# MPCS 51087
# Project 3
# GPU N-Body

Winter 2023

## 1 Background

**Due**: Sunday February 26th @ midnight CT

The N-Body Problem is the problem of predicting the motion of $N$ mutually interacting objects. A common example is self-gravitating bodies, where the forces of interaction described by Newton's Law of Universal Gravitation. The behavior of two body systems can be solved analytically, but larger N-body systems require a computational approach. The most complete model of an N-body system requires an $O(N^2)$ algorithm wherein the total force for each body is computed by summing the forces derived from all other bodies in the system.

Consider the system shown in Figure 1 below:

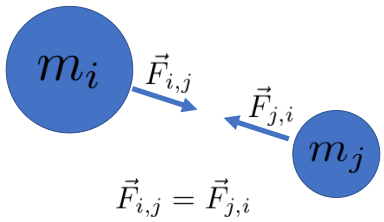


**Figure 1:** Two bodies in 3D space.

To compute the force between two bodies $i$ and $j$, we can use Newton's Law of Universal Gravitation:

$$\vec{F}_{i,j} = G\frac{m_i m_j}{|\vec{r}_{i,j}|^3}(\vec{r}_j - \vec{r}_i) \tag{1}$$

where $\vec{F}_{i,j}$ is the 3D force vector between body $i$ and body $j$, $G$ is the gravitational constant, $m_i$ and $m_j$ are the masses of body $i$ and $j$ respectively, $|\vec{r}_{i,j}|$ is the Cartesian distance between the two bodies, and $\vec{r}_i$ and $\vec{r}_j$ are the 3D Cartesian location vectors of the two bodies. Note that an extra *softening* term is often also used when computing $|\vec{r}_{i,j}|^3$, to enforce a minimum degree of separate so that point particles that pass very close to one another do not experience forces that approach infinity (which can cause numerical instability issues).

With a method for computing the force between any two bodies $i$ and $j$, we can therefore calculate the collective force on body $i$ in any arbitrary system of n-bodies (for instance, Figure 2) by simply computing the forces between body $i$ and all other bodies and summing them up, as in:

$$\vec{F}_i = \sum_{j=0}^{n} \vec{F}_{i,j} \tag{2}$$

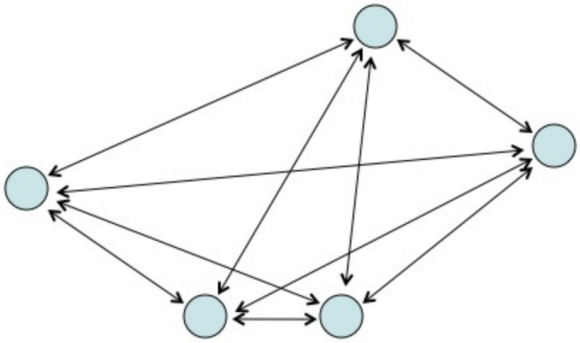

**Figure 2:** Multiple bodies in 3D space.

If we know the total force acting on a particle, we can determine its movement over time by considering several additional equations. First, we can compute a body's velocity as a function of time as follows:

$$\vec{v}_i(t) = \frac{\vec{F}_i}{m_i}t \tag{3}$$

where $\vec{v}_i$ is the body's velocity at time $t$, $\vec{F}_i$ is the total (constant) force on the body, and $m_i$ is the body's mass. We can also write a function for the particle's position as a function of time as follows:

$$\vec{r}_i(t) = \vec{v}_i t \tag{4}$$

where $\vec{r}_i$ is the body's location vector at time $t$, and $\vec{v}_i$ its (constant) velocity. Notable limitations of Equations 3 and 4 are that they both depend on forces remaining constant over time, though in reality as bodies move the forces on them also change. To remedy this limitation, we can then apply a finite difference approximation (similar to what we've done in previous problem sets) to determine the change in velocity and location for some very small timestep $\Delta t$, resulting in the following equations:

$$\Delta\vec{v}_i = \frac{\vec{F}_i}{m_i}\Delta t \tag{5}$$

and:

$$\Delta\vec{r}_i = \vec{v}_i\Delta t \tag{6}$$

These finite difference equations allow us to accurately move the particles forward for a small timestep and then re-calculate the forces on all particles in an iterative manner. Given these ideas, we now have all the tools necessary to simulate a time-dependent n-body system wherein the forces, velocities, and positions of each body are updated in each discrete timestep of the simulation, as described in Algorithm 1 below. Algorithm 1 considers $n$ bodies, each of which has a state consisting of a 3D velocity vector $\vec{v}$, a 3D location vector $\vec{r}$, and a mass $m$.

## 1.1 Serial version: Performance, Testing, & Plotting (20 points)

The first step is to test the provided serial implementation, including 2D animations. The purpose of testing the code in serial is to gauge performance expectations and to facilitate testing the parallel version. Correctness is particularly challenging in this assignment— erroneous codes may still output particle trajectories that appear reasonable, in contrast to previous assignments where coding errors usually resulted in obvious errors in outputted results. Report on the results of some tests that helped you gain confidence in the results. Some ideas for more interesting initial conditions to check correctness might be:

- to start the particles in two or more separate randomized clusters

**Algorithm 1** N-Body Simulation Pseudocode

Input $n$, $\Delta t$, $N$ ▷ $n$ = bodies, $\Delta t$ = timestep, $N$ = number of timesteps
Allocate array of $n$ bodies
Initialize $n$ bodies
**for** $0 \leq t < N$ **do** ▷ Loop over timesteps
    Output particle positions to file
    **for** Each particle $i$ in $0 \leq i < n$ **do**
        $\vec{F}_i = 0$
        **for** Each particle $j$ in $0 \leq j < n$ **do**
            Compute $\vec{F}_{i,j}$ ▷ Equation 1
            $\vec{F}_i = \vec{F}_i + \vec{F}_{i,j}$
        **end for**
        $\vec{v}_i = \vec{v}_i + \frac{\vec{F}_i}{m_i}\Delta t$ ▷ Equation 5
    **end for**
    **for** Each particle $i$ in $0 \leq i < n$ **do**
        $\vec{r}_i = \vec{r}_i + \vec{v}_i \Delta t$ ▷ Equation 6
    **end for**
**end for**

- bias the velocities in different areas of the problem to create global rotation or local swirls
- start all particles at nearly uniform grid points (with small random noise applied to all starting locations)
- have particles start in a galaxy spiral formation
- start all particles in a cluster with high outward velocities
- something else entirely, or some combination of the above

Feel free to collaborate with other students to define simple benchmarks that increase evidence of correctness. Generate an animation using your starting conditions with your serial code using 1,000 particles and 1000 timesteps. Upload your animation somewhere (Dropbox, Google Drive, YouTube, etc.) and include a link to your serial animation in your PDF writeup, as well as briefly discussing what you did to initialize the system and if it turned out as expected

Finally, measure and report on the execution time per timestep for the serial code using 100,000 bodies.

## 1.2 Shared Memory Parallelism (25 points)

Use OpenMP to develop and test a multicore version of your code. Show a comparison of results to the serial code to verify correctness. Show a scaling plot for the 100,000 body configuration as a function of core count. Comment on what you observe.

## 1.3 GPU Implementation (50 points total)

Implement a GPU version of the code using whatever strategy, parameter choice, and optimizations give you the best performance. Show a comparison of results to the serial and OpenMP codes to verify correctness. Report on your best performance compared to the multicore version for the 100,000 particle benchmark.

## 1.4 Compiling, Code Cleanliness, and Documentation (5 pts)

We will compile and execute your code as part of the grading process. You must include a Makefile capable of compiling your code into a runnable executable. Also, while we are not grading to any rigid coding standard, your code should be human readable and well commented so we can understand what is going on.

### 1.5 What to Submit

Your repository should contain:

- Your source code. The serial and parallel versions can all be in the same repository in the form of different functions.
- A working Makefile that compiles your code.
- Your plotting script(s), and any instructions on how to use them (if you are not using the provided plotter)
- A single writeup (in PDF form) containing all plots and discussions asked for in the assignment
- Links in your writeup to your final animations.