Assignment 3

High Performance Programming

Author:

Fredrik Gustafsson

Sara Hedar

Uppsala

February 15, 2019

The Problem

Introduction

When simulating a large number of objects interacting with each others, such as a n-body system of planets and stars, it is important too optimise the code for speed. The time needed to execute a program can be reduced for instance by choosing a faster algorithm, utilise compiler optimisation flags and performing micro optimisation by hand. In this project a simulation for a n-body system was implemented and then different ways to optimise the code was explored.

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:

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

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.

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 implanted as we did not manage to get it to work.

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 simulation was performed through a step function corresponding to advancing the system with one time step. This function was called the same number of time as inputted to the main program. First the acceleration was calculated in a double for-loop for each object, then the velocities and positions were updated in a single for-loop. The complexity of the algorithm implemented was $O(N^2)$, where N is the number of interstellar objects.

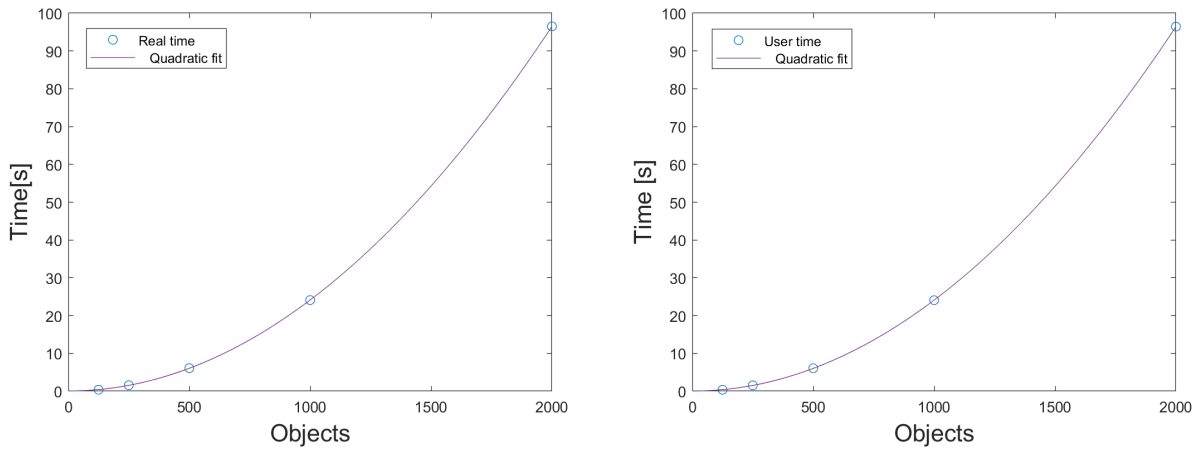
Performance and Discussion

The computations was performed on Unbuntu 18.04.1 LTS on Windows 10, Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2904 Mhz, 2 cores, 4 logical processors. The compiler used was gcc (Ubuntu 7.3.0-27ubuntu1 18.04).

Complexity of the Algorithm

When investigate the complexity of the algorithm implemented all forms of optimisation was omitted. The time to run the program was measured using the `time` command in the Linux terminal. The number of objects to simulate was varied, the following values were used; $N = [125, 250, 500, 1000, 2000]$. The simulation was performed over 200 time steps and the following parameters were used: $G = 100/N, \varepsilon_0 = 10^{-3}, \Delta t = 10^{-5}$.

For each number of object N the time to run the program was measured 5 times, and the fastest was recorded. The results are presented in figure 1. Both real time and user time is plotted as a function of the number of objects and a quadratic fit is made to the recorded times. Clearly both the Real time and the User time follows the quadratic fit, and hence the quadratic complexity of the implemented algorithm is confirmed.



(a) Real time.

(b) User time.

Figure 1: Plots of real and user time as a function of number of objects, with quadratic fits for each case.

Opimisation of the Code

To optimise the code the number of objects was set to 500, the difference in time was Δt = 1E-5 and the simulation was performed for 200 time steps. For each set of settings the time to run the program was measured 5 times with the `time` command in the Linux terminal and the fastest time was kept.

To optimise the code the five compiler flags, -O1,-O2,-03,-Ofast, and -funroll-loops were first tested. Then combinations of -O3 and -funroll-loops, and -Ofast and -funroll-loops were then tested. Finally some modification of the codes was done with loop unrolling of the velocity and position update loop, tested without other flags and with -O3 and -Ofast. Another modification of the code was to inline the three functions; read data, write data and the step function. This was tested both without other optimisations and with the -O3 and -Ofast flags.

	-O0	-O1	-O2	-O3	-Ofast	-funroll-loops	-O3 -funroll-loops	-Ofast -funroll-loops
Real time	6.037	5.615	5.560	5.550	0.196	6.036	5.544	0.457
User time	6.000	5.594	5.547	5.531	0.188	6.016	5.531	0.469
Sys time	0.016	0.016	0.000	0.000	0.000	0.016	0.000	0.000

Table 1: Fastest time recorded for each combination of compiler flags.

	irw	irw and -O3	irw -Ofast	is	froll vel pos	froll vel pos and -O3	froll vel pos and -Ofast
Real time	6.059	5.572	0.197	6.037	6.037	5.588	0.198
User time	6.047	5.573	0.172	6.000	6.000	5.563	0.188
Sys time	0.000	0.031	0.000	0.016	0.016	0.016	0.000

Table 2: Fastest time recorded when anually modifyinh code, with and without flags, irw = inline on read and write function, is = inline step function, froll vel pos = loop unroll on velocity and position update with unroll factor 4.

Discussion

From our tests the shortest time was achieved with -Ofast flag when compiling, using the -funroll-loops flag gave no increase in performance and for -Ofast decreased it. The manual optimisation of the code, inline and loop unrolling once again gave no increase in performance. That loop unrolling gave no increase in performance is not completely unexpected due to how the algorithm looks. The main computational expensive part is in the different loops where the difference comes from the two parts of the move function. Said function has to loop over every element twice in order to calculate the all forces which gives it it's O(N^2) complexity. The second loop just loops over the elements once and is therefore not important for the complexity. In our optimisation we only tried to optimise the computationally cheaper part of the algorithm while keeping the expensive one the same. Had we instead tried to do loop unrolling on the acceleration calculation we might have seen an increase in performance. Another thing which might have affected the performance is that we have a if condition in the innermost for loop however the condition is not full filled 1/N % where N is the number of objects. For large N the percentage of branch miss prediction should therefore be very low.

The usage of the -Ofast flag when compiling is risky since it can introduce rounding errors. However in our case testing our result, after 200 steps, with the provided comparison program and files return a solution correct to the 12th decimal, which is to the precision that the test program tested.

Distribution of Roles

The work load was divided equally between the participants of the group.