Andrew Hsu

# Parallelization Strategies

I used OpenMP to parallelize my otherwise sequential mergesort program. The problem I was having was passing different left and right indexes into each concurrent thread. I think it is difficult since #pragma omp parallel parallelizes an otherwise sequential block of code, so putting different arguments into each thread is not straightforward. To solve this, I used the same logic I did to generate the left and right indexes with pthreads, but I stored these values in a global vector of vectors rather than passing them in as parameters. This way, inside of my #pragma for loop, it can index each vector, and OpenMP will handle the iterations of the for loops.

One concept that this is dependent on is knowing the number of threads that will execute beforehand. To get around this, I have one #pragma parallel sections that just contains one line where I retrieve the number of threads using omp\_get\_num\_threads(). This allows me to get the number of threads so I can calculate the different left and right indexes that will be passed to mergesort for each thread.

Another problem I had to deal with was the issue of the number of elements of the array not dividing evenly among the number of threads. To solve this, I just continued my earlier algorithm, but if we are assigning the left and right index of the very right thread, calculate the remainder of the number of elements and number of threads. This way, if the elements to be sorted don’t divide evenly, the last thread will just take on extra values.

# Organization

Inside of my main method, I first initialize all my necessary variables. After, I read in command line arguments for name, source file, and out file, I load in my array from my source file. From there, I do all my setting up, calling taskHandler() inside of each thread. taskHandler() is the initialization function that kicks everything off, which starts timing and calls mergeSort(), which then calls merge.

# File Description

Mysort.cpp is the main cpp file that performs the heavy lifting of the program. This is the file that contains the main method, taskhandler, mergesort, and merge.

Mysort.h is the header file for mysort and just contains function declarations and defines.

Makefile is the make file compiled by running command “make”

Testfile.txt is a randomly generated textfile used as the test input

# Compilation

To compile, just run command “make” inside of directory

# Execution

The syntax for running is mysort [--name] [sourcefile.txt] [-o outfile.txt]

Including “—name” will not run the program, it will just return Andrew Hsu

# Bugs

There are no bugs I can find right now with my given input files