Sequential all-pairs n-body simulation

This programming assignment is divided into two parts. In this first part you simply build a sequential program and measure its performance as described below. In the second part (PA1(b)) that will follow next week, you will use OpenMP to create a parallel implementation and report on its performance. For this assignment you may work on your own, or together with a partner in the class. The score earned is not affected by working in a team of two.

Problem Description

We consider a simple simulation of point masses interacting through the gravitational force in 2D. We start with a collection of \( n \) bodies \( b_1, \ldots, b_n \). Each body \( b_i \) is described by a scalar mass \( m_i \), a position vector \( r_i \), and a velocity vector \( v_i \) (vectors in 2-space).

For a given configuration of bodies, the force vector \( f_{ij} \) on \( b_i \) as a result of its interaction with \( b_j \) is

\[
f_{ij} = \frac{G m_i m_j r_{ij}}{||r_{ij}||^3}
\]

Where \( G \) is the gravitational constant, and \( r_{ij} = r_j - r_i \) and \( ||r_{ij}|| \) is the length of \( r_{ij} \). The total force \( f_i \) experienced by \( b_i \) as a result of its interaction with all other bodies is

\[
f_i = \sum_{1 \leq j \leq n, \ j \neq i} f_{ij}
\]

Since \( f = ma \), this force results in the acceleration

\[
a_i = \frac{f_i}{m_i}
\]

of body \( b_i \).

To simulate the trajectories of the \( n \) bodies over time, we numerically integrate the equations of motion using a simple forward Euler method with a timestep of \( \Delta t \). Starting from initial positions \( r_i^0 \) and velocities \( v_i^0 \) for all bodies at time \( t = 0 \), we advance the positions and velocities as follows. First, using the equations above and the positions \( r_i^t \) of all bodies at time \( t \), compute the acceleration \( a_i^t \) for each body \( b_i \). Then update the positions and velocities for body \( i \) to time \( t + \Delta t \).

\[
\begin{align*}
    v_i^{t+\Delta t} &= v_i^t + \Delta t \cdot a_i^t \\
    r_i^{t+\Delta t} &= r_i^t + \Delta t \cdot v_i^t
\end{align*}
\]

Our objective is to compute the final state of the system at time \( t = k\Delta t \), i.e. following \( k \) iterations of this update scheme starting from an initial configuration of the \( n \) bodies. In outline this can be done as follows:
Create initial configuration of bodies

\[ \text{for } t = 1 \text{ to } k \]
\[ \text{compute } a_i \text{ for all } 1 \leq i \leq n \]
\[ \text{update } r_i \text{ and } v_i \text{ for all } 1 \leq i \leq n \]

The body of the loop calculates \( n \) accelerations and updates \( n \) positions and velocities per iteration. In a simulation with \( k \) timesteps a total of \( k n^2 \) interactions are calculated. For sufficiently large values of \( n \), the time spent updating velocities and positions will be negligible, so we define the performance of the loop to be \( k n^2 / t_{\text{elapsed}} \) interactions per second, where \( t_{\text{elapsed}} \) is wallclock time in seconds for the loop.

Computational details

Use 64-bit floating-point values (double) for all quantities and in all calculations. A safe configuration of the bodies is equally spaced in a line or a circle in the unit square, with small mass \( m_i = 1.0 \) and no initial velocity \( v_i^0 = (0,0) \). Set the timestep \( \Delta t = 0.001 \) and use \( G = 6.673 \times 10^{-11} \text{ m}^3 \cdot \text{kg}^{-1} \cdot \text{s}^{-2} \). With these parameters the simulation will do nothing of interest for a large number of iterations, and should not encounter any pathologies. The goal of this exercise is to perform the computation quickly; the simulation is not of interest. If you are interested in seeing something happen, you can increase \( G \) to 1.0 to notice gravitational effects in a short simulation (with huge errors). You can perform some simple computational checks on your program. For example, the momentum of the system \( \sum_{1 \leq i \leq n} m_i v_i \) should be conserved. As our simple n-body simulation suffers from various numerical problems, principally due to the poor integrator and unbounded forces in colliding trajectories, the momentum is not well conserved. I will post the values from a small \( n \) simulation to compare to your implementation to judge ballpark similarity.

Programming Assignment

1. Program the simple sequential algorithm for the problem above using C or C++ on Phaedra.
2. Create a modified sequential version of (1) that uses Newton’s third law, \( f_{ij} = -f_{ji} \), to halve the number of force calculations. Whenever \( f_{ij} \) is computed, its contribution may be added to \( f_i \) and subtracted from \( f_j \), eliminating the need to subsequently compute \( f_{ji} \).
3. Check both programs compute nearly the same result (why might they not compute the exact same result?)
4. Optimize the performance of (1) and (2) using the gcc compiler or icc compiler. Do not parallelize the programs at this stage!
5. For values of \( n \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\} \) on the x axis, and using \( k = 4 \) time steps, plot the performance of both (1) and (2) on the y axis (in units of millions of interactions per second). Make 5 runs for each value of \( n \leq 2000 \) to show average performance and variation.
6. Please turn in the performance graph described in (5) and a listing of your program. The basic assignment is not time consuming, but be sure to leave sufficient time to explore performance improvements and to get the data you need. Details about access to phaedra and compilers will be on the class web page and will be covered in class.