Aaron Pham
Professor Anastasiu
Coen 145
December 6, 2022

Program 3 Report

1. I parallelized dijkstra's algorithm in the OMP code by adding parallel for loops inside the dijkstra function given in the serial code. I used 3 OMP parallel for loops for the 3 for loops inside the function. I parallelized dijksta's algorithm in the MPI code by using MPI broadcast and MPI allreduce inside the dijkstra algorithm. I included a barrier block in order to make sure that all processors finish at the same time. Inside the load function, I used MPI Send and Receive to distribute the graph and load them between processors. In the main function, I used MPI_INIT and MPI_Finalize as well as MPI_COMM size and rank to get the number of processors and rank of each processor. I also included an MPI barrier so that my processors go through my main function at the same time.

2. My timing results in **Figure 1** and **Figure 2** show my timing results for my parallel executions with OMP and MPI for the different graph sizes.

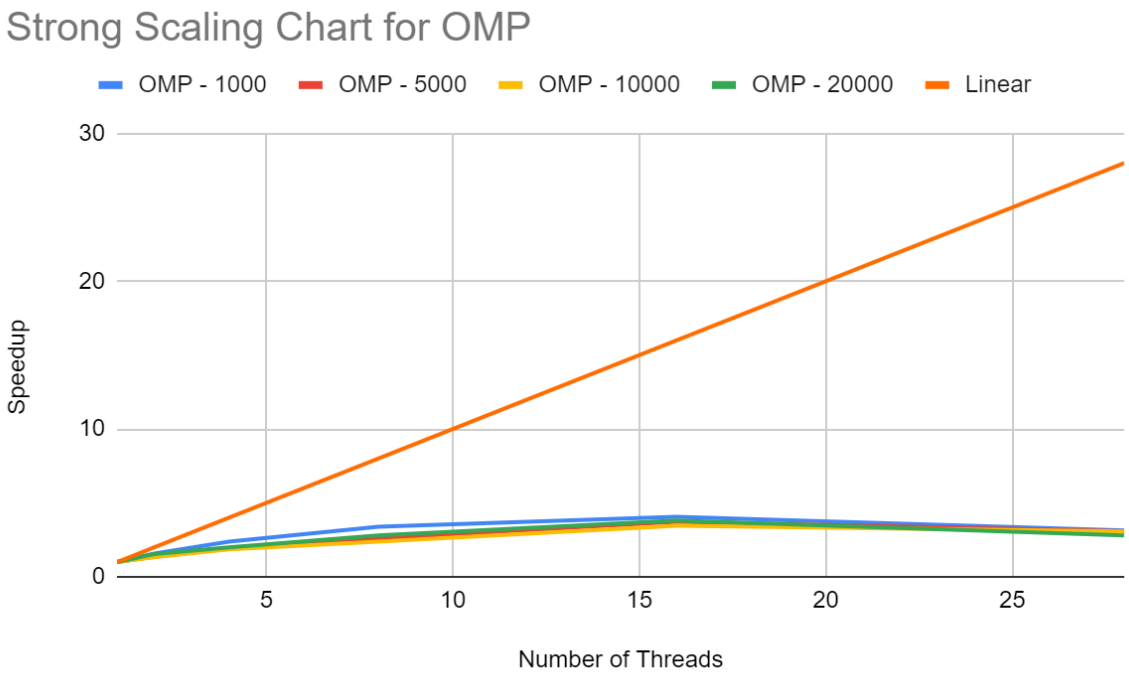

Figure 1: Strong Scaling Chart of speedup for OMP

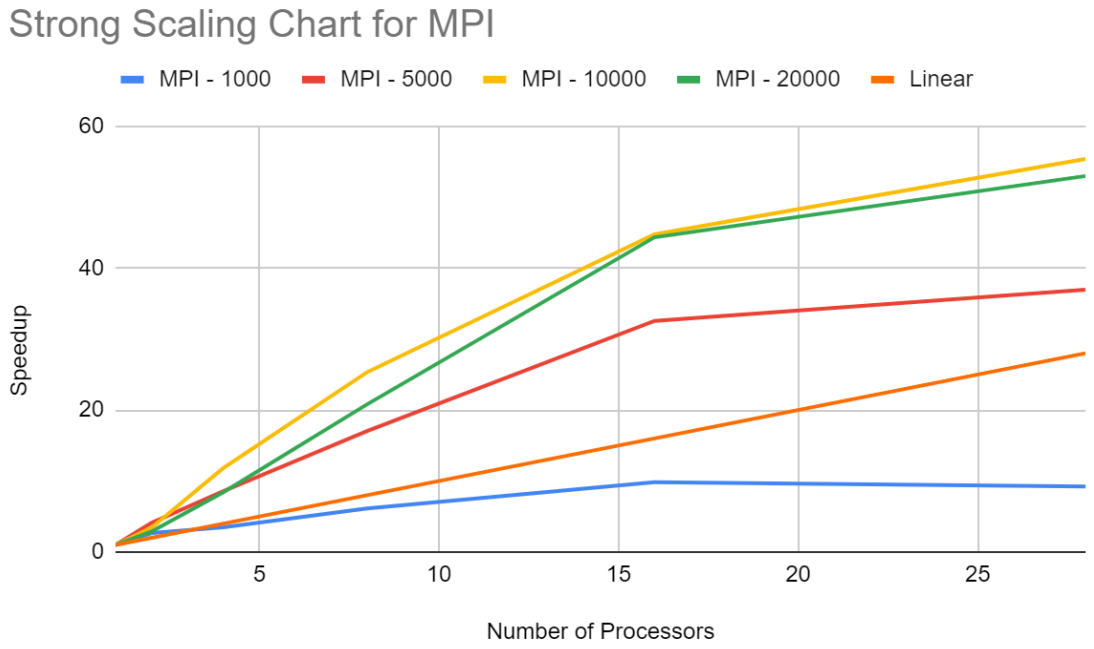

Figure 2: Strong Scaling Chart of speedup for MPI

Analysis

3i. The results show that as the number of threads increased for the OMP, the speedup increased as well, up to a certain point. The speedup for the OMP leveled out after 16 threads. The speedup for OMP was also not that great with a max of around 3. The drop in the speedup for OMP is because the barrier blocks in the dijkstra function must wait for each thread to finish going through the for loop. This causes a lot of waiting and stops the OMP code from having good speedup. On the other hand, as the number of processors increased for the MPI, the speedup increased drastically. Even at 28 processors, the speedup does not level out like it does for the OMP.

3ii. The size of the problem affects the runtime because it increases the amount of work each thread and processor has to do. More work for each thread and processor increases the time it takes each thread and processor to do their work. Increasing the size of the problem does not change the speedup for each thread. Increasing the size of the problem for the MPI, however, causes more speedup. MPI is better at handling bigger sets of data and it shows that the MPI has

much greater speedup in the 10000 and 20000 size graphs than it does in the 1000 and 5000 size graphs.

3iii. My program scales fairly well for the MPI, but not very well with OMP. According to the strong scaling charts, it shows that a further increase in the number of threads will not really increase the speedup. If anything the speedup will continue to level out around 3, or even start to decrease. Increasing the number of processors, however, will continue to increase the speedup.