Results Report

Background

In this project I have taken an already existing SPH code base and parallelized it with MPI and OMP. I added spatial binning and then made it compatible with MPI then added OMP. The MPI isn't done as efficiently as possible but since I needed to output the data sequentially each time frame it was better to do it this way. Each process has the entire array of bins and then it only works on the subset of indices (creating a pseudo sub-array) that I allocated to each process during initialization. Then after each calculation I take the "sub-arrays" of each process and join them then broadcast them to all processes. This was the best choice for me as an MPI novice because each process needed to calculate the interactions based on its neighbours. Since the calculations updated the values each timestep I thought it would be best to share the information very often to get the most accurate results.

Results

Table 1.1: Speed Up vs # of Processes/Threads

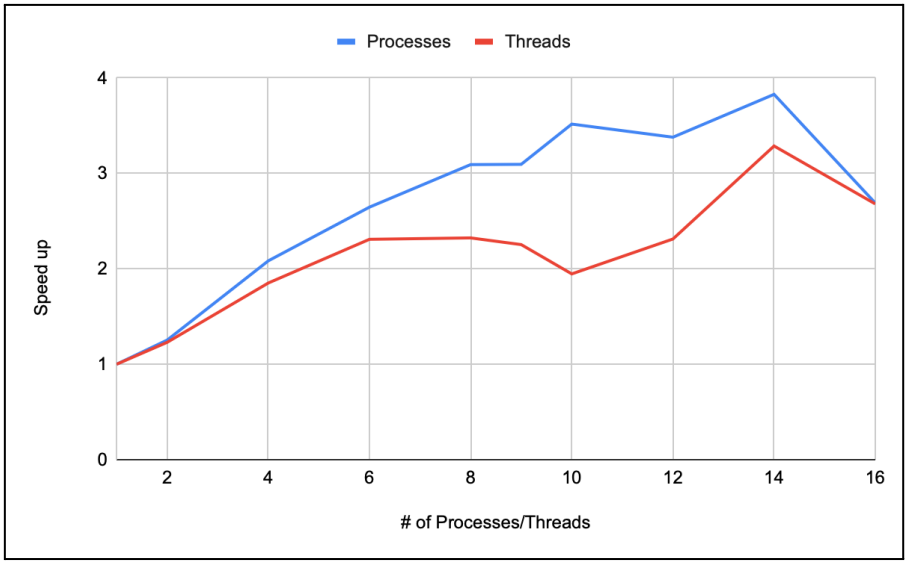


Table 1.2: Hybrid Speed Up vs # of Processes

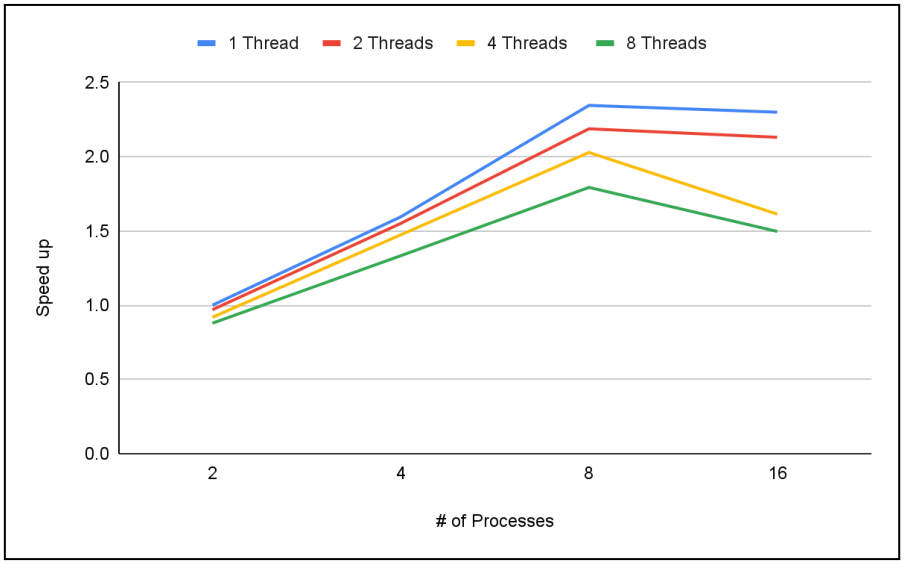
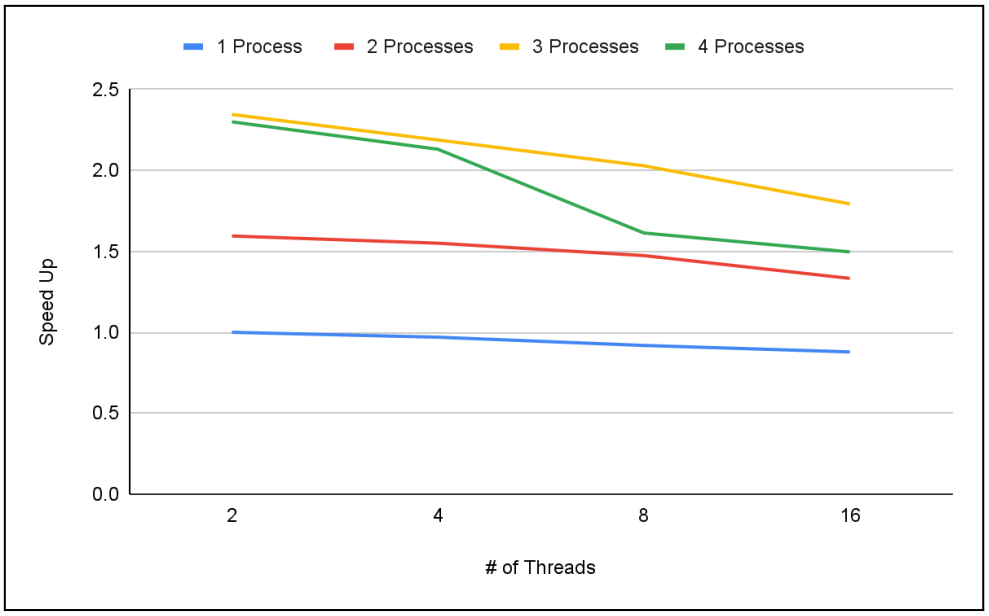


Table 1.3: Hybrid Speed Up vs # of Threads



The results only consist of strong scaling because the main goal of this was to see the speed when parallelizing with MPI vs OMP vs Hybrid (Both). The initial conditions for the computations were: 1000 particles for 200 frames and 200 calculations each frame. As seen in Table 1.3 there is almost a 4x speed when there are 14 processes running the code and about 3.2x speed up for 14 threads. This can be a significant improvement when there are many more computations. As you can also see there is less of a speed up as we get to 16 processes/threads this may be due to a load balancing issue where some processes/threads are becoming redundant and creation of these are becoming useless overhead. You can also see a trend of the speed up taper off as we add more than 10 processes especially. This could also be a result of the aforementioned redundancy and may not be an issue with more computations.

Taking a look at the hybrid table (Table 1.2) the we see the most speed up with 8 processes and for all threads. This is likely the best configuration for these specific parameters since it is such a drastic difference. From table 1.2 we can see that as the number of processes increases the speed up does as well. From table 1.3 it shows a slightly adverse effect on the speed up as we add more threads. This can be due to the small size of the problem where creating many threads and processes is not providing enough of a speed up to make up for the overhead of creating the extra resources.

From this it seems the best use of this code is to run it with multiples nodes or threads but not both and the hybrid solution does fair well with the

Next steps

While you can see that there is much speed up with just this level of parallelization there are still some improvements that can be made to optimise the code. Firstly the major flaw of my code is that it shares all the particles with all the processes and it does that very often. The best option would be for each process to have a sub array to save space and then when each particle moves out of a bin and into another bin that's not in the same process you just send it to that specific process. This would cut down on a lot of communication which would also save time. Then when the particles need to access their neighbours information from another process you can do that same thing this would increase

the communication but only slightly. Another possible improvement would be load balancing. Since the bins are spread across the processes in sequence, if there is a high concentration of particles in a certain area then that process will have a much higher load because of the way the bins are allocated. So adding some sort of load balancing so one process doesn't do the majority of the calculations would be another way to save time. Finally, adding more thread parallelization. Since I focussed mainly on the MPI parallelization there was less thought and intention put behind the thread parallelization sections and there is certainly room for more optimizations by adding more parallel regions and using a more diverse set of tools that the library offers. This will definitely help with the hybrid model especially if you add the OMP with the MPI parallelization in mind.