Andrew Hsu

Lab 1

# Description

My first parallelization strategy was fork join. This was done by first splitting the initial large array into different sections to be performed by different threads. This is done dynamically by dividing the number of total elements by the total number of desired threads. Then, we have NUMTHREADS arrays with the left and right index. Then we can call pthread\_create() and create one thread for every sub array we created. After, we call pthread\_join() to make sure all threads join back together. After this returns, inside of each subarray, the numbers will be sorted. However, in order to sort between the subarrays, we need to call mergeSort one more time to sort the outputs of the 5 threads. Then we can write the sorted data to the output

My second parallelization was bucketSort. This implementation involved splitting the initial large array into a given number of buckets. Similar to mergesort, we can then parallelize the code based on the buckets we have. The only difference is we don’t necessarily need one thread for each bucket. My code for this algorithm did not work because I did not have enough time. If I had a couple more hours, I believe I was on the right track to getting things parallelized. I just needed to dynamically assign threads to buckets, even when the number of threads does not equal the number of buckets. I would have passed the bucketHandler into pthread\_create to have at maximum one thread per bucket while attempting to keep all threads busy.

## Code Organization

My main method has two large if statements – one for bucketsort and one for mergesort. When parallelizing, each of these if statements call a void\* function which is passed into pthread\_create. These functions are bucketHandler and taskHandler, which are responsible for calling bucketsort and mergesort for each individual thread. After that, I have initialization and cleanup functions global\_init and global\_cleanup.

# File Description

Mysort.cpp is the main cpp file. Mysort.h is the header file containing structs, defines, and declarations. Pthreadbarrier.h is a file that is necessary for mac users in order to use <pthread.h>. Test.txt is the input file I was using to test. And Makefile is the makefile run using command “make”. Mysort is the object file.

# Compilation

Everything is compiled using “make”

# Execution

Run command:

./mysort [--name] [source.txt] [-t NUMTHREADS] [--alg=<fj, or bucket>]

If printing name, no code will run.

Keep in mind running bucketsort will run a seg fault. (I tried my best, I think I was close please look at my code).

# Extant Bugs

As of right now, the main bug is that bucketsort does not run. It throws a seg fault when trying to pass in a vector to the void\* argument of pthread\_create.