Parallel execution of a parameter sweep for molecular dynamics simulations in a hybrid GPU/CPU environment

Emmanuel N. Millán[1,2,3], Carlos García Garino[2], and Eduardo M. Bringa[1,3]

[1] CONICET, Mendoza
[2] ITIC, Universidad Nacional de Cuyo
[3] Instituto de Ciencias Básicas, Universidad Nacional de Cuyo, Mendoza
{emmanueln@gmail.com,cgarino@gmail.com,ebringa@yahoo.com}
http://sites.google.com/site/simafweb/

Abstract. Molecular Dynamics (MD) simulations can help to utimingnderstand an immense number of phenomena at the nano and microscale. They often require the exploration of large parameter space, and a possible parallelization strategy consists of sending different parameter sets to different processors. Here we present such approach using a hybrid environment of Graphic Processing Units (GPUs) and CPU cores. We take advantage of the software LAMMPS (lammps.sandia.gov), which is already prepared to run in a hybrid environment, in order to do an efficient parameter sweep. One example is presented in this work: the collision of two clusters is sampled over a multivariate space to obtain information on the resulting structural properties.

Keywords: General purpose GPU, parameter sweep, Molecular Dynamics

1 Introduction

In the last few years, Graphic Processing Units (GPUs) have been used in numerous scientific projects due to their large amount of parallel processors and memory bandwidth. The two primary video cards manufacturers, NVIDIA and AMD/ATI, have created software development kits to use the GPUs for general purpose computing. There are several technologies for developing code that can be executed in a GPU: CUDA [1] from NVIDIA, OpenCL [2], AMD Accelerated Parallel Processing (APP) [3], OpenACC [4]. Atomistic simulation, including Molecular Dynamics (MD), is one of the areas where application of GPUs has grown significantly.

MD simulations follow the trajectories of interacting particles [5], and they have been extremely successful to model a variety of systems specially at the nanoscale [6]. A great number of these simulations require High Performance Computing (HPC) to run, combining hardware and software aspects, and GPUs offer a possible avenue for fast, energy efficient HPC [7–9]. Indeed, in the past

few years MD applications have been developed to run on GPUs, or successful MD CPU codes have been ported to GPUs. For instance, in the area of simulations aimed at chemistry and biology stand the works of Amber [10], NAMD [11] and GROMACS [12]. Furthermore, there are codes that aim to physics and engineering applications, e.g. HOOMD [7], LAMMPS [13], and DL_POLY [14]. These codes can achieve speed-ups of x2-x50 compared to CPU versions. Here we use LAMMPS, which is a mature, open source code, in active development and with a fairly large user community. It can be executed in a hybrid cluster of computers with multiple GPUs in each node using OpenMPI [15] or MPICH [16]. Also can work with CUDA and with OpenCL, using a library called Geryon [17, 8]. LAMMPS can carry out large simulations, which often could not run in a single process, into different cores or threads, using efficient parallelization strategies. For instance, in some tests performed by the authors, a single NVIDIA Tesla C2070 was able to run a simulation with more than ten million atoms. In this work we consider an alternative scenario: a large number of simulations, where each of them can fit into a single core. The simulations differ only by a discrete number of initial parameters, and a parameter sweep is required. This work focus on testing parameter sweeps in a GPU and a multicore CPU, since both technologies are readily available in the market today and they are often combined to improve performance. We developed a script using the Ruby scripting language (`http://www.ruby-lang.org`) that spawns several independent and concurrent processes of LAMMPS with different input parameters in the GPU and CPU. We conducted several tests to evaluate how many concurrent processes could be running at the same time to make maximum use of the GPU and also the available CPU cores in a single workstation. We created a simulation to test the parameter sweep script which consist in the collision of two clusters of atoms. Performance tests show the maximum amount of workload that a GPU can handle by itself, and how CPU cores can contribute to improve performance, handling part of that workload. The main objective of this work is to find the conditions which reduce the completion time (makespan) for a large number of required simulations, trying to maximize the use of the GPU and CPU cores.

2 Parameter Sweep Code

The Ruby script (which is available in the authors website: `http://sites.google.com/site/simafweb/`) is build to execute multiple jobs of LAMMPS in the GPU and/or CPU in parallel. It is necessary to modify certain variables, which are the parameters that the LAMMPS input files are going to use, for example, the initial set of velocities of the atoms in the system. The script builds an array containing all the possible combinations for the chosen input variables and their values. After that, it spawns the desired number of processes in the GPU and CPU iterating the parameter array, taking a set of parameters in each iteration inside each process, and building the input file for LAMMPS. The script changes a skeleton input file and only modifies the desired lines which contain the parameters we are exploring. Each spawned process creates a directory for

each simulation, and maintains the input file and all the output files in that directory, separately of the rest of the simulations. Each time a LAMMPS simulation is finished we write in an output file the parameter set for that run, the run "number", and the final time for that simulation. The script uses that output file for several tasks: first, if the script is interrupted for any reason, it restarts the last simulation that was running; second, we use it to do post-processing calculations with the output files from all simulations. A second Ruby script is used to complete the parameter sweep, calculating needed properties, like averages and standard deviation of relevant quantities, generating another output file with those results.

3 Nanograin Collisions

3.1 Hardware Infrastructure and Software Details

Simulations were executed in a workstation with multiple CPU cores and a single GPU. The characteristics are: AMD Phenom 1055T six cores CPU at 2.8 GHz, with 12 GB of RAM, 1 Tb 7200 RPM SATA hard drive and an NVIDIA Tesla 2050C GPU. This GPU has 448 cores running at 1.15 GHz with 3 GB of ECC memory, it supports single and double precision and has compute capability of 2.0. The Compute Capability (CC) of the GPU indicates which features of CUDA the GPU can execute, for example: CC 1.3+ supports double precision, previous versions can only execute code in single precision. The software was installed in a Linux distribution, Slackware 13.37 64 bit, with LAMMPS version dated 10-Feb-2012, Ruby 1.9.3p194 and CUDA version 3.2. The LAMMPS code was compiled with OpenMPI 1.4.2 and gcc 4.5.3 with -O2 optimizations, the GPU package with CC 2.0 and with double and single precision. The LAMMPS code uses OpenMPI to distribute the jobs between CPU cores in a single workstation and between nodes of a cluster. All the simulations in this work are executed using OpenMPI.

3.2 Physical Scenario

We use one simulation to test the parameter sweep script: a complex collision between two clusters of atoms. A Lennard-Jones (LJ) interatomic potential was used. LJ interactions are computationally simple, but can describe realistic behavior of materials [5]. Simulations can be carried out using dimensionless LJ units, which will be used from now on. Specifying the LJ parameters σ, ϵ, and mass, for a given material allows the conversion to "real" units. The test case we consider is the collision of nanograins, of interest in several contexts, from nanotechnology to astrophysics. There are several studies of such collisions, focusing on the mixing of grain material [18], or on grain fragmentation [19, 20]. There are also several papers where grains are approximated as elastic spheres [21], but irreversible plastic deformation will occur for collision velocities above certain threshold. Accurate determinations of that threshold are not easily carried out, and grain size will affect strain rate and plastic yielding. Here we try to

quantify how much plastic deformation occurs due to the collision. We construct spherical nanograins, of radius R, from solids with face centered cubic (f.c.c.) structure but different orientation. To perform the parameter sweep we use three different relative velocities for the grains, ten different orientations of the lattice, and five impact parameters (distance between the grain centers along the direction perpendicular to their relative velocity), giving 150 independent simulations. In f.c.c. solids, large stress leads to partial dislocations (PDis) travelling rapidly through the material. In this case, the partials dislocations run across the nanograins and are absorbed at the surfaces, leaving behind only stacking faults (SFs) which change the mechanical properties of the material. Therefore, to measure the amount of plasticity, atoms in SFs are counted as defective atoms using a combination of centro-symmetry parameter [22] and coordination number to avoid counting some surface atoms as part of SFs. We also calculate the average and standard error of the number of atoms that are in SFs and PDis, over all simulated orientations, as a function of velocity and impact parameter. With these values we can observe how the material behaves under collisions with different velocities and impact parameters.

3.3 Single Simulation

In order to obtain reference times to use as comparison for the multiple simulations done in the next subsection we executed the collision of nanograins simulation in both CPU and GPU. We selected a single simulation from the parameter sweep pool of inputs and executed the mpirun command in the CPU (using 1,2,4 and 6 cores) and the GPU. The Table 1 shows the wall time for the simulation in 1,2,4 and 6 MPI tasks (or processes, -np parameter in the mpirun command) in CPU and GPU for single and double precision. Each MPI task uses a CPU core, even if it is executing the simulation in the GPU. From the computed CPU times one can see that the GPU (double precision, running one MPI task) has a speed-up 5.39x over the serial case (1 MPI task) CPU and a speedup of 1.45x over six CPU MPI tasks. For GPU single precision the speedups are greater, e.g., 12x for the CPU serial case and 3.4x for the six CPUs MPI tasks. In some cases the GPU is not used to the fullest potential (see Ref. [8]), it can be seen in Table 1 that two processes running in the GPU in parallel give better timing than one process, but there is no further advantage in the addition of new processes because each process must communicate with the rest. This communication is done through the CPU and the data must go through the PCI-Express bus, which is time consuming. For CPU-only cases, the timing decreases as the number of MPI tasks increases, these improvements in timing are thanks to the MPI effectiveness in the CPU architecture, and the bottleneck in communication with the PCI-Express bus is missing; in our tests we are not taking into account MPI communications between hosts of a cluster.

LAMMPS has the ability to do load balancing of processing by splitting the force calculations between the GPU and CPU [8]. We tested this feature and we did not obtain any speedups for the number of atoms we simulate (10000). For further benchmarks and simulations of this feature see the work of Brown *et*

Table 1. Collision between two nanograins, with 10,000 atoms and 200,000 steps.

Num of domains	CPU	GPU double	GPU single
1	2055 s	381 s	163 s
2	1031 s	346 s	146 s
4	668 s	382 s	184 s
6	554 s	487 s	268 s

al. [8]. In the next subsection we conduct several tests cases to show how many parallel jobs the GPU can handle with the final goal of reducing the makespan of multiple independent simulations. Further testing of LAMMPS simulations using CPUs and GPUs can be found in Ref. [23].

3.4 Multiple Simulations

In this section we execute the parameter sweep script for multiple simulations in the GPU and a hybrid scenario using GPU and CPU cores. To obtain reference times to compare with the hybrid scenario first we executed the multiple simulations using only the GPU, we execute in parallel multiple simulations, distributing jobs between the CPU and GPU in order to minimize the makespan. The optimal assignation of resources of CPU and GPU to minimize the makespan it is a problem of job scheduling which is beyond the scope of the presented work. Further investigation in this matter is needed. Certain simulations, when the number of atoms is of the order of 1000 to 10000, can be executed in parallel using only one GPU, providing the amount of memory available on the GPU be large enough to satisfy job memory requirements. The GPU can execute multiple independent processes at the same time, unless the "compute mode" of the GPU is set to "exclusive process", in this mode only one process can be executed in the GPU. We take advantage of this feature and we execute multiple simulations at the same time for the example described above. Each simulation has 10,059 atoms, runs 200,000 steps and produces 12 MB of data spread in three kinds of dump files: atoms in stacking faults, atoms in partial dislocations, both dumped every 20,000 steps, and all atoms in the system, dumped every 25,000 steps. We execute 150 simulations.

Table 2 shows the makespan for the 150 independent simulations in terms of the number of processes considered. The column "Parallel simulations" indicates the number of independent simulations running in parallel. As a reference, the Phenom CPU executed the same simulations in 60,334 seconds, one independent simulation per core, six simulations in parallel. For double precision, the GPU is 1.46x times faster than the six CPU cores, launching four simulations in parallel. For single precision the speedup is 6.96x, for six simulations in parallel.

Figure 2 displays the data collected in Table 2 and one hybrid GPU/CPU case executing five parallel simulations, four of which are GPU processes and one CPU process using two cores. It is important to note the drop in wall time

when using more than one process simultaneously, since the GPU can handle more than one simulation at a time extremely well. The Tesla GPU can run up to ten independent simulations in parallel, and the best speedup is obtained with four simulations in parallel, at 1.39x comparing one GPU processes vs four GPU processes. The number of parallel simulations that the GPU can execute is related to the amount of memory each simulation requires. For example, running eight parallel simulations of the collision of nanograins consumes approximately 28% of the total memory of the Tesla GPU (3 GB), as reported by the "nvidia-smi" command. Simulations with more atoms or more complex potentials will consume more memory and the number of parallel simulations will depend on the amount of memory needed by each simulation. There is also a point at which the amount of parallel simulations will overload the GPU and the makespan will of course increase.

We conducted several tests cases before selecting which distribution of jobs between the CPU and GPU double precision would give the lowest final wall time. Looking at the results in Table 2 we can see that four GPU double precision processes running the 150 simulations give the best timing. In that case, each batch of four GPU simulations takes ∼1110 s to finish, and these four GPU processes use four CPU cores, while the remaining two cores would remain idle, opening up the possibility to execute a "hybrid" case. Using these two CPU cores we obtain that a single simulation takes ∼1031 s (see Table 1). Therefore, to improve the makespan, we executed the parameter sweep script with this configuration: four GPU processes and one CPU process using two cores, running a total of five parallel LAMMPS simulations. After a batch of five simulations, the GPU needed the same amount of time to complete the four simulations as when the CPU was not in use (∼1110 s), for the CPU simulation the time was higher (∼1250 s). With these values we can calculate approximately how much time will it take to run the 150 simulations with different distributions. For example: executing 10 simulations in the CPU and 140 in the GPU would it take ≈ 140/4 * 1110 s = 38850 s for the GPU to finish and ≈ 10 * 1250 s = 12500 s for the CPU to finish, the CPU will finish first than the GPU. In Table 3 we can see that the time to run the 150 simulations for this distribution is 38932 s. Based on those results, we run 20 simulations in the CPU (taking ∼25000 s), and 130 in the GPU (taking ≈ 130/4 * 1110 s = 36075 s), and the calculated final times are close to the ones obtained in the 150 simulations in Table 3. For other kind of problems one could make similar tests and obtain partial times and calculate the nearly optimal distribution and assignation of the available resources to minimize simulations time.

Table 3 shows the results obtained for the multiple simulations distributed between CPU and GPU. The "Parallel simul. CPU" column indicates the number of independent LAMMPS simulations executing in the CPU with each of these simulations using N number of cores (second column, -np parameter in mpirun command). The "Parallel simul. GPU" column indicates how many processes are running in the GPU in parallel with the CPU. The number of parallel simulations that are executing in each case is: "Parallel simul. CPU" + "Parallel

simul. GPU", the number of CPU cores in use in each case is calculated by: "Parallel simul. CPU" * "CPU Processes" + "Parallel simul. GPU". The "Distr." column indicates the amount of simulations executed in the GPU and CPU for each case. In the case of hybrid simulations between CPU and GPU (row four in Table 3), the number of concurrent simulations is five: one simulation (process) in the CPU using two CPU cores and 4 simulations in the GPU. Comparing the best result from Table 2 for double precision (41070 s) to the best result of hybrid CPU/GPU (35511 s) the speedup is only 1.15x times. The speedup of this hybrid case with the six cores in the CPU (60334 s) test is 1.69x. Therefore, for this kind of simulations, executing simultaneously in the CPU and GPU can reduce the makespan.

Table 2. Collision of grains, 150 simulations.

Parallel simulations	GPU double	GPU single
1	57238 s	25233 s
2	41885 s	15409 s
4	41070 s	8755 s
6	41731 s	8659 s
8	41804 s	8304 s
10	41808 s	8760 s

Table 3. Hybrid simulations between CPU and GPU.

Parallel simul. CPU	CPU processes (-np)	Parallel simul. GPU	GPU Double	Distr.	
				gpu	cpu
6	1	0	60334 s	0	150
0	0	4	41070 s	150	0
1	2	4	40038 s	145	5
			38932 s	140	10
			35511 s	130	20
			38658 s	120	30
			48218 s	110	40

Colberg and Hofling [24] performed MD simulations of polymers and concluded that single floating point precision is not sufficient and may result in qualitatively and quantitatively wrong results. However, for the type of simulations presented here, single precision might be a valid option. We selected one simulation from the 150 performed, and calculated the energy difference between double and single precision runs, and the difference was close to the 0.0007%, which is perfectly acceptable, especially given the decrease in wall time. LAMMPS also has a mixed precision option (not used in this work), where positions are stored in single precision, but accumulation and storage of forces, torques, energies, and virials are performed in double precision.

4 Plasticity in Nanograins

Figure 3 shows snapshots of selected simulations for different velocities V. The lowest velocity case shown here (V=0.3) does not include any SFs, and the deformation at the interface is only elastic, as assumed in most granular models [25]. We also include results for V=6, indicating that there is already a large degree of fragmentation and amorphization, and one cannot easily identify line and planar defects. We observe SFs at intermediate velocities, in {111} planes.

Figure 1 quantifies the amount of plasticity, counting SF atoms, vs. velocity. Plasticity increases with velocity and decreases as the impact parameter grows. This is expected because larger velocities and more central collisions increase the resulting stress in the grains, helping with dislocation nucleation. The curve $b = 0$ R SP shows one result for single floating point precision. It can be seen that the SFs for single precision are within the error bars of the double precision run, further validating the use of single precision in this case.

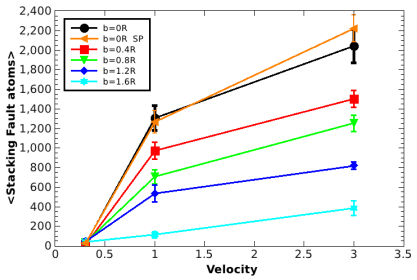

Fig. 1. Average of stacking fault atoms for 10 orientations versus velocity (LJ units), b indicates the impact parameter, and R is the radius of the spherical grains (R=6.7 σ). SP indicates single precision.

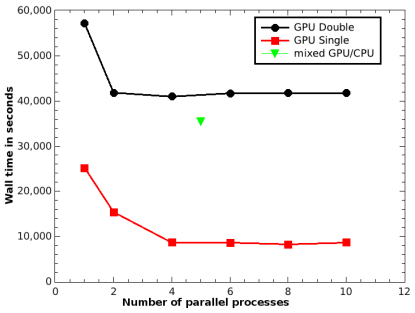

Fig. 2. Collision of grains, 150 simulations. The hybrid GPU/CPU test is for five processes running independent simulations in parallel using six CPU cores, one CPU process with two cores and four GPU processes.

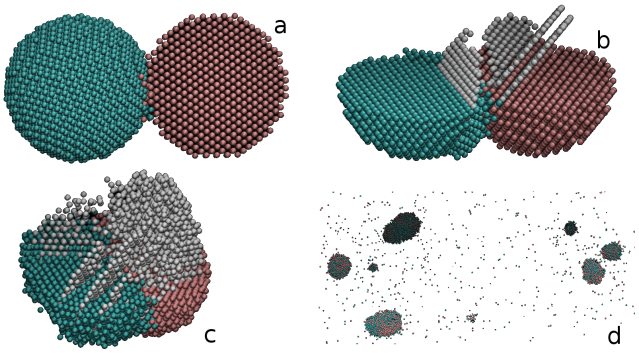

Fig. 3. Collision of grains. Snapshot showing the last step of the simulation for different velocities V (LJ units): a) V=0.3, b) V=1, c) V=3, and d) V=6. Only part of the atoms are shown, to allow the visualization of stacking fault atoms (grey), arranged in planar structures. Grains are showed in teal and red, to display the lack of mixing at low velocities.

5 Conclusions

We presented a strategy to carry out parameter sweeps for the Molecular Dynamics code LAMMPS in a hybrid CPU-GPU environment, using a Ruby script. Although our approach is general and can be applied to a plethora of problems, here we focus on one case which is of interest to us, and calculate timing and speedup factors for that particular case. We first studied the comparative independent performance of the CPU and GPU. Then independent jobs were executed in parallel, with various distributions in order to establish the maximum number of parallel jobs for the minimum makespan possible. Finally, we carried out simulations using a hybrid architecture (CPU/GPU), obtaining an improvement of 15%, respect to non-hybrid results. Of course, this would depends on the size and type of simulation, but similar improvements could be achievable for others types of problems.

Our Ruby script can also be easily modified to make any kind of simulation with LAMMPS and a similar approach could be used to execute code that supports GPU and CPU with minimum changes to the script. The performed tests show that running independent simulations in parallel in both CPU and GPU could improve the final simulation time. In addition, it is important to note that, in all cases tested here, it was better to run more than one simulation in parallel that to run one simulation at a time in the GPU. In the tests performed for double precision, the GPU is 1.46x times faster than the six CPU cores, launching 4 simulation in parallel in the GPU. For single precision the speedup is 6.96x, for 6 simulations in parallel vs six CPU cores. Executing LAMMPS processes in GPU and CPU gives better performance, and we obtained speedups of 1.15x comparing the best result from double precision in GPU (41070 s for four processes in parallel) to the best result of hybrid CPU/GPU (35511 s for four processes in GPU and 1 process with two cores in CPU). The speedup of this hybrid case with the six cores in the CPU (60334 s) test is 1.69x.

The parameter sweep script presented in this work was used to simulate the collision of two nanograins. The number of atoms in Stacking Faults (SFs) provides a measure of plasticity which grows with velocity. We locate a threshold for plasticity, averaged over different impact orientations, for velocities between 0.3 and 1 (LJ units). Beyond V=3 we observe significant fragmentation of the grains, and a measure of plasticity is no longer meaningful. Such information is important for astrophysical applications [21] and will be expanded in the near future.

6 Acknowledgements

E. Millán acknowledges support from a CONICET doctoral scholarship. E.M. Bringa thanks support from a SeCTyP2011-2013 project and PICT2009-0092.

References

1. CUDA from NVIDIA: `http://www.nvidia.com/cuda`

2. OpenCL, The open standard for parallel programming of heterogeneous systems: http://www.khronos.org/opencl/
3. AMD Accelerated Parallel Processing (APP) http://developer.amd.com.
4. OpenACC: http://www.openacc-standard.org/
5. Allen, M.P., Tildesley, D.J.: Computer simulations of liquids. Oxford Science Publications, Oxford, (1987)
6. Anders, C., Bringa, E.M., Ziegenhain, G., Graham, G.A., Hansen, J.F., Park, N., Teslich, N.E., Urbassek, H.M.: Why Nanoprojectiles Work Differently than Macroimpactors: The Role of Plastic Flow. Phys. Rev. Lett. **108** 027601 (2012)
7. Anderson, J.A., Lorenz, C.D., Travesset, A.: General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units. Journal of Computational Physics **227** 5342–5359 (2008)
8. Brown, W.M., Wang, P., Plimpton, S.J., Tharrington, A.N.: Implementing molecular dynamics on hybrid high performance computers short range forces. Computer Physics Communications **182** 898–911 (2011)
9. van Meel, J.A., Arnold, A., Frenkel, D., Portegies Zwart, S.F., Belleman, R.G.: Harvesting graphics power for MD simulations. Molecular Simulation Vol. **34**, Iss. 3 259–266 (2008)
10. AMBER: http://ambermd.org/
11. NAMD: http://www.ks.uiuc.edu/Research/namd
12. GROMACS: http://www.gromacs.org/
13. LAMMPS, Molecular Dynamics Simulator: http://lammps.sandia.gov/
14. DL_POLY Molecular Simulation Package: http://www.cse.scitech.ac.uk/ccg/software/DL_POLY/
15. OpenMPI: http://www.open-mpi.org
16. MPICH2: http://www.mcs.anl.gov/research/projects/mpich2/
17. Geryon: http://users.nccs.gov/~wb8/geryon/index.htm
18. Mariscal, M.M., Dassie, S.A., Leiva, E.P.M.: Collision as a way of forming bimetallic nanoclusters of various structures and chemical compositions. J. Chem. Phys. **123** 184505 (2005)
19. Ohnishi, N., Bringa, E.M., Remington, B.A., Gilmer, G., Minich,R., Yamaguchi, Y., Tielens, A.G.G.M.: Numerical analysis of nanograin collision by classical molecular dynamics. J. Phys.: Conf. Ser. **112** 042017 (2008)
20. Kalweit, M., Drikakis, D.: Collision dynamics of nanoscale Lennard-Jones clusters. Phys. Rev. B **74** 235415 (2006)
21. Ringl, C., Urbassek, H.M.: A LAMMPS implementation of granular mechanics: Inclusion of adhesive and microscopic friction forces. Computer Physics Communications **183** 986–992 (2012)
22. Kelchner, C.L, Plimpton, S.J., Hamilton, J.C.: Dislocation nucleation and defect structure during surface indentation. Phys Rev B, **58** 11085-11088 (1998)
23. Millán Kujtiuk, E., Bringa, E.M., Higginbotham, A., Garcia Garino, C.: GP-GPU Processing of Molecular Dynamics Simulations. High Performance Computing Symposium 2010, 39JAIIO, ISSN: 1851-9326, 3234–3248 (2010)
24. Colberg, P.H., Hofling, F.: Highly accelerated simulations of glassy dynamics using GPUs: Caveats on limited floating-point precision. Computer Physics Communications **182** 1120–1129 (2011)
25. Ringl, C., Bringa, E.M., Bertoldi, D.S., Urbassek, H.M.: Collisions of porous clusters: a granular-mechanics study of compaction and fragmentation. Astrophysical Journal **752** 151 (2012)