CS 267 – Spring 2020 — HW2-1

Hanbing Zhan, Yichen Zhou

Contributions

In terms of the algorithm, Hanbing is responsible for the serial algorithm, including the implementation and test. He plotted the relation of simulation time vs particle size in log-log scale, which indicates the linear complexity of the serial implementation. Yichen is responsible for the parallel part. He also recorded and plotted the data and wrote report for the parallel part.

Overview

This report has two sections: serial and openmp. Serial algorithm explains the implementation and result of the serial code. Openmp algorithms will go over some of sychronization and methods choosed to get parallel execution.

Serial Algorithm

Initialization

Our structure of bin has three fields: idxes, bin_size , capacity. idxes stores the indexes of all particles in that bin. bin_size and capacity represents the current size and the full capacity of that bin. If the size of the bin exceeds its capacity, we would allocate more memory to the bin.

To make sure the average number of particles per bin is 3-4, we first define the number of bins to be the quarter of the number of particles. Then, the init_simulation function defines the height and width of each bin, as well as the number of bins in each row and column.

Specifically, we expect the number of bins in each row and column is the same, which is not true when the number of bins is square number. So we just use the closest factor of the number of bins as our number of bins in each row and column.

Apply Force and Move Particles

In the serial implementation, we scan the particles of all 9 neighbor bins for each particles and calculate their forces. Then, we update the position, speed of each particle in the move function. We choose to remain all the bins and check if any bin has not been in the same bin again by identifying its coordinate. The remove_particle and add_particle function do the remove and add operation for the particle and the corresponding bin.

Results

Our final serial optimization achieves $O(n)$ time. The results are as follows.

Particle Number	Simulation Time (s)	Slope (power of O)
1000	0.500267	1
2000	1.00253	1.002875
4000	2.28836	1.190669
8000	6.19136	1.4359423
16000	13.8177	1.158189
32000	28.7451	1.056798
64000	58.607	1.0277569
128000	121.69	1.0540657
256000	255.351	1.0692711
512000	556.256	1.12326728
1024000	1199.04	1.1080588

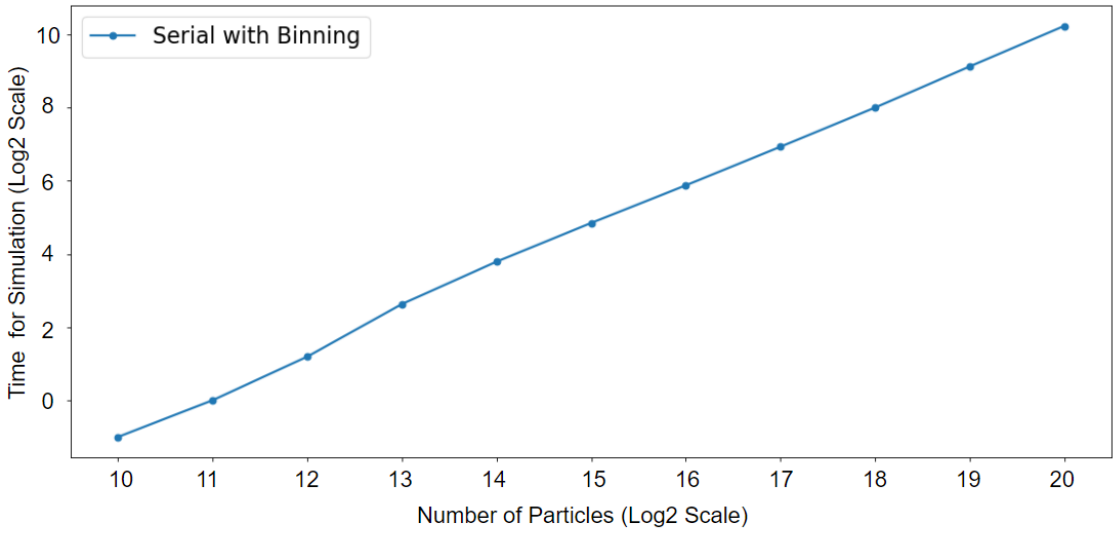

Figure 1: Comparison of simulation time vs particle size for serial implementation in Cori.(log-log scale)

Parallel Algorithm

Initialization

We added two more items to our bin data structure, nei_num and neighbors. We used the nei_num to store the number of valid neighboring bins this current bin has. We used the 1_D array: neighbors to store the index of the neighboring bins.

As in the serial part, we add particles to the bins. Then we initialize the locks for both the bins and particles. We have to ensure that when we apply force or remove or add particles later, there would be no race conditions and deadlocks.

Simulate One Step

As in the serial approach, we go through each bin, and for all the particles in its neighboring bins, we apply the force using this current particle and the other particle in the neighboring bins. We used pragma omp for for our loop worksharing. For each of the thread which is applying force between two particles, we use omp locks to prevent deadlocks.

Then, we update the locations of the particles. Again, we apply pragma omp for loop sharing and set and unset locks around our remove particles and add particles functions.

Results

Our final parallel optimization achieves $O(n)$ time. The results are as follows.

Particle Number	Simulation Time (s)
1000	6.59
2000	7.07
4000	6.61
8000	6.66
16000	6.74
32000	6.89
64000	7.39
128000	8.63
256000	10.67
512000	15.59
1024000	25.72

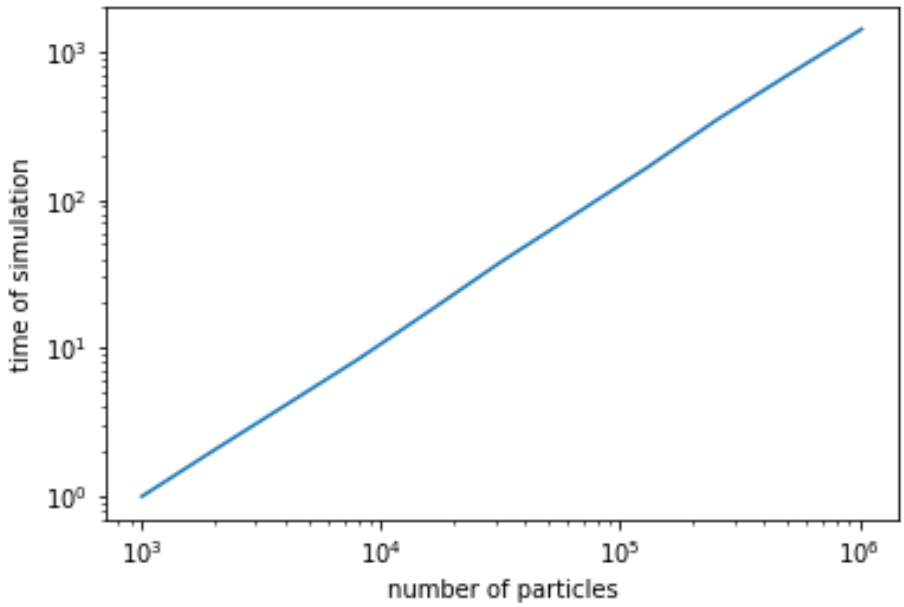

Figure 2: Comparison of simulation time vs particle size for parallel implementation in Cori.(log-log scale)

We tried out different number of processors with the input size of 1024000.From the image, we could see a inverse proportional pattern, so we could see that the simulation time is approximately p times speedup.

Particle Number	**Simulation Time (s)**
68	29.74
64	42.04
32	67.08
16	121.09
8	233.00
4	439.42

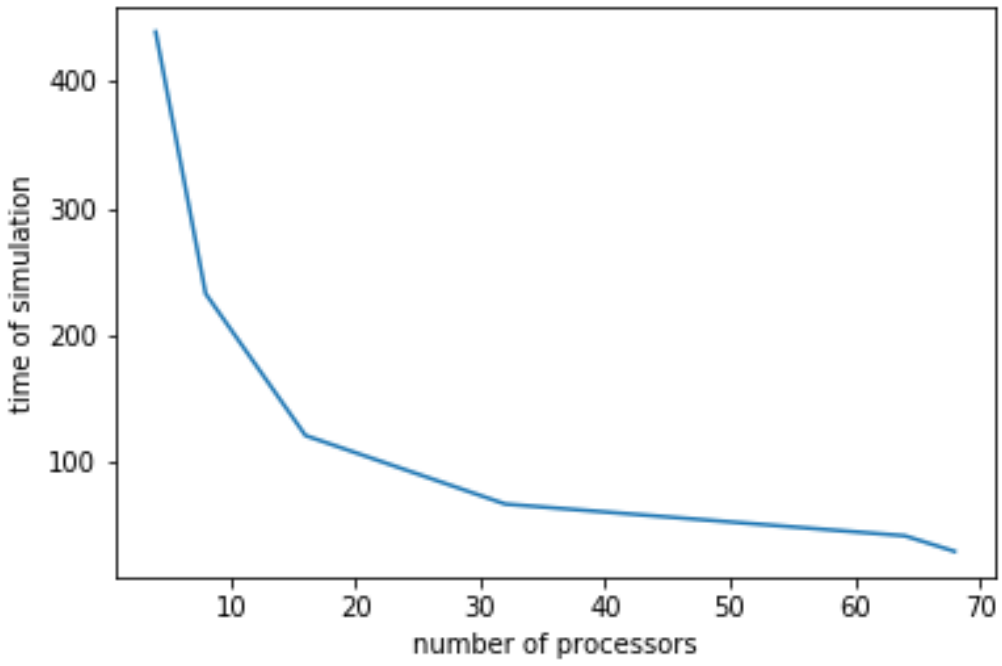

Figure 3: Comparison of simulation time vs number of processors for parallel implementation in Cori.

We tried some approaches to speed up the performance. One thing we tried was to add padding. We once made our bin_list to be a double pointer where each of the bin has 64 byte cache. However, the performance did not speed up. We also tried use Openmp during the initialization, however, this practice slows down the performance. I think in the future, we could optimize by trying to prevent using many-layers nested loops.

For the time, we see that for input size of 1000, we reduced the simulation time from 6.59 seconds to 5.78 seconds if we did not consider the time of setting and unsetting locks. From our observation, the sychronization time scales with number of processors.