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

\[
\begin{align*}
\ddot{r}_i &= \sum_{j=1}^{N} G \frac{m_j}{r_{ij}^3} (r_j - r_i) \\
\end{align*}
\]

\(\begin{align*}
\vec{r}_i(t_0) &= \vec{r}_{i0} \\
\vec{r}_i(t_0) &= \vec{r}_{i0}
\end{align*}\)

✓ 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 another 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 \( U_{ij} = G \frac{m_j}{r_{ij}} \)  \( \Rightarrow \) \( F_{ij} = G \frac{m_i m_j}{r_{ij}^3} (r_j - r_i) \)

\( r_{ij} \rightarrow 0 \Rightarrow F_{ij} \rightarrow \infty \) **UV Divergence**  \( \Rightarrow \) Very small time steps

\( F_{ij} \neq 0 \ \forall r_{ij} \) **IR Divergence**  \( \Rightarrow \) 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** \( U_{ij} = G \frac{m_j}{\sqrt{r_{ij}^2 + \varepsilon^2}} \)
Introduction to the N-Body problem

Time to evolve a Globular Cluster using a very powerful CPU

<table>
<thead>
<tr>
<th>Description</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theoretical Performance (32bit)</td>
<td>$16 \text{ flops/clock cycle} \times 3.40 \times 15 \approx 800 \text{ GFlops}$</td>
</tr>
<tr>
<td>N-Body system particle number</td>
<td>$3 \times 10^5$</td>
</tr>
<tr>
<td>Flops per cycle in N-Body integrators (F)</td>
<td>$\sim 50N^2$</td>
</tr>
<tr>
<td>Fixed time step</td>
<td>$\Delta t = 10^{-3} t_c$</td>
</tr>
<tr>
<td>Total integration time (T)</td>
<td>Relaxation: $t_R \approx 3000 t_c$</td>
</tr>
<tr>
<td>Number of steps to complete (S)</td>
<td>$T/\Delta t \sim 3 \cdot 10^6$</td>
</tr>
<tr>
<td>Flops to execute</td>
<td>$S \times F \approx 10^{19}$</td>
</tr>
<tr>
<td>Computational time</td>
<td>$\sim 5 \text{ months}$</td>
</tr>
</tbody>
</table>

$\sim 4 \text{ yrs ago the needed computing time was } \sim 7 \text{ years}!!$
Why do we use Graphics Processing Units?
Why do we use Graphics Processing Units?

Videogames $\rightarrow$ 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 6th 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**

\[
\begin{align*}
\mathbf{r}_{i,p} &= \mathbf{r}_{i,0} + v_{i,0} \Delta t_{i,0} + \frac{1}{2} a_{i,0} \Delta t_{i,0}^2 + \frac{1}{6} j_{i,0} \Delta t_{i,0}^3 + \frac{1}{24} s_{i,0} \Delta t_{i,0}^4 + \frac{1}{120} c_{i,0} \Delta t_{i,0}^5 + O(\Delta t_{i,0}^6) \\
\mathbf{v}_{i,p} &= \mathbf{v}_{i,0} + a_{i,0} \Delta t_{i,0} + \frac{1}{2} j_{i,0} \Delta t_{i,0}^2 + \frac{1}{6} s_{i,0} \Delta t_{i,0}^3 + \frac{1}{24} c_{i,0} \Delta t_{i,0}^4 + O(\Delta t_{i,0}^5) \\
a_{i,p} &= a_{i,0} + j_{i,0} \Delta t_{i,0} + \frac{1}{2} s_{i,0} \Delta t_{i,0}^2 + \frac{1}{6} c_{i,0} \Delta t_{i,0}^3 + O(\Delta t_{i,0}^4)
\end{align*}
\]

**Evaluation**

Just one evaluation of accelerations per time step

\[
\begin{align*}
a_{ij,1} &= m_j \frac{r_{ij}}{r_{ij}^3} \\
j_{ij,1} &= m_j \frac{v_{ij}}{r_{ij}^3} - 3a a_{ij,1} \\
s_{ij,1} &= m_j \frac{a_{ij}}{r_{ij}^3} - 6a j_{ij,1} - 3\beta a_{ij,1}
\end{align*}
\]

**Corrector**

\[
\begin{align*}
\mathbf{r}_{i,c} &= \mathbf{r}_{i,0} + \frac{\Delta t_{i,0}}{2} (\mathbf{v}_{i,1} + \mathbf{v}_{i,0}) - \frac{\Delta t_{i,0}^2}{10} (a_{i,1} - a_{i,0}) + \frac{\Delta t_{i,0}^3}{120} (j_{i,1} + j_{i,0}) \\
\mathbf{v}_{i,c} &= \mathbf{v}_{i,0} + \frac{\Delta t_{i,0}}{2} (a_{i,1} + a_{i,0}) - \frac{\Delta t_{i,0}^2}{10} (j_{i,1} - j_{i,0}) + \frac{\Delta t_{i,0}^3}{120} (s_{i,1} + s_{i,0})
\end{align*}
\]
HiGPUs: Block Time Steps

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 64 bit 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};

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

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

64 bit used only to reduce round-off errors in evaluating mutual distances
HiGPUs: speed up forces evaluation kernel

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

1:1 approach $\Rightarrow$ We execute $m$ GPU threads $\Rightarrow$ 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: \( S_n = \frac{\Delta T_1}{\Delta T_n} \)

- \( E_n \approx 0.92 \)
- \( \approx 102 \text{ Tflops} \)
- \( \approx 300 \text{ Tflops with Radeon HD 7970} \)
Testing HiGPUs on single, different GPUs
Introduction to the N-Body problem (again)

**Numerical solution = Challenge... why?**

2-body interaction potential \( U_{ij} = G \frac{m_j}{r_{ij}} \)  \( \rightarrow \) \( F_{ij} = G \frac{m_i m_j}{r_{ij}^3} (r_j - r_i) \)

- \( r_{ij} \rightarrow 0 \Rightarrow F_{ij} \rightarrow \infty \) \( UV \) **Divergence**
- \( F_{ij} \neq 0 \ \forall \ r_{ij} \) \( IR \) **Divergence**

Very small time steps

\( O(N^2) \) operations \( (r_{ij}) \)

\( O(N^2) \) **Complex operations**

\( r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \)

**Simplifications**

Approx. methods and/or **Softening parameter**

\( U_{ij} = G \frac{m_j}{\sqrt{r_{ij}^2 + \varepsilon^2}} \)
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

Strategies:
- coordinate transformation
- regular algorithm
- combination of previous points

Algorithmic (Chain) Regularization (Seppo Mikkola):
- Time transformation + Regular algorithm (symplectic leapfrog)
- Chain spatial coordinates
Test regularization schemes

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

$\frac{(\Delta E/E)_{\text{Hermite}}}{(\Delta E/E)_{\text{AR}}} \approx 10^{13} !!!$
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
Some applications
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