MPCS 51087
Problem Set 6
The N-Body Problem on Blue Gene/Q

Spring 2018

Part 1 Due Date: Wednesday, May 30, 2018 by 11:59 pm

Part 2 Due Date: Thursday, June 7, 2018 by noon

1 Part 1 - Running on BG/Q (5 points)

For Part 1 of this assignment, all that is required is that you login to Vesta and compile and run a very brief MPI job on 32 nodes before 11:59 pm of Wednesday May 30. You can use any code you like (e.g., a previous MPI based homework assignment, or something as simple as an MPI "hello world from rank N!"). The purpose of this exercise is to ensure any account issues are resolved well before the deadline for the main portion of the problem set approaches. You do not need to submit anything for this portion of the assignment to us (as we can see the logs on Vesta so can confirm when you've run your first job).

2 Part 2 - The N-Body Problem on BG/Q

2.1 Introduction

The N-Body Problem is the problem of predicting the motion of N objects each of which exerts a force upon each other. A common example is self-gravitating bodies, where the only forces with which they interact are the forces described in 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. To faithfully model the behavior of an N-body system, an $O(N^2)$ algorithm is required wherein the total force for each body is computed by summing the forces derived from all other bodies in the system.

To begin, consider the system shown in Figure 1 below:

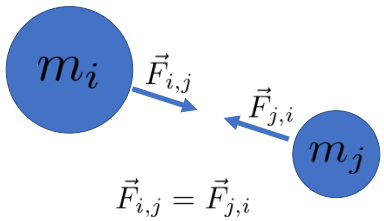

Figure 1: Two bodies in 3D space.

If we wish 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:

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

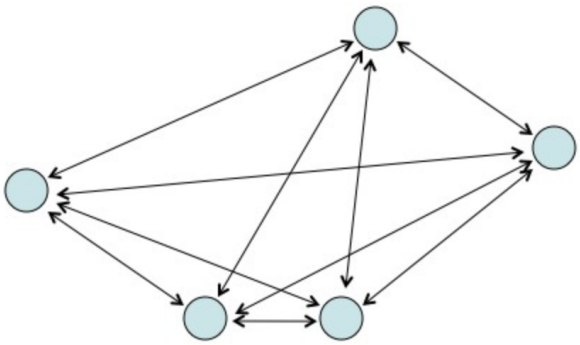

Figure 2: Multiple bodies in 3D space.

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.

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 ≤ t < N do                        ▷ Loop over timesteps
    Output particle positions to file
    for Each particle i in 0 ≤ i < n do
        F_i = 0
        for Each particle j in 0 ≤ j < n do
            Compute F_{i,j}             ▷ Equation 1
            F_i = F_i + F_{i,j}
        end for
        v_i = v_i + (F_i / m_i) Δt      ▷ Equation 5
    end for
    for Each particle i in 0 ≤ i < n do
        r_i = r_i + v_i Δt              ▷ Equation 6
    end for
end for
```

2.2 Serial Implementation & Plotting

We have provided you with a reference serial implementation of the N-Body simulation similar to what we went over in class. You are welcome to build off this code, alter it in any way, or build your own from the ground up. It is available on github at: (`https://github.com/jtramm/nbody_serial`).

Data visualization for a time dependent N-Body simulation is a little more complicated than the simple 2D plots required for previous problem sets. To allow you to focus on the parallel aspects of this assignment, we have provided you with a reference plotting script in python that is capable of creating a `.mp4` video animation given an ASCII particle data file. You are welcome to use your own data format and/or write your own plotting scripts using any plotting program you like, the reference script is just provided to save time for those not interested in doing so. Note – if you decide to use a different plotting scheme, make sure it will work for very large data files on the order of 10 to 20 GB in size.

The purpose of implementing the code in serial is to allow you to compare your results to the parallel version (given the same starting random seed) to ensure the parallel version of your code is working correctly. Correctness is particularly challenging in this assignment, as deeply flawed codes may still output particle activities that look reasonable to the eye, in contrast to previous assignments where coding errors usually resulted in obvious errors in outputted results.

In our serial implementation, we use several default parameters for location, velocity, and mass sampling along with default parameters for G, Δt, and softening. If implementing your own code from scratch, you may want to use these parameters as a starting point to make getting up and running smoother (though you are welcome to change these, as we will discuss in the next section).

2.3 Formulation of Interesting Initial Conditions (5 points)

Once you are up and running with the serial code, come up with your own way of initializing particle states so as to create a simulation animation that's more interesting than the default distribution (which is uniform

in space, velocity, and mass). If building off the provided example serial code, this can be done by altering the `randomizeBodies()` function.

Some ideas for more interesting initial conditions might be:

- 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 plotting script domain to capture effects at different scales from the default.

Note that you may want to adjust the masses of your bodies such that the total mass of the system remains constant regardless of how many bodies are simulated.

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 again.

2.4 Parallel MPI Implementation (25 points)

Your task for this assignment is to parallelize the N-Body problem so as to run on the Vesta Blue Gene/Q supercomputer. In particular, you will be 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.

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.
- 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.

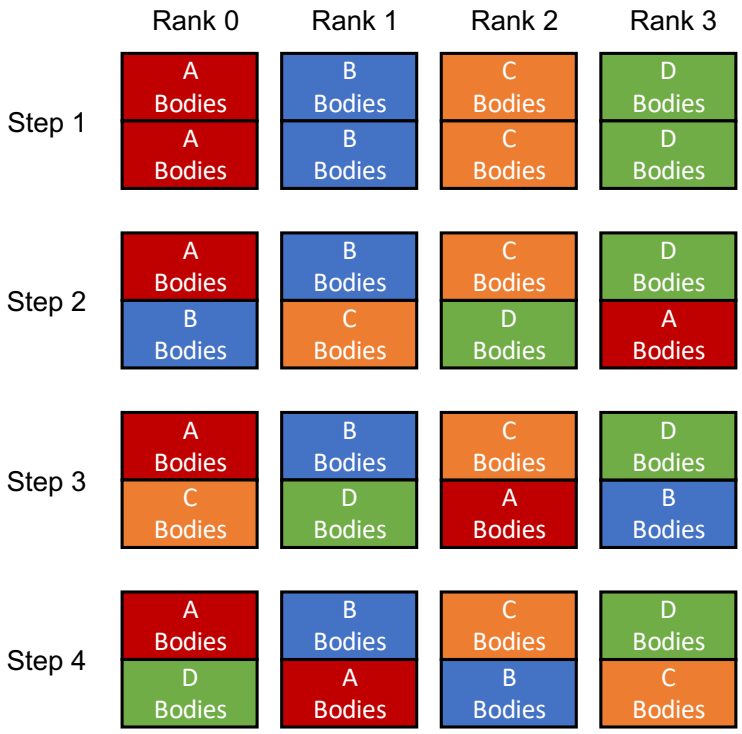

Figure 3: Pipeline example on 4 MPI ranks, showing MPI ranks passing travelling "remote" body sets to the left at each step.

- 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 out 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.

2.5 Parallel File Output (20 points)

In previous assignments serialization to disk usually only occurred once at the end of the simulation, and you were only running on a handful of MPI ranks, so it was not particularly important for your parallel file I/O schemes to be well optimized.

This assignment adds an extra degree of importance to the file I/O problem as you may have up to 65,536 MPI ranks writing data to a file *every iteration*. So, you will need to implement a performant scheme to ensure efficiency when running at scale. We highly recommend using MPI collective File IO operations (e.g., you might wish to adapt this example to your purposes: `https://stackoverflow.com/questions/9777828/writing-a-matrix-into-a-single-txt-file-with-mpi`).

You should be outputting a single data file from your simulation. Do not output individual files from different MPI ranks and stitch them together at the end, as writing to over 1,000 files at once is not a good idea on the BG/Q file system.

Algorithm 2 N-Body Parallel Pipeline Simulation Pseudocode

Input n, Δt, N, M ▷ n = # of bodies, Δt = timestep, N = # of timesteps, M = # MPI ranks
$n_l = n/M$ ▷ Compute number of bodies per MPI rank
Each rank allocates array of n_l local bodies B_{local}
Each rank initializes its n_l local bodies B_{local}
Each rank allocates array of n_l remote bodies B_{remote}
for $0 \leq r < M$ **do** ▷ Initialize all bodies in problem
    Initialize n_l new bodies on Rank 0 ▷ Use randomized initialization method of your choice
    Rank 0 transmits n_l new bodies to Rank r
    Rank r stores new bodies in B_{local}
end for
for $0 \leq t < N$ **do** ▷ Loop over timesteps
    Output particle positions to file (MPI Coordinated)
    $B_{remote} = B_{local}$ ▷ Copy local bodies to remote buffer
    for M Iterations **do** ▷ Pipeline Loop
        for Each body i in B_{local} **do** ▷ OpenMP Parallel For Loop
            $\vec{F}_i = 0$
            for Each body j in B_{remote} **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
        Send B_{remote} buffer to left neighbor rank
        Receive new B_{remote} buffer from right neighbor rank
    end for
    for Each body i in B_{local} **do**
        $\vec{r}_i = \vec{r}_i + \vec{v}_i\Delta t$ ▷ Equation 6
    end for
end for
Return x

Adding to the challenge is that BG/Q is a big-endian architecture, meaning that binary files you output on BG/Q will not be natively readable on your little-endian intel laptop when generating a plot. So, you will either need to write programs to manually flip the endianness of binary output files, or more preferably just output your files in ASCII format for easy portability (recommended).

Finally, when generating animated plots from your outputted data files for very large runs, you should not be doing this task on the login nodes of Midway or Vesta, as the data files for this assignment will get quite large and may take a non trivial amount of time to animate. Instead, you should download your data files and plot locally or allocate a compute node on Midway for generating the plot.

NOTE – you should be sure to use 64 bit types when computing byte offsets for your file i/o routines, as for larger problem sizes you can easily overflow a 32 bit integer. It is recommended you use `MPI_Offset` or `size_t` types for this purpose.

2.6 Testing for Correctness (5 points)

Correctness is particularly challenging in this assignment, as deeply flawed codes 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 problem. However, due to the non-associativity of floating point operations, you will find that you may not be able to generate bit-wise identical results on this problem. 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 tallied correctly.

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

2.7 Strong Scaling on Midway (10 points)

Perform a strong scaling study on the "sandyb" partition of Midway out to 8 nodes, testing both pure MPI and hybrid MPI+OpenMP parallelism styles. In this case, you pure MPI runs should be executed using 16 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 16 threads per rank.

Only test node configurations that are powers of 2 (e.g., 1, 2, 4, 8). Perform the study by node rather than by individual CPU, so that the smallest problem you run is 1 node using all available CPUs. 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 is optimal for your N-Body code on Midway.

2.8 BG/Q Testing

In this portion of the assignment, you will be running on Vesta, a Blue Gene/Q supercomputer at Argonne. The class allocation is for 500,000 core-hours, and you should try to keep your personal usage to below 40,000 core-hours. You can check your total usage to date with the `sbank users` command.

2.8.1 Performance Analysis (10 points)

Run tests to determine if pure MPI (64 MPI ranks per node with 1 thread per rank) vs. hybrid MPI-OpenMP (1 MPI rank per node with 64 threads per rank) is faster on 32 nodes of Vesta for your code. Use the following parameters for your comparison:

- Number of Vesta nodes = 32
- Number of Timesteps = 10
- Number of Bodies = 102,400

Report your findings in your PDF writeup. Is the same parallelism method optimal on both Midway and BG/Q? Why might this be? Use the faster parallelism style for your testing in the following sections.

For your faster parallelism method, also compute the fraction of runtime that File I/O is responsible for. You can measure this either by adding timers to your code, or more simply just re-running with all file I/O

function calls commented out and comparing this "compute only" runtime to the full simulation with File I/O included. Report what percent of runtime at this scale is attributed to file I/O in your PDF writeup. If it is greater than 20 or 30% of runtime, you may need to go back and optimize your File I/O scheme before moving on!

Finally, compare the runtime you observe on 32 nodes of BG/Q Vesta and to your performance results on 8 "sandyb" nodes of Midway. If we assume a BG/Q node uses 80 Watts, and a Midway node uses 400 Watts, which architecture is more efficient in terms of performance per power for your code? That is, which has a higher value for interactions per second per Watt? Report your results in your PDF writeup.

2.8.2 Strong Scaling (10 points)

Perform a strong scaling study on BG/Q Vesta out to 1024 nodes. Perform the study by node rather than by individual CPU, with the smallest problem you run as 32 nodes using all available CPUs and threads. In this manner, the runtime you get for 32 nodes will be taken as the "ideal" performance for 32 nodes. Use the parameters given below for your strong scaling study:

- Number of Timesteps = 10
- Number of Bodies = 524,288

Include a plot of your strong scaling analysis in your writeup. As usual, just test configurations that are powers of 2, i.e., (32, 64, 128, 256, 512, and 1024 nodes).

2.8.3 Production Simulation (10 points)

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

Note that if you are using the included python plotting script, you may want to alter the marker size and increase the bitrate and/or resolution so as to improve the clarity of your high particle count animation.

Note that this job will likely be your highest core-hour count job for the semester, so you will want to be sure your code is ready and you're submitting with the right options. To this end, you should generate a short test animation by running your final code at scale for only 10 iterations to make sure things are looking good before submitting a job for the full 400 iterations. You may find that you will want to adjust your particle initialization parameters (masses, velocities, etc) to accommodate the higher particle counts. Potentially, you may also find that you'll need to optimize your code more to be able to run within a reasonable time window (you should be able to do this simulation in 30-45 minutes on 1024 nodes).

In the event that you are running low on core-hours at this point in the assignment (you should not use more than 40,000), or cannot optimize your code to run within your allocation limits, run a smaller test problem at least and submit those results instead.

2.9 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.

2.10 What to Submit

You submission should be in the form of a zip or tar file containing the following items:

- 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.