Assignment Report

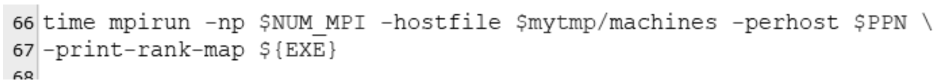
Task

There is a serial code, which is used to calculate the movements and interactions of some particles, needed to be parallelized. 5000 bodies of particles are required to update the velocities and positions for 100 timestep due to the interaction forces.

Discussion of parallelization strategy

In this project, MPI was used to place each body and the interaction forces were calculated by using OpenMP.

Firstly, an MPI parallel region was initialized after definition of some values. Then, it is necessary to check if the amount of "BODIES" could be divisible by number of processes*threads in use. After initialized the original positions of the particles, MPI_Bcast method was used to broadcast some data to each process. In the timestep loop, there were two MPI_Scatter methods used to send data original velocities to other process from root process. Moreover, an OpenMP region was setup to calculate the interaction forces and velocities when a distance existed between the two particles. Finally, the updated velocities were gathered to all the processes by MPI_Allgather method. According to these data, the application completed the update of particles' positions and speeds. In order to get the time of parallelism code operation, a time measure command in "flexi-submitHybrid.sh" was used to measure (see Figure.1).

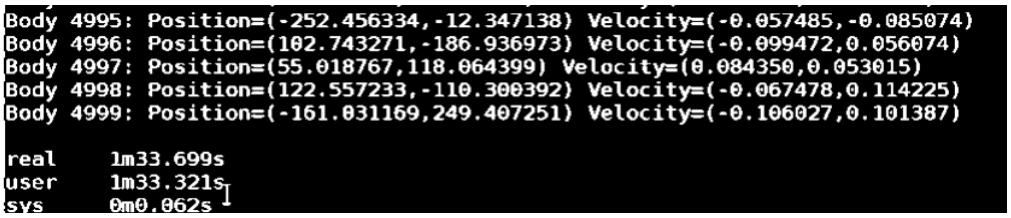


```
time mpirun -np $NUM_MPI -hostfile $mytmp/machines -perhost $PPN \
-print-rank-map ${EXE}
```

Figure 1. Command to measure time of parallelism code

Compared to the provided serial version

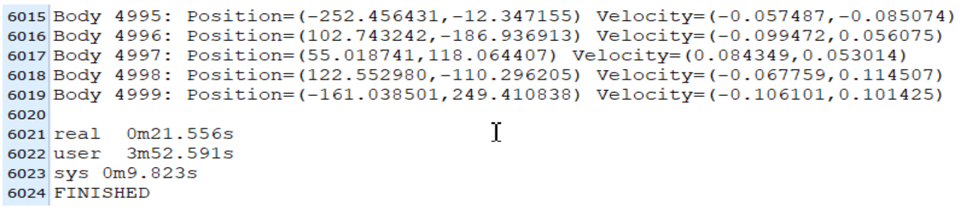
This part is going to compare the accuracy and performance of serial code and parallelism code. After applying and timing the sample code, the outputs were obtained. According the results(Figure 2), the real time of serial code is 1m34s.



```
Body 4995: Position=(-252.456334,-12.347138) Velocity=(-0.057485,-0.085074)
Body 4996: Position=(102.743271,-186.936973) Velocity=(-0.099472,0.056074)
Body 4997: Position=(55.018767,118.064399) Velocity=(0.084350,0.053015)
Body 4998: Position=(122.557233,-110.300392) Velocity=(-0.067478,0.114225)
Body 4999: Position=(-161.031169,249.407251) Velocity=(-0.106027,0.101387)

real    1m33.699s
user    1m33.321s
sys     0m0.062s
```

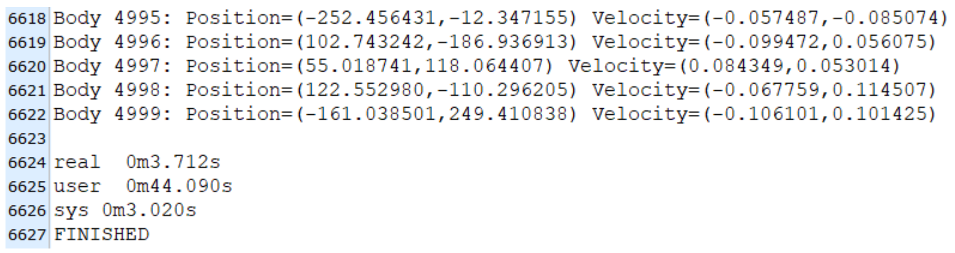
Figure 2. Result and time of serial code

As for parallelism code, the positions and velocities of particles are the same of the output of serial code, while different MPI process or OpenMP threads would influence performance of the code. Thus, in order to get the optimum granularity in single node, it is required to change the amounts of processes or threads several times.



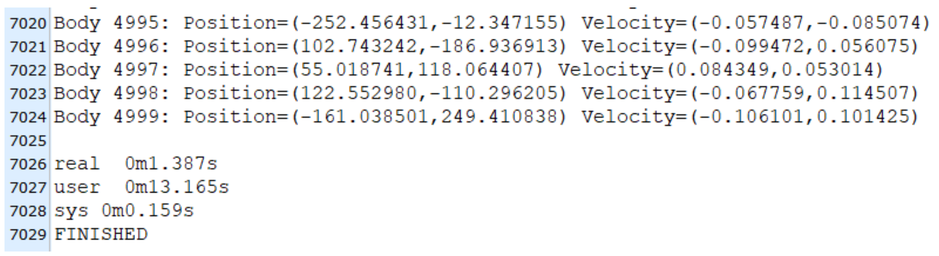
```
Body 4995: Position=(-252.456431,-12.347155) Velocity=(-0.057487,-0.085074)
Body 4996: Position=(102.743242,-186.936913) Velocity=(-0.099472,0.056075)
Body 4997: Position=(55.018741,118.064407) Velocity=(0.084349,0.053014)
Body 4998: Position=(122.552980,-110.296205) Velocity=(-0.067759,0.114507)
Body 4999: Position=(-161.038501,249.410838) Velocity=(-0.106101,0.101425)

real  0m21.556s
user  3m52.591s
sys 0m9.823s
FINISHED
```

Figure 3. 5 MPI processes and 3 OpenMP threads in a single node with 16 cores



```
Body 4995: Position=(-252.456431,-12.347155) Velocity=(-0.057487,-0.085074)
Body 4996: Position=(102.743242,-186.936913) Velocity=(-0.099472,0.056075)
Body 4997: Position=(55.018741,118.064407) Velocity=(0.084349,0.053014)
Body 4998: Position=(122.552980,-110.296205) Velocity=(-0.067759,0.114507)
Body 4999: Position=(-161.038501,249.410838) Velocity=(-0.106101,0.101425)

real  0m3.712s
user  0m44.090s
sys 0m3.020s
FINISHED
```

Figure 4. 8 MPI processes and 2 OpenMP threads in a single node with 16 cores



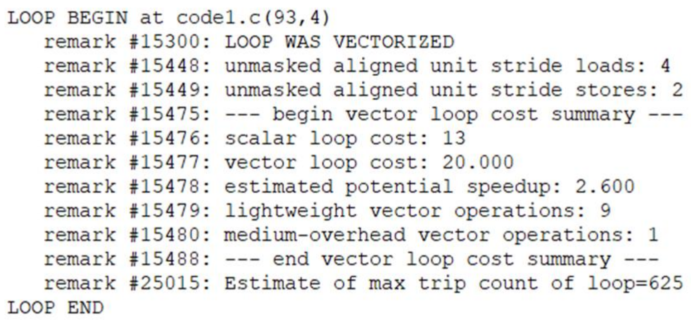
```
Body 4995: Position=(-252.456431,-12.347155) Velocity=(-0.057487,-0.085074)
Body 4996: Position=(102.743242,-186.936913) Velocity=(-0.099472,0.056075)
Body 4997: Position=(55.018741,118.064407) Velocity=(0.084349,0.053014)
Body 4998: Position=(122.552980,-110.296205) Velocity=(-0.067759,0.114507)
Body 4999: Position=(-161.038501,249.410838) Velocity=(-0.106101,0.101425)

real  0m1.387s
user  0m13.165s
sys 0m0.159s
FINISHED
```

Figure 5. 10 MPI processes and 1 OpenMP thread in a single node with 16 cores

According Figure2,3,4, when using 10 MPI processes and one OpenMP thread in one single node with 16 cores, the parallelism code got the optimum granularity. Maybe the dataset is small, OpenMP distribution may use more time. Thus, less OpenMP threads may cause less operation time.

Discussion of compiler report

After using the intel compiler with the -O2 optimization flag, some knowledge could be obtained from the compiler report. Figure 6 showed that how the compiler with -O2 optimize the code.



```
LOOP BEGIN at code1.c(93,4)
   remark #15300: LOOP WAS VECTORIZED
   remark #15448: unmasked aligned unit stride loads: 4
   remark #15449: unmasked aligned unit stride stores: 2
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 13
   remark #15477: vector loop cost: 20.000
   remark #15478: estimated potential speedup: 2.600
   remark #15479: lightweight vector operations: 9
   remark #15480: medium-overhead vector operations: 1
   remark #15488: --- end vector loop cost summary ---
   remark #25015: Estimate of max trip count of loop=625
LOOP END
```

Figure 6. Part of compiler report

According to the report, the compiler with -O2 optimizes more than which with -O1. The compiler does not perform loop unrolling or function inlining when specify the command with -O2, while this option increases both compilation time and the performance of the generated code.

Discussion on scaling code to model galaxy formation

When talking about how to implement the parallel code with much large-scale data, it is required more times firstly. Then, more MPI processes or more threads would also be used to compute. However, if object is the data about particles in the broad universe, a more high-performance computer would be needed to use.