact that LAMMPS is written in C++ and HOOMD in Python which is an interpreted language and is slower that a compiled one (C++).

Figure 5 shows the result for the shock script executed in the Storm Cluster with LAMMPS+MPI, for 1,2,4,6 and 8 cores versus one GPU. This experiment is useful to compare the performance of one GPU of modest features versus a cluster of machines.

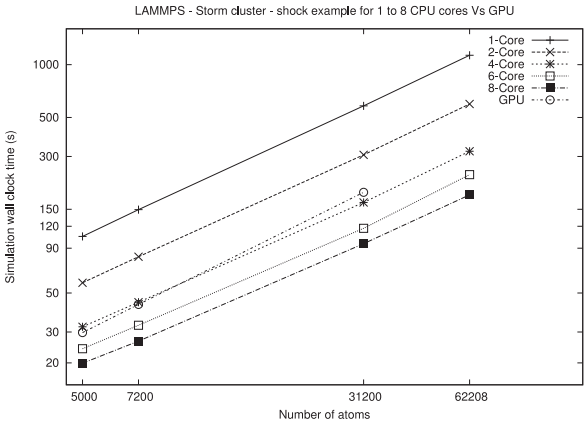

Fig. 5. Shock example for LAMMPS in Storm Cluster, with MPI in multiples CPUs vs one GPU. The GPU behaves as expected, resulting in a performance near of 4 nodes of the cluster Storm.

From Figure 5 it can be observed that, for the experiment with 31200 atoms, the performance of one GPU is very close to 4 nodes of the Storm Cluster and above 2 nodes of the Twister cluster. Because the nodes of the Storm cluster are somewhat old (4 years), the same test was executed in a newer cluster (Twister), and Figure 6 shows the obtained results from this experiment.

Results obtained from the experiment in Figure 6 shows that a GPU has a performance of roughly one node of the Twister cluster (Core 2 Duo with two cores each node). This indicates that for this kind of experiments, one low cost GPU compares with a high end CPU. The cost of the GPU is 105 US$ and the cost of one node of Twister is 630 US$.

Figure 7 shows the results for 1 and 2 CPUs (Athlon64 X2 4600+) versus one GPU because the shock script for HOOMD only works with OpenMP.

Fig. 8 shows a typical shock simulation as the ones described for Fig. 4 but somewhat larger. Those benchmark results only included minimal diagnostic I/O. Here we added the writing of large archives containing the positions of all the atoms every 1000 simulation steps, to follow compression and possible structural changes. The size of the sample, in face center cubic (fcc) unit cells N_x x N_y x N_z , is 25x25x75, with N_{atoms}= 4 x N_x x N_y x N_y = 187,500 atoms. The crystal was oriented to simulate shock wave propagation along the [001] direction. Lattice temperature was initially set to a very low value, 0.1 LJ units, to avoid spurious defect nucleation and evolution due to temperature effects.

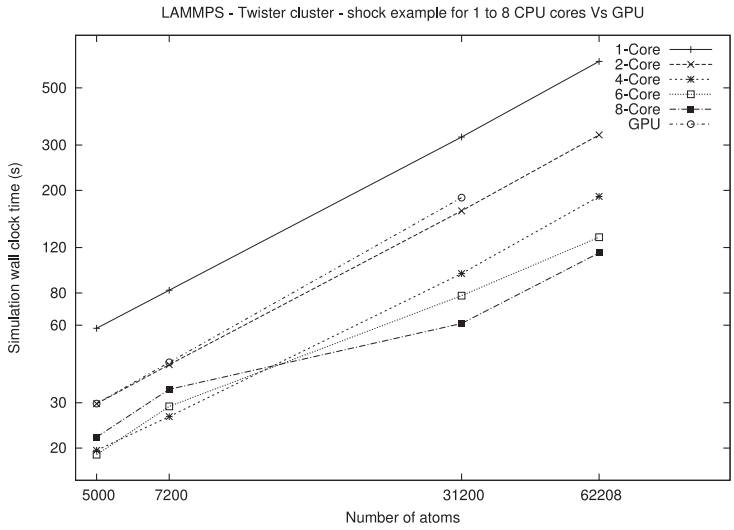

Fig. 6. Shock example for LAMMPS in Twister Cluster, with MPI in multiples CPUs vs one GPU. Being that Twister is a newer cluster than Storm, the performance of the GPU is near one node of the cluster (2 cores).

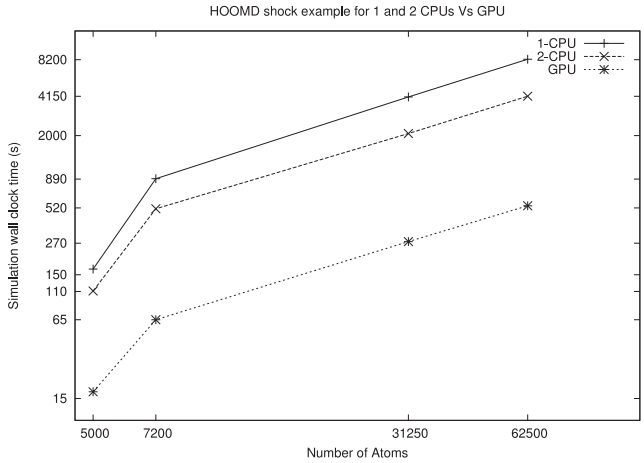

Fig. 7. HOOMD in 1 and 2 CPUs cores versus GPU. Running HOOMD in 2 cores versus 1 core, gives a speedup between 1.5 to 1.99 in the best case (62500 atoms).

Piston velocity was 1.75 LJ units. This is 0.206 of the longitudinal sound velocity along [001], and therefore our figure 7-b can be compared to the top frame in Fig. 1 of ref. [25]. Our cross-section is smaller than in ref. [25], but large enough to obtain representative defect structures. As the wave propagates trough the lattice, planar defects (stacking faults) are nucleated and release the tremendous shear stress generated by the uniaxial compression of the perfect lattice[25]. Similar defect structures and wave propagation can be seen for more complex many-body potential interactions describing fcc metals[30,31], and these defect structures can be experimentally detected by X-ray diffraction[32].

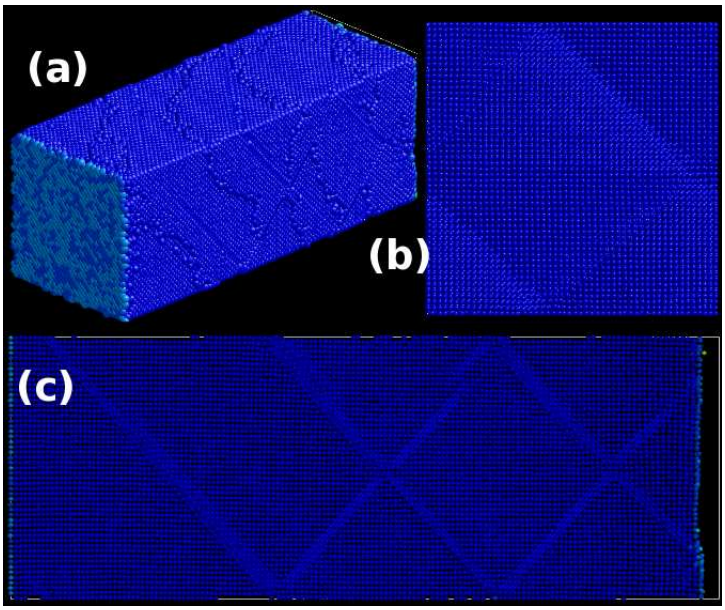

Fig. 8. Shock wave propagating along the [001] direction in a LJ monocrystal. All frames are for 2.0 LJ time units after the shock started. The whole sample is shown in (a), including the piston face. Frame (b) displays a cut perpendicular to the shock propagation. Frame (c) displays a thin slab of material showing the structure along the shock propagation, with the wave having already reached the back surface of the sample. The cross-hatched pattern indicates criss-crossing stacking faults in the crystal lattice, as in ref. [30,25].

5 Summary and Outlook

Using a number of examples we have shown that GPU processing of MD simulations can be an attractive option to CPU processing. The relationship between wall clock timing and cost is extremely good for GPUs when compared to CPU systems, achieving savings of a factor of approximately six.

It is important to point out that discussed results where obtained with a rather legacy and low cost GPU card but up-to-date CPU hardware, except for storm cluster. Moreover, larger improvments can be expected for GPU cards than CPU cores, then further results can still be better than ones presented here.

It is difficult to compare LAMMPS and HOOMD codes performance. While it was not one of main targets of this work can be said that LAMMPS processes faster that HOOMD but this code does not suffer lack of memory drawback for problems with large number of atoms.

There are a large number of physical systems which can be simulated within the constrains given by GPU architectures. Future GPU coding developments allowing the use of many-body and reactive potentials like EAM, MEAM and AIREBO will not require significantly different potential routines and would help solving materials science problems in addition to the biological problems that are currently being solved in GPU platforms. GPU implementations capable of using MPI efficiently for communication between GPU cards which reside in different machines of a cluster would also enhance the scope of possible problems. Recent N-body algorithms[26] show that novel programming paradigms might be needed to achieve this.

6 Acknowledgments

The authors gratefully acknowledge the financial support provided by projects SeCyTP 06/M009 and 06/B194 from National University of Cuyo and project PAE-PICT 2312 granted by National Agency for Scientific and Technological Promotion (ANPCyT). We acknowledge discussions with Joshua Anderson, who added features to HOOMD to make the shock simulations possible.

References

[1] GPGPU, General-Purpose computation on Graphics Processing Units: http://gpgpu.org/about

[2] M. P.Allen, D.J. Tildesley, “Computer simulation of liquids”, Oxford Science Publications, Oxford, 1987.

[3] D. Frenkel, B. Smit, Understanding Molecular Simulations, Academic Press, San Diego CA, 2002.

[4] DLPOLY Molecular Simulation Package: http://www.cse.scitech.ac.uk/ccg/software/DL_POLY/

[5] MDCASK Molecular Dynamics Code for Radiation Damage: https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/mdcask/

[6] GROMACS, GROningen MAchine for Chemical Simulations: http://www.gromacs.org/

[7] G. Giupponi, M. Harvey and G. De Fabritiis, The impact of accelerator processors for high-throughput molecular modeling and simulation, Drug Discovery Today 13, 1052, 2008.

[8] M. 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.
[9] HOOMD, Highly Optimized Object-oriented Many-particle Dynamics: http://codeblue.umich.edu/hoomd-blue/
[10] LAMMPS, Large-scale Atomic/Molecular Massively Parallel Simulator: http://lammps.sandia.gov/
[11] NAMD Scalable Molecular Dynamics: http://www.ks.uiuc.edu/Research/namd/
[12] Playstation: www.us.playstation.com
[13] Nintendo: nintendo.com
[14] Playstation Cluster: http://www.sciencedaily.com/releases/2007/03/070319205733.htm
[15] CUDA, Compute Unified Device Architecture: http://www.nvidia.com/object/cuda_what_is.html
[16] ATI Stream: http://www.amd.com/la/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx
[17] OpenCL, Open Computing Language: http://www.khronos.org/opencl/
[18] BLAS, Basic Linear Algebra Subprograms: www.netlib.org/blas
[19] Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Sven H.M. Buijssen, Matthias Grajewski and Stefan Turek, "Exploring weak scalability for fem calculations on a gpu-enhanced cluster," Parallel Computing, vol. 33, no. 10-11, pp. 685–699, 2007.
[20] Andre R. Brodtkorb , Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik and Olaf O. Storaasli. State-of-the-art in heterogeneous computing. Scientific Programming 18. IOS Press. pp. 1–33, 2010.
[21] Joshua A. Anderson, Chris D. Lorenz, A. Travesset. "General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units". Journal of Computational Physics 227: pp. 5342-5359, 2008.
[22] S. J. Plimpton, J. Comput. Phys. 117, 1995.
[23] Hiroto Kuninaka and Hisao Hayakawa, Simulation of cohesive head-on collisions of thermally activated nanoclusters, PHYSICAL REVIEW E 79, 031309, 2009.
[24] E. M. Bringa and R. E. Johnson, Coulomb Explosion and Thermal Spikes, Phys. Rev. Lett. 88, 165501, 2002.
[25] Timothy C. Germann, Brad Lee Holian, Peter S. Lomdahl, and Ramon Ravelo. Orientation Dependence in Molecular Dynamics Simulations of Shocked Single Crystals, Phys. Rev. Lett. 84, pp. 5351, 2000.
[26] Tsuyoshi Hamada, Rio Yokota, Keigo Nitadori, Tetsu Narumi, Kenji Yasuoka, Makoto Taiji. "42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence". Gordon Bell finalists table of contents Article No.: 62 ISBN:978-1-60558-744-8, 2009
[27] V. V. Zhakhovskiĭ, S. V. Zybin, K. Nishihara, and S. I. Anisimov. Shock Wave Structure in Lennard-Jones Crystal via Molecular Dynamics, Phys. Rev. Lett. 83, pp. 1175, 1999.
[28] E. M. Bringa, K. Rosolankova, R. E. Rudd, B. A. Remington, J. S. Wark, M. Duchaineau, D. H. Kalantar, J. Hawreliak & J. Belak. Shock deformation of face-centred-cubic metals on subnanosecond timescales, Nature Materials 5, pp. 805-809, 2006.
[29] http://lammps.sandia.gov/doc/Manual.html
[30] E.M. Bringa, B.D. Wirth, M.J. Caturla, J. Stolken, D. Kalantar, Nucl. Inst. Meth. Phys. Res. B 2002 56, 2003.

[31] E.M. Bringa, J. U. Cazamias, P. Erhart, J. Stölken, N. Tanushev, B. D. Wirth, R. E. Rudd, and M. J. Caturla. "Atomistic shock Hugoniot simulation of single-crystal copper". J. App. Phys. 96, pp. 3793, 2004.
[32] K. Rosolankova, J.S. Wark, E.M. Bringa and J. Hawreliak, J. Phys. Cond. Matter 18, 6749, 2006.