

A particle-mesh integrator for galactic dynamics powered by GPGPUs

Dominique Aubert¹, Mehdi Amini², and Romaric David²

¹ Observatoire Astronomique, Universite de Strasbourg, France

² CECPV, Universite de Strasbourg, France

Abstract. We present a particle-mesh N-body integrator running on GPU using CUDA. Relying on a grid-based description of the gravitational potential, it can simulate the evolution of self-interacting ‘stars’ in order to model e.g. galaxies. All the steps of the application have been ported on the GPU, namely 1/ an histogramming algorithm with CUDPP, 2/ of the resolution of the Poisson equation by means of FFT with CUFFT and multi-grid relaxation, 3/ of an optimized finite-difference scheme to compute the accelerations of stars and 4/ of an update procedure for positions and velocities. We present several tests at different resolution, and reach a speedup from 2 to 50 depending on the resolution and on the test case.

1 Introduction

By essence, astrophysics lack of laboratory experiments and from this intrinsic limitation emerges the need to rely on numerical simulation in order to understand the observations. Among the different fields of astrophysics, galactic dynamics has been a playground for numerical simulations for almost 50 years and it has been accompanied by numerical cosmology which ignited some of the largest scientific calculations ever made (see e.g. [1–3]). They both heavily relies on the use of N-Body integration techniques. Among the latter, one can cite direct N-Body integration (PP hereafter), Particle-Mesh (PM) and its extensions (P3M, AP3M), Tree-codes and AMR integrators. The recent introduction of ready-to-use API for General Purpose Graphical Processor Units (GPUs hereafter) will strongly impact this domain by providing an easy way to boost the performances of existing codes. Numerical experiments might be made faster and incidentally larger and hopefully more accurate. Several implementations of PP-methods (see e.g. [4, 5]) have previously been made available on GPU. In such integrators, pairwise interactions between stars are explicitly computed, implying a $O(n^2)$ complexity and thus limiting the number n of particles taken into account (typically $10^4 - 10^5$ on a single machine). We present here an attempt to fully develop on GPU a Particle-Mesh integrator for galactic dynamics, a method that can easily model the evolution of millions of bodies. Using CUDA, the API developed by NVidia for its devices, the overall speedup with respect to pure CPU computation spans from 2 to 50 thus promising interesting perspectives for future simulations.

First the principles of PM integrators are briefly described in section II. Then we provide the details of the parallelization using CUDA. Performances compared to CPU computation are presented in the following section. We discuss the limitations, the advantages and the perspectives in the last section.

2 An Overview of the Particle-Mesh Integrator

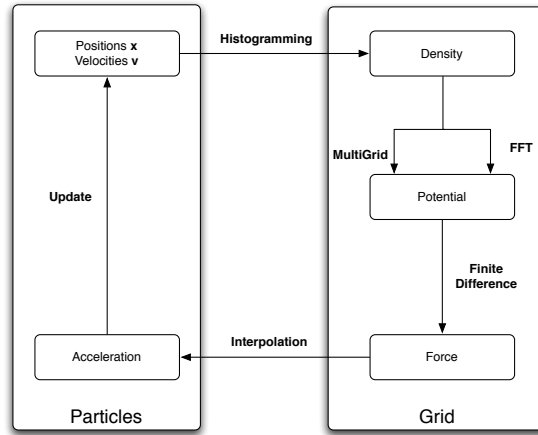


Fig. 1. Flow of operations in a PM calculation. One cycle corresponds to one time step. Operations on the left hand side of the diagram act on 'particles' data, while the right hand side operations act on fixed grid data.

The purpose of N-Body integrations is to simulate the mechanical evolution of a dynamical system due to its inner (and sometimes external) interactions.

To simulate the evolution of the system, the positions and the motions of the stars have to be computed.

The motion of a star at position \mathbf{x} and velocity \mathbf{v} is modified as time goes by according to the laws of motion:

$$\mathbf{x}(t + dt) = \mathbf{x}(t) + \mathbf{v} \times dt \quad (1)$$

$$\mathbf{v}(t + dt) = \mathbf{v}(t) + \boldsymbol{\gamma} \times dt \quad (2)$$

$$\boldsymbol{\gamma} = -\nabla\phi, \quad (3)$$

where $\boldsymbol{\gamma}$ stands for the star's acceleration and $\phi(\mathbf{x})$ stands for the gravitational *potential* applied to the star at its location. The potential is a scalar field and is related to the spatial distribution of matter via the Poisson equation $\nabla^2\phi = 4\pi G\rho(\mathbf{x})$, where $\rho(\mathbf{x})$ stands for the density of stars at a position \mathbf{x} . The density can be computed from the knowledge of the stars positions, which in turn makes

it possible to predict the motion of these bodies through the evaluation of the potential. A handful of methods exist to solve this situation (see e.g. [6, 7] for a review in an astronomical context) and the following sections will focus on the so-called particle-mesh method (PM hereafter).

Running a PM integrator with realistic problem sizes (128^3 and larger) on the researchers desktop machine can be time consuming. We aim at setting the foundations of a fast PM integrator able to run large simulations on desktop machines. It would ease the access to the results of large and more accurate simulations and/or accelerates the mass production of simulated catalogs by quickly providing large sets of small numerical experiments. We implemented all of the steps described involved in such calculation on GPU with CUDA as the SIMD programming mode and the libraries of the CUDA toolkits could fit the steps of our method. We mostly take advantage of the multithreading abilities of these devices to accelerate the computation.

PM-driven N-Body integrations loop over a well-defined sequence of elementary steps, which can be listed as follow and summarized in figure 1:

1. Density evaluation : knowing the position of the particles, the density ρ is evaluated on a 3D regular grid. Following an algorithm described by [8], we implemented an SIMD Nearest-Grid-Point assignment scheme. It relies on the radix sort and the scan primitives included in the CUDPP library. We also used a 'mixed' algorithm which still uses the GPU for finding the nearest cells to the particles but updates the density on the CPU. The calculations are much simpler but involve recurrent data transfers between the host and the device.
2. Potential evaluation: the density ρ being available, the potential is computed on the same 3D grid via the resolution of the Poisson equation. It is usually achieved with FFT or multi-grid (MG hereafter) relaxation. MG techniques are less efficient than FFT but can be used for any kind of boundary conditions (while FFT assumes periodic computational domain). MG can also solve the modified gravity versions of the Poisson Equation [9]. Parallelizations of both have been widely studied and implemented [10, 11]. We developed both versions on the GPU, using the CUFFT API for FFT and writing from scratch a MG solver.
3. Accelerations calculation: they are directly available from the potential using finite differentiation of the potential. We adapted this entire step to the GPU and optimized it from a SIMD point-of-view while taking into account data locality problems.
4. Interpolation: the data representation switches back to a particle description. Each body is being independently assigned an acceleration by interpolation at its position.
5. Velocities and positions update: the accelerations lead to the update of the velocities and the velocity update allows to update the positions. In practice, we used a common leapfrog scheme, where velocities and positions are updated in a staggered fashion. This step is parallel by nature as each particle is being assigned an acceleration and therefore a velocity and a position.

From this point, a new density can be computed and a new time step can be started.

These four steps have been adapted for GPUs and, more important, the entire sequence has been integrated for GPUs in order to limit performance losses due to data transfer from/to the CPU or main memory. That means that, as soon as the input data (initial positions and velocities) are calculated or read from a file, the data is sent to the GPU *once and for all* and needs to be brought back on the host only to be written to the output file.

3 Performances

3.1 Setup of the experiments

The subsequent experiments were performed on three sets of initial conditions. First a Plummer sphere [12] where the particles are distributed isotropically and satisfy a well-defined profile. The velocity distribution is chosen in order to balance the gravity and the sphere remains coherent over long period of time. Second, we simulated an exponential disk, where particles are distributed in a thin plane and velocities are similar to the one measured in real disc-like galaxies. This experiment is set up to be unstable by nature, allowing spiral arms to develop for instance. The third type of simulations consists simply in particles randomly distributed in a cubic space with random velocities. It is similar to cosmological simulations that are ignited from a quasi-uniform distribution of matter. All the simulations are performed on the same volume using 32^3 , 64^3 and 128^3 particles/density cells : smaller problems are of little interests while the next power of 8 (256^3) surpasses the current capacity some routines such as CUFFT and CUDPP sort and compact. These larger situations will be addressed on short term as hardware and software improve and our set of simulation already provides a good insight on the perspectives offered by GPUs. The time step is chosen in order to achieve an energy conservation of $\Delta E/E \sim 10^{-3}$ over a time unit, where the energy is defined as $E = \sum_{\text{particles}} \frac{v^2}{2} - \phi(x)$. The tests were run on a GeForce 8800 GTX device.

For comparison, we developed a CPU version of the PM integrator, written in C, compiled using the Intel C compiler. CPU-driven simulations were performed on Opteron Dual-Core at 2.2 GHz, which also hosts the GPU we used. Let us emphasize that *by no means* the CPU version should be considered as fully optimized version, even though loops were optimized by the compiler. The following results should be more considered as a demonstration of the quick gains that can be achieved on GPUs with moderate development skills.

On the CPU, Fourier transforms were performed with the FFTW 3.1.2 library using the single-float precision version and multi-threading was not enabled. On both the GPU and CPU, FFTs were performed using complex-to-complex transform. The multi-grid calculation involved 3 V-cycles with five levels of restrictions, using 5 pre- and post-recursion smoothing steps. It ensures the same

level of energy conservation as the FFT calculation $\Delta E/E \sim 10^{-3}$. The energy fluctuations were found to be identical between the CPU and GPU versions.

Each of the steps of the integrator is timed separately and ran 1000 times. As explained in section 2, we used two different histogramming procedures, one that involves a partial CPU calculation ("GPU Mixed Histo" hereafter) and one "Full GPU" version that suppresses all CPU calculations at the cost of complex sorting and compact routines.

3.2 Overall performance

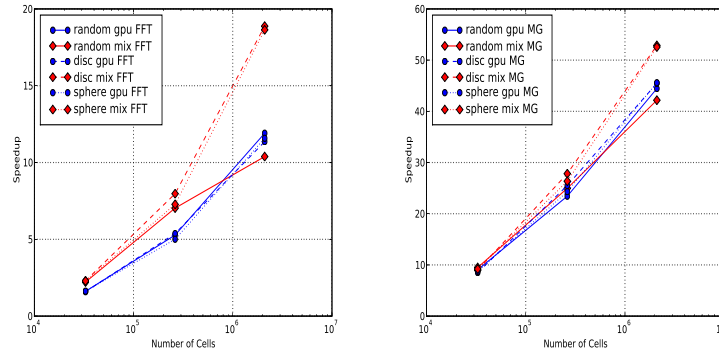


Fig. 2. The overall speedup compared to the CPU version using an FFT (left panel) or a multi-grid (right panel) solver, shown as a function of the number of cells. Please note that y-scales are different in the two panels. Blue lines with circular symbols stand for the full-GPU calculations, while red ones with diamonds stand for the GPU simulations with the histogramming partially performed on the host.

According to our experiments, the GPU versions of the PM integrator can be significantly accelerated, depending of the type of simulation and the number of cells.

As a first broad picture, the figure 2 presents the overall speedup of Full-GPU and GPU-Mixed Histo calculations compared to the CPU version for 32^3 , 64^3 and 128^3 simulations. Using FFTs for the Poisson equations, the gain measured for the Full GPU ranges from a factor 1.6 to 11.5 while the mixed version experiences a speedup from 2.5 for the smallest versions to 18 for the largest simulation. Using the multi-grid relaxation, the speedup is at least a factor 10 for both versions of the histogramming and reaches a level of 43 for the Full-GPU versions and 52 for the GPU-Mix 128^3 calculations. Focusing on the 128^3 simulations, the GPU-mixed versions are almost twice faster than the GPU versions, while Multi-Grid based calculations are almost equivalent if one considers the Full-GPU or GPU-mixed calculations. One can also note that the random calculations are less

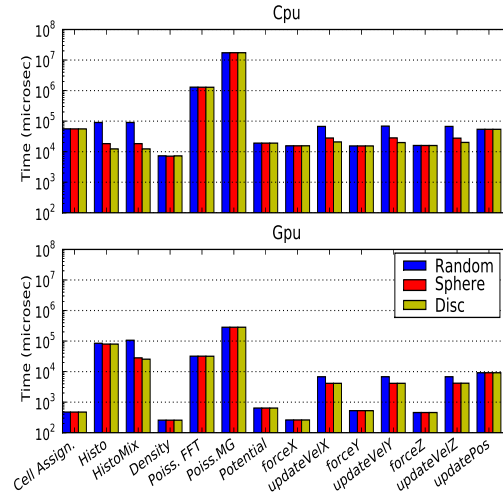


Fig. 3. Absolute timings for the different stages of a single time step for the CPU and the GPU versions. For a given run, histogramming is performed either on the GPU (Histo) or on the CPU (HistoMix). Also the Poisson equation is solved using FFT or Multi-Grid relaxation (MG). For the CPU version, the Histo and HistoMix routines are identical.

efficient when the histogram is performed on the CPU. All these trends result from different limitations and specificities that are detailed hereafter.

3.3 Bottlenecks

CPU and GPU versions are limited by different bottlenecks. It can be seen from Fig. 3 which details the execution times of the different steps involved in a single timestep for the 128^3 simulations. Histogramming on the GPU and Mixed histogramming are shown side by side even if only one of these implementations is used in a given simulation run. The same remark holds for the FFT-based and the MG-based Poisson solvers.

The CPU version depends heavily on the performance of the Poisson solver: about 75 % of a timestep is spent solving the Poisson equation when the FFT solver is called while this proportion is close to a 100% when the MG solver is used.

Meanwhile, the Full-GPU version is more influenced by the density calculation. Histogramming is not naturally suited to SIMD calculations and involves complex operations (sorting/segmented scan). If FFT is used for the Poisson equation, 60 % of its computational time is spent constructing the density. Using MG, the fraction of time spent in the histogramming is lowered to levels of 10% to 20%, but only because MG is less efficient than FFT.

Finally, the Mixed-Histo calculation manages to divide by two the duration of histogramming in the overall calculation compared to the full GPU even though costly transfer are involved (except for the random distribution, see section 3.4). In FFT-based simulations, the same amount of time is then spent in the Poisson solver and in the histogramming. It explains the significant increase of performance observed in the overall calculation when switching to this type of histogram. In MG-based calculation, the weight of the histogramming becomes even less important than it was. A moderate increase in performance is then observed. It is reported in figure 2.

3.4 Sub components analysis

In addition to the absolute timings in Fig. 3, the speedups (compared to the CPU versions) of the sub components of a time step are shown in Fig. 4. From these measurements, we deduce that the overall speedup of the GPU version compared to the CPU results mainly from a large gain in the resolution of the Poisson equation, moderated by the low efficiency of the histogramming and sustained by the speedups achieved in all the other steps of the calculation. The analysis of the individual sub components follow.

Poisson solver From Fig. 3, it can be noted that the resolutions of the Poisson equations are extremely time consuming for the CPU versions and among them the multi-grid calculations are ten times slower than FFTs. The same difference can be noted on the GPU versions, even though the speedup on GPU is 40 (for the FFT) or 60 (for the Multi-Grid). Let us emphasize that the FFT-driven resolution of the Poisson equation involves two Fourier transforms in opposite directions and an isotropic filtering. If we consider only the FFTs, our measurements showed that CUFFT is 2, 16 and 40 times faster than FFTW for the 32^3 , 64^3 and 128^3 experiments. It differs from measurements of [13] with a different GPU but are consistent with the tests of [14].

Important speedups are measured for the multi-grid relaxation, where both GPU and CPU code were written from scratch. Speedup is achieved using the high-level parallelism of the computations involved in the restrictions, prolongations and the Gauss-Seidel iterations. We think greater speedups might be reached by fine-tuning the GPU routines, especially with a greater use of shared memory for the redundant operations of restrictions/prolongation.

Histogramming On the downside, no gain can be observed for the histogramming step on the GPUs. Even worse, this computation can be 5 times slower on 128^3 simulations and can go down to 10 times slower on the full GPU version for 32^3 particles simulations (not studied here otherwise). The GPU-mix histogramming performs the most expensive step on the host but still, the data transfer (transfer rate and latency) results in this computation step being twice slower than computation performed only on the CPU. The mixed version fill the gap

for the sphere and disc simulations, but are at least 15% slower than the CPU versions.

It should be mentioned that higher order assignment schemes exist (CIC, TSC) where particles contribute to more than one cell [6], resulting in the calculation of 8(CIC) or 27 (TSC) histograms per timestep. The CPU or mixed-GPU may suffer from these successive calculations. Conversely, the algorithm described by [8] requires only one sort and may become more competitive as higher assignment schemes are used.

Interestingly, the random case simulations are less favorable to the CPU versions in terms of histogramming: it cannot be as efficient as it writes data in memory due to the fact that particles are spread in all the computation box, presumably due to cache misses. In the two previous cases, the particles were confined in certain sub regions (disc or sphere), ensuring a certain level of cache hits. Meanwhile, the SIMD GPU histogramming routine relies on sort/scan primitives which do not depend strongly on the particle distribution.

Accelerations and updates All the other stages of the calculation are significantly speeded up on GPU, with speedups ranging from 5 to 120 compared to the CPU versions. We noticed that the speedups of velocity updates increase as the particles are spread in a larger portion of the grid, especially comparing disc simulations to random simulations. To compute the velocity of a particle, a given thread (or the CPU) finds the cell it belongs to and uses the associated acceleration. If particles are strongly clustered, memory access scheme gets close to neighboring accesses to the memory, which are more likely to be cached on the CPU. Consequently, speedups are larger as the CPU fails to cache the memory accesses, as it is the case for the random simulation. Also, computing the force/acceleration is faster along the x direction and results from the storage of the grids in memory which favors certain directions in terms of contiguous memory access among threads.

4 Conclusion and perspectives

Using CUDA API we developed a Particle-Mesh N-Body integrator that runs on common graphic devices. All the steps of the algorithm were parallelized and we obtain speedups that ranges from 2 to 50 depending on the size of the problem and the choice of techniques. The density computation and the Poisson solver are the critical part of the implementation. We implemented a full GPU histogramming algorithm and solve the Poisson equation by means of FFT and MG relaxation. For large problems, the resolution of the Poisson equation is at least 40 times faster on the GPU using FFT and 60 times faster using multi-grid relaxation, while the histogram computation is hardly accelerated on GPU. Combined with the 10-100 speedups obtained on all the other steps (cell assignment, acceleration computation/interpolation, velocity/position update) a significant acceleration of the code is observed : for a typical 128^3 data set we achieve speedups of 12 for FFT based calculations and up to 45 for MG based

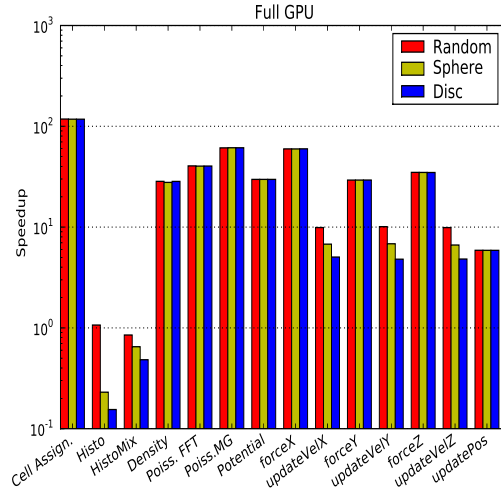


Fig. 4. Speedup for the different stages of a single time step for the the GPU-Mix and full GPU simulations on a 128^3 grid.

ones. If the histogramming is partially performed on the host, higher speedups of 20 for FFT-based and 50 for MG-based simulations are observed.

In a near future, we plan first to assess larger problems (256^3 , 512^3) in order to reach astrophysically relevant resolutions. Using devices with larger memory capabilities and supporting atomic operations, it should be within our reach using the Multi-Grid Poisson solver and an improved version of the histogramming routine (following e.g. [15]). If such large situations can be handled, running large multi-GPU simulations would be the next step to reach the billion particles with GPU speedups. However, it remains still unclear how network based communication may lower the speedups obtained with a single device.

From an astrophysical point of view, the results obtained are encouraging. Furthermore, this PM development is accompanied by two other codes : GPU version of a non-linear FAS Multi-Grid solver for the modified Newtonian dynamics and the CUDA transcription of a cosmological radiative transfer code. For these two other codes, speedups range from 10 to 80 compared to CPU versions. It opens interesting perspectives of an easy access to HPC-like calculations in their desktop machines with a set of well-suited API's enabled for GPUs.

Acknowledgments

D.A. would like to thank R. Teyssier for patience and fruitful comments along the course of this development and T.Keller for technical support. This work is supported by a grant from the *Conseil Scientifique* de l'Université Louis Pasteur.

References

1. V. Springel and al., “Simulations of the formation, evolution and clustering of galaxies and quasars,” Nature, vol. 435, pp. 629–636, Jun. 2005.
2. I. T. Iliev and al., “Simulating Cosmic Reionization,” ArXiv e-prints, vol. 806, Jun. 2008.
3. R. Teyssier and al., “Full-Sky Weak Lensing Simulation with 70 Billion Particles,” ArXiv e-prints, vol. 807, Jul. 2008.
4. S. F. Portegies Zwart, R. G. Belleman, and P. M. Geldof, “High-performance direct gravitational N-body simulations on graphics processing units,” New Astronomy, vol. 12, pp. 641–650, Nov. 2007.
5. T. Hamada and T. Iitaka, “The Chamomile Scheme: An Optimized Algorithm for N-body simulations on Programmable Graphics Processing Units,” ArXiv Astrophysics e-prints, Mar. 2007.
6. R. W. Hockney and J. W. Eastwood, Computer simulation using particles. Bristol: Hilger, 1988, 1988.
7. E. Bertschinger, “Simulations of Structure Formation in the Universe,” ARAA, vol. 36, pp. 599–654, 1998.
8. R. Ferrell and E. Bertschinger, “Particle-Mesh Methods on the Connection Machine,” Contributions to Mineralogy and Petrology, pp. 10 002–+, Nov. 1993.
9. R. Brada and M. Milgrom, “Stability of Disk Galaxies in the Modified Dynamics,” ApJ, vol. 519, pp. 590–598, Jul. 1999.
10. T. Sorensen, T. Schaeffter, K. Noe, and M. Hansen, “Accelerating the nonequispaced fast fourier transform on commodity graphics hardware,” IEEE Transactions on Medical Imaging, vol. 27, no. 4, pp. 538–547, Oct. 2008.
11. D. Goddeke, R. Strzodka, and S. Turek, “Performance and accuracy of hardware-oriented native-, emulated-and mixed-precision solvers in fem simulations,” Int. J. Parallel Emerg. Distrib. Syst., vol. 22, no. 4, pp. 221–256, 2007.
12. J. Binney and S. Tremaine, Galactic Dynamics: Second Edition. Galactic Dynamics: Second Edition, by James Binney and Scott Tremaine. ISBN 978-0-691-13026-2 (HB). Published by Princeton University Press, Princeton, NJ USA, 2008., 2008.
13. H. Merz. (2008) CUFFT Vs FFTW Comparison www.science.uwaterloo.ca/~hmerz/CUDAbenchFFT/ -.
14. P. Demores. (2007) GPU Benchmarking www.cv.nrao.edu/~pdemores/gpu/.
15. G. Stantchev, W. Dorland, and N. Gumerov, “Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu,” Journal of Parallel and Distributed Computing, vol. 68, no. 10, pp. 1339–1349, Oct. 2008.