ECEN 5033 Concurrent Programming

**LAB3 WRITE UP**

# **Introduction:**

This document is about the merge sort algorithm which is used to sort an unsorted text file and send it to a new or existing file in sorted manner.

**# Description of code organization:**

Consists of main.c and Makefile

* The lab3 code has been organized in two files which is the main.c and Makefile. These two programs are compiled with Makefile. It generates executable called mysort on running make.

**# Description of every file**

* Makefile compiles the main.c and generates executable that can be run by typing make
* mysort is the main executable of the project.
* main.c contains the main code of the project. This file takes the inputs as - - name. Name argument prints the full name of the person written the code.
* It also takes a non-option argument which is the unsortedfile.txt. This file will have multiple lines which each line having integer of size 0 till INT\_MAX, which is 2147483647.
* It also takes -o argument which takes option as the sorted file eg.sortedfile.txt which will be created and contain the numbers in sorted manner for the unsorted file, in this case unsortedfile.txt. The numbers will be sorted and arranged line by line in the sorted file, in this case sortedfile.txt.
* The main.c contains the merge sort algorithm functions and function to read and write to the file. Reading the file is done by using char \*\*array and converting the array to an integer 1d array. This array will be passed to the mergesort function, where the recursive mergesort functions are parallelized using OpenMp #pragma omp parallel sections and sorted out.
* The sorted array will then be written to a new or an existing file line by line.
* Merge sort is an out of place sorting algorithm which needs to create duplicate arrays to execute sorting. The time complexity of merge sort is O(nlogn) and in worst case is O(n^2).
* The file is read using getline which takes row of the file as row of the array and columns of the file as columns of the array. This read array is converted using strtol to an integer array. The char \*\*array which reads input of from the unsorted file uses dynamic memory allocation and reallocates for file content greater than size of initially defined array.

# **Parallelization Strategy for Merge sort**

* The unsorted input file copied into an array where the array will be sorted using mergesort.
* The input file is sorted using recursive merge sort algorithm where each of the two recursive mergesort functions are parallelized using the openMP parallel sections which has an implied barrier at the end.
* This creates different threads for those mergesort calls and executes each of those sections parallelly thus parallelizing the serial program with ease.
* The array is then finally merged and is sent to the sorted\_file.txt

**# Compilation instructions**

* Eg of how to run the project. First compile the file using **make** which creates an executable of the name **mysort**.

**# Execution instructions**

* Then run the executable using the command by typing

**./mysort - - name unsortedfile.txt -o sortedfile.txt**  which sorts the unsorted file using merge sort utilizing OpenMP library functions.

**# Extant bugs**

* The input values are less than **INT\_MAX,** hence a value of more than 2147483647 will lead to differences in the actual sorted file run using **sort -n** linux command and the **mysort** executable. Hence, the same should be considered when running the file.

**# Comparison between OpenMP Parallelization strategy and pthread**

* In OpenMP, the left and right sections of array are parallelized using different OpenMP sections.
* Using pragma omp section where the thread which gets to the section will execute the left and right respectively. It has an implied barrier at the end.
* Finally, the main thread merges with the forked thread and final sorted in the sorted array which is output to the text file.
* This is comparatively very easy in execution as it can parallelize a serial program without much effort compared to pthread.
* In pthread, the input file is copied into an array will be divided into sub arrays where each individual thread will work on approximately same number of array elements.
* The low array index, high array index, array to be populated, thread number is calculated and passed for each thread by using a structure of these values.
* New threads are created, and their specific indexes and unique parameters are passed to the merge thread function which will sort individual sub arrays using recursive merge sort.
* The threads are joined on completion of their unique tasks.
* The individual sub arrays are then finally merged and is sent to the sorted\_file.txt
* With pthread, we need to create separate sub arrays of our large array and pass them to the thread main function separately and finally we need to merge all those sorted sub arrays together, This complexity is reduced in OpenMP as we can choose parts of program we need to parallelize and pass them in the OpenMP parallel functions as and when needed.