David Wade

Concurrent Programming

Lab 1

Write-Up

For this lab, parallelization strategies were used to implement the merge sort algorithm from lab 0 as well as for a new bucket sort algorithm. Starting with the merge sort algorithm, my parallelization strategy consisted of splitting the input data into chunks. The number of chunks was equivalent to the number of threads the program would utilize. The data chunks were then stored into a vector. I then launched the threads, where each thread was responsible for sorting a chunk of data. Once all the data chunks were sorted, I then sequentially merged each sorted data chunk. Originally I wanted to do the merge the chunks concurrently, however this task deemed more difficult than expected. For the bucket sort parallelization, I first got the sequential bucket sort algorithm to operate properly. The part of the algorithm that I parallelized was the for loop I used to go through the entire data array and insert the data into the map data structure. Similarly to the merge sort parallelization, I split the data into chunks and made each thread responsible for inserting their corresponding chunk into the vector map data structure. I used a lock when accessing the vector map data structure in order to avoid data races.

The code of this lab is organized into three C++ files and their corresponding header files. There is a file that implements the merge sort algorithm, a file for the bucket sort algorithm, and the Driver file for the whole program.

The merge sort file is actually the same exact merge sort file I submitted for Lab 0. I wanted to avoid changing it at all in order to keep functionality and due to my parallelization strategy not involving altering the algorithm. Because of this, the parallelization part of merge sort is actually located in the driver file. I know this isn't recommended coding practice, but I wanted to ensure functionality with the implementation. Other than that, the Driver file acts as a normal Driver. It extracts the command line arguments and performs the requested algorithm on the given input file, and writes the sorted data to an output file with the name of the user's choice. In addition, the driver also times each implementation of the algorithm and prints the time in nanoseconds to the console. The bucket sort algorithm is completely encapsulated though. The algorithm and the parallelization strategy lies fully in the Bucket Sort file.

In order to compile the program, the user simply needs to be located in the Lab directory and type a single make command and the entire lab will compile and create object files for each .cpp file.

In order to execute the program, the user must do an executable called "mysort". The syntax for the executable is as follows "./mysort [--name] [sourcefile.txt] [-o out.txt] [-t NUM_THREADS] [--alg-<fjmerge,lkbucket>]". If the name argument is given, then the program will display my name and terminate. Otherwise, the program will read the input source file, sort the file in the algorithm of the user's choosing, and then output it to a file named of the user's choosing. If an invalid sorting algorithm is inputted, the program will give a Usage error and remind the user of the proper syntax.

There are no extant bugs with this lab.