Assignement 3
Mary Chrishani Jayaweera
Magnus Sahlin

1 Introduction

The aim of this report is to build and optimize a program that can simulate the forces between masses and thereby simulate the movement of stars and planets. We also want to evaluate the complexity of the program.

1.1 Theory

Simulating the galaxy is an n-body problem. The governing equation include the Newton's law of gravitation for force calculations.

$$\mathbf{f_{ij}} = \frac{Gm_i m_j}{\mathbf{r}_{ij}^3}\mathbf{r_{ij}} = \frac{Gm_i m_j}{\mathbf{r}_{ij}^3}\hat{\mathbf{r}}_{ij} \tag{1}$$

where G - gravitational constant m_i and m_j are masses of the particles $\mathbf{r}_{ij}$ position of particle i relative to particle j

To calculate the force on one star:

$$\mathbf{F_i} = -Gm_i \sum_{j=0, j\neq i}^{N-1} \frac{m_j}{\mathbf{r}_{ij}^2}\hat{\mathbf{r}}_{ij} \tag{2}$$

After the forces are calculated the acceleration and velocity can be determined.

$$\mathbf{a}_t = \frac{\mathbf{F}}{m} \tag{3}$$

Choosing a time step dt, the velocity and position change can also be determined using the Euler Method.

$$\mathbf{v}_{t+dt} = \mathbf{v}_t + \mathbf{a}_t \cdot dt \tag{4}$$

$$\mathbf{p}_{t+dt} = \mathbf{p}_t + \mathbf{v}_{t+dt} \cdot dt \tag{5}$$

2 Implementation

2.1 Plummer Spheres

The particles or the planets and the stars in the simulation in reality have a certain extension in space and are not infinitely small points. To account for this and the instability induced in the simulation when the denominator in equation 6 goes to zero, we add a constant ϵ to the denominator. What this does in practice is setting an upper limit to the magnitude of the force. This constant is set to 0.001 according to instructions.

$$\mathbf{F_i} = -Gm_i \sum_{j=0, j\neq i}^{N-1} \frac{m_j}{\mathbf{r}_{ij}^2 + \epsilon} \hat{\mathbf{r}}_{ij} \tag{6}$$

2.2 Complexity

The simplest implementation is calculating the force between each and every star separately. This is in theory also the most accurate and most performance heavy implementation. For a simulation with 10 stars, each star will have 9 forces acting on it for a total of $10 \cdot 9 = 90$ forces in the simulation. If we would double the number of stars, then every star will have twice as many forces acting on it, but the total number of forces in the simulation have quadrupled, $20 \cdot 19 = 380$. This means that the implementation is of quadratic complexity.

2.3 Hands-on Optimization

One easy optimization is recognizing that each force actually have a duplicate but opposite force from the other star. Implementing that these "twin forces" are only calculated once will in theory make the code twice as fast. However it will not change the complexity of the implementation.

The code was written so as to minimize the use of function calls to a reasonable extent.

3 Performance

Performance was evaluated by measuring the time of a set number of time steps but with different amount of stars in the simulation.

In table 1 we can see that as N increases, the time increases squared. This is according to the specification in the assignment. For small N the pattern is not apparent, since most of the computing time is not directed towards the actual simulation, but rather memory accesses and other initialization things. For the first four measurements, the number of particles are doubled for each test, so the time should be quadrupled. The same is true for the last four.

N [#]	Real [s]	User [s]	System [s]
100	0.092	0.083	0.009
200	0.213	0.209	0.004
400	0.319	0.316	0.004
800	2.284	2.280	0.004
1000	3.747	3.739	0.008
2000	14.004	13.957	0.044
4000	56.885	56.878	0.004
8000	229.6830	229.484	0.177

Table 1: Time for 200 iterations with different N

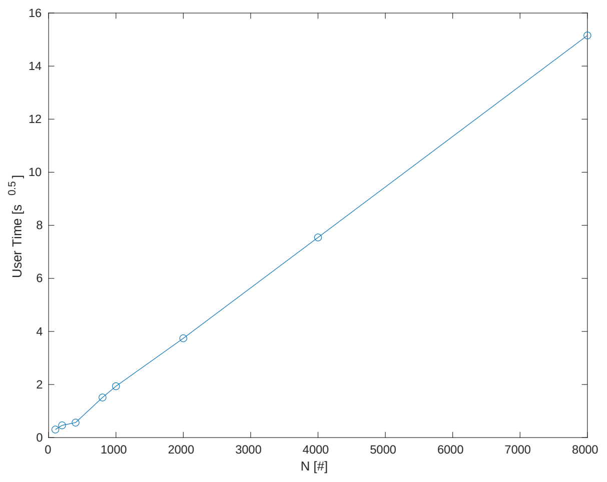

Figure 1: Time for 200 iterations with different N

In figure 1 we can clearly see that when the time is plotted squared rooted, we see a linear relation. This means that the model is of $\mathcal{O}(n^2)$ complexity.

3.1 Compiler Optimization Flags

We found that the Ofast flag is the best for our purpose. Running the same simulation with 3000 stars for 100 iterations without graphics takes 3.771 seconds with the O2 flag, 3.539 second with the O3 flag, and 3.092 seconds with the Ofast flag. For correctness and reproducibility all tests were made on the same computer with processor i5-6200U and the same compiler GCC version 7.4.