COMP 633: Parallel Computing
Programming Assignment 1(a)
Sequential all-pairs n-body simulation

Assigned: Mon Sep 20
Due: Tue Sep 28 (at start of class)

This programming assignment is divided into two parts. In part (a) you simply build a sequential program and measure its performance as described below. In part (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 $\boldsymbol{r}_i$, and a velocity vector $\boldsymbol{v}_i$ (vectors in 2-space).

For a given configuration of bodies, the force vector $\boldsymbol{f}_{ij}$ on b_i as a result of its interaction with b_j is

$$\boldsymbol{f}_{ij} = \frac{G m_i m_j \boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|^3}$$

Where G is the gravitational constant, and $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i$ and $\|\boldsymbol{r}_{ij}\|$ is the length of $\boldsymbol{r}_{ij}$. The total force $\boldsymbol{f}_i$ experienced by b_i as a result of its interaction with all other bodies is

$$\boldsymbol{f}_i = \sum_{\substack{1 \le j \le n \\ j \ne i}} \boldsymbol{f}_{ij}$$

Since $\boldsymbol{f} = m\boldsymbol{a}$, this force results in the acceleration

$$\boldsymbol{a}_i = \frac{\boldsymbol{f}_i}{m_i}$$

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 Δt. 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 $\boldsymbol{a}_i^t$ for each body b_i. Then update the positions and velocities for body i to time $(t + \Delta t)$.

$$\boldsymbol{v}_i^{t+\Delta t} = \boldsymbol{v}_i^t + \Delta t \cdot \boldsymbol{a}_i^t$$
$$\boldsymbol{r}_i^{t+\Delta t} = \boldsymbol{r}_i^t + \Delta t \cdot \boldsymbol{v}_i^t$$

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
for t = 1 **to** k
  compute a_i for all $1 \le i \le n$
  update r_i and v_i for all $1 \le i \le n$
end

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 kn^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 kn^2/t_{elapsed} interactions per second, where t_{elapsed} is elapsed time in seconds between the start and end of 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 $\boldsymbol{v}_i^0 = (0,0)$. Set the timestep $\Delta t = 0.001$ and use $G = 6.673 \times 10^{-11}\ m^3 \cdot kg^{-1} \cdot s^{-2}$. 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 $\sum_{1 \le i \le n} m_i \boldsymbol{v}_i$ should be approximately 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 actually not conserved precisely. I will post the values from a small n simulation for you to compare with 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, $\boldsymbol{f}_{ij} = -\boldsymbol{f}_{ji}$, 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 $n \le 2000$ 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 are on the web page and will be covered in class.