Tyler Petrochko

CPSC 525 Assignment #4

Included Files

I've included the following files in my submission

- serial.c
- parallel.c
- serial_data1.txt
- serial_data2.txt
- serial_data3.txt
- serial_data4.txt
- parallel_data1.txt
- parallel_data2.txt
- parallel_data3.txt
- parallel_data4.txt
- Makefile
- run.sh
- timing.h

The first two files are the source code for my serial and parallel implementation of the assignment respectively. They are both fully functional in that they correctly calculate generate the centers of mass and average velocities when tested against the test data provided in the data/ directory.

The next eight files include the output generated when the serial and parallel implementations are run against the four “actual” data files. The file serial_data4.txt is intentionally blank; as discussed later, the simulation was unable to complete within a reasonable time frame.

The Makefile provided can be used to generate the serial and parallel binaries with the command (see the last section for modules that must be loaded in the development environment).

```
$ make all
```

or simply

```
$ make
```

The file run.sh can be used to generate the .txt output files when run according to the specifications from the assignment, i.e. 8 MPI processes, with 2 processes on each of 4 nodes. To do so, I used the following bash command after SSH-ing into the Omega cluster:

```
$ qsub -I -l procs=8,tpn=2,mem=34gb,walltime=15:00 -q fas_devel
```

I included the file timing.h because my Makefile relied on it, although this could easily be changed.

Performance

The performance for my parallel implementation is as following

Time Steps	Bodies	Execution Time (s)
1024	800	3.754109
1024	1600	15.653886
1024	3200	67.525101
1024	6400	251.359509

While the performance for my serial implementation is as follows

Time Steps	Bodies	Execution Time (s)
1024	800	14.345081

1024	1600	59.697339
1024	3200	248.527077
1024	6400	**??**

The serial program was unable to complete the 6400 body simulation due to time constraints, i.e. it still hadn't completed a single time step after 15 minutes. This suggests that, while the execution time seems roughly linear with a smaller number of bodies, the execution time spikes beyond 3200 bodies. While the parallel implementation performs better in every case, its main advantage is that is still manages to scale execution time linearly at 6400 bodies.

In each parallel_data<x>.txt I've provided a load balance printout at each "checkpoint," but to avoid taking up too much space I'll only include a sample load balance from the 6400 bodies simulation

Time Step	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
0	828	790	814	769	802	808	780	809
128	834	889	754	802	775	775	808	763
256	400	1219	398	1220	400	1169	425	1169
384	294	1302	275	1305	275	1339	308	1302
512	224	1345	208	1369	189	1437	247	1381
640	192	1381	176	1323	159	1497	190	1482
768	169	1399	169	1313	162	1501	169	1518
896	180	1416	156	1293	161	1496	184	1514
1024	196	1389	179	1245	169	1495	201	1526

As illustrated in this breakdown, the load balance quickly becomes an issue. By the 256th time step, the third quadrant contains only 398 bodies while the first quadrant contains 1219. At this point, the quadrants that "own" more bodies (Q1, Q3, Q5, and Q7) are causing a performance bottleneck by forcing the other quadrants to idle while they finish their calculations. To fix this, at the end of each step, the 3D Cartesian plane could be shifted so that the origin is at the center

of mass. This would likely roughly rebalance the bodies between quadrants, allowing the load to stay more even.

Development Environment

To develop, I used the following bash commands to load all required modules. The run.sh script specified handles loading all required modules to run the simulations.

```
$ module load Langs/Intel/15; module load Langs/Intel/15 MPI/OpenMPI/1.8.6-intel15
```