Computational Star Formation
Proceedings IAU Symposium No. 270, 2011
J. Alves, B.G. Elmegreen, J. M. Girart & V. Trimble, eds.

doi:10.1017/S1743921311000706

Numerical Cosmology powered by GPUs

Dominique Aubert[1]

[1]Observatoire Astronomique, Universite de Strasbourg, CNRS,UMR 7550
11 rue de l'Universite, 67000 Strasbourg France
email: dominique.aubert@astro.unistra.fr

Abstract. Graphics Processing Units (GPUs) offer a new way to accelerate numerical calculations by means of on-board massive parallelisation. We discuss two examples of GPU implementation relevant for cosmological simulations, an N-Body Particle-mesh solver and a radiative transfer code. The latter has also been ported on multi-GPU clusters. The range of acceleration (x30-x80) achieved here offer bright perspective for large scale simulations driven by GPUs.

Keywords. methods: n-body simulations, numerical, cosmology

1. GPUs for scientific applications

The last few years have seen the rise of a new technique for parallel calculations which relies on graphics processing units (GPUs hereafter). This type of hardware was originally designed and optimized for applications related to graphics display such as 3D rendering. Compared to the classical CPUs, the GPU architecture favors 'calculation units' against cache and control flow units. It originates from the fact that graphical calculations are numerically intensive but the same set of operations is identically applied to a large number of data (like e.g. polygons rotations). Ideally, the lack of cache is compensated by the intensity of calculations and control flow is unnecessary since the same operations are applied to all data. Furthermore, graphics boards typically contain a few hundreds of multicore units, implying that a large number of clone calculations can be dealt in parallel. If a scientific application can fit in this model, one can expect a significant acceleration.

Due to their original field of applications, GPUs were at first limited to single-float calculations but the latest generation boards are also able to perform double-precision calculations. Single boards can achieve a peak performance of 1.5 Tflop/s in single precision and 500 Gflop/s in double precision. Theoretical bandwidth can be as high as 180 GB/s. Applications which have been ported on GPUs experience typically a x10 to x100 acceleration compared to a single CPU-core calculation. The final improvement rate is dependent on several factors: the suitability of a given application to the GPU programming model, the amount of communication between the host PC and the board and of course the level of optimisation such as memory access patterns.

Several options exists to interact with this hardware for scientific calculations. Originally, the scientists literally wrote their applications in order to mimic graphics calculations, with e.g. shaders. Hopefully, more user friendly solutions exists today. Currently, the most popular one is CUDA, written by the Nvidia company exclusively for its devices. CUDA is a toolkit which contains a compiler plus several libraries and development tools. In practice, one writes a regular C or Fortran code coupled to a set of libraries which provide primitives to interact with the devices, mostly for communications, launching calculations and organizing the parallel task among the data. More recently, the OpenCL standard has been defined by the Khronos group: this standard allows to program on

any multicore architecture in an unified way and is not limited to the devices designed by a specific company.

2. Particle-Mesh N-Body integrator

The particle-mesh (PM) technique relies on a grid-based description of the gravitational field created by a distribution of massive particles. This method is fast but suffers from a low spatial resolution and is usually coupled to a more accurate technique such as direct n-body (PPM) or tree-based calculations (TPM). Once the initial particle distribution is known, the density is calculated on a fixed grid. The Poisson equation is solved usually by means of Fourier-space technique or relaxation and the resulting potential provides the force field at the grid nodes. The force is interpolated at particle positions and the latter can be moved and prepared for the next timestep.

In Aubert, Amini & David 2009, we described a GPU implementation of this technique using CUDA and further technical details can be found in this publication. In our implementation, the Poisson equation can be solved either by FFT (using the cuFFT library) or by our own implementation of the multigrid relaxation technique. Interestingly most of the steps of a PM calculation consist in identical operations performed independently on a large set of data. For instance, updating the velocity of particles can easily be done in parallel. Also the restriction or prolongation operations in the multigrid solver can be performed cell-by-cell without any communication. Even the relaxation can be performed on the cells of the grid completely in parallel. Overall, any operation restricted to the particle-based description or the grid-based description can be fully executed by independent and parallel threads. As a consequence large accelerations (x50 -x100) can be achieved on these operations on GPUs. This is illustrated by the Figure 1.

However, the PM technique implies that several switches between the particle based and the grid based description should be performed. An example is the histogramming step, where particles are projected on a grid using e.g. a CIC interpolating scheme. A first issue is related to the fact that particles can hit the same cell, implying that the number of hits in a given cell cannot be updated in parallel. Moreover particles are distributed 'randomly' in the computational volume leading to non ordered memory accesses in order to update the grid, killing the GPU performances. The same issues are encountered when interpolating the force computed on a grid back on particles. Interestingly, these points were already tackled when vector-based architectures were the norm and a way to circumvent these problems are described e.g. in Ferrel & Bertschinger (1994). Using brute force with atomic operations which serialize

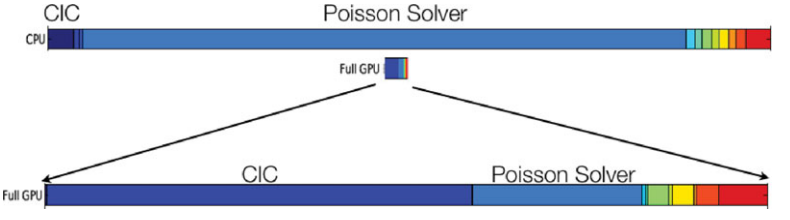

Figure 1. A typical PM timestep on CPU and GPU. Dark blue stand for the time spent in the histogramming step while light blue stands for the resolution of the Poisson equation. The other colors stand for force calculation and position update and are negligible. The bars are on scale and the GPU is globally 30 times faster than CPU, but the relative importance of CIC and Poisson resolution is different.

https://doi.org/10.1017/S1743921311000706 Published online by Cambridge University Press

accesses when required is also an option. In any case the operation remains quite GPU-unfriendly.

As a consequence, the relative importance of the tasks involved in PM calculations differs from classic CPU calculations. Figure 1 compare the two cases: overall a x30 acceleration is achieved on GPU but the importance of the Poisson resolution is reduced compared to the CPU implementation, whereas the CIC-histogramming step becomes dominant. It is due to the fact that the Poisson resolution experiences a better benefit from the GPU implementation than the CIC step. The Poisson resolution fits into the GPU programming model while the histogramming is intrinsically more difficult to parallelise.

3. Cosmological Radiative transfer on multi-GPUs

The second application is described in full details in Aubert & Teyssier (2008) and Aubert & Teyssier (2010). It consists in a cosmological radiative transfer code, called ATON. Given a gas density distribution and a collection of sources, this code solves the radiative transfer using the moment-based M1 technique (Levermore 1984) and computes the chemistry and heating created by the ionizing radiation. Radiation is described as a fluid on a fixed grid and its transport is computed using Godunov-like techniques in an *explicit fashion*. Thanks to the latter choice, the radiative state of cell can be updated in parallel and using the same set of operations, fitting perfectly in the GPU programming model. Chemistry and photo-heating process are purely local and can also be computed in a fully parallel fashion. More generally all hyperbolic solvers are perfectly adequate for GPU-based calculations. For instance, using CUDA, ATON is 80x faster on a GeForce 8800 GTX than on a Opteron 2.7 GHz. It should be noted that such an acceleration is mandatory to use ATON: because of the explicit resolution, the Courant condition on the speed of light forces the calculation to operate on a large number of timesteps. Without GPUs, such a technique would be extremely limited in practice.

Another feature of ATON is that it runs on more than one GPU using an additional MPI layer. Practically, the GPUs are seen only by a single host but the host communicate through regular MPI communications. Each GPU is assigned a sub-volume of the global computational volume and the exchanges between subvolumes are implemented by exchanging ghost layers between the GPUs: the graphics boards send the data on their hosts, the hosts communicates and once ready they send the exchanged data back on their GPUs.

As an application we performed a study of the cosmological reionisation in a set 1024^3 cosmological simulations (see Aubert & Teyssier 2010). The number of required timesteps to evolve the system from $z \sim 15$ to $z = 5.5$ lie between 50 000 and 150 000 depending on the physical size of the simulated volume. These calculations were performed on 128 Tesla GPUs hosted by the Titane Supercomputer of the CEA. Thanks to the acceleration provided by GPUs, such calculations are completed in 3-10 hours. It allowed us to assess the issue of communication costs: since all the exchanged data must go through the PCI-bus, a legitimate question would be to understand how much is lost in communications. Globally, it should be noted that the speedup remains reasonably linear up to 128 devices. Furthermore, our measurements for different parallel configurations show that 10 - 15% of a calculation is spent in MPI+PCI communications. While significant, it remains reasonable and demonstrates the usability of GPUs for large scale multi-device calculations.

https://doi.org/10.1017/S1743921311000706 Published online by Cambridge University Press

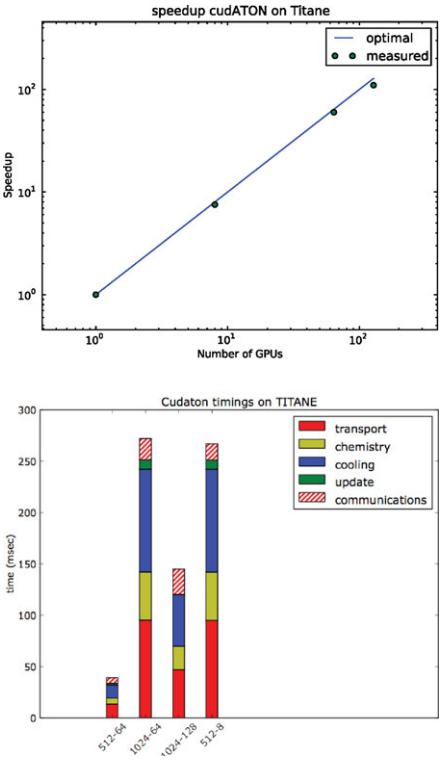

Figure 2. Top: speedup curve achieved by the radiative transfer code ATON as a function of the number of GPUs. Bottom: time spent in the different stages of radiative timestep of aton. The communications are limited to 15% fraction.

4. Conclusion

In addition to these two applications (PM and radiative transfer) we also implemented on GPUs a standard hydrodynamical solver on fixed grid, achieving an acceleration close to x100. It should be noted that such accelerations are by no means exceptional for GPU codes and are obtained in a large range of applications from graphics rendering, to medical applications or genomics. It should also be emphasized that GPU-programming is fairly easy to learn, however the difficulty lies in fitting existing applications into this specific programming model, which can be fairly different than the usual CPU programming, less flexible and closer to the hardware. These difficulties set aside, significant accelerations can be quickly obtained for a moderate developing cost.

References

Aubert, D., Amini, M., & David, R. 2009, *LNCS*,5544,874
Ferrel, E. & Bertschinger, E. 1994, *Int. J. Mod. Phys.*, 933
Aubert, D. & Teyssier, R. 2008, *MNRAS*, 387, 295
Aubert, D. & Teyssier, R. 2010, submitted to *ApJ*, arxiv:1004.2503
Levermore, C. 1984, *Journal of Quantitative Spectroscopy and Radiative Transfer*, 31, 149

https://doi.org/10.1017/S1743921311000706 Published online by Cambridge University Press