CS 415 Parallel Computing

N-Body Computations

March 13, 2012

Erin Keith

Introduction:

This project focused on computing the gravitational interactions of any number of bodies in a defined space. The basic formula to compute the force of one body upon another is:

$$F = \frac{Gm_a m_b}{r^2}$$

where G is the gravitational constant, m_a and m_b are each body's mass, and r is the distance between the two bodies. The total force on a body is the sum of the forces of each additional body upon the reference body.

Each of the forces in two dimensional space have an x and y component, defined by:

$$F_x = \frac{Gm_a m_b}{r^2}\left(\frac{x_b - x_a}{r}\right)$$

and

$$F_y = \frac{Gm_a m_b}{r^2}\left(\frac{y_b - y_a}{r}\right)$$

These forces contribute to the velocity of a body (which also has two components) to change the position of that body in space. A series of frames are included in Figure 1, which illustrate the effect of two bodies on a third smaller body, sending it flying through space.

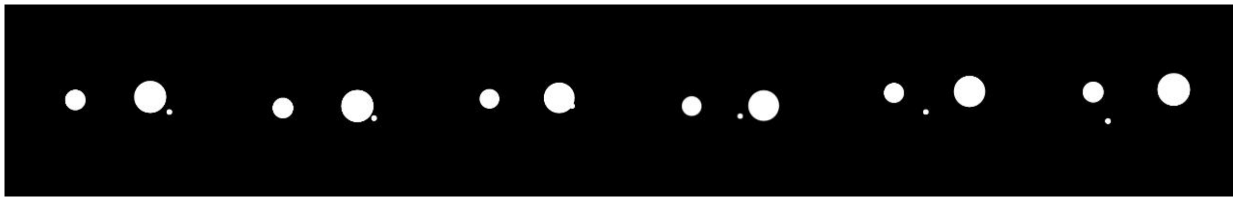

Figure 1: Frames from N-body Animation

Procedure:

Sequential Program

To master the physics and math required by this program, I began with coding the sequential solution. This also enabled me to test the algorithm and develop the animations to make sure that my solution worked.

Since each body is defined by a position and mass, I created a struct to define a body, and created an array to hold all of them. The bodies were initialized to random positions and masses, and their initial velocities were set to zero.

A function was created to calculate the total force of each of the bodies on individual body in the iteration. The force over mass was added to the velocity to update the new velocity, which was added to the old position to determine the new position.

I noticed that the bodies were exiting the "universe" too quickly, so I imposed conditions that kept them within bounds, which helped improve the results. After watching some simulations on YouTube, I realized that the bodies may still be interacting, but just out of the small boundaries I had set. After increasing the size of the universe, I was able to increase the iterations significantly, which made for longer more interesting animations and some parameters that would make implementation analysis more meaningful.

Part II: Parallel

My approach to parallelizing the problem was to assign each body to a processor. Separating the problem in this manner lends itself directly to Type 3 pipelining, as explained in our Parallel Programming book. The master process initialized a position array and a mass array. The master passed the mass array one time for storage in each of the slaves.

With each iteration, the master passed the position array to the first slave, which passed it to the subsequent slave, etc. until the position array reached the last slave. After each slave received and sent the positions, it computed its own force, velocity and new positions. The new positions got sent back to the master and the whole process was repeated.

Analysis:

As usual, I did not have enough time to get a broad enough data set to properly analyze the results. What I did notice is that there was no speedup of parallel implementation over sequential for such a small number of bodies. Table 1 displays the runtime values collected and Figure 2 provides a graph of these times.

Table 1: Sequential and Parallel Runtimes for N-Bodies

Number of Bodies	Sequential Time (sec)	Parallel Time (sec)
10	0.03	2.97
20	0.3	4.5
30	0.59	4.9
40	0.6	5.2
50	0.94	
60	1.32	

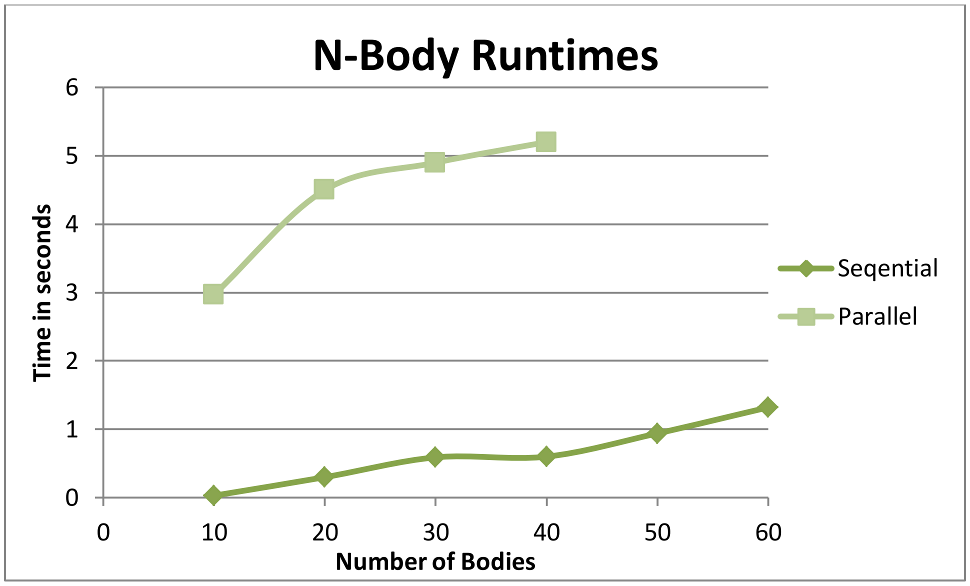

Figure 2: Runtimes for Sequential and Parallel N-Body Implementation

The communication time must have been what was bogging down this implementation. My theory is that this implementation of the parallel algorithm would not see speedup until a significant N was reached to overcome the communication time.

Conclusion:

I am not very satisfied with the results of my parallel implementation. I was very curious to see if the parallel implementation would level out as the sequential times crept toward it, but by the time I was ready to run it using a larger amount of processors, the grid was slammed. I did notice some discussion of using some MPI functions that haven't been discussed in class or the book yet, which certainly begs further looking into, as they would significantly reduce the amount of communication. Obviously the Barnes-Hut algorithm would reduce runtimes, but now I'm more interested in conquering the $O(n^2)$ speedup problem.