Selected Methods in *N*-body Simulations

Aleksy Bałaziński [1]

[1]Faculty of Electrical Engineering, Warsaw University of Technology

Motivation

The dominant force over large distances is the gravitational force. According to the law of universal gravitation, the evolution of a system of N bodies with masses m_i and position vectors $\mathbf{x}_i$ is described by N equations

$$\ddot{\mathbf{x}}_i = -G \sum_{j \neq i} \frac{m_j}{|\mathbf{x}_{ij}|^3} \mathbf{x}_{ij}, \tag{1}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$. Direct application of (1) is the basis of the so-called *particle-particle* (PP) method. The method is characterized by $O(N^2)$ time complexity. Assuming that 50 ns are required to perform the floating-point operations under the summation symbol, $N = 50,000$, and 200 iterations, the simulation would take around **three hours** to complete. Therefore, it is evident that more efficient algorithms are needed to make simulations of this scale feasible. In this work we focused on two methods developed by Hockney and Eastwood [1]: particle-mesh (PM) and particle-particle particle-mesh (P^{3}M).

The PM method is very well suited for parallelization, therefore, we hypothesize that significant computational savings can be achieved by implementing the algorithm on a GPU. A careful analysis of the P^{3}M method shows that the short-range correction part of the algorithm can be slightly modified to allow for a lock-free parallelization. We explore this possibility and describe the results.

Particle-mesh method

The particle-mesh method can be described as the following sequence of four steps:

1. Assign masses to mesh points,
2. Solve the Poisson's equation on the mesh,
3. Calculate the field strength at mesh-points,
4. Find the field strength at each particle's location by interpolation.

In the implementation, we use the convolution method to find the potential. The method uses the mesh-discretized version of the integral form of Poisson equation,

$$\phi(\mathbf{x_p}) = V \sum_{\mathbf{p}'} G(\mathbf{x_p} - \mathbf{x_{p'}}) \rho(\mathbf{x_{p'}}), \tag{2}$$

where G is the Green's function (potential due to unit mass). The right-hand side of (2) is a convolution sum over a finite set of mesh points. If we assume periodic boundary conditions, we can apply the discrete Fourier transform to both sides and use the convolution theorem to conclude that

$$\hat{\phi}(\mathbf{k}) = \hat{G}(\mathbf{k}) \hat{\rho}(\mathbf{k}). \tag{3}$$

The currently supported assignment schemes are NGP, CIC, and TSC. The user can choose between two-point and four-point central finite difference in step 3.

Particle-particle particle-mesh method

The P^{3}M algorithm is a hybrid method: Forces between distant particles are calculated using the PM method, whereas, for particles lying closely together, the PP method is used. The total force applied to particle i is

$$\mathbf{F}_i^{\text{SR}} + \mathbf{F}_i = \sum_{j \neq i} (\mathbf{f}_{ij}^{\text{tot}} - \mathbf{R}_{ij}) + \mathbf{F}_i, \tag{4}$$

where $\mathbf{F}_i \approx \sum_{j \neq i} \mathbf{R}_{ij}$ is the *mesh force* computed using the PM method and $\mathbf{R}_{ij}$ is a prescribed *reference force*. In (4), $\mathbf{f}_{ij}^{\text{tot}}$ is the total force between particles i and j.

Our implementation supports two types of reference forces: the force between two spheres with uniformly decreasing density and the force between two solid spheres.

Short-range correction parallelization

To avoid double-counting, the particle i residing in cell $\mathbf{q}$ has to look for its neighbors in a subset $\mathcal{N}$ of the immediate neighborhood of $\mathbf{q}$. More specifically, define

$$\mathcal{N}(\mathbf{q} = (q_1, q_2, q_3)) = \{(q_1 + t, q_2 - 1, q_3 + s), (q_1 + s, q_2, q_3 - 1), (q_1 - 1, q_2, q_3) \mid s, t = -1, 0, 1\}.$$

Thus $|\mathcal{N}| = 13$, which is half of the size of the immediate neighborhood.

The short-range correction part of the P^{3}M method is shown below.

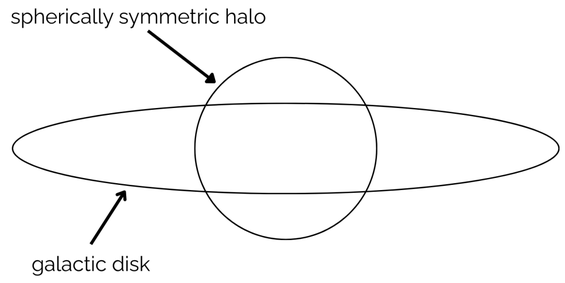
Algorithm 1 Short-range correction

```
for each chaining cell q do
    for each q_n ∈ N(q) ∪ {q} do
        for each i ∈ q do
            for each j ∈ q_n do
                if |y_i − y_j| > r_e then
                    break
                UpdateShortRange(i, j, q, q_n)
```

This procedure can be parallelized by splitting the work done in the outmost loop between some number of threads. In doing so, extra care has to be taken to avoid data races. By the construction of the set $\mathcal{N}$, it is possible to split the short-range force into 14 parts, each of which is accessed by only one thread at a time.

Galaxy model

The model of a galaxy used as a test bed for the implementation is a simple one. The galaxy is assumed to comprise only two parts: a thin disk and a spherically symmetric halo. The disk consists of tens of thousands of particles, each representing some number of stars. The halo is simulated as a fixed external gravitational field.



The disk particles are sampled from a radial distribution

$$p(r) = \frac{3}{\pi R_D^2} \left(1 - \frac{r}{R_D}\right),$$

where R_D is the radius of the disk and $r \leq R_D$.

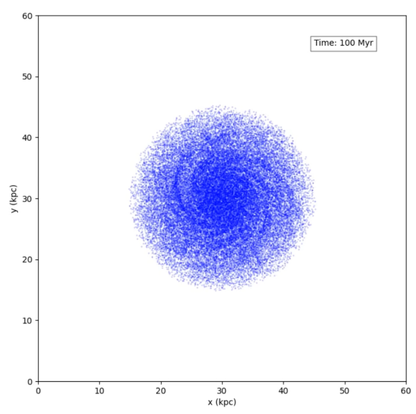
The density profile of the halo is analogous to the one used for the disk, save for the fact it is 3-dimensional, i.e.

$$\rho(r) = \begin{cases} \rho_0 \left(1 - \frac{r}{R_H}\right), & r \leq R_H \\ 0, & \text{otherwise,} \end{cases}$$

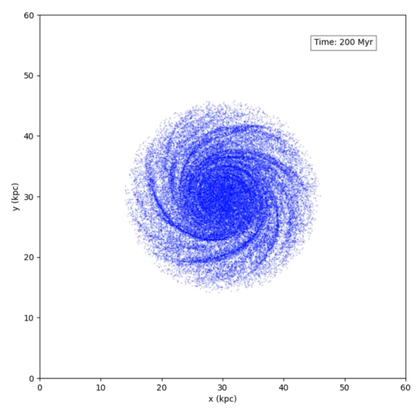
where R_H is the radius of the halo.

Results

In both methods $N = 50,000$ particles were used. Cell size H and time-step length DT were set to $60/64 = 0.9375$ kpc, and 1 Myr respectively. The evolution of the system over 200 Myr is shown below.

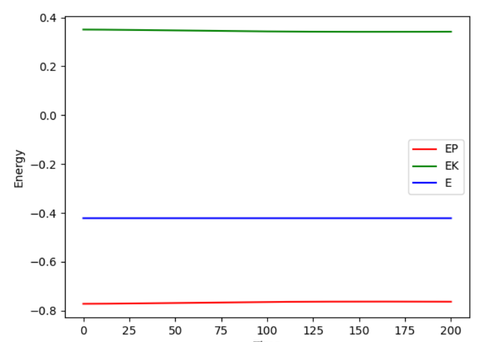


(a) $t = 100$ Myr

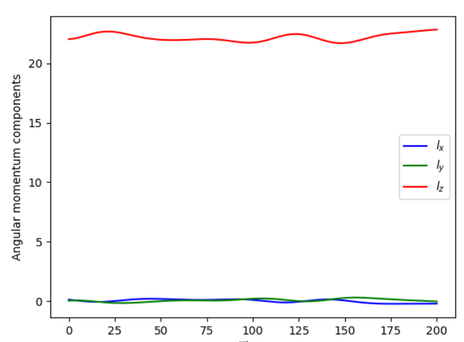


(b) $t = 200$ Myr

Graphs of energy and angular momentum components vs. time are shown below.



(a) Energy



(b) Angular momentum

The implementation was tested on a system equipped with an Intel Core i7-9750H CPU and an NVIDIA GTX 1650 GPU. To efficiently perform fast Fourier transforms, it relies on external libraries: FFTW for the CPU and cuFFT for the GPU.

The total runtime (200 iterations) of the PM method was approximately **15 seconds on the CPU** and **4 seconds on the GPU**.

In stark contrast, the P^{3}M method was significantly more resource-intensive, with a runtime of around **1 minute and 50 seconds**. The contribution of the PM component was negligible; a hybrid approach (executing the PM method on the GPU and the short-range correction on the CPU) yielded only marginal improvements in total runtime.

References

[1] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. CRC Press, 1st edition, 1988.