HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2023
PROJECT: THE BARNES-HUT METHOD

Relation to previous assignment: This assignment is a continuation of the previous Assignment 3; you are allowed to use your code from Assignment 3 as a starting point. However, the results you submit for this assignment should be complete and not dependent on the previous assignment, you are not allowed to write "see the previous report" or anything like that, your final code and report should be complete and independent of the previous assignment. Someone who has never heard of the previous Assignment 3 should still be able to read your report and use your code.

1. Problem Setting

The background for this assignment is the same as for the previous Assignment 3. Newton's law of gravitation is the same and the expression for the force that corresponds to so-called Plummer spheres is the same

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

We will also use the same values of the parameters $G = 100/N$ and $\epsilon_0 = 10^{-3}$.

What is new in this problem is the time integration and the algorithm used to compute the forces; instead of the simple $\mathcal{O}(N^2)$ algorithm used before, we will now use the Barnes-Hut method which, if implemented correctly, should have better computational complexity. For time integration we will use the second order **Velocity Verlet** time integration method

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2,$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{\mathbf{a}(t)+\mathbf{a}(t+\Delta t)}{2}\Delta t.$$

Using a second order accurate method allows us to take larger time steps while keeping the numerical accuracy, thus improving the efficiency of the algorithm.

2. Divide-and-conquer (Barnes-Hut)

For many problem settings, the number of operations required for computing the force in the N-body problem can be substantially reduced by taking advantage of the idea that the force exerted by a group of objects on object i can be approximated as the force exerted by one object with mass given by the total mass of the group located at the center of gravity of the group of objects. This is a reasonable

Date: March 31, 2023.

approximation as long as $h/r \ll 1$, where h is the maximum distance between any two objects in the group and r is the distance to the group. In other words, the group of objects must be sufficiently far away such that they can be approximately treated as one object.

This reduces the number of operations because if there are M objects in the group, then $M-1$ fewer forces need to be calculated in order to determine their net effect on object i. The way to handle this algorithmically is to store the particles in a quadtree and compute the forces on each object recursively. The idea behind a quadtree is to recursively subdivide the domain into four squares, and subdivide those squares into further sub-squares, and so on, until only one object remains in each square, as shown in Figure 2.1.

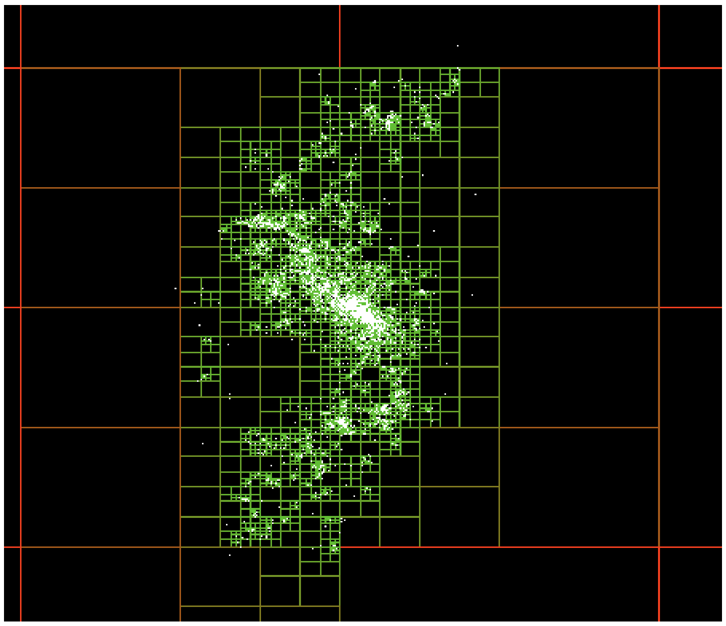

FIGURE 2.1. Adaptive grid in which each object is stored in its own quadrangle

The specific algorithm that employs the quadtree to compute the forces on a group of objects is known as the Barnes-Hut algorithm. Specifically, the Barnes-Hut algorithm involves the following high-level steps:

(1) Build the quadtree for the particles
(2) Recursively compute the mass and center of mass of each quadrangle by adding up all of the masses contained within that quadrangle. (Center of mass: $X_{cm} = \sum_i x_i m_i / M$, $Y_{cm} = \sum_i y_i m_i / M$ where $M = \sum_i m_i$)
(3) Compute the forces on each object by recursively traversing the quadtree.

The key ingredient of the Barnes-Hut algorithm is to decide whether or not to traverse down the branches of a node and compute the forces resulting from the particles contained within its branches, or just to stop at the current node and assume all of its particles can be lumped into an equivalent mass at its center of mass. This is determined by considering the value θ defined as

$$\theta = \frac{\text{width of current box containing particles}}{\text{distance from particle to center of box}} .$$

The current θ value is compared to a threshold $\theta_{\max} \in [0, 1]$ and the box is treated as an equivalent mass if $\theta \leq \theta_{\max}$. In case $\theta > \theta_{\max}$, the algorithm instead traverses down the branches of the quadtree.

One aspect of the algorithm that may at first seem tricky is building the quadtree. A simple approach is to start with an empty root and insert particles into the tree one at a time, creating new partitions as needed. That is just one possibility however, and not necessarily the best or most efficient choice. You are free to do it in any way you want.

You can assume that the particle coordinates are always within a given interval $(x_{\min}, x_{\max})$ and $(y_{\min}, y_{\max})$; we are only interested in simulations where all particles stay in that domain. In the unexpected situation where any particle moves outside, it is OK to simply print an error message and stop the simulation or you can also allow for a larger domain in building the tree. The particles are from start inside a unit box (0,1) and (0,1) but it might be a good idea to extend the box from start to avoid stopping the algorithm early.

The special case of two particles happening to end up at exactly the same point can be tricky to handle when the quadtree is created. In practice that should be extremely unlikely in our simulations; if your program detects that two particles are at exactly the same position, then it is OK to just print an error message and stop the simulation.

3. Parameter tuning

Since the Barnes-Hut method means that forces are computed approximately depending on the $\theta_{\max}$ value, it helps to have an idea of the desired accuracy of the simulations in order to know what $\theta_{\max}$ value is appropriate.

One way of quantifying the accuracy of the simulations is to use the results of a simulation with the straightforward $\mathcal{O}(N^2)$ algorithm in Assignment 3 as a reference, and check how far away from that result you get when using the Barnes-Hut method with different choices of $\theta_{\max}$. Note that you then also need to use the **Symplectic Euler** time integration to get the same numerical accuracy. For example, setting $\theta_{\max}$ to 0 should give the same results within rounding errors. Verify first that your tree data structure works as it should by comparing your results in this way.

Once you have confirmed that your tree structure works it is time to find a suitable value on $\theta_{\max}$. Define the target accuracy in the following way: consider a simulation of 2000 stars with starting conditions as given by `ellipse_N_02000.gal` simulated with Symplectic Euler for 200 time steps and $\Delta t = 10^{-5}$. For this case we want the maximum difference between particle positions (`pos_maxdiff`) for the reference simulation and the Barnes-Hut simulation to be smaller than 10^{-3}. So, to check what is a reasonable choice of $\theta_{\max}$, run such simulations (2000 stars, 200 time steps) with different $\theta_{\max}$-values and check how the maximum difference between particle positions is affected. From this you can determine what is a reasonable value of $\theta_{\max}$: choose a value that is as large as possible while still keeping the maximum difference between particle positions below 10^{-3} for the case with 2000 stars, 200 time steps. If you have implemented the Barnes-Hut algorithm correctly, the $\theta_{\max}$ value you find using the procedure above should be somewhere between 0.02 and 0.5. If you find that you need a smaller $\theta_{\max}$ value than 0.02 to reach the target accuracy, that indicates that your implementation is not working properly. Use the $\theta_{\max}$ value you found and study how your Barnes-Hut implementation scales with N. Is the complexity $\mathcal{O}(N)$, $\mathcal{O}(Nlog(N))$ or $\mathcal{O}(N^2)$?

Next, verify that your Velocity Verlet time integration is of 2:nd order, i.e., set up a convergence study where you show that the error behaves as $\mathcal{O}(\Delta t^2)$ as $\Delta t \to 0$. Take for example the input file `sun_and_planets_N_3.gal`, use $\theta_{\max} = 0$, $\Delta t = 10^{-8}$ and $2 \cdot 10^6$ time steps as the "true" solution. Then see how fast the `pos_maxdiff` decreases as you go from $\Delta t = 10^{-3}$ and 20 time steps down to the "true" solution. Verify also in the same way that your Symplectic Euler is only of 1:st order.

When you have the Velocity Verlet time integration and Barnes-Hut working correctly you can start to optimizing the values $\theta_{\max}$ and Δt to get a minimal run time while keeping a reasonable accuracy (of your choice). Choose a problem with large enough of stars to get a good parallelism, compute a reference solution using a small time step and $\theta_{\max} = 0$. Then choose an acceptable error tolerance that the stars can maximally deviate from and tune in $\theta_{\max}$ and Δt. Finally, compare how much faster you can solve the problem compared to the straight forward algorithm in Assignment 3 keeping the same error tolerance. The simulation time ($T = \text{nsteps} \times \Delta t$) can be chosen such that run time doesn't become unreasonably long.

Be aware of that both the Symplectic Euler and Velocity Verlet can become unstable when the stars are close to each other, then you might need to decrease the time step. In the provided input files the stars are close and you need to choose the time step very small to get stability. If you see that your results are behaving strangely try to decrease the time step for the problem.

Finally, don't forget to optimize the code using the techniques you have learnt in the course and parallelize the algorithm with Pthreads or OpenMP to run as fast as possible.