UPPSALA
UNIVERSITET

Assignment 4
High Performance Programming

Written by:
Group 10
Jakob Gölén
Parwand Batti

March 1, 2022

1 Problem

In this assignment the evolution of galaxies with different number of stars was simulated by calculating the force between particles using their positions, velocities and masses. The algorithm used in assignment for calculating the forces was Barnes-Hut algorithm. The idea behind Barnes-Hut algorithm is to reduce the number of force computations by approximating the force exerted by a group of particles on a single particle i as the force exerted by one particle located at the center of mass of a group of particles.

The first step in Barnes-Hut algorithm is to store the particles in a tree structure called quadtree. Quadtree divides basically the domain into four branches recursively and if the branches contains more than one particles, the branches are subdivided into further branches. This procedure is repeated until each particle is located in own branch.

When the quadtree is created, the mass and center of mass of each branch are computed recursively by adding up the masses of particles in a given branch and dividing that by the total mass.

Last step in the Barnes-Hut algorithm is to compute the forces on each particle by creating a new quadtree at each step and traversing the quadtree recursively.

By considering the value of θ below, one can decide whether to traverse down the branches and compute the forces or not.

$$\theta = \frac{d}{r}$$

where d is the width of the current region with particles and r is the distance between the particle and the center of the mass. The value of θ was compared to a threshold $\theta_{\max} \in [0, 1]$. In the case with $\theta > \theta_{\max}$ and at least one branch is not empty, the algorithm traverses down the branches. Otherwise, the algorithm computes the forces for the whole region in the case with $\theta <= \theta_{\max}$.

Assuming there are N particles in the system, The forces are calculated using equation (1).

$$\overline{F}_i = -Gm_i \sum_{i=0}^{N-1} \frac{m_j}{(r_i + \epsilon_0)^3} \hat{r}_i \tag{1}$$

where G is the gravitational constant and is set to $\frac{100}{N}$. r_i is the distance between particle i and the calculated center of mass and $\hat{r}_i$ is the normalised direction. Epsilon is a small number which smooths the simulation and it is chosen as 10^{-3}.

The sympletic euler was used here to find out the particles movements in time. Using the forces calculated in equation (1) together with particles masses , the acceleration can be computed. Then the velocities and positions of each particle are updated at the next step, see equation (2).

$$\begin{cases} \overline{a}_i^n = \frac{\overline{F}_i^n}{m_i} \\ \overline{v}_i^{n+1} = \overline{v}_i^n + \Delta t \cdot \overline{a}_i^n \\ \overline{x}_i^{n+1} = \overline{x}_i^n + \Delta t \cdot \overline{v}_i^{n+1} \end{cases} \tag{2}$$

The input data and the equations for the force calculation and the step calculation was given in the assignment description. [1]

2 Solution

All of the tests were performed on a `AMD Ryzen 5 5600H with Radeon Graphics @3.301Ghz` running on `Ubuntu version 20.04` using virtual box. The program was compiled with `gcc version 9.3.0`.

First, the code was implemented without the Barnes-Hut method. The code was then modified to implement the Barnes-Hut method and thereafter it was parallelized. The program reads input containing information about how many particles to simulate, the starting configuration of the system (which was read from a file and contained information about starting position, velocity, mass and brightness of each particle), the number of steps to take, size of the time step and if graphics should be used or not. Later on, the program also took a $\theta_{\max}$ value to compare, and the number of threads that should be used for parallelization.

The info about each particle was stored in an array of structs and a function was written to calculate the combined force of all particles on one particle using equation 1. Then, two for-loops were initialized, one that advanced the time and one that went through each particle and calculated the combined force on that particle and then updated the position and velocity using equation 2. When the program has advanced the desired steps, the loops end and the final information about each particle is written into an output file which could then be compared to a reference file in order to verify the accuracy of the algorithm.

When the program was working correctly, the next step was to implement the Barnes-Hut method. For this, two functions were added and the function for calculating forces was modified. The new function was a function that created a quadtree. First, the root of the tree was dynamically allocated as a struct containing information about what region the node should contain, pointers to the four branches of each node, and the combined mass and center of mass of each particle in the region the node corresponded to. Next, the number of particles in the region was calculated in a for loop and their mass and center of mass was added to the nodes mass and center of mass. If there was only one or zero particles in the node, no subbranches should be created so the four pointers to the sub branches were set to `NULL`. If there were zero particles in the node, a special flag was set to signal that the node was empty.

If there were more than one particle in the node. Four sub branches were dynamically allocated and the function calls itself recursively inside the four sub branches. The next function was a function to delete a quadtree. For this, the function simply checked if it was `NULL`. If it wasn't, it went into the four sub branches and called itself recursively. When all four sub branches were `NULL`, it freed the nodes memory and set the node to `NULL`. With `NULL`, it was confirmed that the program contained no memory leaks.

Next, the force function was modified. Instead of checking the force contribution of all other particles on a particle, it instead starts at the root and checks if the node is

empty (no particles). If that is the case, the function exits since there are no forces to calculate and the function ends. If not, the function calculates a θ using equation 1 and compare it to the θ_{max} defined by the user. For the case $\theta \leq \theta_{max}$, all particles in the node are considered to be one particle with the combined mass and center of mass of all particles in the region and calculates the force from that particle from that on the current particle. If $\theta > \theta_{max}$, that means that the current particle is close to the center of mass of the particles in the node and the function then goes into the four sub branches and calls itself recursively. This then leads to the force from all other particles being involved, either as a part of a larger mass or on its own.

When the new functions had been added and the force function had been modified. The for loops were modified so that at every new time step, a new quadtree was created, and when the forces had been calculated and the particles had been updated, the tree was deleted so a new one could be created for the next timestep.

When the Barnes-Hut algorithm had been implemented, the code was parallelized. For the parallization, Open MP was used due to it's ease to implement into already working code. Three main parts of the program was parallelized, the functions for creating and deleting the quadtree, and the loop calculcating the force for each particle.

The function for creating a quadtree was parallelized using tasks. If there was more than one particle in a node, one thread created four tasks with, where each task took care of one of the four sub branches in the node, since each of those tasks were independent from each other. The function deleting a quadtree could be parallelized in the same way, with one thread creating four tasks and each task deleted one of the four sub branches.

Lastly, the loop calculating forces on each particle could be parallelized, since the force from all particles on one particle was not dependant on the force from all particles on another particle. The critical argument was added if graphics was turned on, to ensure that all particles was deleted before drawing the particles for the next step.

3 Performance and discussion

In order to observe the performance of the algorithm, the code was running for four different cases and the execution time was plotted as function of N. All cases were executed for 100 steps and $\Delta t = 10^{-5}$. Firstly, the code was executed in serial with and without optimal $\theta_{\max}$. Thereafter, the code was running even in serial with and without optimal $\theta_{\max}$.

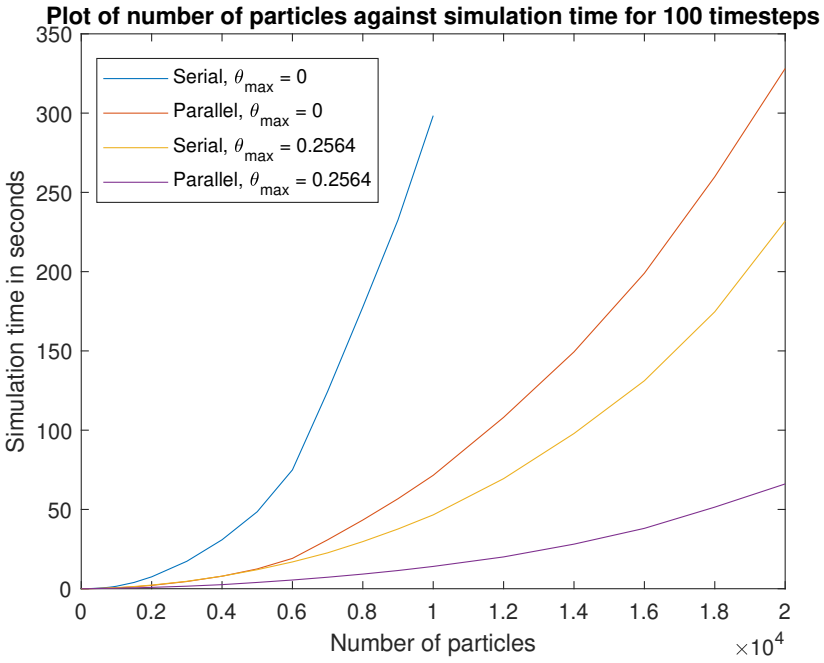

Figure 1: Plot of time compared to number of particles

The optimum $\theta_{\max}$ was found by investigating the `pos_maxdiff` for different measurements using $\epsilon_0 = 10^{-3}$, 2000 stars, 200 timesteps and $\Delta t = 10^{-5}$. Then the optimum $\theta_{\max}$ was found to be 0.2564. Since this value is between 0.02 and 0.5, the implementations seems to be working correctly.

By looking at Figure 1, it is obvious how big of an improvement a suitable θ_{max} makes, since the serial code with a suitable θ_{max} is even faster than the parallel code where θ_{max} is set to zero, and galaxy simulations which took 5 minutes now takes just 45 seconds.

The implementation was hard to optimize in serial, but it was easier to optimize it in parallel. A lot of `malloc` and if-statements made the code slower. On the other hand, by increasing $\theta_{\max}$ and reducing the accuracy, the code was running much faster.

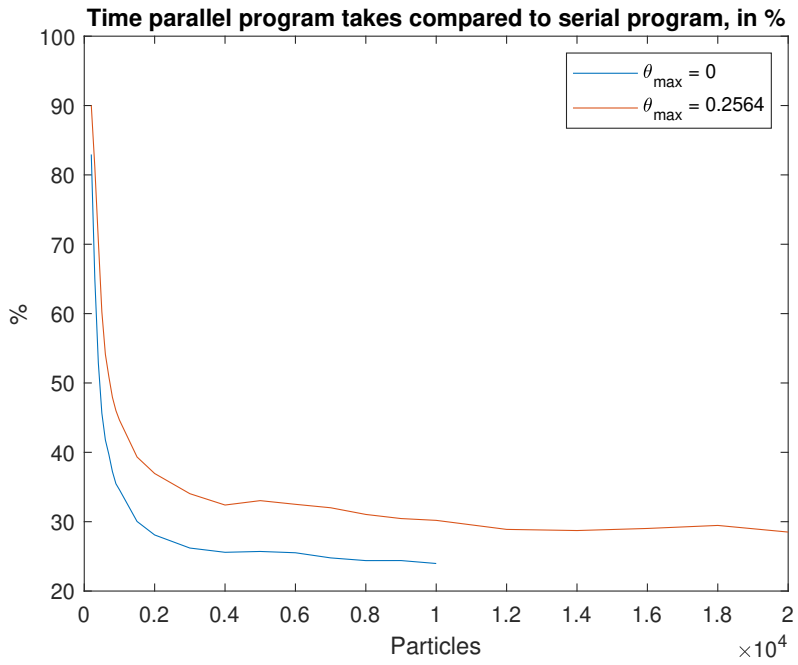

Figure 2: Time in % of parallel when compared to serial

The parallelization seems to be working correctly, by looking at Figure 2, the execution time of the parallel code is about 25% of the serial code, which is what to be expected when using four threads, when setting θ_{max} to the optimal value, the execution time increases slightly to around 30% of the serial code.

The results shows that the performance is improved using the optimum $\theta_{\max}$ and parallelization. However, the execution time is a bit higher than what we expected. This can be due to the implementation is not perfect and can be improved both in serial and parallel. For instance, calculating the forces recursively wasn't parallelized due to variable dependency. This part of the code consuming a lot of time and we would save a lot of time if this could be parallelized in an efficient way.

References

[1] Assignment 4. The barnes-hut method, high performance programming, 2022. Uppsala University.