HPP A3

Emanuel Bodin, Torsten Malmgård

February 2020

1 Introduction

The N-body problem is about predicting the individual motion of a group of objects interacting with each other in a gravitational space, for example the motion of the sun, stars and planets. By using Newtons law of gravitation, eq 1, describes the force two particles i and j exert on each other.

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

For a space consisting of N bodies, eq 1 can be rewritten to eq 2, where ϵ_0 is a small number, here 10^{-3}.

$$\mathbf{F_i} = -Gm_i \sum_{j=0, j\neq i}^{N-1} \frac{m_j}{(r_{ij}+\epsilon_0)^3}\mathbf{r_{ij}} \tag{2}$$

By using the symplectic Euler time integration, the velocity $\mathbf{u_i}$ and position $\mathbf{x_i}$ of particle i can be determined by the following formulas:

$$\mathbf{a_i^n} = \frac{\mathbf{F_i^n}}{m_i} \tag{3}$$

$$\mathbf{u_i^{n+1}} = \mathbf{u_i^n} + \Delta t \mathbf{a_i^n} \tag{4}$$

$$\mathbf{x_i^{n+1}} = \mathbf{x_i^n} + \Delta t \mathbf{u_i^{n+1}} \tag{5}$$

This straight forward solution becomes computationally expensive for large values of N, the total number of calculations will grow as $\mathcal{O}(N^2)$. This assignment is about implementing a code that performs a simulation of N particles for a certain number of time steps in a two dimensional space. The initial position and masses of the particles will be read from a file, after the simulation the final result will be written to a file which then can be compared with the some reference output files to check the correctness of the simulation.
When having a solution that yields correct result, the code should be optimized using both compiler flags and code changes.

2 Solution

[H] Since the data from the input files is structured such as in fig 1, a struct containing six double number is very suitable for storing information for a certain particle. An array of size N could then be allocated storing data for each particle. The code first extracts the parameters from the command line call, setting some constant variables. It then allocates a particle array and reads the particle data from the desired input file, storing each particle in the array. The code the moves on to the calculation part where it first allocates a new particle array where it can store the particle values for the next times steps, before it

moves on to the actual calculations. The calculations consists of three loops, the outer most loops through the desired time steps, the middle one every particle to determine the total force it is exerted to and the innermost the rest of the particles.After the loops finished, we wrote all the particle data to a gal file to be able to compare the result with some reference output.

```
particle 0 position x
particle 0 position y
particle 0 mass
particle 0 velocity x
particle 0 velocity y
particle 0 brightness
particle 1 position x
particle 1 position y
particle 1 mass
particle 1 velocity x
particle 1 velocity y
particle 1 brightness
```

Figure 1: Data structure of input file.

3 Performance and Discussion

First measurements with the -O1 took almost 1 minute 30 seconds on a MacBookPro 17 using a 2,3 GHz Dual-Core Intel Core i5. Recompiling with -02 tripled the performance and using -O3 increased it even further, using Ofast gave a marginaly better performance. See fig. 3 for a comparison of the results. Further optimization using vectorisation was tested. Using the flag -fopt-info-vec-missed in the compiler gave some insights on what improvements could be made. Unfortunately this showed that most loops where nested andour inner loop contained an if-statment, making vectorization impossible. The other loops

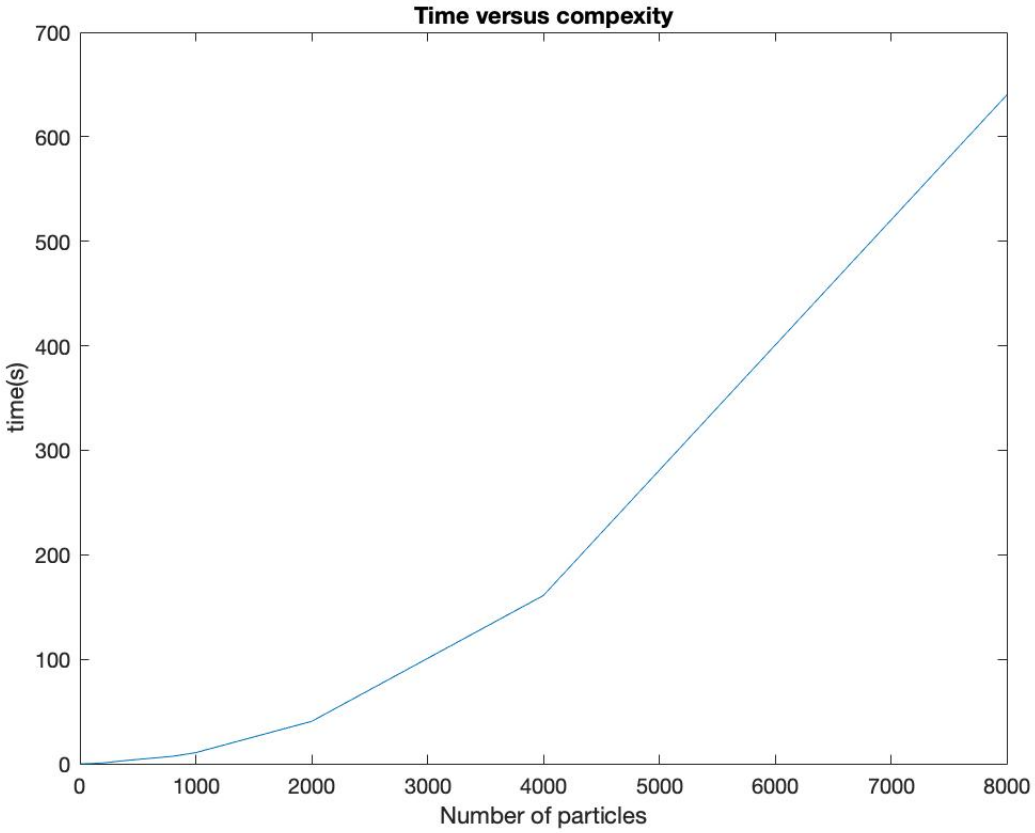

Figure 2: Measured time for simulations with different amounts of particles.

Table 3: .

Optimization	Time [s]
O1	96
O2	28.5
O3	26.7
Ofast	3.09