Mina Vu

Assignment 4 Report

CS515 Parallel Programming

Spring 2022

**Programming with MPI**

Assignment 4 gave us the opportunity to program with file IO and MPI to experience message passing between multiple processors. Starting off with a sample sequential bucket sorting program, we created a similar program using file IO and two parallel versions with MPI that also read and wrote to files. All three programs are complete and produced outputs as expected when tested with varying numbers of processors and integer counts.

Reading and writing to a file was quite simple, especially with the non-parallel version. The only thing to keep in mind was that to find the size of the file, we needed to seek the end of the file. This meant that later reads of the file required that we rewind the pointer first. This was interesting to me as it reminded me of the old VHS tapes. MPI file IO for reading was also fairly straight-forward because every processor read the same number of integers from the file just at a different offset which correlates to the rank of that processor. Writing, on the other hand, was a bit more thought-provoking because each processor may write different amount of numbers to the file. Their offsets do not correlate to their rank but instead to the number of integers in their divided buckets/buffers.

MPI collective routines were the most challenging part of this assignment. In particular, a collective routine that sends and receives different chunks of data at different processors required more work. This is because each processor needs to know the exact number of items it is to expect to receive from all other processors. Thus, a collective non-vector version of the routine must be used before the collective vector routine can work properly. I used MPI\_Alltoall and MPI\_Alltoallv in the last program to accomplish the sending and receiving of different buckets to the correct processor. I also used the collective routine MPI\_Allreduce to get the total count of each receiving buffer at each processor in order to allocate enough memory to the receiving buffer. I ran into a lot of errors when not doing this.

While I was testing all of my programs with different configurations, I found another bug that was a result of me not understanding the steps fully and missing an important half-step. In the first MPI program, I tried to gather all bucket items from other processors into the bucket[P][bucsize] variable thinking that it would save space and lines of code because data is already in there for one processor. I neglected that the bucsize variable is only the max between 2\*(N/P)/P and 8 (I changed this number from 4 in both MPI programs after office hours) which may be less than the count of items that processor will receive from all others. Upon changing this to match the step that I missed, my program worked wonderfully.