Aditi, Dennis, Rafay, Sezim, Jared

CS 205 Final Project Proposal

Multi-threaded Gravity Simulator

The aim of our project is to create a gravity simulator. Given the masses, initial positions, and initial velocities of every object in a universe, the program should determine the objects' positions at every future point in time. Each object can have a thread associated with it, which computes what the state of that object should be at the next time step (i.e., determine its new position and velocity), given the current state of the universe. The changes in the system over time can be displayed to the user in a visualization.

Many of the large performance gains we anticipate to capture in our project will come through task-level parallelism. This is because the significant computational bottleneck in such a physical simulation will be having to calculate the velocities, position, and trajectories of all the objects in our system. This is a task that is naturally suited for parallelism. Moreover, since the motion of each object will be dependent on the other objects in the system, our parallelism must account for ways for the different objects in the system to dynamically interact with one another; the result is that no part of the application can be trivially or embarrassingly parallelised.

We plan to parallelize the calculation of the location and state of each object, sync them to the same state every fraction of a second in a memory-consistent model. We will focus on shared memory models due to the need to coordinate the position and interaction between every object on the map using Newtonian physics. We can use the MIMD machine model for our problem.

The size of memory depends on how many objects we are simulating, but we estimate it to be between 32 MB to 1 GB. Given that each core has access to 69GB of memory, the size of our problem is definitely reasonable for the resources that we have access to.

A single run of the simulation takes any time from 30 seconds to 5 minutes. The solution to our problem of identifying where everything will be after a certain timeframe will require a high volume of output due to the snapshot recording, every moment in time, of the different bodies of objects located within the frame. So, we expect that in the sequential code, as the number of objects increase, the performance will drop significantly and time needed for simulation go up. This is where parallelization will be of use.