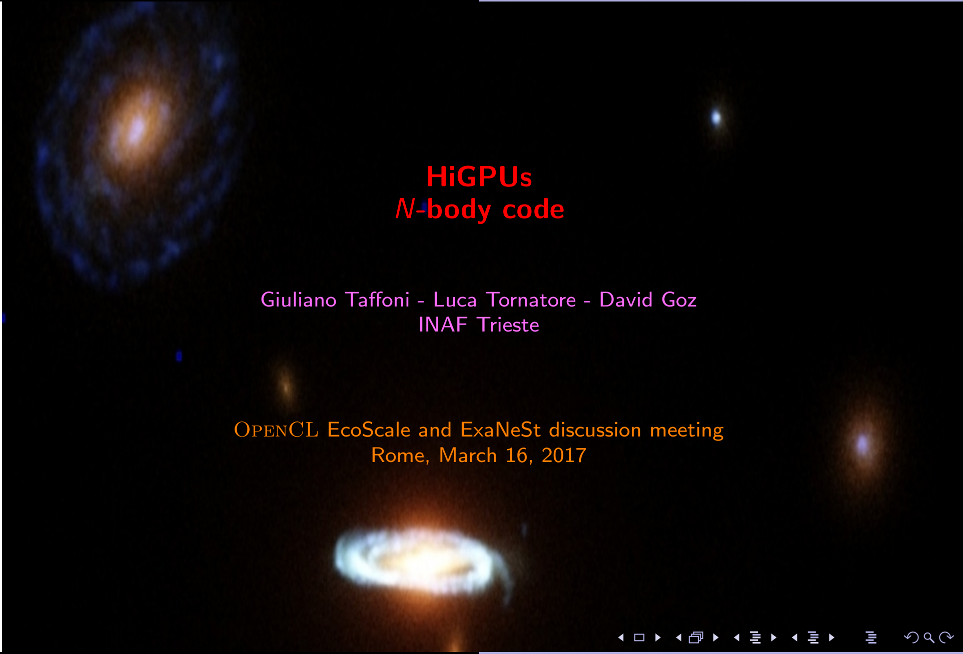


HiGPUs *N*-body code

Giuliano Taffoni - Luca Tornatore - David Goz
INAF Trieste

OpenCL EcoScale and ExaNeSt discussion meeting
Rome, March 16, 2017

N-body problem

The study of the motion of N point-like masses having initial positions and velocities $\mathbf{r}_{i,0}$, $\mathbf{v}_{i,0}$, $i = 1, 2, ..., N$ interacting through a pair-wise force that depends only on their positions is known as the N-**body problem**.

Its applications can be found on both small and large scales starting from nuclear physics up to astrophysical problems.
In this latter case, the interacation force is gravity, and the problem is referred as the *classical gravitational newtonian N-body problem*.

As a matter of fact, the gravitational N-body problem is explicitly solvable only for $N < 3$ while, for $N \geq 3$, the procedure to get a solution is exclusively numerical.

Two types of N-body algorithms:

- the so-called *collision-less*, where a body sees the background potential of the rest of the stellar system (e.g. Barnes-Hut treecode which scale as $O(N \log N)$ with N bodies);
- expensive *collisional* one or *direct summation*, in which all gravitational forces are integrated for all bodies to take into account the graininess of the potential and individual time step; they tipically scale as $O(N^2)$.

GPUs can very efficiently be used for this kind of problem, due to their highly parallel structure and computational speed (see e.g. the work of Portegies Zwart et al. 2007; Hamada & Iitaka 2007; Belleman et al. 2008; Berczik et al. 2011; Nitadori & Aarseth 2012; Capuzzo-Dolcetta et al. 2013, Berczik et al. 2013).

The numerical solution of the N-body problem

The newtonian interaction potential between a point mass m_i and another mass m_j is given by:

$$U_{ij} = \frac{Gm_i m_j}{\mid \mathbf{r}_j - \mathbf{r}_i \mid} \equiv \frac{Gm_i m_j}{r_{ij}} = U_{ji}$$

where $\mathbf{r}_i$ and $\mathbf{r}_i$ are the position vectors of the i-th and j-th bodies, G is the gravitational constant and $r_{ij} \equiv \mid \mathbf{r}_j - \mathbf{r}_i \mid$ represents the euclidean distance betweeen the two bodies.
There is the so called *double-divergence* of the two-body interaction potential:

- close encounters ($r_{ij} \to 0$) yield to an unbound force between interaction bodies ($F_{ij} \to 0$) producing an unbound error in the relative acceleration. This divergence is often faced introducing a *softening parameter*, ϵ, in the interaction potential which becomes (loss of resolution at scales of the order of ϵ and below)

$$U_{ij} = \frac{Gm_i m_j}{\sqrt{r_{ij}^2 + \epsilon^2}}$$

- the resultant force acting on every body of an N-body-system requires summation over $N - 1$ pair-wise contributions, yielding to an $O(N^2)$ computational complexity (e.g. $N \simeq 10^{11}$ for a typical galaxy); moreover the evaluation of r_{ij} requires the evaluation of the irrational square root which needs more than one floating point operation.

HiGPUs *N*-body code

HiGPUs (R. Capuzzo-Dolcetta, M. Spera, D. Punzo 2013) is an *N*-body code suitable for studying the dynamical evolution of stellar systems composed up to 10 millions of stars with the precision guaranteed by direct summation of the pair-wise forces.
The code is written combining tools of C and C++ programming languages and it is parallelized using Message Passing Interface (MPI), OpenMP, and OpenCL to allow the utilization of GPUs of different vendors.

The code implements the Hermite's 6th order time integration scheme (Nitadoi & Makino 2008) with block time steps, allowing both high precision and speed in the study of the dynamical evolution of star systems.

The coarse-grained parallelization is such that a one-to-one correspondence between MPI process and computational nodes is established and each MPI process manages all the GPUs available per node.

Hermite's step

Let us assume that the i-th particle, has at time $t_{c,0}$, a position $\mathbf{r}_{i,0}$, a velocity $\mathbf{v}_{i,0}$ an acceleration $\mathbf{a}_{i,0}$, a *jerk* $\dot{\mathbf{a}}_{i,0}$, a *snap* $\ddot{\mathbf{a}}_{i,0}$, a *crackle* $\dddot{\mathbf{a}}_{i,0}$ and an individual time step $\Delta t_{i,0}$. Calling m (with $m \leq N$) the number of particles belonging to the same time-block, which have to be evolved to the same time $t_{c,0} + \Delta t_{i,0}$, the generic Hermite's step is composed by various substeps:

(1) Prediction step, with $O(N)$ complexity: positions, velocities and accelerations of all particles are predicted using the following equations:

$$\begin{aligned}
\mathbf{r}_{i,pred} &= \mathbf{r}_{i,0} + \mathbf{v}_{i,0}\Delta t_{i,0} + \frac{1}{2}\mathbf{a}_{i,0}\Delta t_{i,0}^{2} + \frac{1}{6}\dot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{3} + \\
&+ \frac{1}{24}\ddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{4} + \frac{1}{120}\dddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{5}, \\
\mathbf{v}_{i,pred} &= \mathbf{v}_{i,0} + \mathbf{a}_{i,0}\Delta t_{i,0} + \frac{1}{2}\dot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{2} + \frac{1}{6}\ddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{3} + \\
&+ \frac{1}{24}\dddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{4}, \\
\mathbf{a}_{i,pred} &= \mathbf{a}_{i,0} + \dot{\mathbf{a}}_{i,0}\Delta t_{i,0} + \frac{1}{2}\ddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{2} + \frac{1}{6}\dddot{\mathbf{a}}_{i,0}\Delta t_{i,0}^{3}.
\end{aligned}$$

Hermite's step

(2) Evaluation step, with $O(Nm)$ complexity: the accelerations of $m \leq N$ particles as well as their first and second time derivatives are evaluated using the above predicted data. The mutual interaction between the i-th particle and the remaining $N-1$ is described by the following relations:

$$\begin{aligned}
\mathbf{a}_{i,1} &= \sum_{\substack{j=1\\ j\neq i}}^{N} \mathbf{a}_{ij,1} = \sum_{\substack{j=1\\ j\neq i}}^{N} m_j \frac{\mathbf{r}_{ij}}{r_{ij}^3}, \\
\dot{\mathbf{a}}_{i,1} &= \sum_{\substack{j=1\\ j\neq i}}^{N} \dot{\mathbf{a}}_{ij,1} = \sum_{\substack{j=1\\ j\neq i}}^{N} \left(m_j \frac{\mathbf{v}_{ij}}{r_{ij}^3} - 3\alpha_{ij}\mathbf{a}_{ij,1} \right), \\
\ddot{\mathbf{a}}_{i,1} &= \sum_{\substack{j=1\\ j\neq i}}^{N} \ddot{\mathbf{a}}_{ij,1} = \sum_{\substack{j=1\\ j\neq i}}^{N} \left(m_j \frac{\mathbf{a}_{ij}}{r_{ij}^3} - 6\alpha\dot{\mathbf{a}}_{ij,1} - 3\beta_{ij}\mathbf{a}_{ij,1} \right),
\end{aligned}$$

where $\mathbf{r}_{ij} \equiv \mathbf{r}_{j,pred} - \mathbf{r}_{i,pred}$, $\mathbf{v}_{ij} \equiv \mathbf{v}_{j,pred} - \mathbf{v}_{i,pred}$, $\mathbf{a}_{ij} \equiv \mathbf{a}_{j,pred} - \mathbf{a}_{i,pred}$, $\alpha_{ij} r_{ij}^2 \equiv \mathbf{r}_{ij} \cdot \mathbf{v}_{ij}$, $\beta_{ij} r_{ij}^2 \equiv v_{ij}^2 + \mathbf{r}_{ij} \cdot \mathbf{a}_{ij} + \alpha_{ij}^2 r_{ij}^2$.

Hermite's step

(3) Correction step with complexity $O(m)$: positions and velocities of the mentioned m particles to be updated are corrected using the above evaluated accelerations and their time derivatives:

$$\begin{aligned}
\mathbf{v}_{i,corr} &= \mathbf{v}_{i,0} + \frac{\Delta t_{i,0}}{2}\left(\mathbf{a}_{i,1}+\mathbf{a}_{i,0}\right) - \frac{\Delta t_{i,0}^2}{10}\left(\dot{\mathbf{a}}_{i,1}-\dot{\mathbf{a}}_{i,0}\right) + \\
&+ \frac{\Delta t_{i,0}^3}{120}\left(\ddot{\mathbf{a}}_{i,1}+\ddot{\mathbf{a}}_{i,0}\right), \\
\mathbf{r}_{i,corr} &= \mathbf{r}_{i,0} + \frac{\Delta t_{i,0}}{2}\left(\mathbf{v}_{i,corr}+\mathbf{v}_{i,0}\right) - \frac{\Delta t_{i,0}^2}{10}\left(\mathbf{a}_{i,1}-\mathbf{a}_{i,0}\right) + \\
&+ \frac{\Delta t_{i,0}^3}{120}\left(\dot{\mathbf{a}}_{i,1}+\dot{\mathbf{a}}_{i,0}\right).
\end{aligned}$$

The individual time steps for m particles are, thus, updated, by mean of the so called *generalized Aarseth criterion* (Nitadori & Makino 2008).

$$\Delta t_{i,1} = \eta \left(\frac{A^{(1)}}{A^{(p-2)}}\right)^{\frac{1}{p-3}},$$

where η is linked to the accuracy required for the simulation, and

$$A^{(s)} \equiv \sqrt{\left|\mathbf{a}^{(s-1)}\right|\left|\mathbf{a}^{(s+1)}\right| + \left|\mathbf{a}^{(s)}\right|^2}.$$

GPU load optimizations

Particles are sub-divided in several groups that share the same time step (to avoid time synchronization among the N bodies).
Suppose that the stars to be updated are m (with $m \leq N$), the GPUs to use are $Ngpu$ and that every GPU is capable to run $Nthreads$ in parallel (full occupancy of the device):

- every GPU handles $N/Ngpu$ bodies;
- the simplest parallelization scheme of the forces calculation would be such to run m threads per GPU and calculate the partial accelerations due to $N/Ngpu$ bodies;
- such approach guarantees good load of the GPU if $m \simeq Nthreads$;
- significant degrading of performance if $m \ll Nthreads$.

The **Bfactor** variable:

- it's a factor that multiplies the total number of GPU blocks of threads by using the formula:

$$B_{MAX} = \left[\frac{N}{Ngpu * TpB} \right]$$

where TpB indicates the number of GPU threads per block.

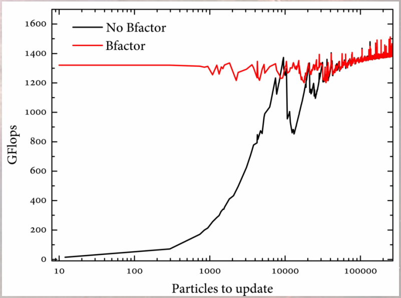
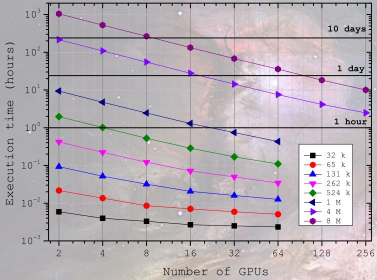
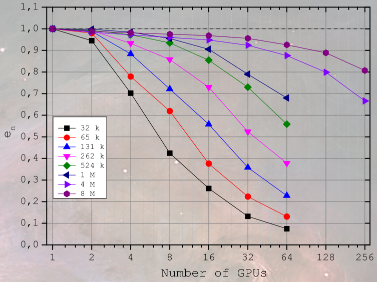
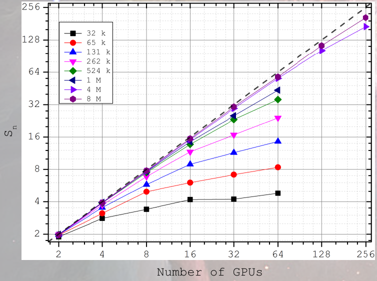


Figure 1: The effect of *Bfactor* optimization on the performance of a GeForce GTX TITAN Black. Reproduced from Spera 2015.

Code scalability, speedup and efficiency







Benchmarking performed using IBM PLX @CINECA (2 nVIDIA Tesla M2070 x node) from R. Capuzzo-Dolcetta & M. Spera 2013:

- Top left: time needed to integrate N-body systems ($32k \leq N \leq 8M$) over one time unit as a function of N_{GPU};
- Bottom left: speedup as a function of N_{GPU};
- Top right: efficiency as a function of N_{GPU}.

Code profiling

Index	Section	Used resource	Notation
1	Each node determines the particles to be updated and their indexes indexes are stored in an array named *next* containing *m* integer elements	CPU (OpenMP)	Δt_{next}
2	Each node copies to its GPUs the array containing indexes of *m* particles and the predictor step of N/N_{GPU} stars is executed	GPU	Δt_{pred}
3	Each node computes the forces (and their higher order derivatives) of *m* particles due to N/N_{GPU} bodies	GPU	Δt_{eval}
4	Each node reduces the calculated forces and derivatives of *Bfactor* blocks	GPU	Δt_{redu}
5	Each node adjusts conveniently the reduced values	GPU	Δt_{repo}
6	The CPUs receive the accelerations from the GPUs	GPU $\rightarrow$ CPU	Δt_{DtoH}
7	The MPI_Allreduce() functions collect and reduce accelerations from all the computational nodes	CPU(MPI)	Δt_{mpi}
8	Corrector step and time step update for *m* particles	CPU	Δt_{corr}
9	The reduced accelerations (and derivatives) and the corrected positions and velocities of *m* particles are passed to the GPUs of each node	CPU $\rightarrow$ GPU	Δt_{HtoD}

Table 1: The main sections of HiGPUs, which are performed at each time step. The "convenient adjustment" mentioned in the description of the 5th section refers to the re-organization of the computed and reduced accelerations and derivatives in one array only (instead of three) to improve the performance of the subsequent data transfer from the GPU to the CPU.

Simulation with $N = 2^{20}$ ($\simeq 10^6$) bodies

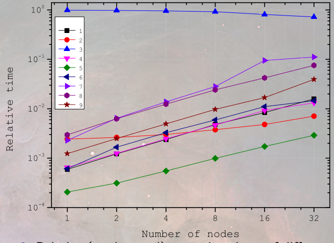


Figure 2: Relative (to the total) execution times of different parts of HiGPUs. Reproduced from R. Capuzzo-Dolcetta & M. Spera 2013.

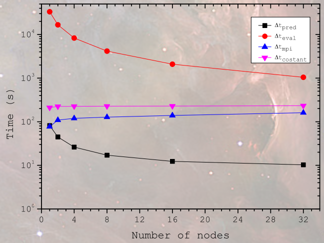


Figure 3: Times needed to complete the evaluation step, the predictor step, the MPI communications, and other sections grouped together. Reproduced from R. Capuzzo-Dolcetta & M. Spera 2013.