High Performance Computing

MPCS 51087

Problem Set 5

The N-Body Problem on Blue Gene/Q

Spencer Zepelin

August 29, 2019

Compiling and Running the Code

The program was written in C with the gnu99 standard. Source code for the program can be found in the directory “src_nbody” containing the files main.c, utils.c, parallel.c, serial.c, nbody_header.h, and a makefile. The mode in which the program runs—serially, with MPI parallelism, or with hybrid MPI/OpenMP parallelism—will be determined at compile time.

A makefile has been provided for ease of compilation. Modifying the values for “MPI” and “OPENMP” at the top of the makefile will determine which mode is compiled. If both are “no”, the program will run serially. If only “MPI” is “yes”, the program will be compiled for MPI parallelism. If both are “yes”, the program will be compiled for hybrid parallelism. Note that exclusive OpenMP parallelism has not been built into this project, so setting “OPENMP” to “yes” while “MPI” is “no” will result in a serial run.

Setting “OPTIMIZE” to “yes” add the “-O3” flag to any compilation.

Once the proper options have been selected, running “make” will create the executable “nbody”. “make run” will execute the serial version of the code with the default arguments, and “make mpi” will execute a parallel version of the code with the same default arguments and two MPI ranks.

The default arguments are as follows:

Number of bodies = 1000
Size of timestep = 0.2
Number of time steps = 200
Number of threads = 1

Running the serial executable with your own arguments should look as follows:

./nbody <#bodies> <size_timestep> <#timesteps>

Running either parallel executable with your own arguments should look as follows:

mpirun -n <#ranks> ./nbody <#bodies> <size_timestep> <#timesteps> <#threads>

In any case, the data generated by the program will be output to “nbody.dat”.

The source code for generating animations is in the Jupyter Notebook “Generate_Animation.ipynb” in the “src_nbody” directory. “Endian-ness”, inputs, and outputs can be modified therein as necessary.

Batch scripts and job commands for both Midway and BG/Q can be found in the “src_batches” directory.

Code Design

The parallel code builds from the serial code provided. It begins by determining the optimal layout of ranks in a 1d pipeline. From there, the lead rank writes to file the header containing the number of bodies and number of iterations. At present, an assertion enforces that the total number of bodies is divisible by the number of ranks. In a future version a more robust scheme —either adding more bodies for even division over ranks or supporting uneven work sizes— could be implemented. After each rank allocates memory for their own bodies, they randomly generate the starting conditions (threading with OpenMP if enabled). Ranks allocate additional buffers for writing and shipping data before the main loop commences.

At each iteration, each rank first packs up the necessary data to ship and write. Next, the ranks collectively write their position data to file, calculating offsets from the timestep, number of bodies, and their own rank. Each rank calculates the internal forces between its own particles before beginning the pipeline. Once the pipeline commences, position and mass data from each rank are ferried around the pipe, stopping at every rank to allow that rank to calculate the forces its own particles experience from the foreign particles. Upon completing the pipeline, each rank updates the velocities and positions of its own bodies, and the next iteration commences.

All OpenMP functionality is IFDEF’d such that it will be stripped out of the program at compile time if OpenMP is not enabled. This avoids creating added overhead for singly-threaded, pure MPI executables.

Certain optimizations could be made to improve the performance of this code by eliminating redundant buffers and data copies. That said, the additional resources contribute significantly to the simplicity of this code and help to enhance its readability.

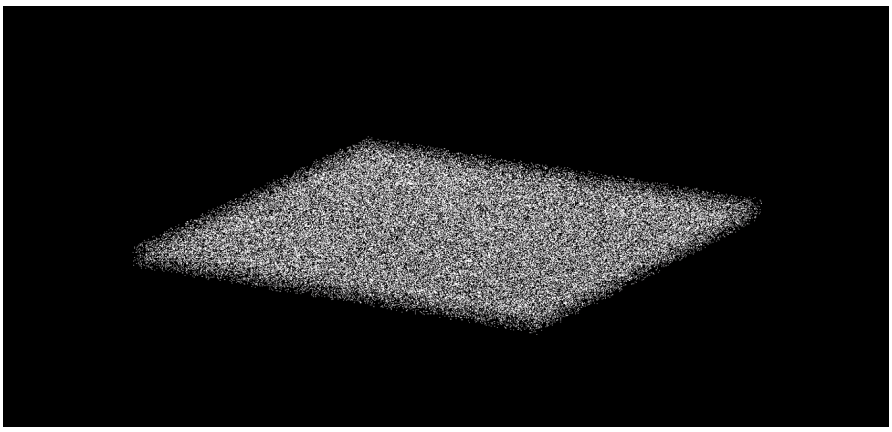
Initial Conditions

The starting position of each body is chosen from a square domain on the X and Y axes centered at the origin. The starting value on the Z axis is chosen from a domain one tenth the size of that of the other two. Given a random distribution, the initial positions roughly describe a flattish square, as seen in the visualization below of just over 100,000 bodies. The appearance of a parallelogram results from the angle of view.

Starting velocities are linearly proportional to the distance from the origin in each dimension. In other words, the initial velocity of every particle points away from the origin, increasing in magnitude as the distance from the origin increases. As recommended, the mass of the total

system is kept on the same order of magnitude by having randomly generated masses scaled down by the total number of bodies.

An animation of 1,000 bodies over 1,000 time steps of magnitude 0.2 each can be found here.



Correctness Testing

Simulations were run in serial and in parallel with the following parameters:

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

To ensure OpenMP parallelism did not result in any data divergence, the parallel version was run with two OpenMP threads.

The comparison script and generated data can be found in the directory “src_compare”. The script, “compare.py”, can be run with Python3 and numpy installed and should be run as follows:

python3 compare.py <nbody file 1> <nbody file 2>

With assertion checks, the script will make sure that the headers are identical and every position of every body at every timestep differs by less than 1e-10. Assuming the data meet those conditions, a success message will be returned to the user.

For the generated data, the following was run…

python3 compare.py cmp_serial.dat cmp_parallel.dat

…and was met with success, indicating that we were able to generate approximately the same results between serial and parallel implementations for the above simulation parameters. Divergence remains present, however, as can easily be seen by running:

diff cmp_serial.dat cmp_parallel.dat

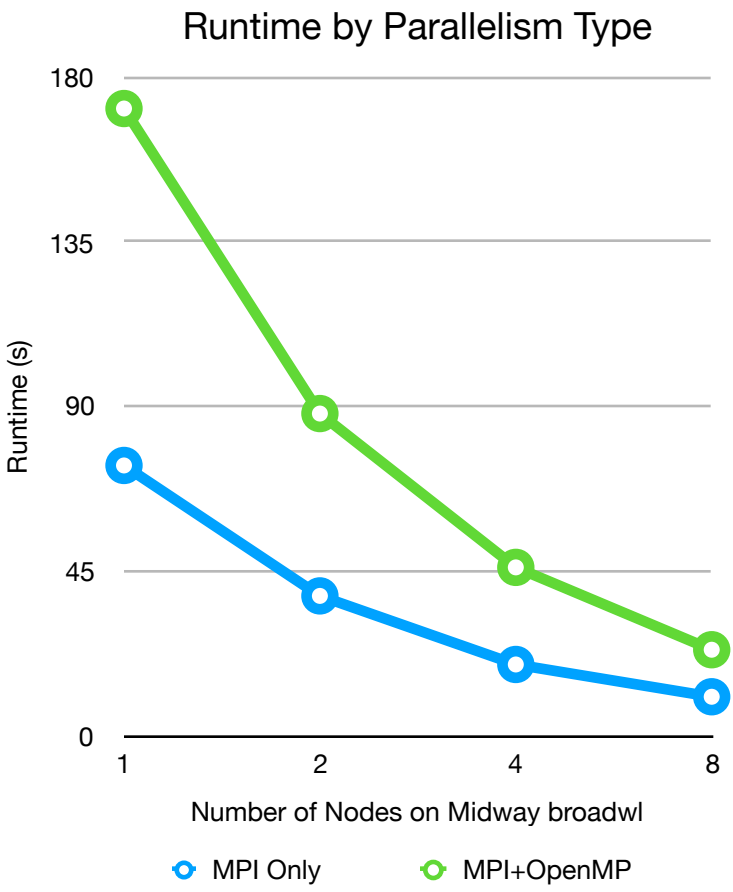
Note: The error checking script does not account for “endian-ness”, so the data should match that of the machine on which it is tested. The script could be modified in a later version to implement handling of “endian-ness”.

Strong Scaling on Midway

A strong scaling study was performed on the Midway broadwl partition testing both pure MPI and hybrid MPI/OpenMPI parallelism with the following parameters on 1, 2, 4, and 8 nodes:

Number of Timesteps = 10
Number of Bodies = 100,352

For pure MPI, the number of ranks was equivalent to 28 times the number of nodes. For hybrid runs, the number of ranks was equivalent to the number of nodes with 28 threads on each rank.



Strong Scaling Comparison

Nodes	MPI Only Runtime (s)	MPI+OpenMP Runtime (s)
1	73.90	171.50
2	38.23	88.09
4	19.48	46.03
8	10.65	23.51

In all trials on Midway, the pure MPI implementation outperformed hybrid parallelism by approximately a factor of two as seen in the chart and data above. These results would seem to indicate that splitting the problem onto more ranks more than accounts for the additional overhead incurred by a larger pipe and certain non-threaded calculations. Though I did not anticipate this result, the data lead me to conclude, as of this point, that pure MPI is the superior parallelism method for this N-Body problem on Midway.

Blue Gene/Q Performance Analysis

The table below shows the measured runtime values of the pure MPI executable vs hybrid parallelism. Both trials were run a second time with all IO functionality stripped in order to calculate the percentage of time being spent on IO. The following parameters were used for the tests:

Number of Vesta nodes = 32
Number of Timesteps = 10
Number of Bodies = 100,352

For hybrid parallelism, each node had a single rank with 64 threads. For pure MPI, each node had 64 ranks.

Vesta Mode Trials

Run Type	Ranks	Threads per Rank	Runtime (s)	Optimized Runtime (s)
Pure MPI	2048	1	25.05	13.17
Pure MPI (compute only)	2048	1	24.51	12.43
Pure MPI IO Percentage			2.156%	5.619%
Hybrid	32	64	26.95	13.12
Hybrid (compute only)	32	64	26.56	12.99
Hybrid IO Percentage			1.447%	0.991%

Hybrid parallelism performed much more comparably to pure MPI on Vesta as compared to the trials performed on Midway. I speculate that the performance of the hybrid continues to improve relative to pure MPI as the overhead of the pipeline continues to increase. Pure MPI runs on 8 nodes on Midway only used 224 ranks whereas on Vesta they used 2,048 ranks. Still, pure MPI generally appears to be outperforming. That said, the hybrid version of the program outperformed the pure MPI version using optimized compilation. As a result, I elected to investigate further and perform the strong scaling study on both versions of the program

IO typically accounted for less than 3% of the total runtime of the program and in the worst case still accounted for less than 6%, soundly meeting the bar for our desired IO performance.

The table below shows the data used to calculate performance efficiency in terms of power by calculating the interactions per second per Watt. As the last column illustrates, the overall

efficiency of Blue Gene/Q in terms of interactions per second per Watt outperforms Midway despite a slightly higher runtime. Given that computing resources can be as expensive to power as they are to build, it makes sense that efficiency would be emphasized on a world class machine like BG/Q.

Machine	Nodes	Watts per Node	Total Watts	Interactions per Timestep	Timesteps	Measured Runtime (s)	Interactions per Second	Interactions per Second per Watt
Midway broadwl	8	500	4000	1.01E+10	10	10.65	9.46E+09	2.36E+06
Blue Gene/Q Vesta	32	80	2560	1.01E+10	10	13.12	7.68E+09	3.00E+06

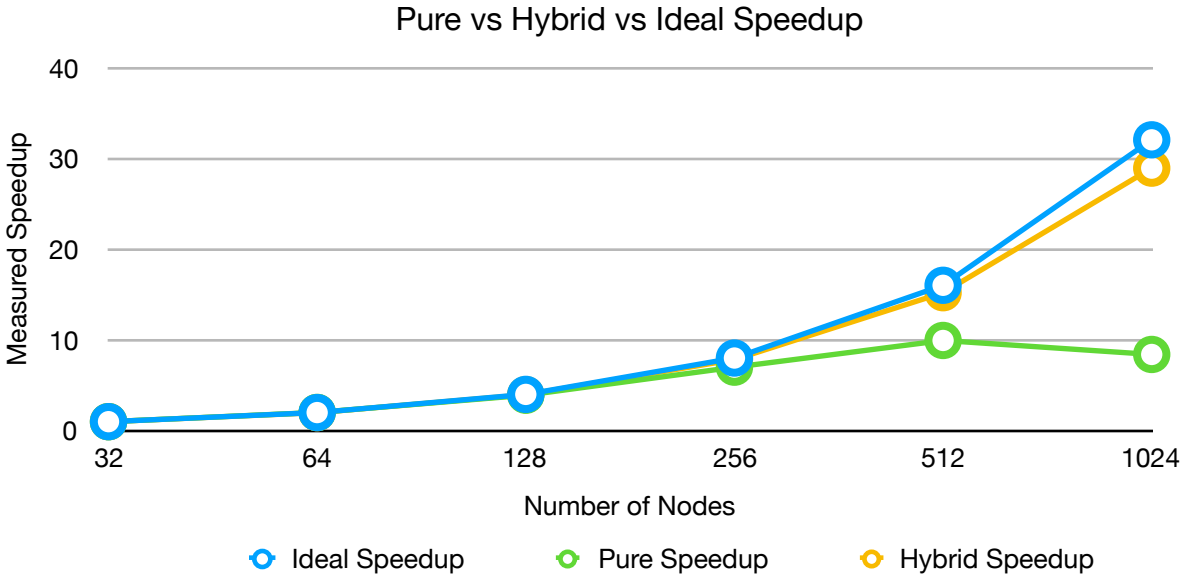
Strong Scaling

Given the comparable performance of both modes during performance analysis, I ran the strong scaling study on both. Those data are presented in the table and plot below. Trials were performed with the following parameters:

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

Vesta Strong Scaling Study

Nodes	Pure Runtime (s)	Hybrid Runtime (s)	Ideal Speedup	Pure Speedup	Hybrid Speedup
32	335.7	355.3	1	1.00	1.00
64	166.6	178.4	2	2.02	1.99
128	85.9	89.5	4	3.91	3.97
256	48.0	45.4	8	6.99	7.83
512	33.8	23.4	16	9.93	15.18
1024	40.0	12.3	32	8.39	28.89



As I suspected might be the case, the hybrid version began to outperform the pure MPI version at very high node counts. In fact, hybrid speedups were just short of ideal. In stark contrast, I measured a surprisingly decline in performance of the pure MPI program scaling up from 512 to 1,024 nodes. While this results merits further investigation, I speculate that the cost of the pipeline and the expense of running calculations in serial on each rank overcame the benefits of smaller work sizes per rank.

Production Simulation

The production simulation consists of 786,432 particles over 800 timesteps. Given that it took over six hours in the queue to receive compute resources for each of the strong scaling trials on 1,024 nodes, I instead ran the production simulation over 512 nodes. Resources were quickly allocated for the trial, and the test ran in 4,150 seconds, just over 69 minutes. As such, I would expect the same trial run over 1,024 nodes to complete in slightly over half that much time—around 35 minutes—which places it squarely within the performance expectations for the large scale trial. Summary statistics for the run have been calculated in the table below.

Nodes	512
Watts per Node	80
Total Watts	40,960
Bodies	786,432
Interactions per Timestep	6.185E+11
Timesteps	800
Total Interactions	4.948E+14
Measured Runtime (s)	4,150
Interactions per Second	1.192E+11
Interactions per Second per Watt	2.911E+06

The animation of the large-scale production run can be seen here.

One of my favorite animations created in working on this project was generated on Midway with just over 100,000 bodies on 800 timesteps. That mid scale animation can be seen here.