INAF-OATs Technical Report 223: Direct *N-body* code designed for heterogeneous platforms

D. Goz[a], L. Tornatore[a], S. Bertocco[a], G. Taffoni[a]

[a]*INAF-Osservatorio Astronomico di Trieste, Via G. Tiepolo 11, 34131 Trieste - Italy*

Abstract

The ExaNeSt H2020 project aims at the design and development of an exascale ready supercomputer with a low energy consumption profile but able to support the most demanding scientific and technical applications. The project will produce a prototype based on hybrid hardware (CPUs+accelerators) implementing a co-design approach where scientific applications requirements are driving the hardware design. Astrophysical codes are playing a fundamental role to validate the exascale platform. We present strategies adopted in order to port on heterogeneous platform a state-of-the-art N-body code (called ExaHiGPUs) based on high order Hermite's integration scheme with a direct evaluation of pair-wise particle forces.

Keywords: `ExaNeSt`, N`-body solvers`, `CPU`, `GPU`, `accelerators`, `heterogeneous architecture`, ExaHiGPUs

1. Introduction

The usage of heterogeneous computing in scientific research appears to be inevitable. Indeed, computing power in existing peta-scale machines already is mainly issued by accelerators, and this will be exacerbated even further on the future exa-scale platforms that will involve millions of specialized parallel

Email addresses: `david.goz@inaf.it, ORCID:0000-0001-9808-2283` (D. Goz), `luca.tornatore@inaf.it, ORCID:0000-0003-1751-0130` (L. Tornatore), `sara.bertocco@inaf.it, ORCID:0000-0003-2386-623X` (S. Bertocco), `giuliano.taffoni@inaf.it, ORCID:0000-0002-4211-6816` (G. Taffoni)

compute units. Thus, algorithms will be designed for general systems with different devices, and likely with complex memory hierarchies.

Various accelerators are employed in heterogeneous computing, including highly parallel devices, such as Graphic Processor Units (GPUs) and Field Programmable Gate Arrays (FPGAs). GPUs offer high floating-point throughput and memory bandwidth than multi-core Central Processing Units (CPUs).

The ExaNeSt project will produce a prototype based on low power-consumption ARM64 processors, FPGAs as accelerators and low-latency interconnections implementing a co-design approach where scientific applications requirements are driving the hardware design [1].

For programming on heterogeneous platforms, programmers can employ OpenCL[1] language, facing the challenge of writing efficient code for hybrid (CPUs+accelerators) architecture.

There exist many scientific problems which would greatly benefit from hybrid resources. In this report we investigate the feasibility of implementing state-of-the-art direct N-body code on heterogeneous hardware.

1.1. N-body solvers running on hybrid computing platforms

N-body solvers provide the backbone for different scientific and engineering applications, such as astrophysics, nuclear physics, molecular dynamics, fluid mechanics and biology. The numerical solution of the direct N-body problem is still considered a challenge despite the significant advances in both hardware technologies and software development. The main drawback related to the direct N-body problem relies on the fact that the algorithm requires $O(N^2)$ computational cost. In practice many variations of the naive algorithm are used, for instance, implementing high order Hermite integration schemes [2] and block time-stepping. These variations can eliminate most of the standard parallelization method for N^2 algorithm, requiring huge effort to maximize performance.

There are some N-body codes designed to speed up the classical N-body

[1] www.khronos.org/opencl/

problem for astrophysics using GPUs, for example, φGPU code [3], φGRAPE code [4], NBODY6 code [5], MYRIAD code [6], and HiGPUs code [7, 8, 9]. In all of them, the GPU is fed by the host CPU with the gravity equation of data in the form of coordinates, velocities and masses of particles, and it handles calculating the forces for the data points.

1.2. Motivation

An intensive co-design is essential to build an Exascale system. Applications must be re-engineered to exploit the ExaNeSt platform based on heterogeneous architecture.

The work presented in this report aims to devise a new implementation, called ExaHiGPUs, tailored for heterogeneous architecture of the direct N-body code HiGPUs [10, 7, 9]. The choice of the application is motivated by the fact that HiGPUs is a state-of-the-art direct N-body code, based on the Hermite 6th order time integrator, that has been widely used for simulations of star clusters with up to ~ 8 million bodies [11, 12], and of galaxy mergers [13].

2. Implementation

The 6th order Hermite integration scheme consists of three stages (fully described in [2]): a predictor step that predicts particle's positions and velocities; an evaluation step to evaluate new accelerations, their first order (*jerk*), second order (*snap*), and third order derivatives (*crackle*); a corrector step that corrects the predicted positions and velocities using the results of the previous steps.

In the following we describe our implementation that follows the one implemented in the HiGPUs[2] code, but with substantial modifications aimed to fully exploit heterogeneous architecture.

1. **DEVICE EXPLOITABLE**: HiGPUs code has been designed to exploit the compute capabilities provided by GPUs. The entire Hermite scheme is

[2] http://astrowww.phys.uniroma1.it/dolcetta/HPCcodes/HiGPUs.html

implemented and optimized using OpenCL kernels in such a way to fully exploit this kind of devices[3]. Since the design of OpenCL is such that kernels can be run in a wide range of architectures, we decided to drop specific GPU optimizations as implemented in the original code allowing us to test the Hermite integrator on any OpenCL-compliant device (CPUs or accelerators).

2. **PARALLELIZATION SCHEME**: a one-to-one correspondence between MPI processes and computational nodes is established and each MPI process manages all the OpenCL-compliant devices of the same type available per node (device type is selected by the user). Inside of each shared-memory computational node parallelization is achieved by means of OpenMP environment. Hence, in our implementation, the host code is parallelized with hybrid MPI+OpenMP programming, while the device code is parallelized with OpenCL. The user is allowed to choose at compile time if the application uses MPI or OpenMP, or both, or neither.

3. **DECOMPOSITION OVER HOST AND DEVICE**: The Hermite integration is performed on the selected OpenCL-compliant device(s). Our algorithm uses a share-time step scheme that integrates all particles, while the original implementation adopts a block time-step scheme [14]. Thus, in a simulation with N-particles using n devices, during the evaluation stage each device deals with N/n particles and evaluates $N(N/n)$ accelerations and their derivatives, subsequently collected and reduced from the all set of computational nodes. In our implementation the evaluation of the time step, by means of the so called *generalized Aarseth criterion* [2], the total energy and the angular momentum of the system are performed on the device as well. The latest quantities are periodically evaluated during the simulation in order to check the accuracy of the integration scheme.

Our implementation requires that particle data is communicated between

[3]The optimization strategy to guarantee an optimal load of the GPU is described in detail in [7]

the host and the device at each share-time step, which gives rise to synchronization points between host and device(s). Accelerations, *jerk*, *snap* and time step computed by the device(s) are retrieved by the host on every computational node, reduced and then sent back again to the device(s).

4. **KERNEL OPTIMIZATIONS**: OpenCL kernels have access to distinct memory regions distinguished by access type and scope. *Local* memory provides read-and-write access to work-items within the same work-group and it is specifically designed to reduce the latency of data transactions. The *evaluation* kernel is the computationally most expensive part of the Hermite algorithm. This makes the calculation of the accelerations a good candidate for exploiting the local memory of the device. When the *evaluation* kernel is issued to the device(s), each work-item goes through the following steps:

   i) each work-item in the work-group caches one particle from *global* memory into the *local* memory. The total number of cached particles is therefor equal to the work-group size (user free parameter);

   ii) partial acceleration, *jerk* and *snap* for each work-item are calculated and stored in registers using particles cached in local memory;

   iii) steps i) and ii) are repeated until all particles handled by the device have been read (avoiding to sum up the self interaction);

   iv) results are stored in global memory ready to be read by the host.

   The previous scheme implies $N(N/n)$ calculations performed by the device, requiring internal synchronization due to the fact that *local* memory is limited. However, the exploiting of local *memory* is generally accepted as the best method to reduce global memory latency.

5. **KERNEL VECTORIZATION**: since the majority of OpenCL-compliant devices supports vector instruction set, we vectorized all kernels of the application. Vectorizing code can effectively improve memory bandwidth because of regular memory access, better coalescing of these memory accesses and reducing the number of loads/stores (each load/store is larger).

6. **PRECISION**: double-precision (DP) in inter-particles distance and acceleration is mandatory in order to minimize the round-off errors. However, while the full IEEE-compliant DP arithmetic is increasingly efficient in available GPUs, it is still extremely resource-eager and performance-poor in other accelerators like FPGAs. As an alternative, the extended-precision (EX) numeric type can represent a trade-off in porting our applications on devices not specifically designed for scientific calculations. An EX-number provides approximately 48 bits of mantissa at single-precision exponent ranges. Our application can be run using DP, EX or single precision (SP) arithmetic (user-defined at compile time). EX-arithmetic is implemented as proposed in [15].

3. Results

The host hardware we used to develop and validate the serial application (single node, i.e. without using MPI) is a workstation with one Intel Core i7-3770 running at 3.40 GHz and one NVIDIA GeForce GTX 1080 graphics card in the PCI Express (16×) bus. The workstation runs a Linux Ubuntu SMP kernel version 4.15.0-20 generic and graphics card driver NVIDIA 390.48.

We compare the energy E and the angular momentum L of the N-body system during the simulation with the values at the start of the simulation. Latest quantities must remain constant within an isolated system. We determine the relative errors $\Delta E/E$ and $\Delta L/L$ using the following equations:

$$\frac{\Delta E}{E} = \frac{|E_{start} - E(t)|}{E_{start}}, \quad \text{and} \quad \frac{\Delta L}{L} = \frac{|L_{start} - L(t)|}{L_{start}} \tag{1}$$

where E_{start}, L_{start} and $E(t)$, $L(t)$ are the energy and the angular momentum at the start and at a given time of the simulation, respectively.

Figure 1 and figure 2 show the relative errors of energy and angular momentum of the system as a function of time (in code unit), respectively. The simulation was carried out with 4096 particles. As relevant result, adopting SP-arithmetic, round-off error accumulates during the simulation, while it is

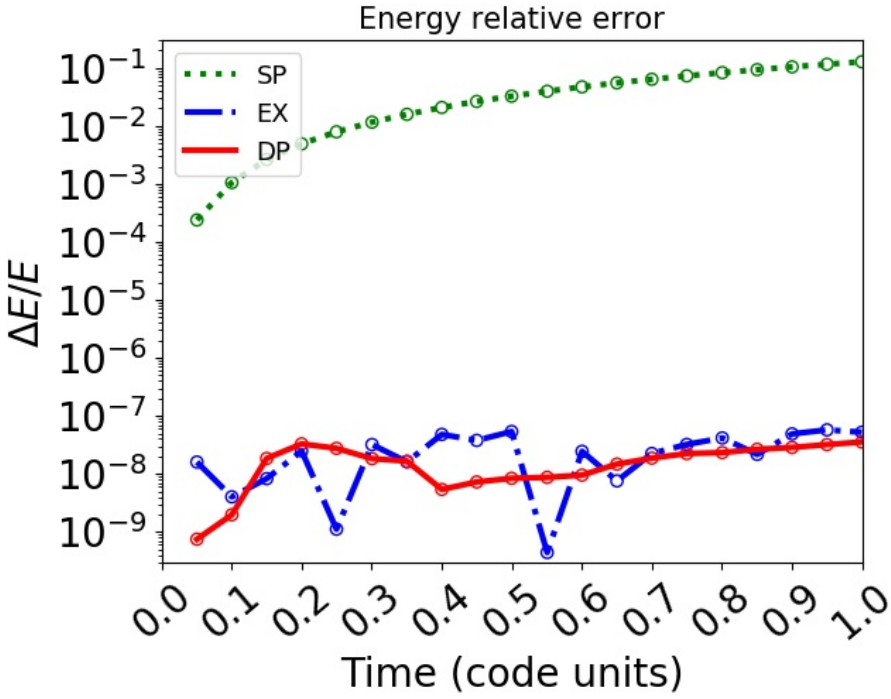

Figure 1: Relative error $\Delta E/E$ for DP-arithmetic (continuous red line), EX-arithmetic (dot-dashed blue line) and SP-arithmetic (dotted cyan line) as a function of the integration time (in code unit).

roughly constant using EX or DP-arithmetic. The test presented suggests that EX-arithmetic can be adopted for N-body problem ensuring to keep control over the accumulation of the round-off error during the simulation. This approach allow us to require only SP compute capability to the accelerator. Future tests will be aimed to verify the parallel implementation of ExaHiGPUs across computational nodes.

4. Conclusions

Astronomers will be forced to re-engineer their applications in order to exploit new Exascale computing facilities based on heterogeneous hardware

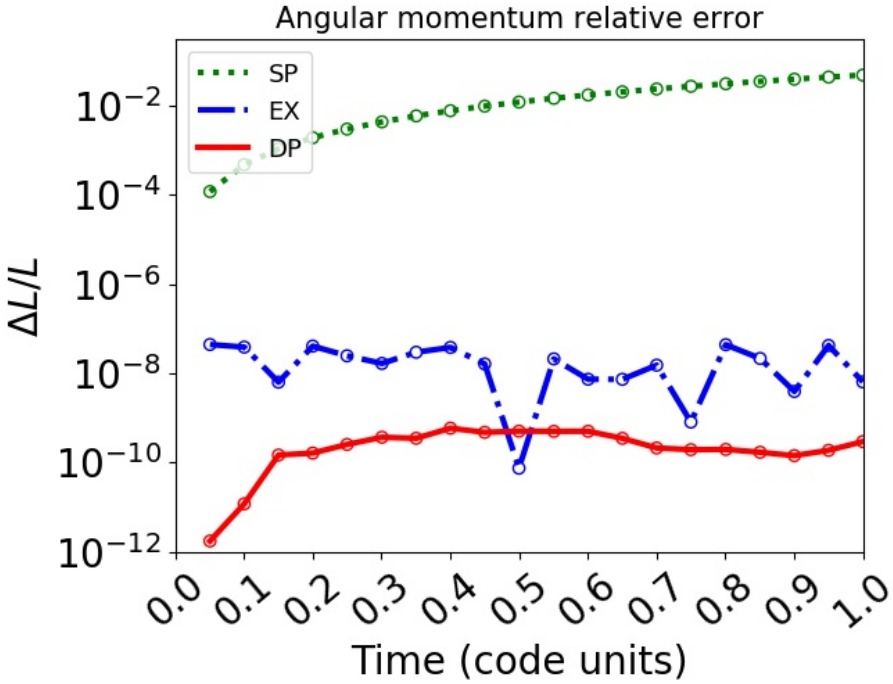

Figure 2: Relative error $\Delta L/L$ for DP-arithmetic (continuous red line), EX-arithmetic (dot-dashed blue line) and SP-arithmetic (dotted cyan line) as a function of the integration time (in code unit).

(CPUs/GPUs/FPGAs). We devised and validated a direct N-body integrator, based on the 6th order Hermite integration scheme, designed for heterogeneous platforms.

In future work we will describe and test the parallel implementation of ExaHiGPUs in a computational cluster.

5. Acknowledgments

This work is supported by the European Union's Horizon 2020 Research and Innovation Programme under the ExaNeSt project (Grant Agreement No. 671553).

This research has been made use of IPython [16], Scipy [17], Numpy [18] and MatPlotLib [19].

References

[1] G. Taffoni, G. Murante, L. Tornatore, D. Goz, S. Borgani, M. Katevenis, N. Chrysos, M. Marazakis, Shall numerical astrophysics step into the era of Exascale computing?, in: Astronomical Data Analysis Software an Systems XXVI (ADASS XXVI), Astronomical Society of the Pacific Conference Series, 2016.

[2] K. Nitadori, J. Makino, Sixth- and eighth-order Hermite integrator for N-body simulations, New Astronomy 13 (2008) 498–507. arXiv:0708.0738, doi:10.1016/j.newast.2008.01.010.

[3] P. Berczik, K. Nitadori, S. Zhong, R. Spurzem, T. Hamada, X. Wang, I. Berentzen, A. Veles, W. Ge, High performance massively parallel direct N-body simulations on large GPU clusters., in: International conference on High Performance Computing, Kyiv, Ukraine, October 8-10, 2011., p. 8-18, 2011, pp. 8–18.

[4] S. Harfst, A. Gualandris, D. Merritt, R. Spurzem, S. Portegies Zwart, P. Berczik, Performance analysis of direct N-body algorithms on special-purpose supercomputers, New Astronomy 12 (2007) 357–377. arXiv:astro-ph/0608125, doi:10.1016/j.newast.2006.11.003.

[5] K. Nitadori, S. J. Aarseth, Accelerating NBODY6 with graphics processing units, MNRAS 424 (2012) 545–552. arXiv:1205.1222, doi:10.1111/j.1365-2966.2012.21227.x.

[6] S. Konstantinidis, K. D. Kokkotas, MYRIAD: a new N-body code for simulations of star clusters, Astronomy and Astrophysics 522 (2010) A70. arXiv:1006.3326, doi:10.1051/0004-6361/200913890.

[7] R. Capuzzo-Dolcetta, M. Spera, D. Punzo, A fully parallel, high precision, N-body code running on hybrid computing platforms, Journal of Computational Physics 236 (2013) 580–593. arXiv:1207.2367, doi:10.1016/j.jcp.2012.11.013.

[8] R. Capuzzo-Dolcetta, M. Spera, A performance comparison of different graphics processing units running direct N-body simulations, Computer Physics Communications 184 (2013) 2528–2539. arXiv:1304.1966, doi:10.1016/j.cpc.2013.07.005.

[9] M. Spera, Using Graphics Processing Units to solve the classical N-body problem in physics and astrophysics, ArXiv e-printsarXiv:1411.5234.

[10] R. Capuzzo-Dolcetta, M. Spera, D. Punzo, A fully parallel, high precision, N-body code running on hybrid computing platforms, Journal of Computational Physics 236 (2013) 580–593. arXiv:1207.2367, doi:10.1016/j.jcp.2012.11.013.

[11] M. Spera, R. Capuzzo-Dolcetta, Rapid Mass segregation in small stellar clusters, ArXiv e-printsarXiv:1501.01040.

[12] M. Spera, M. Mapelli, A. Bressan, The mass spectrum of compact remnants from the PARSEC stellar evolution tracks, MNRAS 451 (2015) 4086–4103. arXiv:1505.05201, doi:10.1093/mnras/stv1161.

[13] E. Bortolas, A. Gualandris, M. Dotti, M. Spera, M. Mapelli, Brownian motion of massive black hole binaries and the final parsec problem, MNRAS 461 (2016) 1023–1031. arXiv:1606.06728, doi:10.1093/mnras/stw1372.

[14] S. L. W. McMillan, S. J. Aarseth, An O(N log N) integration scheme for collisional stellar systems, The Astrophysical Journal 414 (1993) 200–212. doi:10.1086/173068.

[15] A. Thall, Extended-precision floating-point numbers for gpu computation (2006) 52doi:10.1145/1179622.1179682.

[16] F. Perez, B. Granger, Ipython: A system for interactive scientific computing, Computing in Science Engineering 9 (3) (2007) 21–29. doi:10.1109/MCSE.2007.53.

[17] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, [Online; accessed 2015-09-13] (2001–).
URL http://www.scipy.org/

[18] S. van der Walt, S. Colbert, G. Varoquaux, The numpy array: A structure for efficient numerical computation, Computing in Science Engineering 13 (2) (2011) 22–30. doi:10.1109/MCSE.2011.37.

[19] J. Hunter, Matplotlib: A 2d graphics environment, Computing in Science Engineering 9 (3) (2007) 90–95. doi:10.1109/MCSE.2007.55.