4F03 Final Project Report

Group Members: Jiacheng Yang (yangj10 - 400005278), Ruitian Kang(kangrt - 1422650), Hongbing Jia (jiah10 - 4000161657, 6F03), Joel Straatman (straatjc - 001314676)

Abstract

This report will focus on the discussion of an implementation of N-body simulation using MPI library (and also OpenMP).

Part 1

Clarification: Due to the overwhelming demanding of computing resources of GPU8 at the time when the data is collected, the CPU utilization of our implementation becomes extremely unstable (sometimes around 20%-30%) and the time we calculated becomes pretty unreliable at the same time, so we uses processors from mpihost, gpu1, gpu2, gpu3 and gpu4 (mpihost slots=4; gpu1 slots=8; gpu2 slots=8; gpu3 slots=8; gpu4 slots=4) to conquer this problem. Meanwhile, the final time in our report still comes from the root processor on mpihost. Overall, We believe that due to the very good capability of gpu1, gpu2 and gpu3 (Intel Xeon), the speedup and efficiency of our program is more satisfying.

The following data runs in a pure MPI implementation with 10 * 10 number of subsets, 0.5 time interval, 2:1:1 light particles number : medium particle number : heavy particle number (But the final version of our code is MPI and oepnMP hybrid).

For 1 Node (Serialized Version):

Number of Particles	Timing	SpeedUp	Efficiency
2000	29.36	-	-
4000	118.84	-	-
8000	468.66	-	-
16000	1887.82	-	-

For 2 Nodes:

Number of Particles	Timing	SpeedUp	Efficiency
2000	22.07	1.33	0.665
4000	87.32	1.36	0.68
8000	349.05	1.34	0.67
16000	1433.42	1.31	0.655

For 4 Nodes:

Number of Particles	Timing	SpeedUp	Efficiency
2000	18.59	1.57	0.3925
4000	67.48	1.76	0.44
8000	269.71	1.73	0.43
16000	982.80	1.92	0.48

For 8 Nodes:

Number of Particles	Timing	SpeedUp	Efficiency
2000	7.01	4.18	0.52
4000	27.40	4.33	0.54
8000	112.21	4.17	0.52
16000	439.22	4.29	0.53

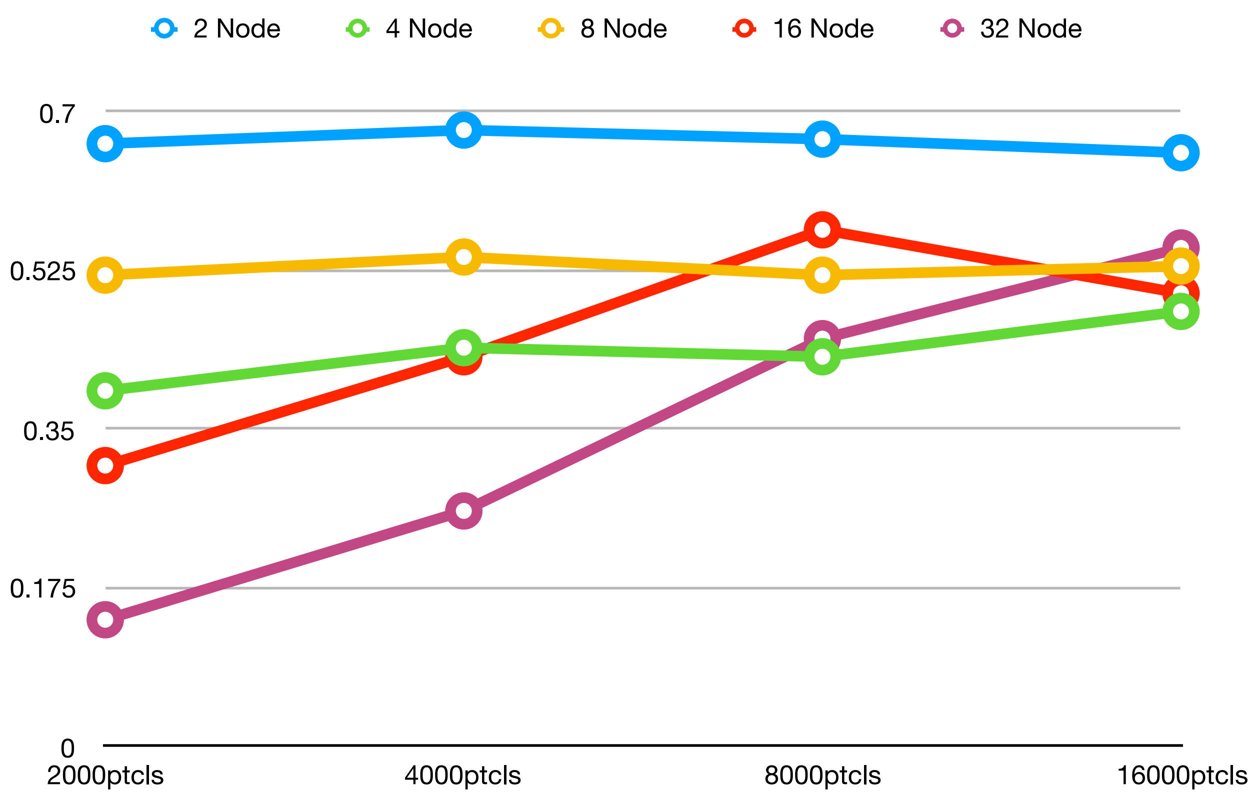
For 16 Nodes:

Number of Particles	Timing	SpeedUp	Efficiency
2000	5.84	5.02	0.31
4000	17.03	6.97	0.43
8000	51.06	9.17	0.57
16000	233.31	8.09	0.50

For 32 Nodes:

Number of Particles	Timing	SpeedUp	Efficiency
2000	6.26	4.69	0.14
4000	14.02	8.47	0.26
8000	32.48	14.42	0.45
16000	106.15	17.78	0.55

Efficiency Plot:



Part 2

The implementation we used is weakly scalable. Because when we increase the problem size and the number of nodes at the same time, for example, 2000 and 2 nodes, to 4000 and 4 nodes, to 8000 and 8 nodes and to 16000 and 16 nodes, the efficiency is roughly the same if excludes some external influences (my classmates' programs have to share some computing power and thus tank my cpu utilization at certain points). This makes perfect sense in this case. Because the parallel approach we used was dividing tasks among different processors and they should do same amount of same tasks at the same time. When weakly scale, each processor should still maintain the fact that they have good amount of work to do.

On the other hand, we think this implementation is not strongly scalable because when keep the size of problem same and increase the number of processors at the same, the efficiency sees an obvious and dramatic drop. Because when using more and more processors, the performance from starting time becomes more significant since the average execution of each processor is shortened. Besides, the cost of communication and the root processors' awakening other processors also becomes more significant.

Part 3

The plot of 16 nodes is supposed to follow the same pattern as the plot of 32 nodes because in these two cases, using so many processors to handle small size problems would be an overkill and see performance loss due to overhead. But in our graph, there is a drop of efficiency when dealing with 16000 particles. I think this is caused by the unstable environment the program was running in since the computing resources were fought over between different programs of all the users.

Part 4

We will use three sets of comparison to think of this question.

Only MPI for 1 processors:
the total time need for 2000 particles is 23.7875
the total time need for 4000 particles is 95.1483
the total time need for 8000 particles is 381.4249
the total time need for 16000 particles is 1524.5459

OpenMP + OpenMP for 1 processors 4 threads for each:
the total time need for 2000 particles is 23.581053
the total time need for 4000 particles is 93.682706
the total time need for 8000 particles is 372.974773
the total time need for 16000 particles is 1485.305249

Only MPI for 2 processors:
the total time need for 2000 particles is 17.907173
the total time need for 4000 particles is 71.604987
the total time need for 8000 particles is 302.856733

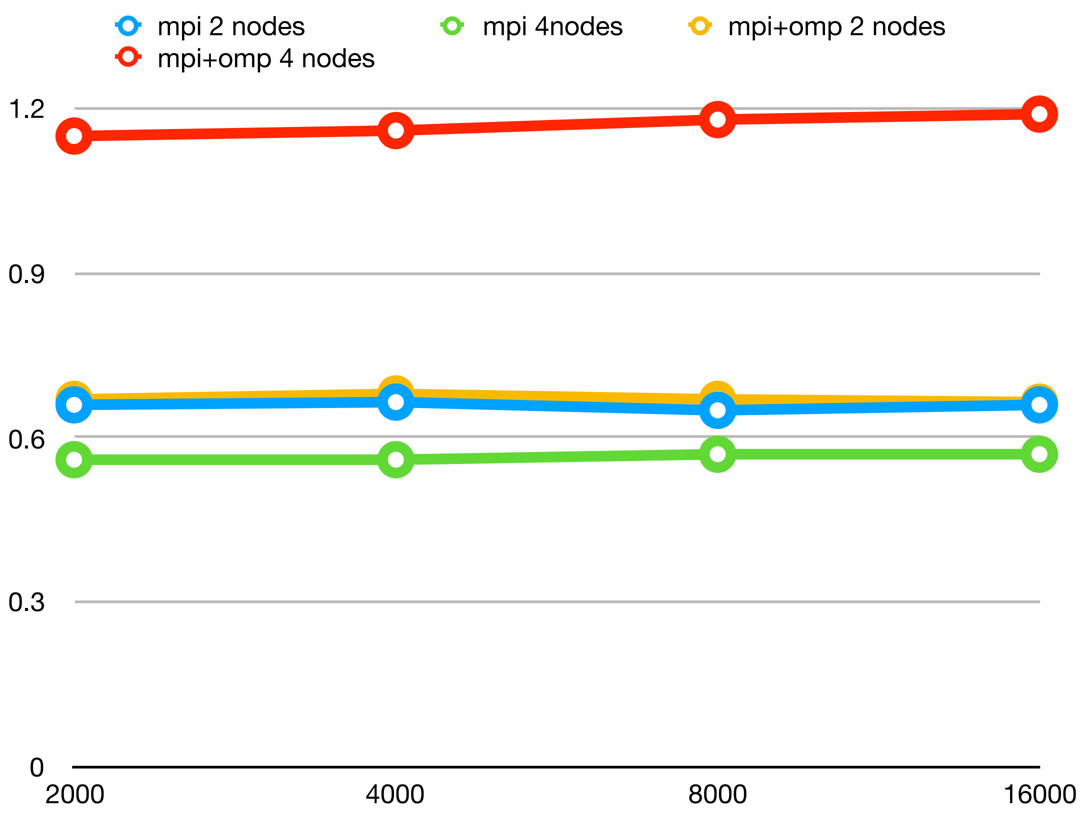
the total time need for 16000 particles is 1149.215183

OpenMP + OpenMP for 2 processors 4 threads for each:
the total time need for 2000 particles is 17.586538
the total time need for 4000 particles is 69.871908
the total time need for 8000 particles is 278.739792
the total time need for 16000 particles is 1113.814781

Only MPI for 4 processors:
the total time need for 2000 particles is 10.462528
the total time need for 4000 particles is 41.807931
the total time need for 8000 particles is 167.005208
the total time need for 16000 particles is 667.500826

OpenMP + OpenMP for 4 processors 4 threads for each:
the total time need for 2000 particles is 5.414594
the total time need for 4000 particles is 19.882772
the total time need for 8000 particles is 79.057298
the total time need for 16000 particles is 312.857476

Efficiency Plots Comparison (PER PROCESSOR):



Conclusion:

To conclude, overall, when combining OMP and MPI, the implementation would have a much higher CPU utilization and as the size of problem goes up, the performance difference would be higher and higher.

Performance Tunes we did:

We changed force calculation subroutines into inline functions.