MPCS 51087
Project 4
Distributed Memory N-Body

Spring 2022

1 Introduction

- Milestone 1 (Sections 1.1-1.3) due Sunday May 22 @6pm
- Complete Assignment: due Sunday, May 29, @6pm (May 26 @midnight for graduating students)

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:

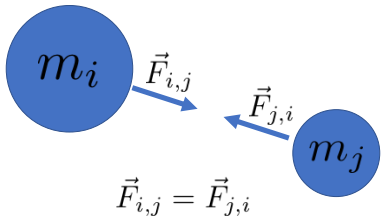

Figure 1: Two bodies in 3D space.

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:

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

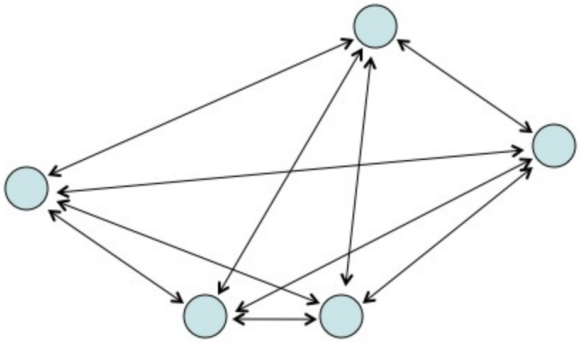

Figure 2: Multiple bodies in 3D space.

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 Δt, 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 Implementation: Performance, Testing, & Plotting (20 points)

The first step is to complete and test a serial implementation, including 2d animations. The purpose of implementing the code in serial is to guage 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. Feel free to collaborate with other students to define simple benchmarks that increase evidence of correctness. Finally, measure and report on the time per iteration for the serial code using 102,400 bodies.

Algorithm 1 N-Body Simulation Pseudocode

Input n, Δt, N ▷ n = bodies, Δt = 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

1.2 Formulation of Interesting Initial Conditions (5 points)

Next, experiment with initial particle states that create interesting animations. Some ideas for more interesting initial conditions might be:a

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

You are welcome to tweak the physical parameters of the problem (G, Δt, initial velocity and mass ranges, softening, etc) as you see fit – just make sure your program remains stable and doesn't require more than 500 or so timesteps to be interesting. You may also want to adjust the plot domain to capture effects at different scales from the default.

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. If you end up changing your parameters again later on in the assignment to work better for high particle counts, you do not need to regenerate this animation.

1.3 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 102,400 body configuration as a function of core count. Comment on what you observe.

1.4 Parallel MPI Implementation (50 points total, broken down below)

Your task for this assignment is to parallelize the N-Body problem using MPI and OpenMP so as to allow for hybrid parallelism.

Parallelization of Algorithm 1 should come in the form of decomposing particles across MPI ranks and then building a "pipeline" to exchange particle sets between ranks. No single rank is allowed to store all particles at any one point in time. For decomposing N total particles, this can be accomplished by having each of the M MPI ranks initializing and storing $N_{local} = N/M$ local particles. Each rank will be responsible for computing forces and updating its local particle velocities and positions throughout the entire simulation. While ranks can compute the forces between each of their N_{local} particles locally without communication, they will need a way to compute the forces between their local particles and all other particles residing on the other MPI ranks in the simulation. For this assignment, you will be implementing a "pipelining" scheme wherein each subset of particles will be passed around between all MPI ranks, as shown in Figure 3. Note that a full pipeline, where particle sets visit all MPI ranks, must be performed for each timestep of the simulation. The N-Body parallel pipeline scheme you will be implementing is outlined in Algorithm 2.

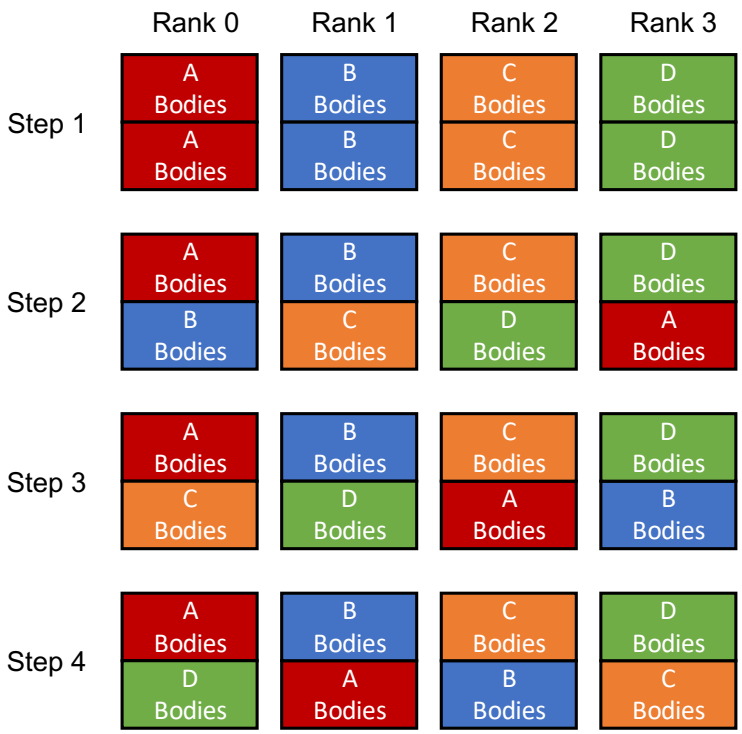

Figure 3: Pipeline example on 4 MPI ranks, showing MPI ranks passing travelling "remote" body sets to the left at each step.

As is shown in Figure 3, each of the 4 MPI ranks in the example sample a subset of N bodies, with each being responsible for computing the total forces and updating the velocities and positions for their set each timestep. The particles each rank begins with will be known as that rank's "local" particles that it will always hold onto and track for the duration for the simulation. A series of pipelined communication steps are performed where all ranks shift a "remote" set of particles to their left, compute forces on their "local" particles from local-remote particle interactions, and then continue shifting the remote particles around to their left until all particle sets have visited all ranks.

A few things to note regarding your implementation:

- Note that in this pipeline process, only the "local" sets have their velocities updated every communication step. The visiting "remote" particles are read-only.

Algorithm 2 N-Body Parallel Pipeline Simulation Pseudocode

1: Input n, Δt, N, M ▷ n = # of bodies, Δt = timestep, N = # of timesteps, M = # MPI ranks
2: $n_l = n/M$ ▷ Compute number of bodies per MPI rank
3: Each rank allocates array of n_l local bodies B_{local}
4: Each rank initializes its n_l local bodies B_{local}
5: Each rank allocates array of n_l remote bodies B_{remote}
6: **for** $0 \leq r < M$ **do** ▷ Initialize all bodies in problem
7: Initialize n_l new bodies on Rank 0 ▷ Use randomized initialization method of your choice
8: Rank 0 transmits n_l new bodies to Rank r
9: Rank r stores new bodies in B_{local}
10: **end for**
11: **for** $0 \leq t < N$ **do** ▷ Loop over timesteps
12: Output particle positions to file (MPI Coordinated)
13: $B_{remote} = B_{local}$ ▷ Copy local bodies to remote buffer
14: **for** M Iterations **do** ▷ Pipeline Loop
15: **for** Each body i in B_{local} **do** ▷ OpenMP Parallel For Loop
16: $\vec{F}_i = 0$
17: **for** Each body j in B_{remote} **do**
18: Compute $\vec{F}_{i,j}$ ▷ Equation 1
19: $\vec{F}_i = \vec{F}_i + \vec{F}_{i,j}$
20: **end for**
21: $\vec{v}_i = \vec{v}_i + \frac{\vec{F}_i}{m_i}\Delta t$ ▷ Equation 5
22: **end for**
23: Send B_{remote} buffer to left neighbor rank
24: Receive new B_{remote} buffer from right neighbor rank
25: **end for**
26: **for** Each body i in B_{local} **do**
27: $\vec{r}_i = \vec{r}_i + \vec{v}_i\Delta t$ ▷ Equation 6
28: **end for**
29: **end for**
30: Return x

- You are not allowed to allocate space for all n total bodies on any single MPI rank. You should only be allocating enough space to hold the local bodies plus buffer(s) to store the travelling remote bodies.
- You may wish to allocate a second "remote" buffer to make coordination of lines 23 and 24 of Algorithm 2 easier. I.e., you may want to have one buffer for outgoing bodies and another for incoming bodies, so they can be sent/received simultaneously.
- In Lines 6-10 of Algorithm 2, you are generating chunks of bodies in serial on a single MPI rank and then distributing them to their home ranks. Alternatively, you could generate the randomized bodies locally on their home ranks. However, it is desirable for this assignment to have the ability to generate identical particles regardless of if running in serial or parallel. Use of the method of generation & transmission defined in Lines 6-10 will allow you to use the same parallel stream to check for identical output animations between smaller serial and parallel runs so that you can check for bugs and ensure that your parallel code is working as expected. There are other ways of ensuring reproducible results that you are welcome to try out instead if you like.

1.4.1 Demonstrating Correctness (15 points)

Correctness is particularly challenging in this assignment, as incorrect implementations may still output particle activities that look reasonable to the eye, in contrast to previous assignments where implementation errors usually resulted in visually obvious problems in outputted results.

To test your parallel implementation for correctness, run a simulation in both serial and parallel and compare the data output files. You should perform your verification simulation using the following parameters:

- Number of Timesteps = 10
- Number of Bodies = 128
- Number of Parallel MPI Ranks = 4

You should be able to generate identical results out to 10 decimal places on this configuration (due to the non-associativity of floating point operations, bit-wise reproducibility is unlikely). Furthermore, you may notice that results totally diverge for higher particle counts across greater numbers of timesteps and processor counts. That said, the parameters given above will be sufficient for providing a basis to test and debug your code so as to ensure particles are being exchanged and forces computed correctly.

In your PDF writeup, report if you were able to get identical results for these simulation parameters.

1.4.2 Strong Scaling on Midway (15 points)

Perform a strong scaling study on the "caslake" partition of Midway using 1,2, 4, and 8 nodes, testing both pure MPI and hybrid MPI+OpenMP parallelism styles. In this case, your MPI–only runs should be executed using 24 MPI ranks per node with 1 thread per rank, and your hybrid MPI+OpenMP runs should be executed with 1 MPI rank per node with 24 threads per rank.

Perform the study by node rather than by individual CPU, so that the smallest problem you run is 1 node using all available CPUs (ie 24 ranks in the all-MPI configuration). This is sometimes done in HPC as it is often impractical for larger problem sizes to be able to execute a code on a single CPU, so we will use a single full node as the smallest division possible. Use the parameters given below for your strong scaling study:

- Number of Timesteps = 10
- Number of Bodies = 102,400

Plot your strong scaling curves on a single plot, with the y-axis as runtime (rather than speedup), so that both the pure MPI and hybrid MPI-OpenMP methods can be directly compared. Include your plot in your PDF writeup, and draw a conclusion as to which parallelism method performs better for your N-Body code on Midway.

1.4.3 Production Simulation (15 points)

Run your code with 1,048,576 particles for 400 timesteps and generate an animation of your results. Upload your animation to somewhere in the cloud and submit a link in your PDF writeup.

1.5 Compiling, Code Cleanliness, and Documentation (5 pts)

We plan on compiling and running your code as part of grading it. 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.

1.6 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.