Uppsala Universitet

High Performance Programming
1TD062

Parallel Barnes-Hut simulation

Agelii, Carl.

September 20, 2024

Introduction and problem description

An N-body simulation is a simulation of N bodies under the influence of a force acting on them. In this case, we look at the gravitational force. Each particle affects every other particle through gravity. This could, for example, simulate the dynamics of a galaxy of stars. The gravitational force between two particles is:

$$F_{ij} = -\frac{Gm_i m_j}{r_{ij}^3}\mathbf{r}_{ij}, \tag{1}$$

where m is a mass, $G = 100/N$ is the gravitational constant, $\mathbf{r}_{ij}$ is the distance vector from particle i to j and r_{ij} is the magnitude of the vector. For this simulation, the domain is the box in the $xy-$plane given by $[0,1] \times [0,1]$. Also, we add a small number $\epsilon = 10^{-3}$ to the distance in order to prevent division by 0.

The Barnes-Hut algorithm

With a "brute force" method, solving the N-body problem would result in a time complexity of $O(N^2)$, which is very inefficient for large values of N. The Barnes-Hut algorithm aims to lower this complexity by exploiting the fact that a group of objects far away from a particle can be approximated as a single, large particle. This can significantly reduce the number of calculations. The way the algorithm works is by first building a quadtree structure for the particles, where each node is a square in the plane and has one or zero particles. Its children further divides the parent into four more nodes, and so on. Only leaf nodes contain particles. When iterating over the particles, we recursively calculate the force from each node. If the particle is sufficiently far away, the entire node containing particles is used for the calculation, avoiding traversing down the tree. The box is treated as a single particle if:

$$\frac{\text{Width of current box}}{\text{Distance from particle to box center}} < \theta_{max}, \tag{2}$$

where $\theta_{max} \in [0,1]$ is a parameter of the simulation.

In order to simulate the system, one has to use numerical integration from the calculated forces, since $F = ma$. In this study, we use both the Velocity Verlet and Symplectic Euler methods. Velocity Verlet reads:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2, \tag{3}$$

$$v(t+\Delta t) = v(t) + \frac{a(t) + a(t+\Delta t)}{2}\Delta t. \tag{4}$$

Symplectic Euler reads:

$$v(t+\Delta t) = v(t) + a(t)\Delta t, \tag{5}$$

$$x(t+\Delta t) = x(t) + v(t+\Delta t)\Delta t. \tag{6}$$

From equation 1, we can exploit the fact that

$$F_i = m_i a_i \implies a_i = \frac{F_i}{m_i} = -\frac{Gm_i m_j}{m_i r_{ij}^3}\mathbf{r}_{ij} = -\frac{Gm_j}{r_{ij}^3}\mathbf{r}_{ij}, \tag{7}$$

meaning multiplying by m_i in each iteration can be avoided.

Experiments

In order to verify the correctness of the Barnes-Hut algorithm, some experiments were conducted. First off, the correctness of the tree itself was tested. θ_{max} was set to 0 and Symplectic Euler was used to integrate. Then, it was compared to reference output data and performed very well, with a absolute difference of 0 for different values of N. Next, in order to understand what a good value of θ_{max} is, another experiment was done. Symplectic Euler was used on 2000 particles. The parameters was set as: $\Delta t = 10^{-5}$ and 200 time steps. θ_{max} was tuned so the absolute positional difference to the reference file was less than 10^{-3}. The error is plotted against θ_{max} can be seen in figure 1. When θ_{max} was set to 0.252, the absolute difference was calculated to 0.000983..., which was lower than the given tolerance.

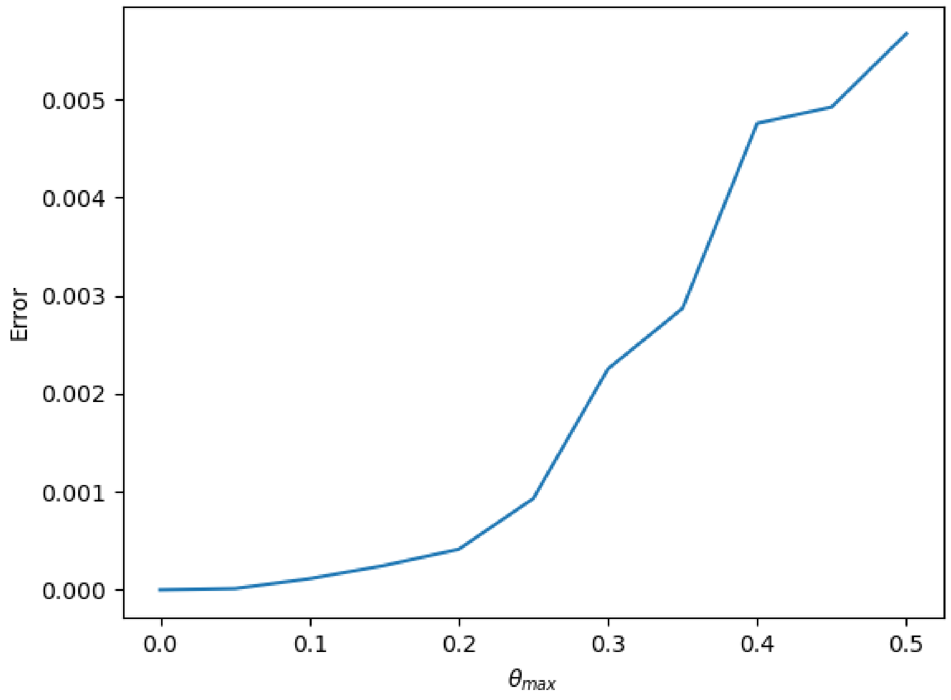

Figure 1: The error as a function of θ_{max}.

Now, the time complexity of the algorithm was studied using the found θ_{max}. Timings were taken for different number of N and can be seen in a loglog-plot in figure 2.

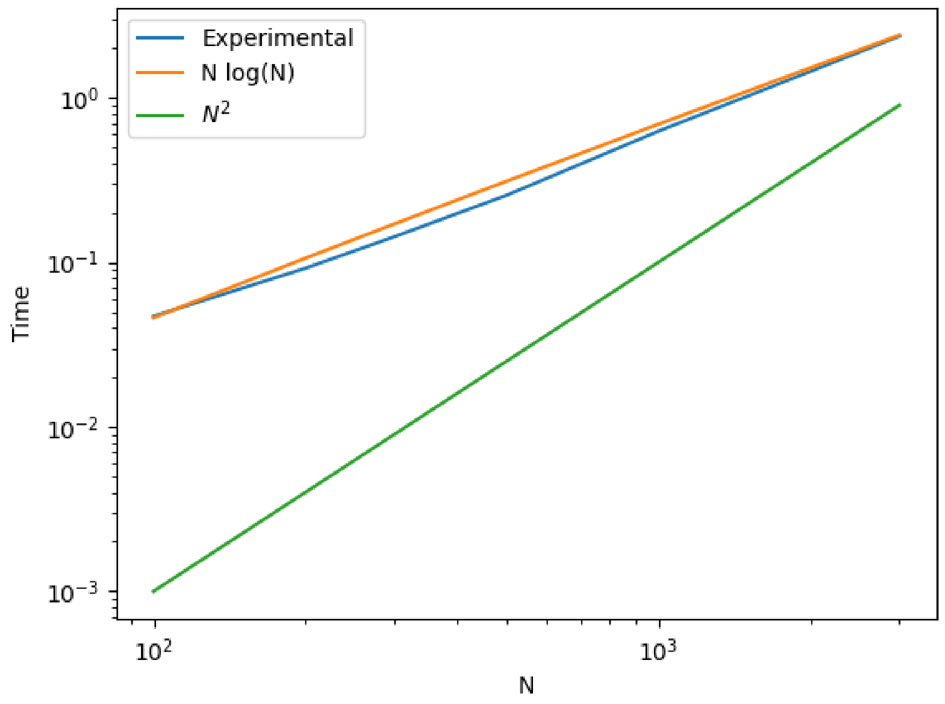

Figure 2: Time complexity study for the Barnes-Hut algorithm.

The plots in figure 2 are shifted in order to more clearly show the slope of the curves. Clearly, the experimental values has similar slope to the $N \log N$ slope. Linear regression gave a slope of ≈ 1.17. $N \log(N)$ has a slope of ≈ 1.16 and N^2 naturally has a slope of 2. This means that the experimental values are closest to the $N \log(N)$ slope, which is expected.
After this, we look at how well the algorithm could be parallelized. In order to do so, we again look at the case where $\Delta t = 10^{-5}$ and 200 time steps. θ_{max} was again set to 0.25. Then timings of the algorithm was done using different number of threads. The results can be seen in table 1.

Table 1: Execution Time (seconds) for Different Number of Threads

Threads	**N=100**	**N=500**	**N=1000**	**N=2000**	**N=5000**
1	0.141	1.415	3.762	9.504	30.433
2	0.092	0.788	2.064	5.114	16.162
4	0.072	0.47	1.166	2.808	8.763
8	0.055	0.291	0.699	1.67	5.011
16	0.051	0.217	0.489	1.086	3.119
32	0.06	0.197	0.404	0.958	2.43

A plot for the speed up can be seen in figure 3.

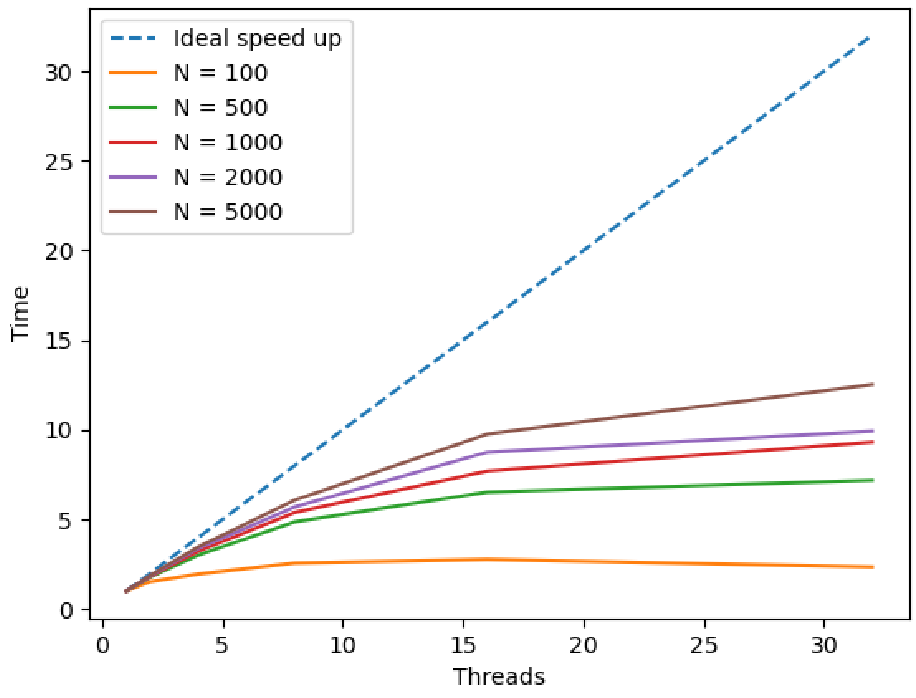

Figure 3: Speed up for different problem sizes.

Another experiment was issued, regarding the convergence rate of the numerical integration schemes. The study was done on three particles and the scheme looked like:

1. Calculate a reference simulation using $2 \cdot 10^6$ steps and $\Delta t = 10^{-8}$. Thus the simulation was run until $T = 2 \cdot 10^{-2}$ time units.

2. Calculate the absolute positional difference between the reference data and another simulation with fewer time steps but still choosing Δt so T remains the same. We start with $\Delta t = 10^{-3}$.

3. Lastly, continue to calculate the error as Δt becomes smaller and smaller. Here, we take $\Delta t = 10^{-3}$, 10^{-4}, 10^{-5}, 10^{-6} and 10^{-7}.

The results from the scheme for the two algorithms can be seen in a loglog-plot in figure 4.

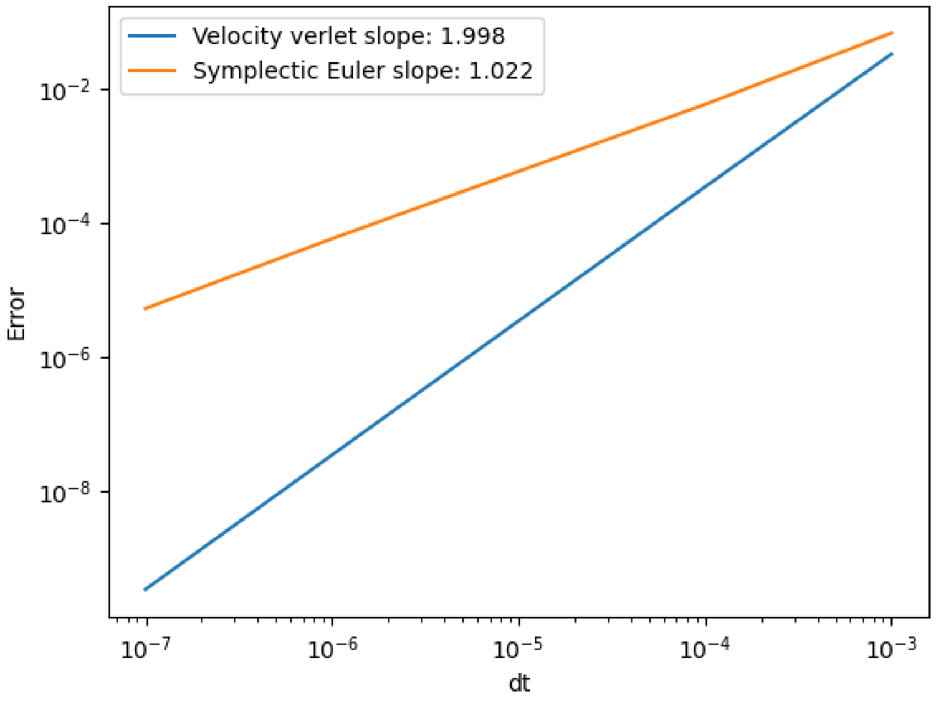

Figure 4: Loglog-plot of the experimental convergence rate of equation 3 and equation 5.

Lastly, we look at how the Barnes-Hut algorithm compares to the regular $O(N^2)$ method. Again, a reference simulation was run using velocity verlet and a very small $\Delta t = 2 \cdot 10^{-7}$, $5 \cdot 10^3$ steps, $N = 5000$ stars and $\theta_{max} = 0$. Consequently, $T = 10^{-3}$ time units. The error tolerance was set to 10^{-4}. Now, the parameters were tuned to get a simulation that was as fast as possible given the error tolerance. We use Velocity Verlet time integration, so in theory, it should perform better when compared to the $O(N^2)$ algorithm which uses Symplectic Euler. An error of $9.42 \cdot 10^{-5}$ was achieved when $\theta_{max} = 0.21$ with 100 time steps which means $\Delta t \approx 10^{-5}$. This simulation took approximately 1.456 seconds, but 25 threads were used. Using the straight forward algorithm, it took 63 seconds to get the same accuracy, and $\Delta t = 2.5 \cdot 10^{-6}$.

Conclusions

The Barnes-Hut scheme seems to work well, and the experiments proved useful for evaluating the performance. By looking at figure 2, it is clear that the scheme has a $N \log N$ time complexity, which is a huge improvement over the regular $O(N^2)$ scheme, especially for large N. The convergence study yielded very good results, and the time integration schemes converged at the expected rates. The plot in figure 3 seems to yield very bad results at first glance. This is because the speed up is not close at all to the ideal one. However, keep in mind that the speed up is very sensitive, especially for higher thread counts. For example, by looking at the column for $N = 5000$ in table 1, the time for 16 threads gives an approximate speed up of ten, but the ideal speed up would have granted a time of 1.9 seconds. This 1 second difference could be because of overhead, but also, because of the tree building and destroying in each time step, which takes more time for larger values of N. Furthermore, at least in my implementation, this tree building could not be parallelized, inducing further speed bumps. It was also experimented with different sort of schedules in OpenMP in order to handle the fact that the Barnes-Hut algorithm potentially can introduce load imbalances. The guided and static schedules performed virtually the same, but the dynamic one was much slower. This is probably because of the overhead of creating tasks.

The final experiment performed fairly well time wise, achieving a much faster time than the straight forward algorithm. Even if the straight forward algorithm were to be implemented with an ideal parallelization, the Barnes-Hut algorithm would be almost twice as fast. This would be even more so if the value of N would be larger, because of the time complexity of the algorithms.

A potential optimization would be to rework the tree instead of destroying it in each time step, so the dynamic memory allocation and tree traversal would not be necessary in each time step. Additionally, parallelizing the creation and destruction of the tree, if feasible, could potentially yield benefits. The data structure for the particles could also be problematic, if it introduces cache misses. A structure of arrays could potentially be better than an array of structures, because of vectorization and cache utility.