Compiling and Running the Program

The submission folder contains 3 sub-folders, 1 pdf file, and 1 env.output file. The table below explains the usage of each sub-folder.

Folders/Files	Comments
data/	This folder stores the raw data given for this assignment
parallel/	Parallel folder holds all the source code, scripts, and related results for task 2 of the assignment; task2.c is included in this folder.
serial/	Serial folder holds all the given code and results for the assignment
env.out	Environment output for my development environment
report.pdf	The writeup of this assignment

Table 1, Top-tier directory

My work of this assignment is in the *parallel* folder, in which one can find *task2.c, Assignment3BuildAll.sh, Makefile, and Assignment3RunAll.sh.* The *data* folder is simply a container of the datasets, and the *serial* folder stores all the source code fetched from Dr.Sherman's course repository.

One can execute the following steps to **compile** the parallel program.

1. *Bash$ cd parallel*
2. *Bash$ ./Assignment3BuildAll.sh*

Up to this point, one should be able to see a binary file, *task2*, generated.

To **run** the program, one can execute:

1. *Bash$ qsub Assignment3RunAll.sh*

The command above submits a job to *Torque* and run the parallel program using the *actualdata1* dataset found in the data folder. In order to run the program with a different dataset, one can open *Assignment3RunAll.sh* (figure 1) and change every occurrence of *actualdata1* to another dataset name. Once done changing, one can run the same *qsub* commands as shown above.

```
1  #!/bin/bash
2  #PBS -l procs=8,tpn=4,mem=46gb,walltime=30:00
3  #PBS -q cpsc424
4  #PBS -j oe
5
6  cd $PBS_O_WORKDIR
7
8  module load Langs/Intel/14
9
10 time mpiexec -n 8 task2 < ../data/actualdata1 > ./results/actualdata1_c.out
```

Figure 1, Assignment3Runll.sh contents

To see the results, one can do while in the *parallel* directory:

1. *Bash$ cd results*
2. *Bash$ vi actualdata1_c.out*

All pre-executed results are saved in the same *results* folder. More details on the contents of the *parallel* folder can be found in the parallel program section of this report.

Serial Program

Before looking into the parallel program (the program of my work), I want to first give a quick guide to the serial program of the assignment. All source files of the serial program are taken from Dr. Sherman's course repository and stored in serial folder. The files and their usages are presented in the table below (table 2).

Folders/Files	Comments
data/	This folder stores the raw data given for this assignment
results/	This folder stores the results (center of mass and average velocity) of all 4 actual datasets
serial.c	Serial source code provided by Dr. Sherman
Makefile	Makefile provided by Dr. Sherman for the serial program
***.o**	Object files after compilation

Table 2, serial folder

All the code in this folder is kept same as Dr. Sherman's source code. I simply ran the program for 4 actual datasets. The performance results for the each data set are shown below.

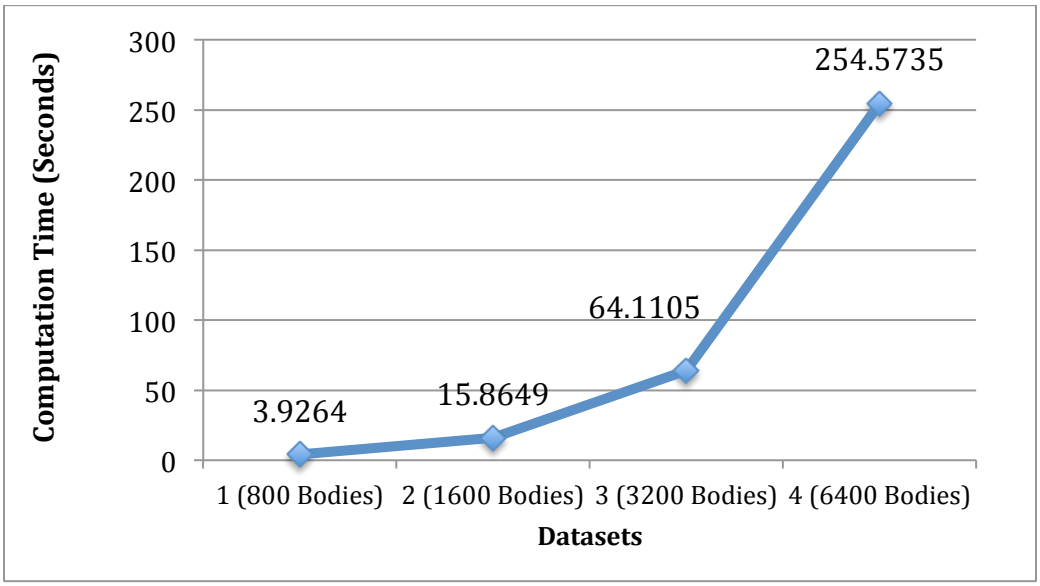

Figure 2, serial performance

From figure 2, we can see that the computation time increases exponentially while the problem size is only doubling on each data set.

I have also been using the results of each dataset to validate the correctness of my parallel program. More details will be provided in the parallel program section. Since serial program does not have any communication overheads, the total run time can be approximated as computation time.

Parallel Program

The parallel program uses only collective MPI operations for communication. All source files, execution results, and compilation/execution scripts are stored in this folder. The description of each is shown as table 3 below.

Folders/Files	Comments
Assignment3BuildAll.sh	The compilation script for task 2, parallel program.
Assignment3RunAll.sh	The Torque submission script for task 2, parallel program.
Makefile	Makefile for compiling task2 executable
task2.c	Program entry of the parallel program. This is where main() resides.
calc.c	Source code contains all calculation related functions
calc.h	Header file for calc.c
comm.c	Source code contains all communication (MPI) related functions, including buffer management functions.
comm.h	Header file for comm.c
global.h	Source code holds all globally defined variables, buffers, and object.
results/	This folder stores all the execution results for the datasets.
***.o**	Object file generated by compilation

Table 3, parallel directory

Looking inside *results* folder, all the execution results are named *actualdataX_c.out,* where X is the numbering of each dataset. I have printed out the center of mass and average velocity in the same way as the serial program for verification purpose. The results have shown that all the calculation is correct and nearly identical to the numerical results from the serial program. I have also printed out the computation time of each process in the files.

The computation time includes the estimation of distance unit, force, acceleration, and velocity calculation, and the final calculation of center of mass as well as the average velocity. The table below presents a numeric view of the computation time (table 4). I avoid presenting a graphical chart since chart is less informative for such large number of data points.

process\Dataset	1(800 Bodies)	2(1600 Bodies)	3(3200 Bodies)	4(6400 Bodies)
0	4.60E-05	5.20E-05	5.29E-05	7.20E-05
1	1.69E-05	2.41E-05	2.69E-05	3.10E-05
2	2.60E-05	3.51E-05	3.72E-05	6.20E-05
3	1.69E-05	2.41E-05	2.60E-05	3.10E-05
4	3.51E-05	4.10E-05	4.10E-05	5.60E-05
5	1.81E-05	2.69E-05	2.69E-05	5.20E-05
6	2.81E-05	3.48E-05	3.60E-05	5.39E-05

7	1.81E-05	2.69E-05	2.69E-05	3.20E-05
Maximum	**4.60E-05**	**5.20E-05**	**5.29E-05**	**7.20E-05**

Table 4, computation time on parallel processes (seconds)/Load balance

From table 4, I can see that the pure computation overhead in all cases in all processes is at microsecond scale. While considering the load balance factor, the maximum computation time spent on any of the datasets is still in microsecond scale. Therefore, I conclude that by splitting the computation into 5 concurrent octants, I can largely reduce the calculation overheads to bare minimum. The dominant factor of the execution time is the communication time caused by MPI operations.

In the *results* folder, one can also find 4 files named *totalRunTimeX.out,* where X denotes the numbering of the datasets. For example, the total run time for *actualdata1* is recorded in file, *totalRunTime1.out.* Since the computation time of the serial program can be approximated as total run time, figure 3 below compares the total run time of the parallel program with that of serial program.

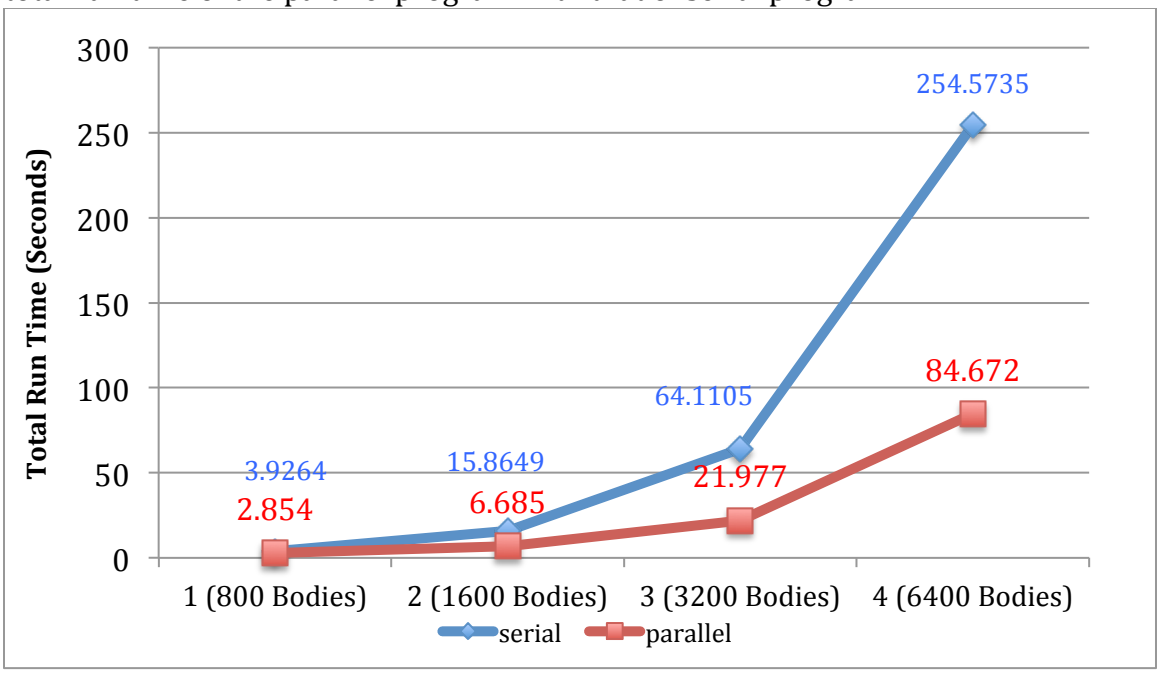

Figure 3, Total Run Time serial vs parallel

From figure 3, the red line and value are data from parallel program whereas the blue line and value are for serial program. From the figure, I can see 2 facts: 1. The total run time is dominated by communication time, given the computation time is in microsecond scale (table 4); 2. Parallelized program can largely shorten the total run time. Though the parallelized run time is much shorter than that of serial program. It is worth noting that the time is more than doubled while I only 2-fold the number of bodies, and the larger problem size yields higher reduction in total run time; therefore, I consider the parallel program not only improves the execution time but also the scalability of the program. In this assignment, I cannot determine if

the program is strongly or weakly scalable since I do not vary the total number of cores used in the program.