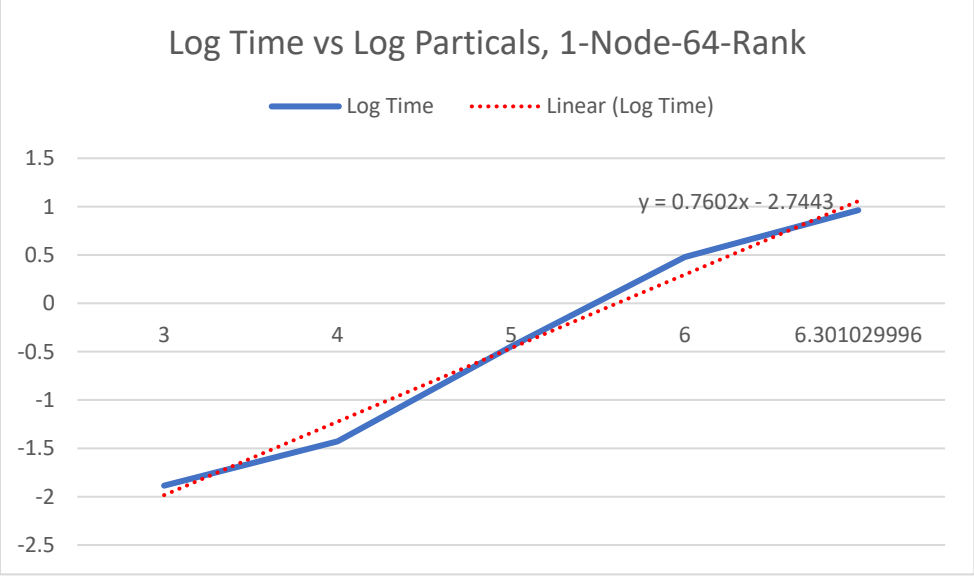
CS 5220 HW4 Report

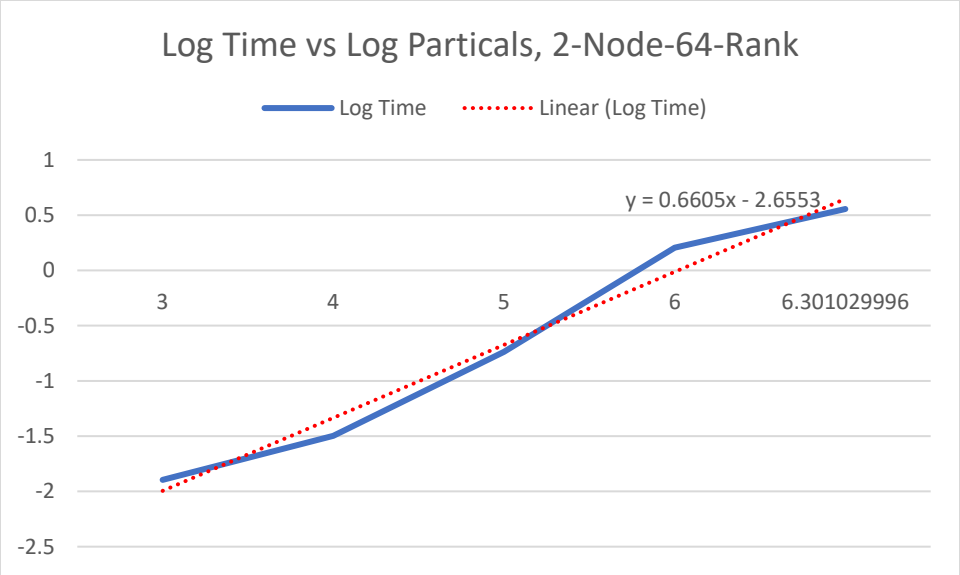
Name: Santiago Lai, Cornell id: zl345, NERSC username: Zhengnan

In this report, I will discuss each of the optimizations I attempted in serial and shared memory implementation, from data structures and synchronization I used to some design choices in my code. Lastly, I will describe my OpenMP code performance vs the idealized speedup, in addition to a breakdown of runtime into computation time/ synchronization time and communication time.

0. Plot in log-log scale for my parallel code:

The two plots for runtime of 1-node-64-rank parallel and 2-node-64-rank parallel code are shown below. In a log-log scale, both plots achieved a slope of near 1 (1.067, 0.8358, and 0.8655, respectively). The results show that my implementation successfully improved the performance from quadratic runtime to linear runtime.





1. Data structures

I followed the instructions in the handout to perform particle binning. In each rank, I used the std::vector to store the particles that belong to that rank. I also initialized an array of indices to keep track of the indices of particles that belong to each row. Finally, when re-distributing particles, I will use two vectors to receive message from rank above and from rank below, then perform re-binning accordingly.

2. Design choices

Similar to what I did in HW3, by experimentally setting the bin size to different multiples of cut-off distance, I found that the runtime is the best when the size is set to 1.2 times the cut off distance. Since the acceleration of a particle is actually not necessary, I removed all references to the accelerations and modified them to equivalent expressions.

Lastly, I used a row layout in my implementation so that each process is in charge of several consecutive rows. To address the ghost particle issue, I extended the row number of each process so that now it also contains the row above and below it. This is trading memory

over communication, as each process can now directly access the ghost particles without the need to communicate with other ranks.

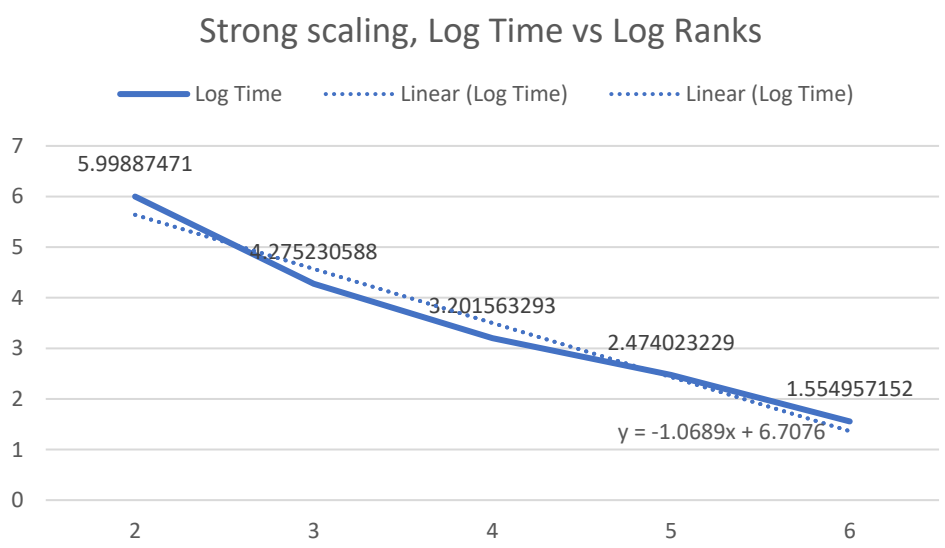
3. Communication in distributed memory implementation

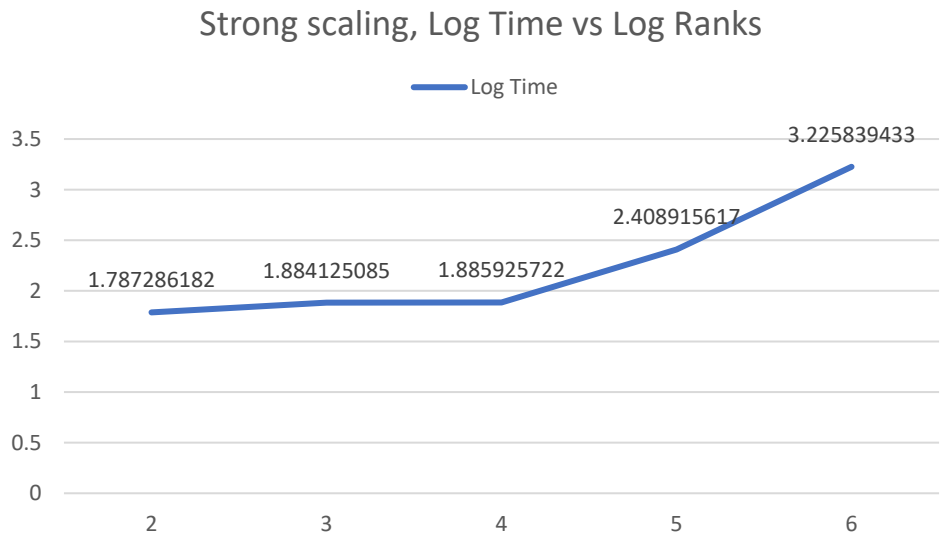
In my implementation, communication is applied mainly in two parts: one in particle redistribution and the other in gathering particles. In particle redistribution, under the assumption that particles will only travel to neighbor bins, I first let each process send a message (with different label) to the processes above and below it. After that I let each process receive the message from the processes above and below it, then perform re-binning to complete one simulation.

The second part is using gather function to get the information of particles at the end of each simulation. To do this I first gather the number of particles that belong to each processor, and compute displacements according to it. Finally, I used MPI_Gather with computed counts and displacements to collect all information back to 0^{th} rank.

4. Speedup by parallel implementation

Please see the plots for the speedup by my parallel code with strong scaling/ weak scaling. In strong scaling, I tested the performance with 4/8/16/32/64 ranks on 1 node with problem size fixed to be 1e6. In weak scaling, I tested the performance where problem size is increased proportionally to the number of ranks, and the problem size is 2e6 when there are 64 ranks. We see that in strong scaling, the slope is close to -1, which implies that we are close to the idealized p-times speedup. In weak scaling, the time are roughly the same for the first few numbers of ranks, and increases drastically after that. This means that the speedup is bad when number of ranks is high.

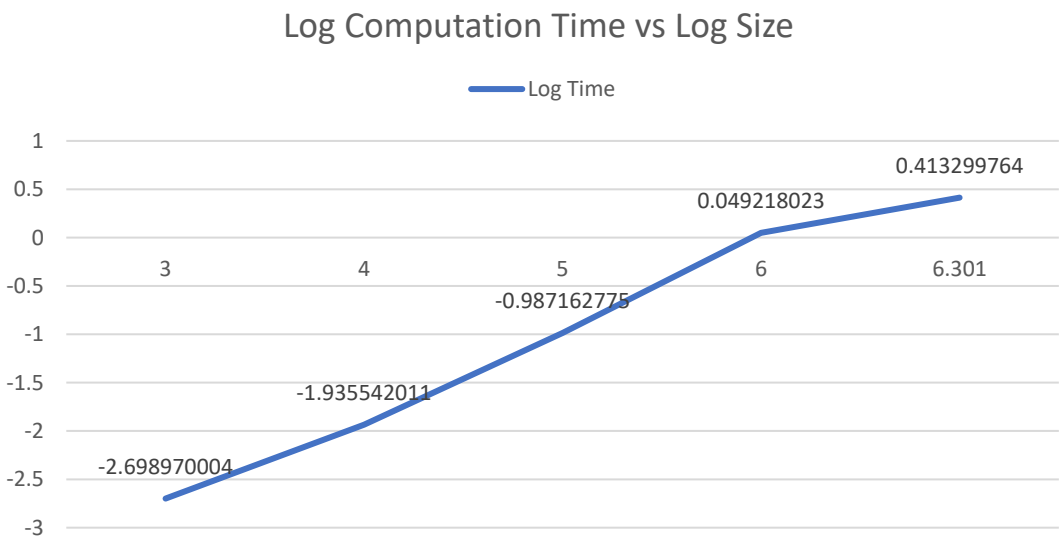


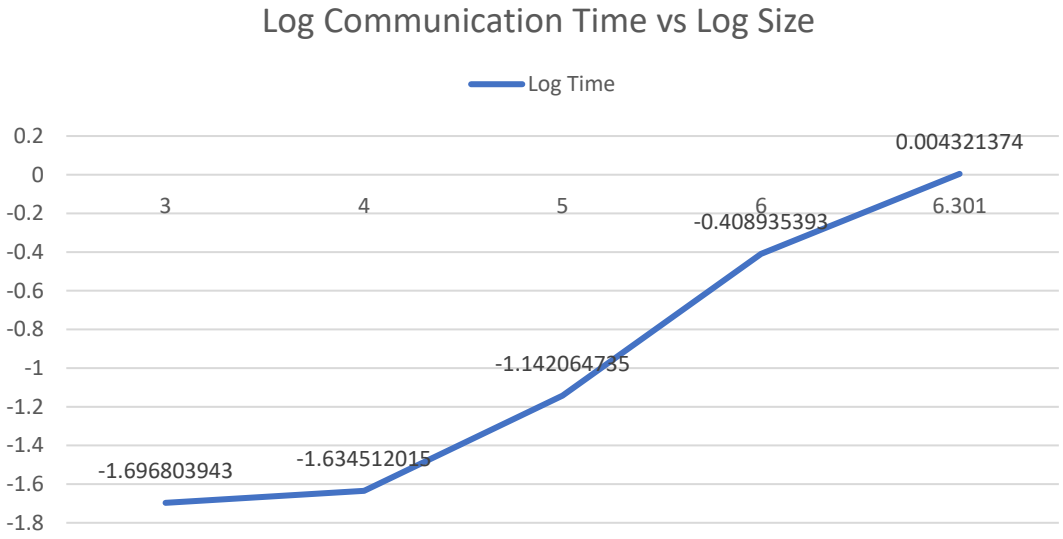


It is hard to do better because there's a significant portion of my code that is non-parallelizable, i.e., putting the particles into bins. It implies that my parallel efficiency will approach zero as the number of processors increases.

5. Runtime decomposition for parallel implementation

I plotted the runtime breakdown for 1e5 particles with 64 ranks on different numbers of particles.





We see that the both computation time and communication time scales approximately linearly when the problem size increases, which is consistent with our intuition.