HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2018
ASSIGNMENT 4: 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, the expression for the force that corresponds to so-called Plummer spheres is the same, and we use the symplectic Euler time integration method in the same way as before. We will also use the same values of the parameters $G = 100/N$, $\epsilon_0 = 10^{-3}$ and timestep $\Delta t = 10^{-5}$.

What is new in this assignment is 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.

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 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

Date: February 6, 2018.

those squares into further sub-squares, and so on, until only one object remains in each square, as shown in Figure 2.1.

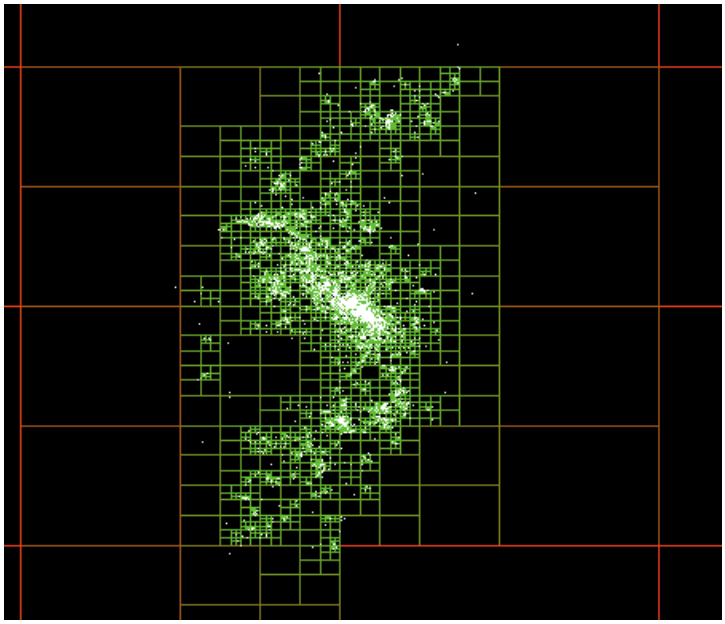

FIGURE 2.1. Adaptive grid in which each object is stored in its own quadrangle

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.

(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 of most efficient choice. You are free to do it in any way you want.

You can assume that the particle coordinates are always within the (0,1) interval; 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.

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. Files included with the assignment

Download the `Assignment4.tar.gz` file from the Student Portal and unpack it. Inside it you find four subdirectories in the same way as for the previous Assignment 3, the only difference is that some larger input files are now included. (Since the Barnes-Hut method has better complexity, you should hopefully be able to handle simulations for larger numbers of particles.)

4. Assignment

You must write a program that solves the given equations of motion for a galaxy with the given initial conditions, using the Barnes-Hut method. Your final program should be written as efficiently as possible.

Start by writing a code that gives correct results, then think about how it can be optimized. Remember to use both compiler optimization flags and your own code changes, so that your final code becomes as efficient as possible.

Input to your program: The program that you create should be called `galsim` and it should accept six input arguments as follows:

```
./galsim N filename nsteps delta_t theta_max graphics
```

where the input arguments have the following meaning:
`N` is the number of stars/particles to simulate
`filename` is the filename of the file to read the initial configuration from
`nsteps` is the number of timesteps
`delta_t` is the timestep Δt
`theta_max` is the threshold value $\theta_{\max}$ to be used in the Barnes-Hut algorithm
`graphics` is 1 or 0 meaning graphics on/off.

If the number of input arguments is not six, the program should print a message about the expected input arguments and then stop.

Note that compared to the previous assignment, one more input argument has been added, giving the value of $\theta_{\max}$ to be used in the simulation.

Output from your program: The program `galsim` should generate a result file `result.gal` in the same way as before.

Implementation: First check that your implementation of the straightforward $O(N^2)$ algorithm is working properly. When you are confident that your code works correctly for the $O(N^2)$ algorithm, implement the Barnes-Hut algorithm. When that also works properly, move on to measure performance and consider possible optimizations.

Portability: You can choose freely which computer you want to use for your performance testing in this assignment — you can use one of the university's Linux computers, or your own computer, or some other computer you have access to. However, even if you have used a different computer for your performance testing, you must still ensure that the code is portable so that it can be compiled and run on the university's Linux computers. This is necessary to make sure that your teachers can test your code.

Report: You must write a report that motivates the efficiency of your implementation using time measurements and a description of the optimisation techniques you have used or attempted to use. In this report, also analyse how the computational time depends on `N`. Remember that one of the course goals involves written communication — the assignment reports is part of how we examine this goal and we expect the highest level of quality. Write clearly and concisely, using figures and tables as appropriate.

When writing your report, remember that *reproducibility* is important: whenever you write a report that includes some timing measurements, or other computational experiments of some kind, you should make sure that the report includes all information necessary so that the reported results become reproducible. The reader of your report should be able to reproduce your results. Your report should make it clear exactly what it is you are reporting, if you have some timings it should be explained precisely what you were measuring: how was the measurement made, was it for the whole program run or for some specific part of the code, what parameters were used, how many timesteps, etc. Details about the used computer, CPU model, compiler version, and compiler optimization options should also be included.

Note that timing results should not include calls to graphics routines. When measuring timings, run your code with graphics turned off to be sure that graphics routines are not disturbing your timings.

Group: You should work in groups of two students/group. (If you have strong reasons why you need to or really want to work alone, discuss that with your teacher, you may be allowed to work alone.)

The group should decide on a distribution of roles during the exercise and very shortly describe it in the final report. For example, who did the most programming and debugging, measuring performance and generating figures/tables, writing the report. If both of you have contributed equally to everything, then write that. If you focused on different things, make sure each of you still undestands what you have done and how each part of your code works.

5. Input data

As in the previous assignment, unpacking the `Assignment4.tar.gz` file gives a directory called `input_data` with various input files that you can use. It is similar to before, the only difference is that there are some larger files included now.

5.1. **Check your own results by comparing to reference output data.** Use the `compare_gal_files.c` program to check the accuracy of your results.

You should verify your results in two ways: firstly, check that when the $O(N^2)$ algorithm (i.e. running your program with `theta_max`=0) is working correctly by comparing to the provided reference output data. Then you should only see very small differences due to rounding errors. Secondly, check that your Barnes-Hut implementation works by running it with different `theta_max`-values and checking that for large enough test cases, the Barnes-Hut algorithm speeds up your program while maintaing acceptable accuracy. See the subsection about "Target accuracy" below.

5.2. **Target accuracy .** 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 use the results of a simulation with the straightforward $O(N^2)$ algorithm as a reference, and check how far away from that result you get when using the Barnes-Hut method with different choices of $\theta_{\max}$.

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 for 200 timesteps, using $\epsilon_0 = 10^{-3}$ 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 timesteps) 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 timesteps. Then use the $\theta_{\max}$ value you found for your remaining tests, e.g. when studying how your Barnes-Hut implementation scales with N.

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 accueracy, that indicates that your implementation is not working properly.

Note that the `ellipse_N_xxxxx_after200steps.gal` files provided with the assignment are simply results of simulations using the straightforward $O(N^2)$ algorithm; you can create such files yourself by running your own code.

6. Deliverables

It is important that you submit the assignment in time. **See the deadline in the Student Portal.**

You should package your code and your report into a single `A4.tar.gz` file that you submit in the Student Portal. There should be a makefile so that issuing "make" produces the executable `galsim`.

Unpacking your submitted file `A4.tar.gz` should give a directory `A4` and inside that there should be a makefile so that simply doing "make" should produce your executable file "`galsim`". The `A4` directory should also contain your report, as a file called `report.pdf`.

In addition to the report, your `A4` directory should also include a text file called `best_timing.txt` that should contain just three lines: on the first line, just one number giving the fastest time that you were able to achieve for the input case `ellipse_N_05000.gal` (5000 particles) with $\Delta t = 10^{-5}$ and 100 timesteps, using the $\theta_{\max}$ value you found in Section 5.2. (Wall time for the whole program run, measured using the `time` command.) On the second line, the name of the CPU model used. On the third line, the used $\theta_{\max}$ value. (This text file will be used by your teachers to create a plot or table showing the reported performance of the different student groups, for different CPU models.)

Apart from `report.pdf` and `best_timing.txt` files, your submission should only contain C/C++ source code files and makefile(s). No object files or executable files should be included, and no input/output (.gal) or other binary files.

Part of our checking of your submissions will be done using a script that automatically unpacks your file, builds your code, and runs it for some test cases, checking both result accuracy and performance. For this to work, it is necessary that your submission has precisely the requested form.

To be sure that your submission has the correct form, you should check it yourself before submitting, using the `check-A4.sh` script that is included with the assignment. To use the `check-A4.sh` script, first set execute permission for it, then copy your `A4.tar.gz` file into the Assignment4 directory, and then run `./check-A4.sh` when standing in the Assignment4 directory. The `check-A4.sh` script will then try to unpack your `A4.tar.gz` file, check that it has the expected contents, and compile and run your program trying to check that it seems correct. If the checking is successful, the output from the script should say "Congratulations, your A4.tar.gz file seems OK!". If you do not get that result, look carefully at the script output to figure out what went wrong, then fix the problem and try again.

If you have used your own computer for this assignment, please verify that your code is portable to the university's Linux computers before you submit. (You may find that some change is needed to make the code portable there, e.g. adding `-lm` to link to the math library.)

The report shall be in pdf format (called `report.pdf`) and shall contain the following sections:

- The Problem (very brief)

- The Solution (Describe the data structures, the structure of your codes, and how the algorithms are implemented. Are there other options, and why did you not use them?)
- Performance and discussion. Present experiments where you investigate the performance of the algorithm and your code. Describe optimizations you have used, or tried to use, and the measured effect of those optimization attempts. Include a figure with a plot of measured execution time as a function of N for the $\mathcal{O}(N^2)$ and the Barnes-Hut cases. What seems to be the complexity of your Barnes-Hut implementation?

Submission

When you are done, upload your final `A4.tar.gz` file in the Student Portal. Note that the uploaded file should have precisely that name, and that you should have checked it using the `check-A4.sh` script before uploading it.

Questions?

If there are any questions about this assignment, e-mail to `elias.rudberg@it.uu.se`.