Using GPUs to Solve the N-Body Problem in Astrophysics

Mario Spera (mario.spera@oapd.inaf.it)
Postdoctoral researcher at INAF – OAPD
Workshop: GPU computing in High Energy Physics
Pisa, September 10-12

Outline

- ✓ Theoretical Introduction
- ✓ Numerical Introduction
- ✓ Why do we use GPUs ?
- ✓ The direct N-Body code HiGPUs
- ✓ Regularization methods
- ✓ Conclusions

Introduction to the N-Body problem

General Definition

The study of the motion of N point-like particles interacting through their mutual force that can be expressed according to a specific physical law

Gravitational N-Body problem

$$\begin{cases} \ddot{\boldsymbol{r}}_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} G \frac{m_j}{r_{ij}^3}(\boldsymbol{r}_j - \boldsymbol{r}_i) \\ \boldsymbol{r}_i(t_0) = \boldsymbol{r}_{i0} \\ \dot{\boldsymbol{r}}_i(t_0) = \dot{\boldsymbol{r}}_{i0} \end{cases}$$

NGC6388, HST

- ✓ $6N$ first-order scalar equations in $6N$ unknowns → Cauchy's problem
- ✓ No (useful) explicit solution for $N > 2$ → *Qiu-Dong Wang, 1991, CMDA 50, 73-88*
- ✓ We need a numerical approach to obtain a solution

Introduction to the N-Body problem

Numerical methods

Direct summation: the force acting on the particle i is computed as the complete sum of the contribution due to all the other $N - 1$ particles in the system

Approximation schemes:the direct sum of inter-particle forces is replaced by an other mathematical expression lighter in terms of computational complexity

Grid methods: codes that are based on the solution of the Poisson's equation on a grid, leading to a discrete force field

Introduction to the N-Body problem

Numerical solution = Challenge… why?

2-body interaction potential $U_{ij} = G\frac{m_j}{r_{ij}}$ $\Rightarrow$ $\boldsymbol{F}_{ij} = G\frac{m_i m_j}{{r_{ij}}^3}(\boldsymbol{r}_j - \boldsymbol{r}_i)$

$r_{ij} \to 0 \Rightarrow F_{ij} \to \infty$ *UV Divergence* Very small time steps

$F_{ij} \neq 0 \; \forall \, r_{ij}$ *IR Divergence* $\mathrm{O}(\mathrm{N}^2)$ operations (r_{ij})

$\mathrm{O}(\mathrm{N}^2)$ Complex operations $\Rightarrow$ $r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$

Simplifications

Approx. methods and/or Softening parameter $U_{ij} = G\frac{m_j}{\sqrt{{r_{ij}}^2 + \boldsymbol{\varepsilon}^2}}$

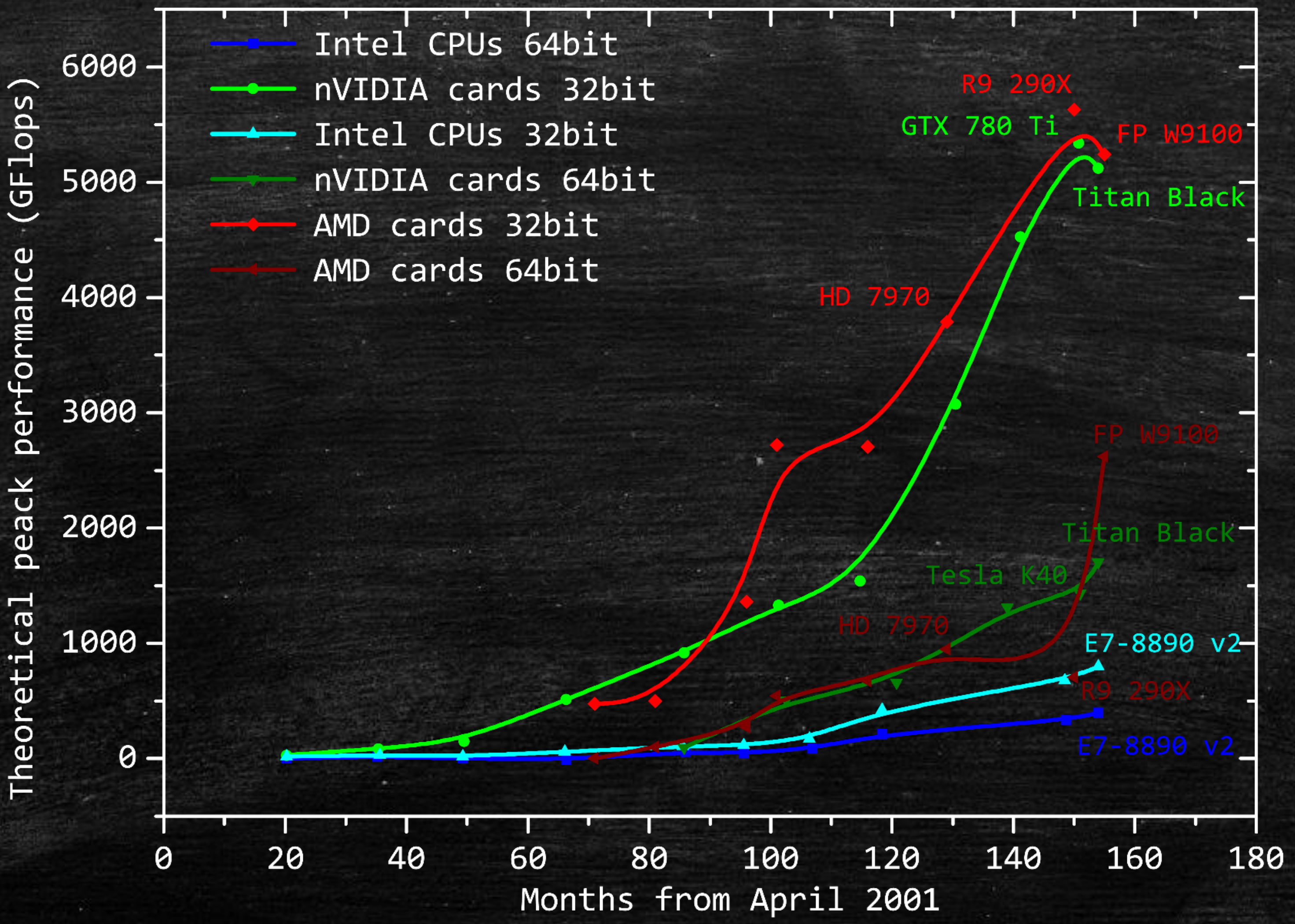
Introduction to the N-Body problem

Time to evolve a Globular Cluster using a very powerful CPU

Intel Xeon E7-8890 v2, 15core @3.40 GHz (turbo)	
Theoretical Performance (32bit)	16 flops/clock cycle * 3.40 * 15 ~ **800 GFlops**
N-Body system particle number	$3 \cdot 10^5$
Flops per cycle in N-Body integrators (F)	$\sim 50N^2$
Fixed time step	$\Delta t = 10^{-3} t_c$
Total integration time (T)	Relaxation: $t_R \simeq 3000 t_c$
Number of steps to complete (S)	$T/\Delta t \sim \mathbf{3 \cdot 10^6}$
Flops to execute	$S * F \simeq \mathbf{10^{19}}$
Computational time	~ **5 months**

~4 yrs ago the needed computing time was ~**7 years** !!

Why do we use Graphics Processing Units ?

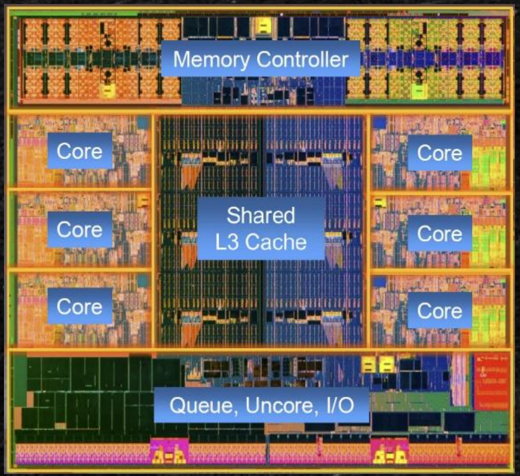


Why do we use Graphics Processing Units ?

Videogames → Science

- ✓ *Up to 5700 GFlops (32bit)*
- ✓ *Up to 2700 GFlops (64bit)*

CPU internal arch



Single core (up to 4 GHz)

Number of cores (< 15)

GPU internal arch

Single core (< 1 GHz)

Number of cores (up to 3000)

The direct N-Body code HiGPUs

- ✓ **HiGPUs : Hermite integrator on GPUs**

  Capuzzo-Dolcetta, Spera, Punzo, JCP 236, March 2013 p. 580-593

  - http://astrowww.phys.uniroma1.it/dolcetta/HPCcodes/HiGPUs.html

  - *AMUSE Package* *http://amusecode.org/*

- ✓ High precision

  Very good relative energy conservation (mixture of single and double precision)
- ✓ High performance

  Scientific results in short times
- ✓ Highly parallel

  It can run efficiently on the most modern supercomputers in the world
- ✓ Very easy to use

- ✓ Hermite 6^{th} order integrator (PEC schemes)
- ✓ Block Time Steps (IR divergence : $O(N^2) \rightarrow O(mN)$)
- ✓ C and C++
- ✓ CUDA + MPI + OpenMP *(to fully exploit hybrid supercomputers)*
- ✓ **OpenCL**

HiGPUs: The Hermite 6th order time integration scheme

PREDICTOR

$$\boldsymbol{r}_{i,p} = \boldsymbol{r}_{i,0} + \boldsymbol{v}_{i,0}\Delta t_{i,0} + \frac{1}{2}\boldsymbol{a}_{i,0}\Delta t_{i,0}^2 + \frac{1}{6}\boldsymbol{j}_{i,0}\Delta t_{i,0}^3 + \frac{1}{24}\boldsymbol{s}_{i,0}\Delta t_{i,0}^4 + \frac{1}{120}\boldsymbol{c}_{i,0}\Delta t_{i,0}^5 + O\left(\Delta t_{i,0}^6\right)$$

$$\boldsymbol{v}_{i,p} = \boldsymbol{v}_{i,0} + \boldsymbol{a}_{i,0}\Delta t_{i,0} + \frac{1}{2}\boldsymbol{j}_{i,0}\Delta t_{i,0}^2 + \frac{1}{6}\boldsymbol{s}_{i,0}\Delta t_{i,0}^3 + \frac{1}{24}\boldsymbol{c}_{i,0}\Delta t_{i,0}^4 + O\left(\Delta t_{i,0}^5\right)$$

$$\boldsymbol{a}_{i,p} = \boldsymbol{a}_{i,0} + \boldsymbol{j}_{i,0}\Delta t_{i,0} + \frac{1}{2}\boldsymbol{s}_{i,0}\Delta t_{i,0}^2 + \frac{1}{6}\boldsymbol{c}_{i,0}\Delta t_{i,0}^3 + O\left(\Delta t_{i,0}^4\right)$$

Just one evaluation of accelerations per time step

EVALUATION

$$\boldsymbol{a}_{ij,1} = m_j \frac{\boldsymbol{r}_{ij}}{r_{ij}^3} \qquad \boldsymbol{j}_{ij,1} = m_j \frac{\boldsymbol{v}_{ij}}{r_{ij}^3} - 3\alpha\boldsymbol{a}_{ij,1} \qquad \boldsymbol{s}_{ij,1} = m_j \frac{\boldsymbol{a}_{ij}}{r_{ij}^3} - 6\alpha\boldsymbol{j}_{ij,1} - 3\beta\boldsymbol{a}_{ij,1}$$

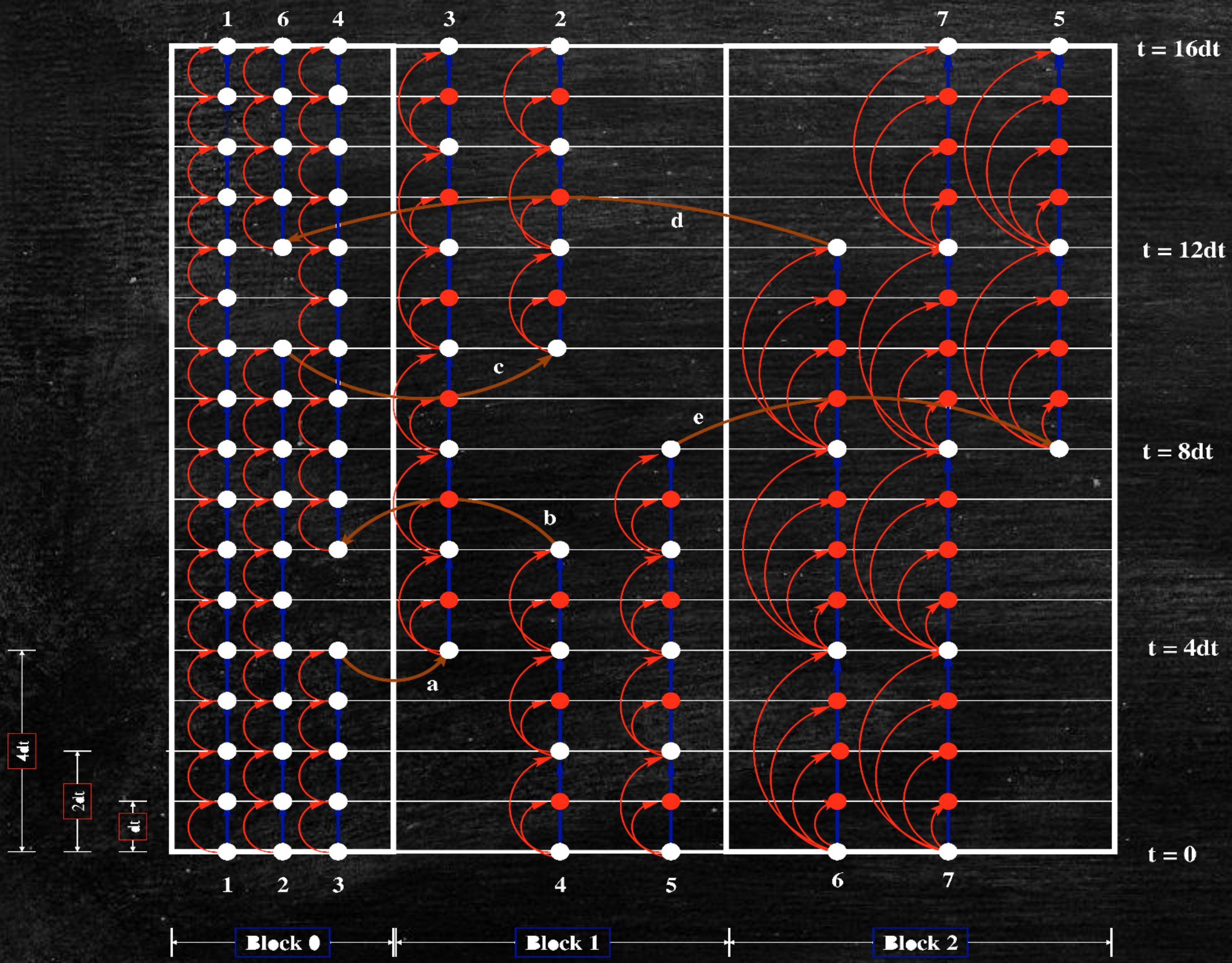
CORRECTOR

$$\boldsymbol{r}_{i,c} = \boldsymbol{r}_{i,0} + \frac{\Delta t_{i,0}}{2}\left(\boldsymbol{v}_{i,1} + \boldsymbol{v}_{i,0}\right) - \frac{\Delta t_{i,0}^2}{10}\left(\boldsymbol{a}_{i,1} - \boldsymbol{a}_{i,0}\right) + \frac{\Delta t_{i,0}^3}{120}\left(\boldsymbol{j}_{i,1} + \boldsymbol{j}_{i,0}\right)$$

$$\boldsymbol{v}_{i,c} = \boldsymbol{v}_{i,0} + \frac{\Delta t_{i,0}}{2}\left(\boldsymbol{a}_{i,1} + \boldsymbol{a}_{i,0}\right) - \frac{\Delta t_{i,0}^2}{10}\left(\boldsymbol{j}_{i,1} - \boldsymbol{j}_{i,0}\right) + \frac{\Delta t_{i,0}^3}{120}\left(\boldsymbol{s}_{i,1} + \boldsymbol{s}_{i,0}\right)$$

HiGPUs: Block Time Steps

Image taken from Konstantinidis, S. and Kokkotas, K. D., A&A 522 A70, 22pp.



Higher parallelization efficiency

It ensures exact time synchronization among particles

It takes into account the different time scales involved in an N-Body system

Computational complexity per time step : $O(mN)$

HiGPUs: speed up forces evaluation kernel

```
double4 myPosition = {0.0, 0.0, 0.0, 0.0};
float4 myVelocity = {0.0f, 0.0f, 0.0f, 0.0f};
float4 myAccelera = {0.0f, 0.0f, 0.0f, 0.0f};

double4 acc = {0.0, 0.0, 0.0, 0.0};
double4 jrk = {0.0, 0.0, 0.0, 0.0};
double4 snp = {0.0, 0.0, 0.0, 0.0};
```

HiGPUs uses 64bit just for positions and accelerations

```
barrier(CLK_LOCAL_MEM_FENCE);
 shPos[threadIdx] = plocal;
 shVel[threadIdx] = vlocal;
 shAcc[threadIdx] = alocal;
barrier(CLK_LOCAL_MEM_FENCE);
```

HiGPUs uses shared memory (much faster than global memory)

```
float4 dr = {plocal.x - myPosition.x,
             plocal.y - myPosition.y,
             plocal.z - myPosition.z,
             0.0f};

float distance = dr.x * dr.x +
                 dr.y * dr.y +
                 dr.z * dr.z;
```

HiGPUs performs intermediate operations in 32bit to speed up the forces evaluation

64 bit used only to reduce round-off errors in evaluating mutual distances

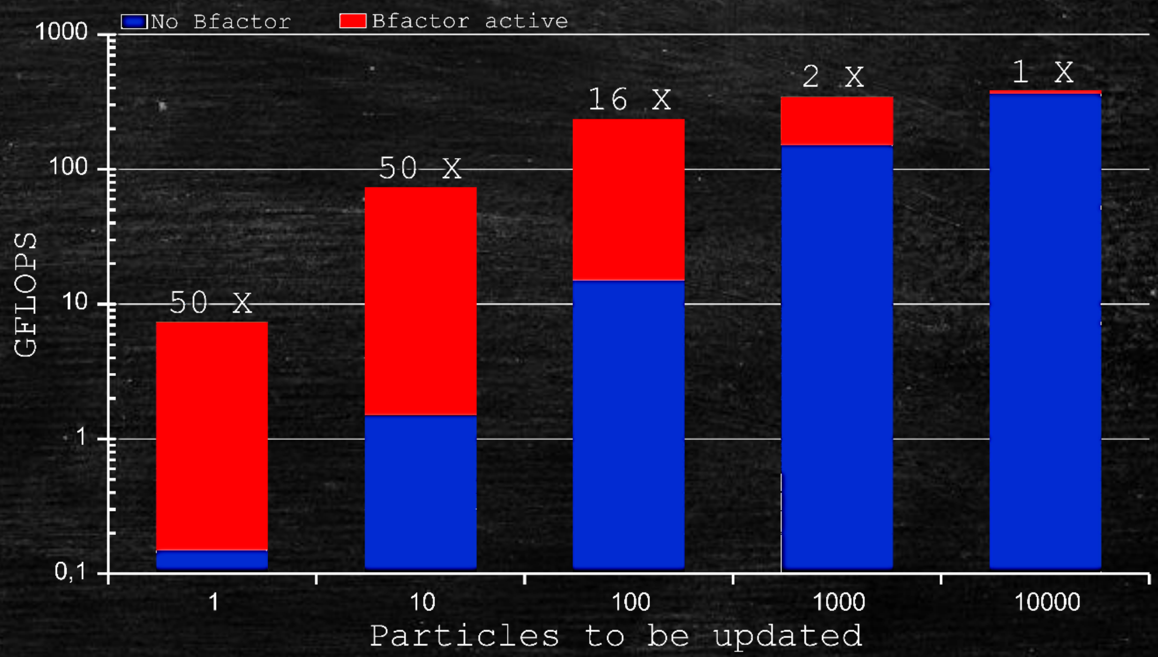
HiGPUs: speed up forces evaluation kernel

Block Time Steps → we need to update m ($m \leq N$) particles per time step

1:1 approach → We execute m GPU threads → They cannot be enough to fully load the GPU

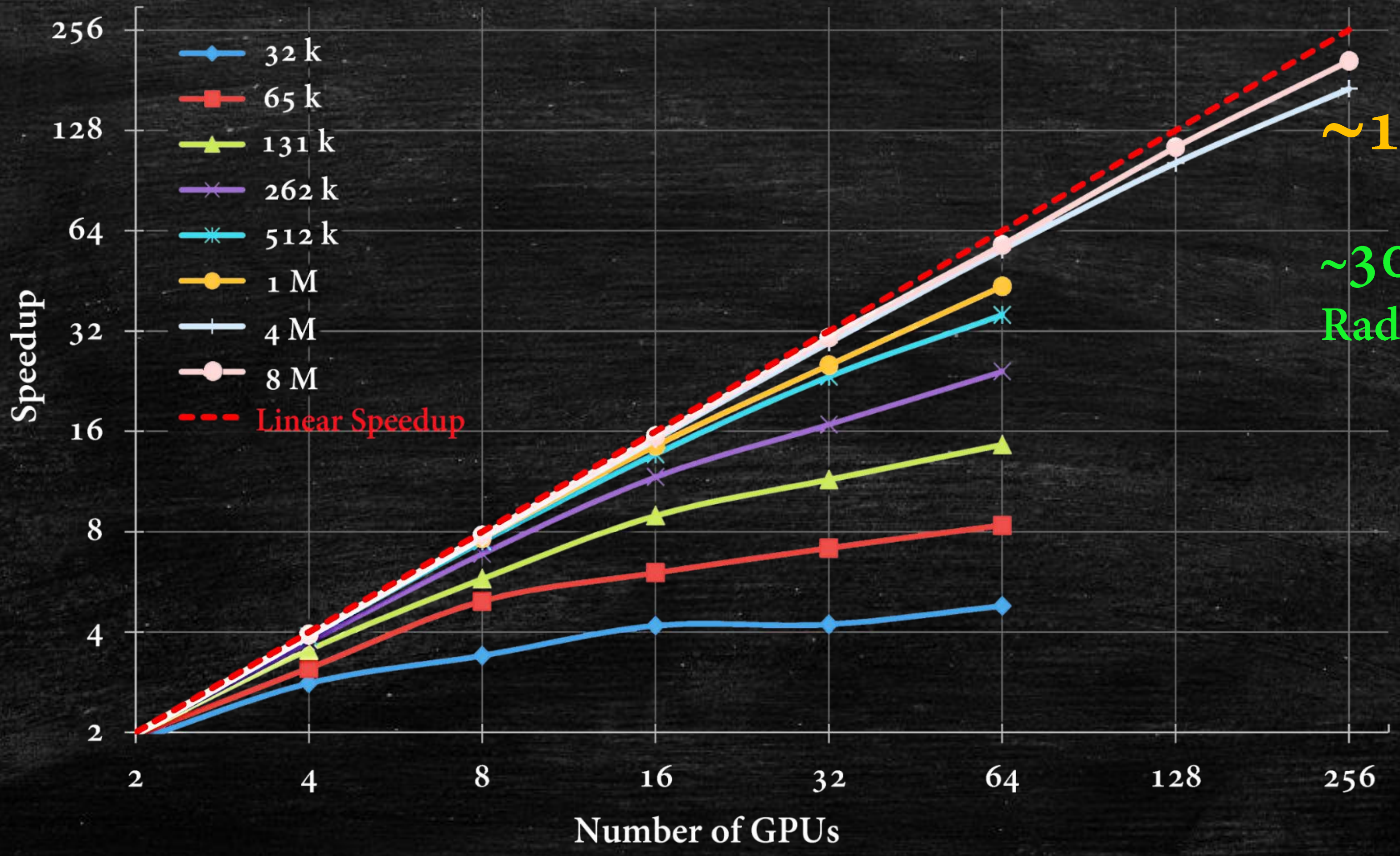
(Up to $5 \cdot 10^4$ threads in parallel for a common GPU)

Bfactor variable: split further the work among the gpu threads (inside a single GPU) until the number of running threads is greater than the maximum number of parallel threads that the GPU can execute in parallel



Test HiGPUs on a GPU supercomputer

Speedup : $S_n = \frac{\Delta T_1}{\Delta T_n}$

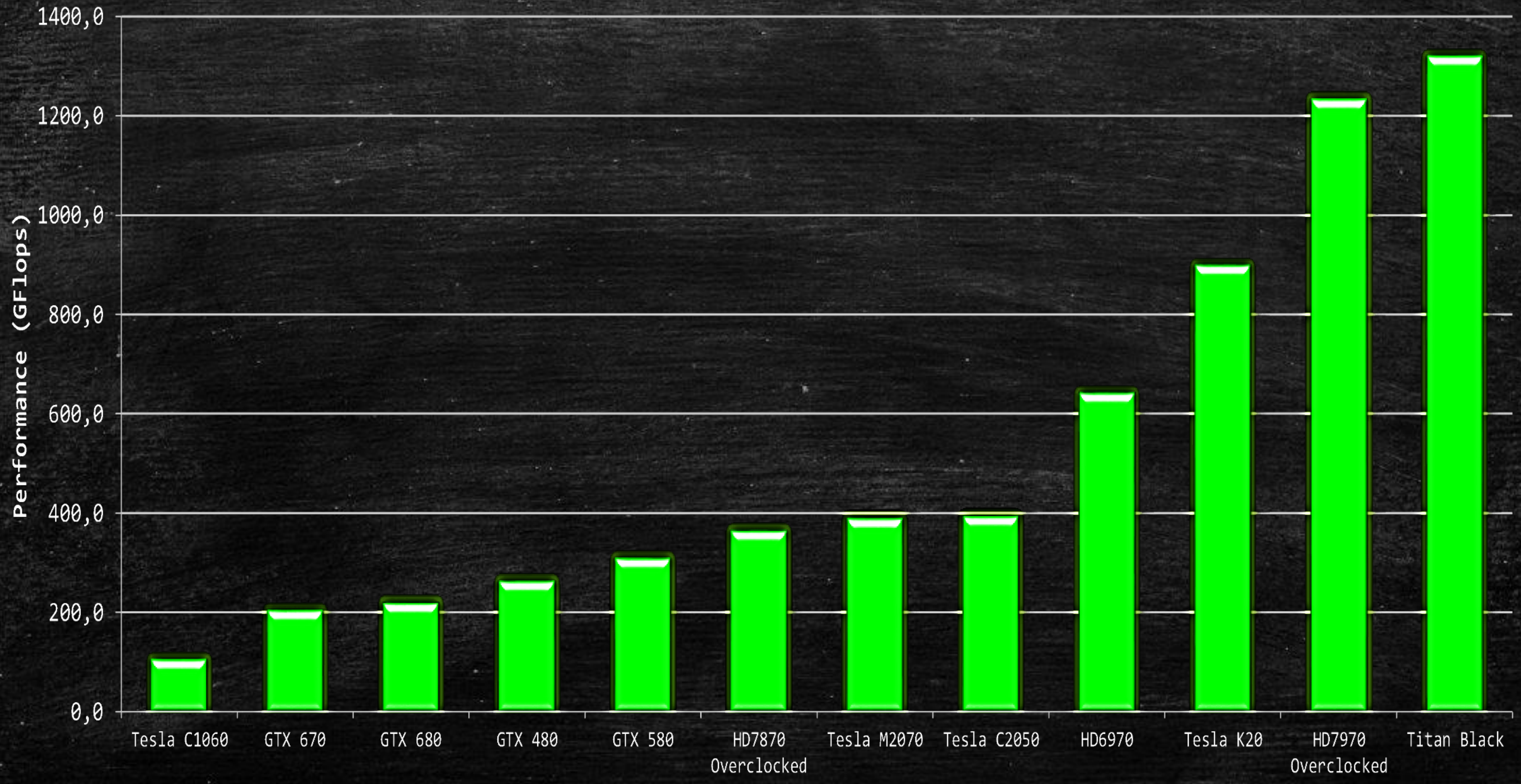


$E_n \sim 0.92$

~102 Tflops

~300 Tflops with Radeon HD 7970

Testing HiGPUs on single, different GPUs



Introduction to the N-Body problem (again)

Numerical solution = Challenge… why?

2-body interaction potential $U_{ij} = G\frac{m_j}{r_{ij}}$ $\Rightarrow$ $\boldsymbol{F}_{ij} = G\frac{m_i m_j}{{r_{ij}}^3}(\boldsymbol{r}_j - \boldsymbol{r}_i)$

$r_{ij} \to 0 \Rightarrow F_{ij} \to \infty$ *UV Divergence* Very small time steps

$F_{ij} \neq 0 \ \forall \, r_{ij}$ *IR Divergence* $O(N^2)$ operations (r_{ij})

$O(N^2)$ Complex operations $\Rightarrow$ $r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$

Simplifications

Approx. methods and/or Softening parameter $U_{ij} = G\frac{m_j}{\sqrt{{r_{ij}}^2 + \boldsymbol{\varepsilon}^2}}$

Regularization methods

General Definition

Methods that try to remove the UV divergence of the 2-body interaction potential obtaining a «regular» expression for the pair-wise force

Difficulties :

- ✓ Implementation
- ✓ Hardware acceleration
- ✓ Integration

Not widely used

Strategies:

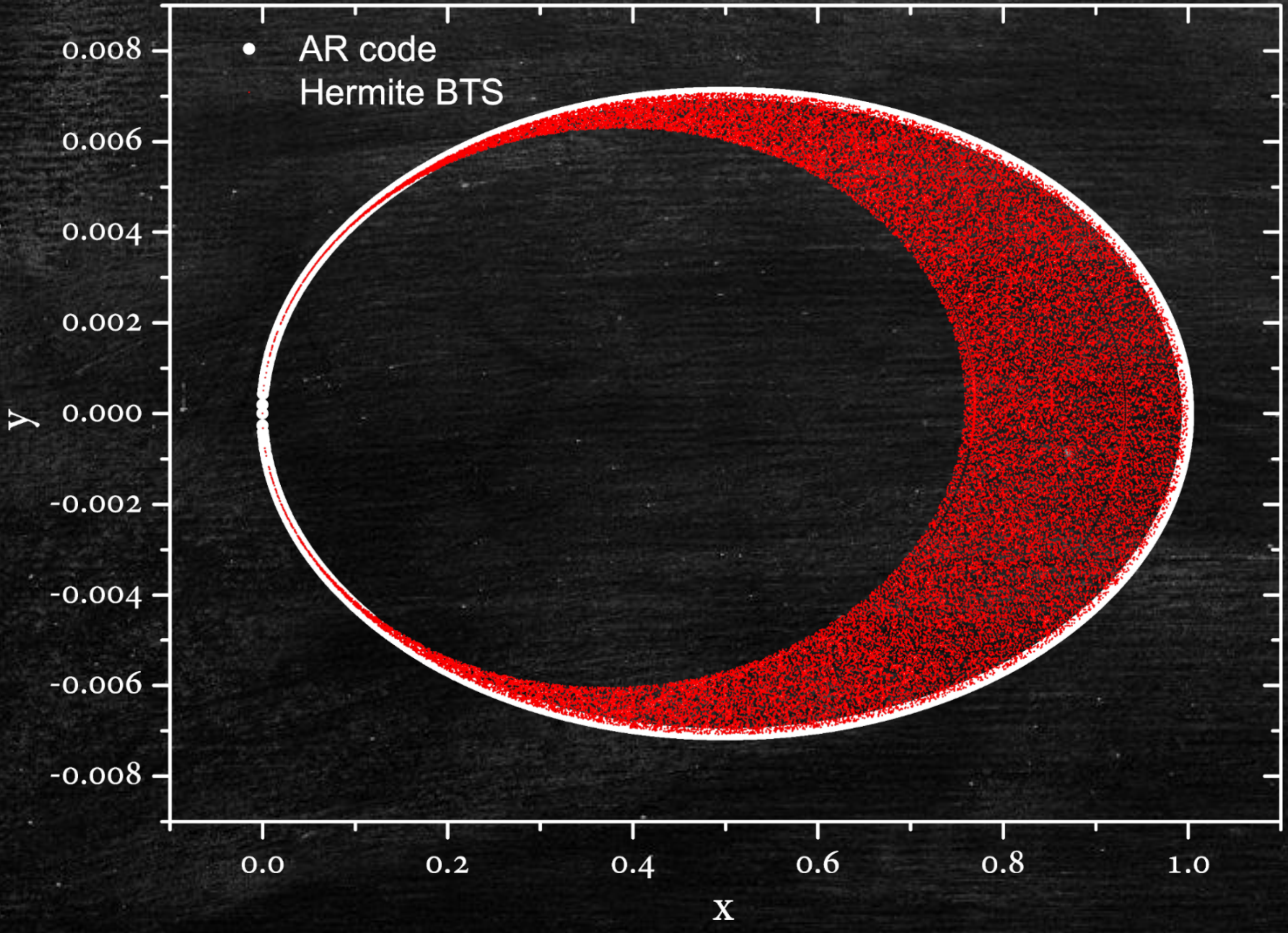
- ✓ *coordinate transformation*
- ✓ *regular algorithm*
- ✓ *combination of previous points*

Burdet-Heggie
Kustaanheimo-Stiefel

Algorithmic (Chain) Regularization *(Seppo Mikkola):*

- ➢ Time transformation + Regular algorithm (symplectic leapfrog)
- ➢ Chain spatial coordinates

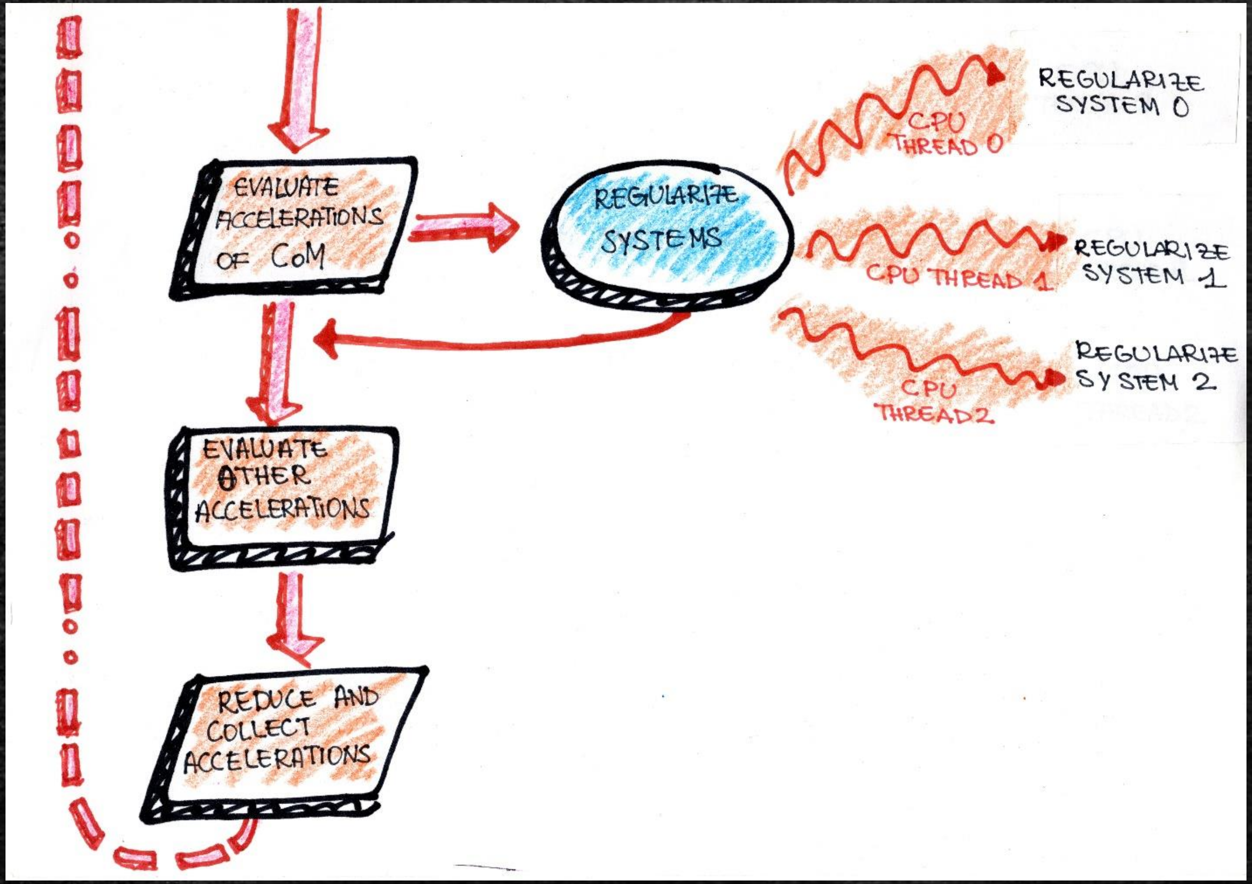
Test regularization schemes



- ✓ 2body problem
- ✓ $\frac{m_1}{m_2} \simeq 10^5$
- ✓ $\simeq 10^4$ revolutions
- ✓ $e \simeq 0.9999$

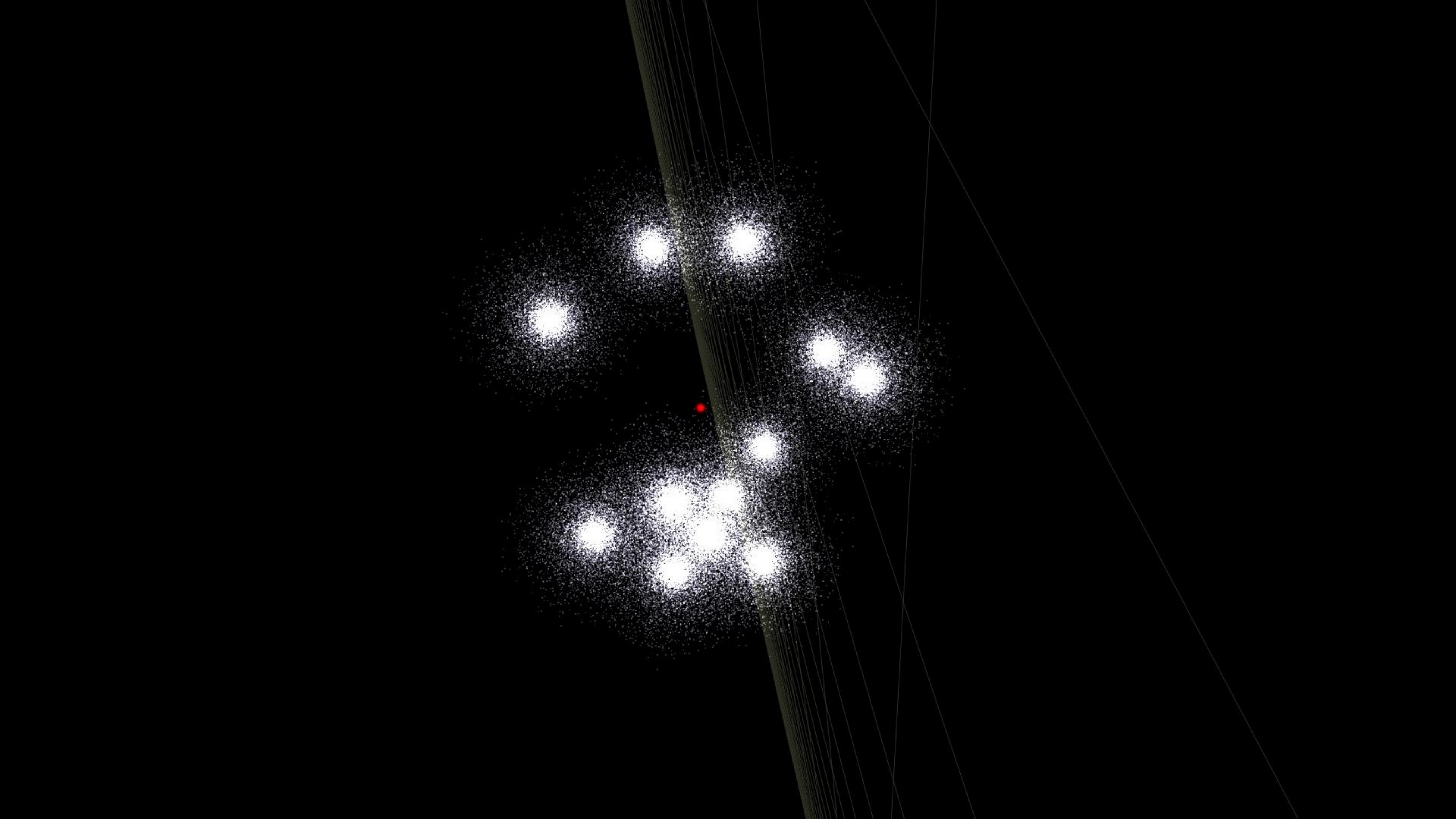
$$\frac{\left(\frac{\Delta E}{E}\right)_{Hermite}}{\left(\frac{\Delta E}{E}\right)_{AR}} \simeq 10^{13} \text{ !!!}$$

Implementing a regularization scheme in the framework of a GPU code: HiGPUs-R



The GPU kernel is asynchronous

The GPU can work in background while the CPU performs the regularization process in parallel by means of OpenMP



Conclusions

- ✓ GPUs can accelerate the numerical integration of N-Body systems
- ✓ The direct N-Body code HiGPUs shows very good scalability on GPU clusters and exhibits very good performance on single, different GPUs
- ✓ Regularization methods for the N-Body problem can take advantage from the GPU asynchronous kernel execution
- ✓ The growth of scientific applications that can run on GPUs is exponential