HPP Assignment 3

Charlotta Jaunviksna 910618-0186, Amanda Norberg 910902-0405

February 7, 2018

1 Introduction

According to Newton's law of gravitation, a particle is acted upon by other particles by a gravitational force. This total force on a target particle has a contribution from each of the surrounding particles. Each gravitational force contribution is proportional to the mass of both the target and neighboring particle as well as the distance between the two particles in question. The total gravitational force on a target particle can be computed as the sum of all force contributions.

The goal is to write a program which simulates an arbitrary number of particles' evolution over a given number of time steps. The program will be developed with special regard to optimisation and aims to be as computationally efficient as possible.

The calculation of the gravitational force acting on a particle demands multiple computations in each time step. These are further explained in section 1.1.

1.1 Theory

The equation used to compute the total gravitational force working on a particle i by all other particles in a distribution of N particles is given by

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

where G is the gravitational constant, m_i and m_j are the masses of the particles i and j, r_{ij} is the distance between the particles, and r_{ij} is the normalized distance vector.

For $r << 1$ the computation becomes unstable. For this special case, the following equation is introduced

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

where ϵ_0 is a small number preventing the function value from diverging for small values of r_{ij}.

The velocity of a particle acted upon by this force can be calculated as

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \Delta t \frac{\mathbf{F}_i^n}{m_i} \tag{3}$$

which then makes it possible to calculate the new position of the target particle:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{u}_i^{n+1} \tag{4}$$

For integrating in time the *symplectic Euler method* was used.

2 Method

2.1 Implementation

A program named *galsim* was implemented in C. The executable took five user specified input arguments

N	-	number of particles to simulate
filename	-	input file in .gal format with the particles' inital configuration
n_steps	-	number of time steps
delta_t	-	time step size
grapichs	-	0 or 1 specifying if graphics were to be used during the simulation

Focus was firstly put on creating a code which performed the given task of simulating the particles. The program consisted of two files: *galsim.c* and *particle_functions.c.*

galsim was the main program file which read information about particles from .gal files into a buffer. The data in the buffer was read into structs that represented particles and held information about mass, current position, current velocity and brightness. The file then run the simulation over the given number of time steps, calculating the new position of each particle in each time step from the gravitational force acting on each particle by the others. When a simulation had finished, the results were stored at the same format as the input configuration file.

The functions were gathered in the file *particle_functions* and called upon by the main program file. Table 1 presents the functions of the program together with a short explanation.

Table 1

Function name	Summary
get_abs_dist	Computes the absolute distance between two particles in two dimensions
get_part_dist_1D	Returns the distance between two particles in one dimension
get_force_1D	Computes the force acting on a target particle by all other particles in one dimension
get_vel_1D	Returns the velocity of the target particle in one dimension
get_pos_1D	Computes the new position of the target particle in one dimension using the other functions

The functions were divided like this as to avoid superfluous input in each function. No global variables were used. To avoid that the computation of the new position used all variables all the time, a new function was written for each part of the computation that only used the variables that were necessary for that specific part.
Simulations were run for different number of particles and time steps and the result files were checked against some given reference files. This was done with the executable file *compare_ gal_files* which compared the final particle positions in the two files.

2.2 Optimisation

The code was continuously optimized during the work. Examples of this were the following:

- The memory for the array containing the particle structs was allocated statically and not dynamically.
- Loop unrolling was used instead of an inner loop when reading and writing initial and final particle configurations. The unroll factor used was 6, as the particle attributes were this many.
- The buffer used for reading the input data was re-used when the results from the simulation were to be written to an output file.
- No global variables where used. Instead, variables of great importance where passed as input arguments to the functions using them.

When the program worked properly, further modifications for better performance were investigated. The following approaches were tested and the runtime before and after were measured.

- The -O3 compiler flag was added.
- The -Ofast compiler flag was added.
- Restrict keywords were added for pointers used as input parameters to particle functions.
- Constant variables were added where possible.
- The -funroll-loops compiler flag was added.
- The type float was used for floating point variables instead of double.

3 Result and discussion

For all simulations the time step size was set to 0.00001.

Table 2 shows the maximum difference between final particle positions between the result file and the reference file for different number of particles and time steps.

Table 2

N	nsteps	pos_maxdiff
10	200	0.036670123553
100	200	0.188637088235
500	200	0.049003791944
1000	200	0.035680276975
2000	200	0.02872175284
3000	100	0.005959215524

As seen in table 2, the error decreases when the number of particles increases with an exception for when N is set to 100. In general, the result is not very good as the errors should be within a few orders of magnitude from the size of rounding errors.
The final optimisation techniques used were adding of the -Ofast compiler flag, adding of restrict keywords and using of constant variables where possible. This, because these shorted the runtime remarkably.

Runtimes before and after final choice of optimisations are shown in Table 3.

Table 3

N	nsteps	time_std	time_opt
10	200	0.012s	0.009s
100	200	0.546s	0.082s
500	200	9.526s	1.957s
1000	200	37.755s	7.878s
2000	200	2m31.047s	31.534s
3000	100	2m51.433s	36.104s

Changing to using the type float instead of double for floating point variables shorted the runtime but it turned out that the input file was not read properly. This was discovered after the initial particle attributes had been controlled with grapchics. Initial parameter configurations had been read wrongly. Casting the doubles read from the initial configuration file into float was tested. This made the program run properly but resulted in a runtime that was about the same as if doubles were used though out the whole process. This resulted in using going back to using double as type for floating point variables.

The adding of the compiler flag -funroll-loops did not make any difference in runtime and was therefore not used.