HPP Assignment 4

Charlotta Jaunviksna 910618-0186, Amanda Norberg 910902-0405

February 16, 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 with the use of the Barnes Hut method to calculate the force exerted on each particle.

In a N-body problem, the forces exerted on a particle by a cluster of particles, can be approximated to that by the force exerted by a single object with the total mass off all the particles in the cluster and with center of mass in the center of the cluster. This can be used to compute the forces on each particle with the Barnes Hut algorithm that uses a quad tree to divide up the particles. Then the forces can be recursively computed by traversing up the tree.

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. Because of this, the following equation is introduced instead

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

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.

A cluster of particles can be seen as one single object when calculating the force they exert on a distant particle. This approximation is reasonable if $h/r << 1$, where h is the maximum distance between two particles in the cluster, and r is the target particles distance to the cluster. In this implementation the use of this approximation will be determined by a variable theta where

$$\theta = \frac{\text{width of current node}}{\text{distance to particle from center of node}}. \tag{5}$$

θ will be compared to a threshold value $\theta_{max} \in [0,1]$ to see if the approximation is feasible. If $\theta \leq \theta_{max}$ the cluster can be approximated as a single object.

The Barnes Hut algorithm uses this approximation by firstly dividing the domain into four equally large squares which in turn are divided in to four new squares. This is repeated until every particle in the space is placed in a square of its own. The structure is called a *quad tree* and the squares are called *quadrangles*. The next step in the algorithm is to compute the mass and center of mass of every quadrangle recursively. The third and last step is to, by traversing the quad tree, compute the force on every particle.

2 Method

2.1 Implementation

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

N	-	number of particles to simulate
filename	-	input file in .gal format with the particles' initial configuration
n_steps	-	number of time steps
delta_t	-	time step size
theta_max	-	threshold value for Barnes Hut algorithm
graphics	-	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 three files: *galsim.c*, *particle_functions.c* and *tree_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 and current velocity. The file then ran 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 computing the particles' movements 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 contribution from one particle to the sum used for force computation (See equa
get_vel_1D	Returns the velocity of the target particle in one dimension
get_pos_1D	Computes the new position of the target particle

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.

In *tree_functions* all functions concerning the use of a quad tree were placed. The first section of functions viewed in table 2 are functions concerning the creation of a quad tree: creating and splitting nodes. The second section are functions that are used to compute the center of mass of each node.

Table 2

Function name	Summary
new_node	Creates new empty node with specified top left corner coordinates and width
isempty	Checks if node is empty
isleaf	Checks if node is a leaf (have a particle stored)
ispointer	Checks if node is pointer (have children but no particle stored)
split_node	Creates four children to the node, inserting the node's particle into matching child node
get_quadrant	Gets quadrant (node) in which to place particle
match_coords	Checks if particle's coordinates matches those of quadrant
insert	Inserts particle into quad tree
have_same_pos	Checks is a particle and a node has the same position
get_theta	Calculates theta for target particle and specified node
calc_cm	Recursively calculates and sets the center of mass of all nodes in the quad tree
set_cm	Sets the mass and coordinates of an empty center-of-mass-struct
update_cm	Sets the center of mass of mother node from the childrens' centers of mass.
calc_forcesum	Recursively calculates the 1D force on a target particle

The correctness of the program was checked by using graphics, which showed the expected behavior of the particles. Simulations were run for different numbers of particles and time steps and the result files were checked against some given reference files with values from a straightforward implementation with complexity $O(N^2)$. This was done with the executable file *compare_gal_files* which compared the final particle positions in the two files. θ_{max} was set to zero during this check.

The acceptable value of θ_{max} was determined by running a simulation with 2000 particles over 200 time steps, choosing a value where the result did not differ more than 10^{-3} when compared with the reference file.

The occurrence of memory leakage was investigated with `valgrind` and no errors or leakages were found.

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 particle attribute *brightness* was left out in the struct as this was not used in the program.
- 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.
- Constant variables were used where possible.
- Using of boolean short circuit. The most unlikely of the expression in if-statements using && was put first.

Further modifications for better performance were investigated. The following approaches were tested and the runtime before and after were measured with the `time` command.

- The -Ofast compiler flag was added.
- The -funroll-loops compiler flag was added.
- Function inlining was used for functions called within other functions.

3 Result and discussion

For all simulations the time step size was set to 0.00001 and the threshold value for the Barnes Hut algoritm (θ_{max}) was set to 0.025. The CPU used was Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz.

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

The final optimisation techniques used, in addition to those used during implementation, was to use the -Ofast flag.

Runtimes after final choice of optimisations are shown in Table 3, where *time_ std* is the runtime before optimisation and *time_ opt* is after.

Table 3

N	nsteps	time_std_Barnes_Hut	time_opt_Barnes_Hut
10	200	0.017s	0.015s
100	200	0.267s	0.121s
500	200	2.831s	1.219s
1000	200	7.878s	3.341s
2000	200	19.684s	8.973s
3000	200	32.158s	15.684s
5000	200	1m3.253s	32.866s
10000	200	2m27.811s	1m16.602s
20000	100	3m14.653s	1m36.855s

Figure 1 shows the runtime after the Barnes Hut implementation had been optimised, plotted against the number of particles used in the simulation. As seen, the time increases with $O(N)$. This was a great improvement compared to the implementation without the Barnes Hut algorithm of complexity $O(N^2)$.

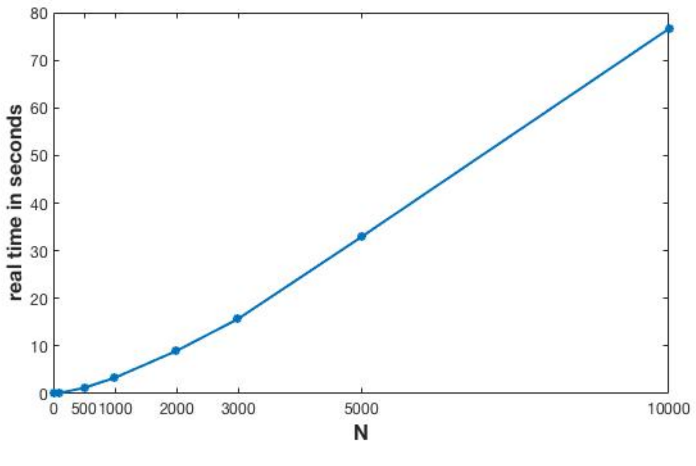

Figure 1

Distribution of work

The coding, optimisation and writing of the report was equally divided between the members of the group.