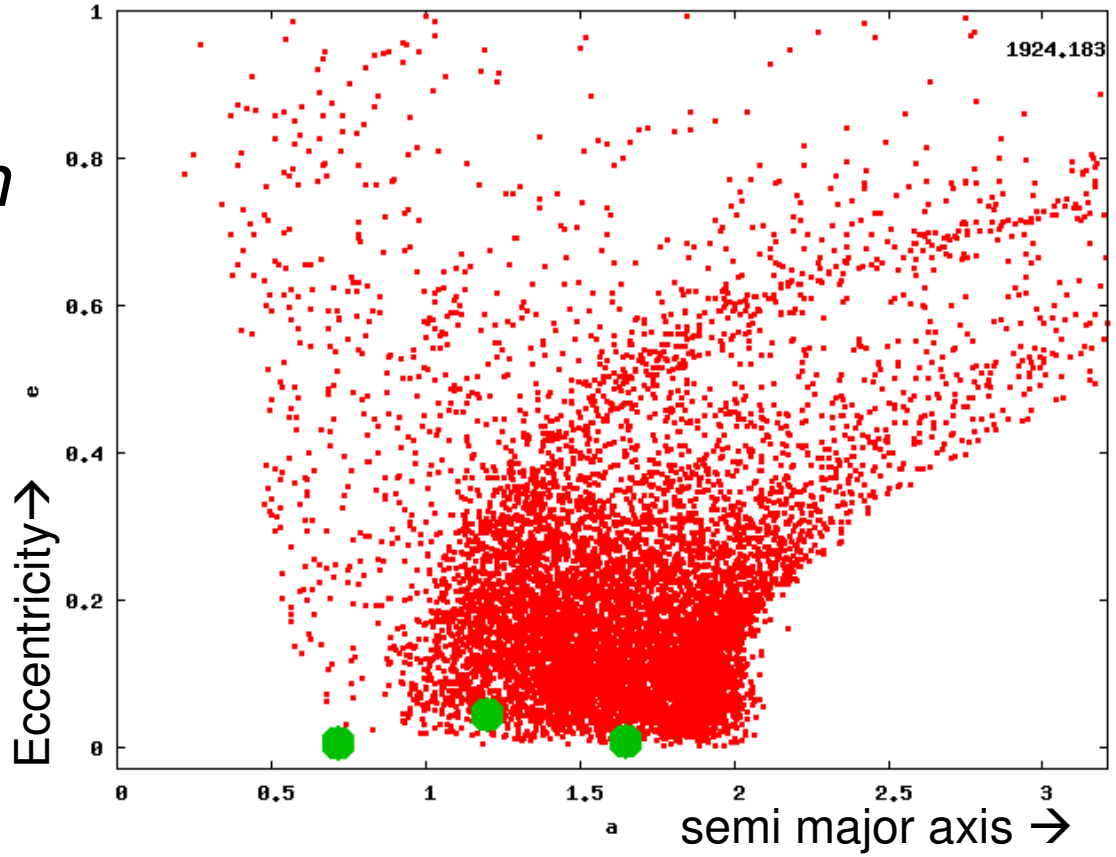
Simulating Self Gravitating Planetesimal Disks on the Graphics Processing Unit (GPU)

Alice C. Quillen
Alex Moore
University of Rochester

Poster
DPS 2008



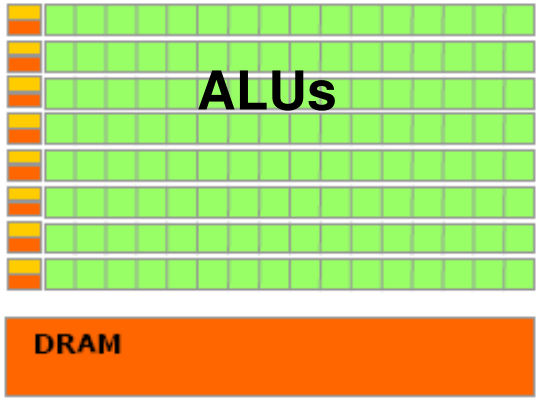
Abstract

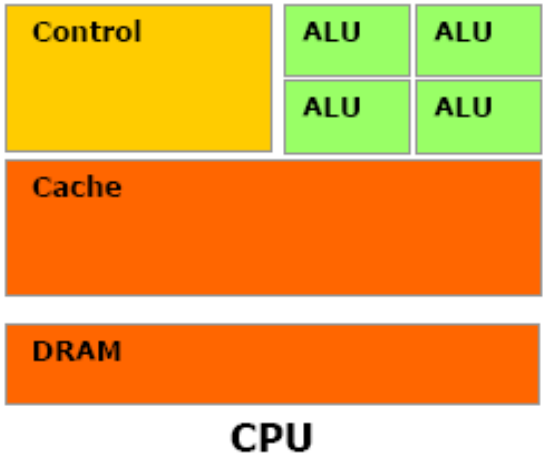
Planetesimal and dust dynamical simulations require computations of many gravitational force pairs as well as collision and nearest neighbor detection. These are order N^2 or $O(N^2)$ computations for N the number of particles. I describe an efficient parallel implementation of an N-body integrator capable of integrating a self-gravitating planetesimal disk in single floating point precision. The software runs on a GPU (graphics processing unit) and is written in CUDA, NVIDIA's programming language for video cards. The integrator is a parallel 2nd order symplectic that works in heliocentric coordinates and barycentric momenta.

A brute force implementation for sorting interparticle distances requires $O(N^2)$ computations, limiting the numbers of particles that have been simulated. Algorithms recently developed for the GPU, such as the radix sort, can run as fast as $O(N)$ and sort distances between a million particles in a few hundred milliseconds. We explore improvements in collision and nearest neighbor detection algorithms and how we can in future incorporate them into our integrator.

Parallel computations on the GPU

- Graphics cards are a cheap and new parallel computing platform
- On a single chip: large number of arithmetic computation units (ALUs) **and** large on-board memory. The fraction of the chip devoted to arithmetic computation units exceeds that on CPUs.
- Optimization of code: maximizing the numbers of computations while minimizing slow memory transfers. Having a large on-board memory is extremely efficient as it bypasses many of the man hours involved in writing code that shares information across processors ---a problem of MPI (message passing interface) approaches to parallel computing.





N-body planetesimal systems in single precision floating point

- Most video applications do not require double precision floating point calculations
- Current GPUs either are not capable of doing double precision calculations or have a reduced number of units devoted to this and so are not as fast.
- Precision limit for single precision is ~$10^{-7.}$ Compare this to the ratio of the mass of Earth to the Sun ~10^{-6}. The force from the Sun swamps that from the Earth if single precision is used and the terms added.
- To carry out N-body calculations in a circumstellar disk in single precision the central force term **must be removed from the force computation sum.**
- We work in a coordinate system that allows interaction terms to be separated from Keplerian evolution.

The Hamiltonian for the N-body system in heliocentric coordinates and associated barycentric canonical momentum separates into 3 terms (Duncan et al. 1998).

$$H = H_{Sun} + K_{Kep} + H_{Int}$$

H_{Int} The interaction term contains the interparticle force pairs but **lacks** the force from central mass!
O(N²) computations.

$$H_{Int} = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} -\frac{G m_i m_j}{2|\mathbf{Q}_i - \mathbf{Q}_j|}.$$

H_{Sun} is a correction caused by working in a heliocentric frame rather than an inertial frame.
O(N) computations.

$$H_{Sun} = \frac{1}{2m_0} \left| \sum_{i=1}^{n} \mathbf{P}_i^2 \right|^2$$

H_{Kep} Keplerian evolution for each particle separately
O(N) computations.

$$H_{Kep} = \sum_{i=0}^{N} \left(\frac{\mathbf{P}_i^2}{2m_i} - \frac{G m_i m_0}{|\mathbf{Q}_i|} \right)$$

Second order Symplectic Integrator

Evolution in timestep τ

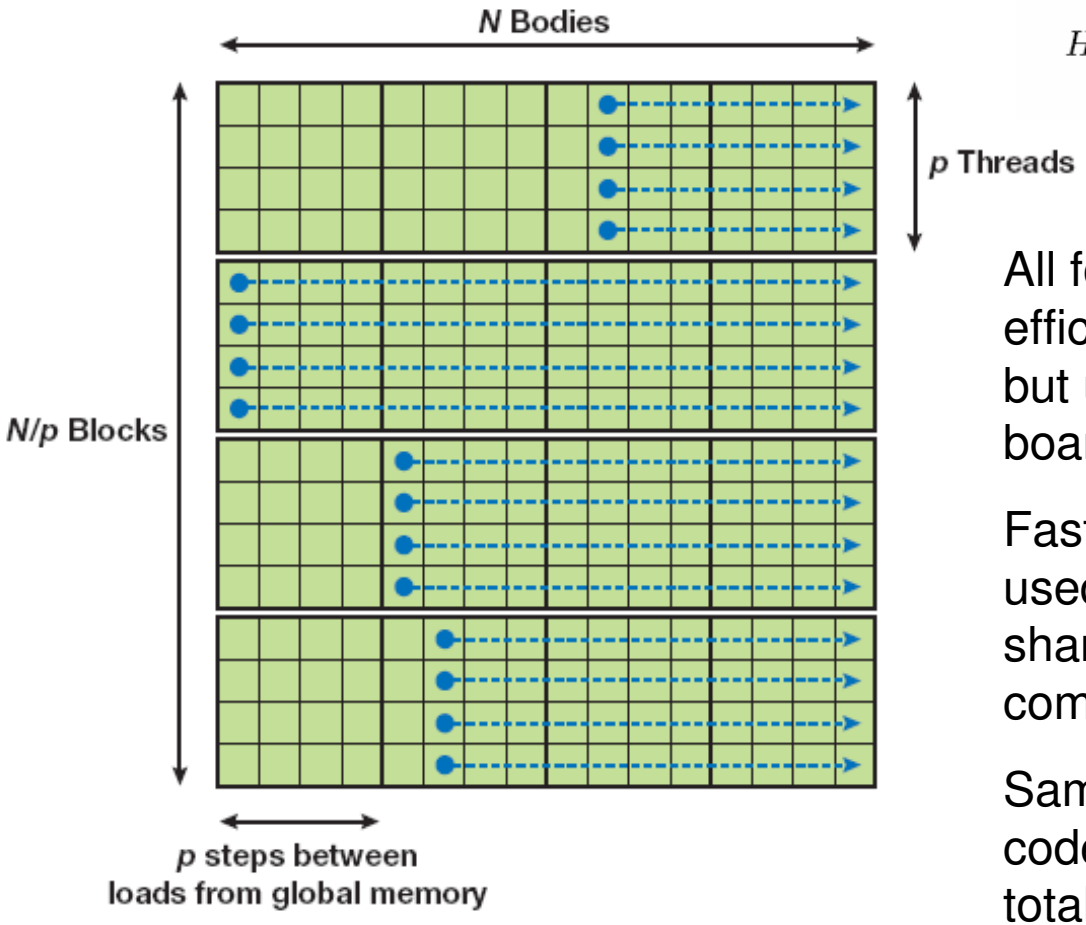
$$E_{Sun}\left(\frac{\tau}{2}\right) E_{Kep}\left(\frac{\tau}{2}\right) E_{Int}\left(\tau\right) E_{Kep}\left(\frac{\tau}{2}\right) E_{Sun}\left(\frac{\tau}{2}\right)$$

- To advance a timestep we use a series of evolution operators each based on a term in the Hamiltonian.
- The interaction evolution kernel is called **once** per timestep because this is the most computationally intensive step.
- The drift and Keplerian evolution kernels are called twice per timestep but they require only *O(N)* computations.

Memory layout and Kernels

- All particle coordinates reside in GPU global memory. Heliocentric/barycentric coordinates used on GPU.
- Particle coordinates only transferred on+off the GPU for initial setup and outputs.
- Separate Kernels written for each term in Hamiltonian. A kernel is code that runs on the GPU but called from the CPU.
- Shared memory on GPU used for computational intense force pair calculation

Interaction Kernel on the GPU



$$H_{Int} = \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} -\frac{G m_i m_j}{2|\mathbf{Q}_i - \mathbf{Q}_j|}.$$

All force pairs calculated as efficiently as on a GRAPE board but using ~20 processors on board a video card

Faster code when shared memory used. p coordinates transferred to shared memory for each computation tile.

Same tiling and almost identical code can be used for calculating total energy as for calculating forces

Nyland et al. 2008 GPU Gems3

Kepstep Kernel

$$H_{Kep} = \sum_{i=0}^{N} \left(\frac{\mathbf{P}_i^2}{2m_i} - \frac{Gm_i m_0}{|\mathbf{Q}_i|} \right)$$

- f and g functions used to advance particle positions.

$$\begin{aligned} \vec{\mathbf{x}}_1 &= f\vec{\mathbf{x}}_0 + g\vec{\mathbf{v}}_0 \\ \vec{\mathbf{v}}_1 &= \dot{f}\vec{\mathbf{x}}_0 + \dot{g}\vec{\mathbf{v}}_0. \end{aligned}$$

- These functions are computed with an iterative solution of the universal differential Kepler's equation (e.g.,see Prussing & Conway 1993; chapter 2). A Universal form is used so that hyperbolic, parabolic and eccentric orbits can be computed with the same equation. It's important that the algorithm be robust --- never give NANs. No case statements are required for different classes of orbits facilitating parallel implementation.

7 Iteration solution of the Universal Kepler's equation on the GPU

Kepler's equation must be solved iteratively.
Usually the iteration is done until the equation is solved within a desired precision level.

```
compute_first_iteration();
while(computed_difference() < desired_precision){
      compute_next_iteration();
}
```

However this makes little sense on a GPU where all particles are integrated simultaneously. If one particles pops out of the while statement early it does not speed up the code because this particle has to wait for the other particles to exit the loop.
Fortunately we can chose an iteration method that rapidly converges. Using the Laguerre cubic convergence method, I find that 7 iterations is enough for all possible initial conditions to achieve double precision accuracy. On the GPU we just iterate 7 times no matter what. We trade computations for complexity which could have slowed the code.

```
compute_first_iteration();
compute_next_iteration();  compute_next_iteration();
compute_next_iteration();  compute_next_iteration(); compute_next_iteration();
```

Angular momentum conservation enforcement in the Kepstep kernel

- In our first implementation we noted a steady loss in angular momentum for particles after 10^6 timesteps were taken. We pinpointed the problem finding that it was due to errors in single precision computation in the Kepstep that don't average to zero.
- We got rid of the problem by insisting that angular momentum is conserved during the step.

$$\vec{\mathbf{L}}_1 = (f\dot{g} - g\dot{f})(\vec{\mathbf{x}}_0 \times \vec{\mathbf{v}}_0) = (f\dot{g} - g\dot{f})\vec{\mathbf{L}}_0.$$

Angular momentum conservation implies that

$$f\dot{g} - g\dot{f} = 1.$$

We solve for one of these 4 functions, after using the universal Kepler's equation to iteratively find the other 3. This ensures that errors in angular momentum average to zero rather than accumulating.

Drift Kernel

Requires a sum of all momenta. This is done with a parallel reduction sum (Harris et al. 2008) on the GPU.

$$H_{Sun} = \frac{1}{2m_0} \left| \sum_{i=1}^{n} \mathbf{P}_i^2 \right|^2$$

References:

- Duncan, M. J., Levison, H. F., & Lee, M. H. 1998, A multiple timestep symplectic algorithm for integrating close encounters, AJ, 116, 2067
- Nyland, L., Harris, M., & and Prins, J. 2008, Chap 31, Fast N-body simulation with CUDA, in GPUGems3, edited by Hubert Nguyen, 2008, Addison-Wesley, Upper Saddle River, NJ, page 677
- Prussing, J. E. & Conway, B. A. 1993, Orbital Mechanics, Oxford University Press, Inc., New York, New York
- Harris, M., Sengupta, S, & Owens, J. D. 2008, Chap 39, Parallel Prefix Sum (scan) with CUDA, in GPUGems3,

Going beyond N-body: Identifying Collisions and close approaches

Regime of sparse collisions: Collision timescale is longer than an orbital timescale.

For Gravitating bodies:

- N-body uses a smoothing length to prevent spurious large velocity kicks caused by close approaches. Smoothing length is chosen based on typical mean particle spacing or Hill sphere radii.
- We would rather not use smoothing length but instead identify which particles are likely to approach within a Hill sphere. The 2 body problem can be solved exactly, including collisions leading to planetary growth or planetesimal destruction.

For dust particles:

- We can integrate particles that only feel gravity of massive bodies (and not each others) but can be sufficiently numerous to collide.

Algorithms for close approach and collision detection

For gravitating bodies:

- Typical travel distance in each timestep can be chosen to be small compared to Hill sphere.
- For planetesimals Hill sphere is large enough that a grid with each cell this size can be made.
- Procedure for rapidly identifying nearest neighbors: create a hash table for each particle on grid, followed by parallel sort. Can be done faster than the interaction step.

For dust

- So numerous and small that we could never simulate them all.
- Possible approaches:
  - sample density distribution with a grid so collisions probabilities can be computed from the density field. Drawback is that this is sensitive to the grid size and the number of particles and time used to construct the density distribution.
  - Inflate dust particle size and use approach above for gravitating bodies.
  - Create an SPH type kernel and use this to compute collision probabilities. Creating an SPH kernel can also be done rapidly in parallel on the GPU as sorting by distance can be done rapidly.

More information

- Alex Moore's talk on Tuesday at the DPS 2008 meeting
- http://arxiv.org/abs/0809.2855, Planet Migration through a Self-Gravitating Planetesimal Disk
- http://astro.pas.rochester.edu/~aquillen/research1.html , for code examples