Uppsala University

High-Performance Programming

1TD062

N-Body Problem

Authors:
Lukas Billera
Filip Helgsten
Anton von Nandelstadh

February 6, 2026

1 Problem

For particle i and j, the gravitational force exerted on each other is given by (1)

$$\mathbf{f}_{ij} = -\frac{G m_i m_j}{r_{ij}^3}\,\mathbf{r}_{ij} \tag{1}$$

where G is the gravitational constant, m_i, m_j are the masses, and

$$\mathbf{r}_{ij} = (x_i - x_j)\,\mathbf{e}_x + (y_i - y_j)\,\mathbf{e}_y. \tag{2}$$

Hence, we can compute the total force experienced by particle i, using (3), where we use the small constant ϵ_0 to increase the robustness. This is because the force computation without it is inherently unstable for $r_{ij} \ll 1$. We get the formula

$$\mathbf{F}_i = -G m_i \sum_{j=0,\,j\neq i}^{N-1} \frac{m_j}{(r_{ij} + \epsilon_0)^3}\,\mathbf{r}_{ij}. \tag{3}$$

Now, to compute the particle paths, we use the symplectic Euler algorithm, given by (4)

$$\mathbf{a}_i^n = \frac{\mathbf{F}_i^n}{m_i}, \tag{4}$$

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \Delta t\,\mathbf{a}_i^n, \tag{5}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t\,\mathbf{u}_i^{n+1}. \tag{6}$$

2 Solution

2.1 Data structures

We store the positions, mass, velocities and brightness in a buffer of size $6 \times N \times$ sizeof(double), which we initialize by reading straight from the initialization file. We can then reference the values of particle i starting from buffer[i] and ending with buffer[i+5]. Since our buffer is small, we can put it on the stack, so we incur minimal memory-related constraints in our code.

Listing 1: Reading initial values into buffer.

```c
    FILE *file;
    file = fopen(filename, "r");
    double buffer[6 * N];

    /*Read in the initial state*/
    fread(buffer, sizeof(double), 6 * N, file);
    fclose(file);
```

2.2 Algorithm

The main loop is just a double for loop, where we do the most straightforward implementation of the algorithm described above. This is not the fastest solution, but we use it because we've been told to. Clearly, based on the nested for loops iterating over all of the data, it is $\mathcal{O}(N^2)$. The other sensible algorithm for n-body simulation is Barnes-Hut, which is $\mathcal{O}(nlogn)$. However, the algorithm we use is more accurate and simpler to implement. We use constant values $\epsilon_0 = 10^{-3}$, $\Delta t = 10^{-5}$ and $G = \frac{100}{N}$. G is not actually a constant but scales inversely with the number of particles. This to scale the system reasonably for different values of N.

Listing 2: Main loop.

```c
     for (int time = 0; time < nsteps; time++) {
        for (int n = 0; n < 6 * N; n += 6) {
            /* Compute Fx and Fy values*/
            double Fx = 0, Fy = 0;
            for (int m = 0; m < 6 * N; m += 6) {
                if (m == n) continue;
                double xdiff = buffer[n] - buffer[m];
                double ydiff = buffer[n + 1] - buffer[m
                   + 1];
                double diff = sqrt(xdiff * xdiff + ydiff
                    * ydiff);
                Fx += -G * (buffer[m + 2] / pow((diff +
                   e0), 3)) * xdiff;
                Fy += -G * (buffer[m + 2] / pow((diff +
                   e0), 3)) * ydiff;
            }

            /*Update buffer values*/
            double ax = Fx;
            double ay = Fy;
            buffer[n + 3] += dt * ax;
            buffer[n + 4] += dt * ay;
            buffer[n] += dt * buffer[n + 3];
            buffer[n + 1] += dt * buffer[n + 4];
        }
    }
```

3 Performance

With the bulk of the code finished, several different optimization flags were tested for efficiency gains. The final product particularly concluded with an `-Ofast` flag, a `-march=native` flag and a `const` cast of the graphics variable. The variation of these parameters is shown below. All runs are compared to a standard fully optimized run with 3000 particles and 100 steps which took 20.785 seconds.

Optimization	Without opt. [s]	Improvement [s]
`-Ofast`	141.409	120.624
`-march=native`	21.245	0.460
`const` graphics variable	20.937	0.152

Table 1: Comparison of the performed optimizations, and their respective time difference calculated with respect to a standard run of the final program which took 20.785 seconds.

To examine the actual time complexity of the algorithm, the execution times for different particles counts N is observed. The graph below shows how execution time varies with N.

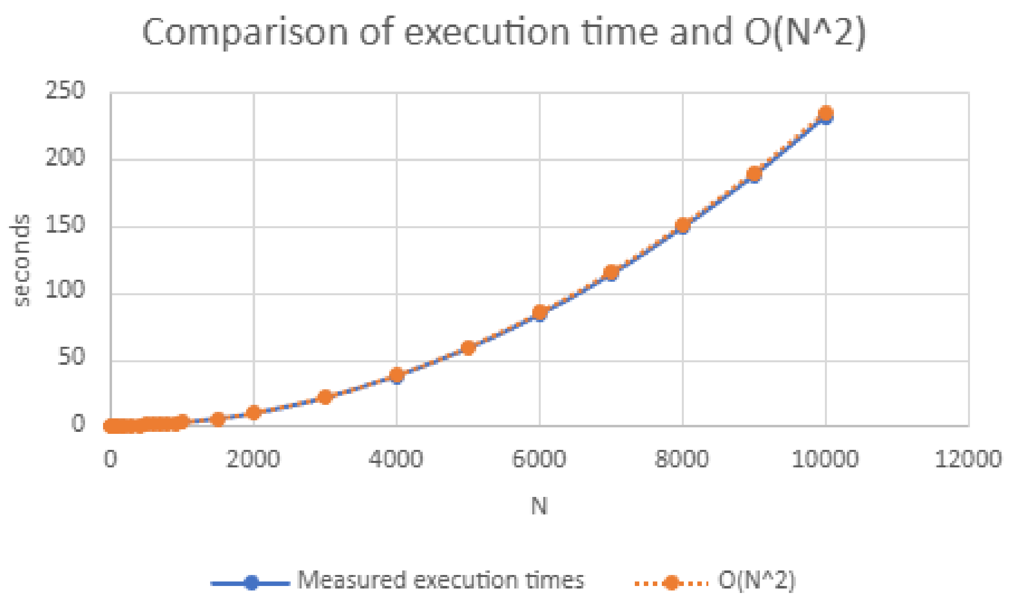

Figure 1: The figure shows the actual execution times in blue and the corresponding $\mathcal{O}(N^2)$ approximation in dotted orange.

4 Discussion

Judging from the table of optimizations, the main optimization is obviously the `-Ofast` flag, which reduced the execution time by about 86%. The other optimizations that were made had little impact compared to this, but at least the `-march=native` flag had some impact. It is however unclear whether the `const` cast was of any importance.

The measured execution times overlaps very well with the $\mathcal{O}(N^2)$ approximation which confirms that this time complexity was actually reached.

5 References

We used chatGPT and Copilot for debugging, e.g. asking "why does this code give a segfault".