Assignment 4

High Performance Programming

Author:

Fredrik Gustafsson

Sara Hedar

Uppsala

February 21, 2019

The Problem

Introduction

When simulating a large number of objects, such as a n-body system of planets and stars, the choice of algorithm and optimisation of the code is important. To reduce the computation time compared to a naive $O(n^2)$ the Barnes-Hut algorithm was implemented to simulate the n-body system. To further improve performance the code was optimised using compiler flags and micro optimisations.

Theory

The equations governing the attracting force between objects follows Newton's law of gravity:

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

where G is the gravitational constant, m the mass and $r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$. However there is a built in instability when $r_{ij} << 1$, to avoid this we modify equation 1 into:

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

where N is the total number of object. To then update the velocity and position of a object we use equation 2-5.

$$\boldsymbol{a_i} = \frac{\boldsymbol{F_i}}{m_i} \tag{3}$$

$$\boldsymbol{u}_i^{n+1} = \boldsymbol{u}_i^n + \Delta t \boldsymbol{a}_i^n \tag{4}$$

$$\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \boldsymbol{u}_i^{n+1} \tag{5}$$

in which $\boldsymbol{u}_i^n$ is the velocity of object i at time step n and $\boldsymbol{x}_i^n$ is position of object i at time step n.

Algorithm

The naive way to do the n-body simulation is to simply loop over each object i and calculate the force from each object j, $i \neq j$, with equ1-5. Such approach however would have a time complexity of $O(n^2)$, a better algorithm for this problem would be the Barnes-Hut algorithm. The Barnes-Hut, algorithm works by sorting the objects in a quadtree. This then allows us to, if the distance from our object is sufficiently large, replace several nodes with one node with a centre of mass, and total mass. Both the centre of mass and the total mass of a node gets updated when the node is split or an object is added further down in the tree quadtree. To calculate if the distance is sufficiently large we compare a value θ to θ_{max} where θ_{max} is chosen and θ is calculated with the equation

$$\theta = \frac{\text{Width of current box containing particles}}{\text{distance from particle to center of box}} \tag{6}$$

The Solution

The initial condition to the system of interstellar objects was provided as an binary file containing floating point values of type `double`. For each object 6 values was supplied; position in the two dimensions, velocity in the two dimensions, mass of the object and brightness of the object. Both mass and brightness was considered constant during the simulations while both the positions and velocities were updated. When the simulations was completed the results were to be written to a binary file on the same format as the input data (initial conditions provided).

The problem was divided into three parts; loading the initial conditions from a binary file, performing the simulation and storing the final solution into a binary file. The problem was solved in on single .c file and one function was written for each part. The number of objects to simulate, the input file, the number of time steps, the size of the time step and an option to use graphics was given as input parameters to the program. The graphics option was not implemented.

The data for each object (position, velocity, mass and brightness) was stored in the composite data structure called `struct` provided in C. A dynamically allocated array was used to keep track of all of the `struct`. Memory was allocated once and the entire input file was read at once to reduce overhead.

The interstellar object were stored in a quadtree. Each node contained a reference to it's four children, the total mass of the particles further down in the tree, the centre of mass of the particles further down in the tree, the coordinates of the centre of the region, the distance form the centre of the region to the boundary, a flag determining if the node is a leaf node and the stored data of an interstellar object

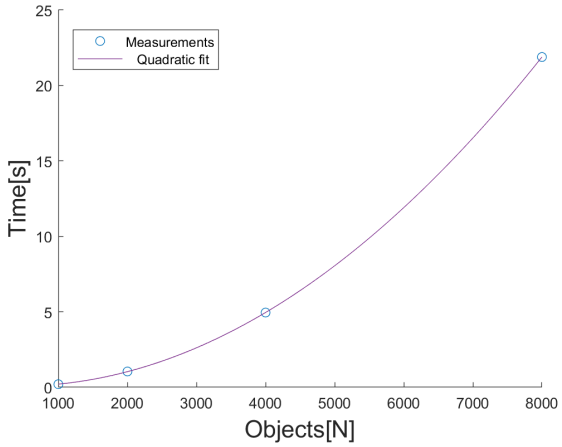
The simulation was performed through a time loop where each iteration corresponded to advancing the system with one time step. First each object was sorted into a quadtree. Then the acceleration of each particle was computed using the Barn-Hut algorithm and the velocity was updated. Next the position for each particle was updated in a separate loop, as not to interfere with the calculation of the acceleration. Finally in the time loop the tree was deleted. The complexity of the algorithm implemented was $\mathrm{O}(N\log(N))$, where N is the number of interstellar objects.

Performance and Discussion

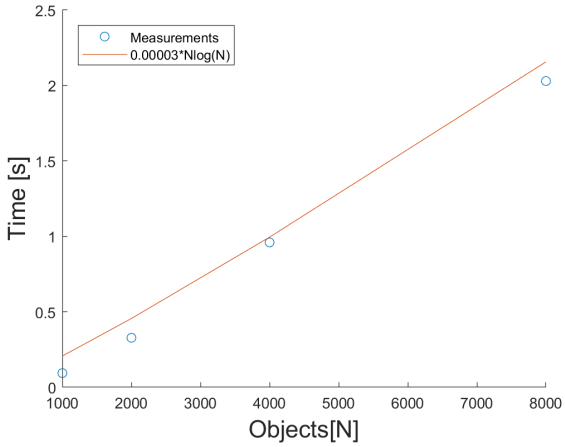
The computations was performed on Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, Unbuntu 18.04.1 LTS on Windows 10. The compiler used was gcc 7.3.0.

Complexity of the Algorithm

To measure the complexity of the Barnes-Hut algorithm a value for θ_{max} was chosen such that the error didn't exceed 0.001. This value was found to be 0.25 however one could try to find a more exact value however doing this optimisation would only yield a marginally better time performance. The algorithm with $\theta_{max} = 0$ and $\theta_{max} = 0.25$ together with quadratic fit and a function respectively is shown in figure 1a and 1b below.



(a) $\theta = 0$.



(b) $\theta = 0.25$.

Figure 1: Plots of user time as a function of the number of simulated objects for $\theta_{max} = 0$ and $\theta_{max} = 0.25$. In the figure for $\theta_{max} = 0$ a quadratic fit is added. In the figure for $\theta_{max} = 0.25$ the function $f(N) = 0.00003 * N * \log(N)$ is added. The simulation was performed over 10 time steps, after the code was optimised.

Optimisation of the Code

After having ensured that the code fulfils the criteria, max_posdiff$\leq$ 0.001 for $\theta_{max} \geq 0.0$, and having found the highest θ_{max} that still fulfils it optimisation of the code started. The first thing which was tested was different optimisation flags. The results for these tests with N = 1000, steps = 200 and dt = 1e-5 is shown below in table1.

Table 1: The modifications done to the different models and the error before and after modifying

	-O0	-O1	-O2	-O3	-Ofast	-funroll-loops	-march=native
Real	16.961	15.107	14.951	14.919	2.799	16.988	16.972
User	16.516	14.828	14.656	14.547	2.688	16.672	16.703
Sys	0.000	0.016	0.000	0.047	0.016	0.000	0.016

	-Ofast -funroll-loops	-Ofast, -march=native	-Ofast, -march=native, -funroll-loops	attribute packed for tree node
Real	2.801	2.722	2.664	2.648
User	2.625	2.609	2.547	2.547
Sys	0.000	0.000	0.016	0.016

Next a measurement of the time used by the function which calculates the acceleration was done. The resulting time became 0.43s, total real time 0.528s, if one then where to manually optimising the code this function would be a good candidate. One optimisation which could be done is changing the pow function call to multiplications instead. Other things which we did with the code was to add the const keyword to our G variable since it will not change at runtime. We also looked at using the restrict keyword however

since non of our functions uses more than one of each kind of pointer this should not affect the performance.

We kept the problem with padding of structures in mind when designing them, doubles and pointers above ints. We also changed some of our smaller declarations from heap to being stac. Another change which could be done was saving our resulting acceleration in an array and then do the update of velocities outside of the loop which calculates accelerations. However such a change would require mor memory since we'd need to store the resulting acceleration on each object, i, instead of using it directly to calculated the new velocities. Finally we used the valgrind tool to confirm that no memory leaks was possible, that we had freed all heaps.

Discussion

The shortest execution time was received when the the -Ofast, -funroll-loops, -march=native compiler flags where used. The option -march=native optimises the code for the computer architecter where the program is compiled. Hence it may lead to reduced performance on other computer architectures.

When using the -Ofast flag the compiler accepts larger rounding errors, which should be kept in mind. To investigate the effect on the accuracy of the simulation when using the -Ofast flag the result was compared with the output from the same simulation but compiled with the -O3 flag instead. We tried this for both $\theta = 0$ and $\theta = 0.25$ over 200 time step no differences were detected.

One thing which might have improved our performance is to reduce the number of if conditions and thereby decrease branch miss prediction. However we can not see how such a change could be implemented since a recursive function utilises if statements to determine if we've reached a base case or not. If statements are also utilised in order to determine in wihch quadrant to add new nodes for which 2 if statements are required.

A clear improvement of the execution time was achieved compared to the quadratic algorithm implemented in the previous assignment. With the quadratic algorithm the simulation for 5000 objects over 200 time steps the execution time was 8.995s. With the Barn-Hut algorithm with $\theta = 0.25$ the execution time was 8.995s. Though with $\theta = 0$ the new program had an execution time of 1m37.461s, which is far worse then the original quadratic implementation. This shows that the constant of the implemented Barnes-Hut algorithm is very large and can be improved. Though for large number of objects we still prefer the implemented Barnes-Hut algorithm, as shown by the measured execution times for 5000 objects over 200 time steps. However when using the algorithm one should keep in mind that it uses approximations which leads to errors in the resulting positions of the objects.

In order to further improve our program more care could be taken with storing the data to increase data locality. For example the brightness of the interstellar objects are not changed during the simulation and does not need to be included in the structs. However in both cases the read and write functions would nee to be rewritten and the number of calls to fwrite would have to increase.

Distribution of Roles

Sara wrote the majority of the code, Fredrik wrote the majority of the report. The optimisation of the code was equally shared.