HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2017
ASSIGNMENT 3: THE GRAVITATIONAL N-BODY PROBLEM

1. The N-body problem

1.1. **Governing equations.** Newton's law of gravitation in two dimensions states that the force exerted on particle i by particle j is given by

$$\mathbf{f}_{ij} = -\frac{Gm_i m_j}{r_{ij}^3}\mathbf{r}_{ij} = -\frac{Gm_i m_j}{r_{ij}^2}\hat{\mathbf{r}}_{ij}\,,$$

where G is the gravitational constant, m_i and m_j are the masses of the particles, r_{ij} is the distance between the particles, and $\mathbf{r}_{ij}$ is the vector that gives the position of particle i relative to particle j. $\hat{\mathbf{r}}$ is the normalized distance vector. If $\mathbf{e}_x$ and $\mathbf{e}_y$ are unit vectors in the x and y directions, respectively, then

$$\mathbf{r}_{ij} = (x_i - x_j)\,\mathbf{e}_x + (y_i - y_j)\,\mathbf{e}_y\,,$$

so that

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

and

$$\hat{\mathbf{r}}_{ij} = \mathbf{r}_{ij}/r_{ij}\,.$$

Given a distribution of N particles, a straight-forward calculation of the force exerted on particle i by the other $N-1$ particles is given by (using C-style indexing)

$$\mathbf{F}_i = -Gm_i \sum_{j=0, j\neq i}^{N-1} \frac{m_j}{r_{ij}^2}\hat{\mathbf{r}}_{ij}.$$

Note that by using another formula for the pairwise forces, other types of particle systems governed by Newtonian mechanics can be modeled, e.g in electrodynamics and molecular dynamics.

There is a built-in instability in the given formulation when $r_{ij} \ll 1$. To deal with this, we introduce a slightly modified force that corresponds to so-called Plummer spheres as follows:

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

where ϵ_0 is a small number (we will use 10^{-3}). This effectively caps the maximum force between two particles and acts as a smoother for the simulation.

Date: February 3, 2017.

Using the *symplectic Euler*[1] time integration method, the velocity $\mathbf{u}_i$ and position $\mathbf{x}_i$ of particle i can then be updated with

$$\begin{aligned} \mathbf{a}_i^n &= \frac{\mathbf{F}_i^n}{m_i}\,, \\ \mathbf{u}_i^{n+1} &= \mathbf{u}_i^n + \Delta t \mathbf{a}_i^n\,, \\ \mathbf{x}_i^{n+1} &= \mathbf{x}_i^n + \Delta t \mathbf{u}_i^{n+1}\,, \end{aligned}$$

where Δt is the time step size and $\mathbf{a}_i$ is the acceleration of particle i. Note that this straightforward solution, which we will use in this assignment, becomes computationally expensive for large values of N, since the total number of operations required to compute the forces on all N particles at each time step grows as $\mathcal{O}(N^2)$. There are algorithms that can be used to improve the scaling, but in this assignment we stick with the straightforward $\mathcal{O}(N^2)$ solution.

2. Problem Setting

In this assignment you will implement a code that calculates the evolution of N particles in a gravitational simulation. You will calculate the motion of an initial set of particles that approximates the evolution of a galaxy.

The simulation is done in two spatial dimensions, so the position of each particle will be given by two coordinates (x and y). There will be N particles with positions in a $L \times W$ dimensionless domain (Use $L = W = 1$ in your simulations). This means that the x and y values will be between 0 and 1.

The particle masses and intitial positions and velocities of the particles will be read from a file when the program starts, then simulated for a given number of timesteps, and in the end the final masses, positions and velocities will be written to a results file. (The masses will of course be the same as in the beginning, but they will be written to the results file anyway so that it can be used as input for another simulation later.)

Since fewer bodies have less mass with which to stick together, we want gravity to scale inversely with the number of bodies. Set $G = 100/N$, $\epsilon_0 = 10^{-3}$ and timestep $\Delta t = 10^{-5}$.

3. Files included with the assignment

Download the `Assignment3.tar.gz` file from the Student Portal and unpack it. Inside it you find five directories:

- `file_operations` : includes files that you can use for reading and writing the binary files used as input and output for your program. The functions provided can read/write an array of double precision numbers from/to a file. Use them if you want, or write your own code for that if you like, just make sure that your files follow the specified format (see below).

[1]The symplectic Euler method is a version of the explicit Euler scheme which is the most basic example of a partitioned Runge-Kutta method (PRK). For the standard Euler scheme, the formula for $\mathbf{x}_i^{n+1}$ would read $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{u}_i^n$. The symplectic Euler scheme preserves global properties of the gravitational system (e.g. total energy), while the standard Euler does not.

- `compare_gal_files` : a program that can be compiled into a separate executable that can be used to compare different simulation results; you can use this to check the correctness of your results.
- `input_data` : a set of input files that you can use when testing your program.
- `ref_output_data` : a set of reference output files that you can use to check that your results are correct. The `compare_gal_files` program can be used to compare your result files to the reference output files.
- `graphics` : a small example code showing how to use graphics routines in the "X" window system commonly used in Linux. Being able to plot the galaxy evolution on the screen is a nice way of seeing that the code works, and helpful for debugging. The small test code given here is just an example, if you know of another way of doing graphics that you prefer, use that instead. To make the "X" graphics code work you may need to install some additional library packages related to X development, e.g. `libX11-devel` in Fedora or `libx11-dev` in Ubuntu.

4. Assignment

You must write a program that solves the given equations of motion for a galaxy with the given initial conditions. 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.

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

```
./galsim N filename nsteps delta_t 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
`graphics` is 1 or 0 meaning graphics on/off.

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

Output from your program: The program `galsim` should simulate the stars/-particles for the given number of timesteps. If run with `graphics` set to 1, the program should show the stars moving on the screen during the simulation. In the end, the final positions and velocities of the stars should be saved to a file called `result.gal` using the same binary file format as for the input file.

File format used for initial configuration (and storage of result): The binary file format used will simply consist of a sequence of `double` numbers giving the mass, position, and velocity of each particle. The order is as follows:

```
particle 0 position x
particle 0 position y
particle 0 mass
particle 0 velocity x
particle 0 velocity y
particle 1 position x
particle 1 position y
particle 1 mass
particle 1 velocity x
particle 1 velocity y
...
```

So, the total file size will be `N*5*sizeof(double)`. Note that the same format should be used for the output file with the final result from the simulation. This means that a result of a previous simulation can later be used as starting point for a new simulation.

Implementing the straightforward $O(N^2)$ algorithm and verify for some small cases that your code works properly (about small test cases, see below under input data). When you are confident that your code works correctly, move on to measure performance and consider possible optimizations.

Graphics is nice to have and recommended, but not mandatory. If you have difficulties making the graphics routines work on the computer you are using, it is OK to skip the graphics. The important thing for us is performance, and not how to use graphics routines. But it is strongly recommended use graphics if you can, it is a nice way to illustrate what you are doing and makes it easy to see if your code works properly for the small test cases. The graphics routines do work on the Linux computers in the lab rooms, so you can at least plot things there even if you are unable to make it work on your own computer.

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.

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.

You may work in groups of up to three. You need to formally form a group in the Student Portal to be able to submit. If you work alone, you anyway need to form a "group" of 1 person in the Student Portal before you can submit.

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 a report. If the group members have contributed equally to everything, then write that. If you focus on different things, make sure everyone in the group still undestands what you have done and how each part of your code works.

5. Input data

Unpacking the `Assignment3.tar.gz` file gives a directory called `input_data` with various input files that you can use. The smallest ones with just a few particles are good to use to verify that your forces and timestepping are working properly.
`circles_N_2.gal` : two stars with equal mass moving in circles.
`circles_N_4.gal` : four stars with equal mass moving in circles.
`sun_and_planet_N_2.gal` : one heavy particle and one lighter particle orbiting the heavy one, like a planet around a sun.
`sun_and_planets_N_3.gal` : sun and two planets.
`sun_and_planets_N_4.gal` : sun and three planets.

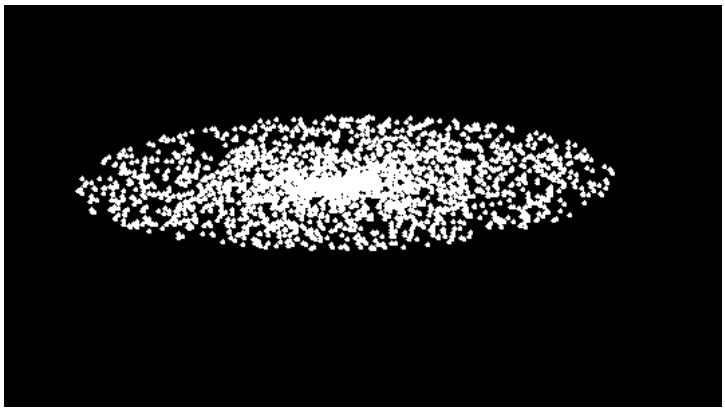

Figure 5.1. Example of an initial distribution of a galaxy with 2500 stars

For the larger cases that are more interesting for performance evaluation, the input files have the following form:

`ellipse_N_01500.gal`

where the number, 1500 in this case, is the number of stars N.

Those initial distributions have an elliptic shape similar to Figure 5.1, and their evolution may lead to (for example) somehing like Figure 5.2.

5.1. **Note: simulations will always be unstable for long times.** The small test cases with for example particles moving in circles, or like a sun with a few planets, are good for verifying that the force computation and time stepping is working properly. However, even when the computations are done correctly this

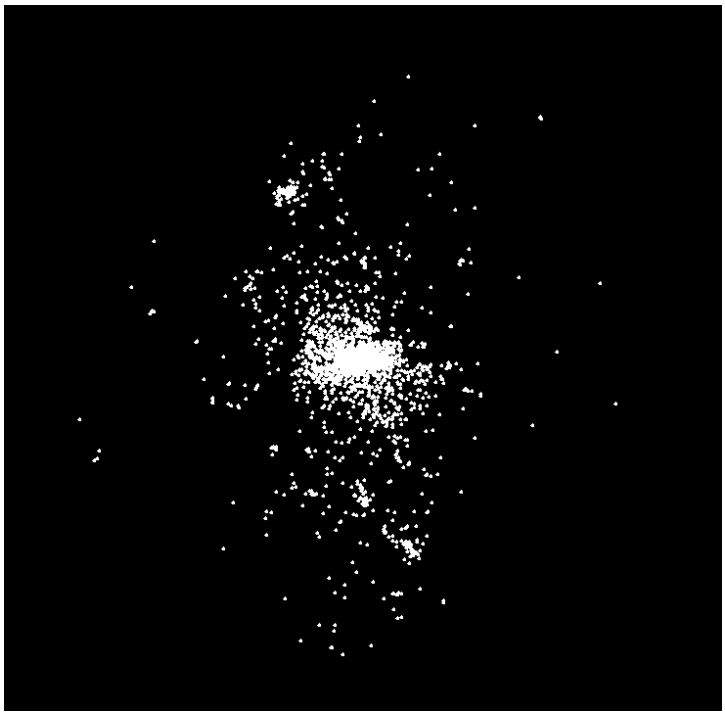

FIGURE 5.2. Example of a possible distribution of a galaxy with 2500 stars after some time (this picture is for a different initial distribution, you should not expect to get precisely this result)

kind of simulations will become unstable when running for a long time since small errors (there is always some effect of rounding errors) will accumulate over time and sooner or later this will always be seen.

So, when looking at the small simulations of a few planets etc. you should mostly focus on the initial behavior, for example check that the particles move in circles or orbit the sun in the expected way a few times, but if looking at it for longer times it is not strange that the simulations "go crazy", that is to be expected.

5.2. **Check your own results by comparing to reference output data.** To make it easier for you to check your own results, the `Assignment3.tar.gz` file contains in the `ref_output_data` directory a few reference galaxy distributions in the following files:
`ellipse_N_00010_after200steps.gal`
`ellipse_N_00100_after200steps.gal`
`ellipse_N_00500_after200steps.gal`
`ellipse_N_01000_after200steps.gal`
`ellipse_N_02000_after200steps.gal`
and a small program `compare_gal_files.c` for comparison of galaxy files.

As the filenames suggest, those files contain galaxy distributions after 200 timesteps. So, for example, if you run your own simulation for 200 timesteps starting from

`ellipse_N_01000.gal` your result should be close to the reference result in `ellipse_N_01000_after200steps.gal`.

The `compare_gal_files.c` program outputs the maximum difference between particle positions as `pos_maxdiff`. Since the coordinates for particle positions are supposed to be between 0 and 1, for one timestep the errors in positions should be small, on the level of rounding errors for the floating-point precision used. Errors will then of course accumulate during the simulation, but since the reference output data are for relatively short simulations, only 200 timesteps, error accumulation should not be too bad. Errors should still be within a few orders of magnitude from the size of rounding errors.

The `compare_gal_files.c` program also outputs the maximum difference between particle velocities as `vel_maxdiff`. Those error values will be larger since the velocities have larger magnitude, but the relative error should be small also for the velocities. If you want to verify that, check how large the velocities are during the simulation and use that information to determine the relative error.

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 `A3.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 `A3.tar.gz` should give a directory A3 and inside that there should be a makefile so that simply doing "make" should produce your executable file "galsim". The A3 directory should also contain your report, as a file called "report.pdf".

Apart from your pdf report, 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 can test it yourself as follows. Start by creating a separate directory for your test, and copy into that directory two files: your `A3.tar.gz` file and the `ellipse_N_00100.gal` input file. Then cd into that directory so that if you do "ls" you just see those two files listed. Then issue the following commands:

```
tar -xzf A3.tar.gz
cd A3
make
./galsim 100 ../ellipse_N_00100.gal 200 1e-5 0
```

Then you program should run a simulation for the given number of timesteps (200) and a `result.gal` file should be created containing the final result.

The report shall be in pdf format 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. Include a figure with a plot of measured execution time as a function of N to confirm the expected $\mathcal{O}(N^2)$ complexity.

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