ECEN 5033 LAB 3 Write-Up

Name: Nitik Gupta

Comparison between OpenMP and Fork-Join

OpenMP was developed to make the process of parallelization of the program much easier. In this class we used both fork-join and OpenMP to parallelize some part of our code.

- In pthread fork-join method, we used pthread_create function and after the creation of the threads, we joined the threads to the master thread again. Inside the threads, we have to use either Barriers or locks to synchronize the threads and detect and remove data races from the software. So, there are lot of the things that needed to be implemented carefully for successful implementation of parallelizing the code.
- In OpenMP, we have to just use #pragma omp to parallelize the program. To make a local thread variable, we use the shared clause and the shared clause for the shared variables.   The threads join at the end of the brackets automatically. The thread number can be changed using the export on terminal or by default use the number of hardware threads on the machine. We just need to add the

In the end it can be said it is much easier to implement OpenMP for bigger projects as it helps in getting the locks or barriers to be used inside the function without using pthread primitives.

Parallelization Strategy:

Using the OpenMP, we parallelized the code part which can be done simultaneously and won't affect other process by making the program faster. First we use the #pragma omp parallel to parallelize the program. The code part which will be parallelized are enclosed between the curly bracket ({}) . The number threads is by default the number of hardware threads.

Code Organization:

The code is organized based on their common functionality and their usage. There are 3 different c++ files in the project: main.cpp, util.cpp and mergesort.cpp.. As the name suggests, util file contains all the miscellaneous functions used in the utility of the main function. Similarly, mergesort contains the functions used for the sorting algorithm. And the main file contains the main function. The header files for each file contains the function declarations of the functions used in C++ file. In the next topic, the functionality of each function is given in detail.

File Description:

- main.cpp: Contains the main function where the command line arguments are captured using the getopt_long function. These arguments are then processed for their respective functionality.
- makefile: makefile for the project which contains the commands to compile the code ad create the object files of each program and then combine in the mysort object file for the execution of the whole program.
- mergesort.cpp: Contains the function definitions required to perform mergesort operation on the given numbers.

- mergesort.h: Contains the function declarations required to perform mergesort operation on the given numbers.
- util.cpp: Contains the function definitions of all the miscellaneous utility functions that performed inside the main function like print, write to file etc.
- util.h: Contains the function declarations of all the miscellaneous utility functions that performed inside the main function like print, write to file etc.

Compilation Instructions:

The makefile can used to help in compilation. Just call make or make all to compile the object file and use them to execute the program. To delete all the object files, you can use make clean in the command line.

Execution Instructions:

- After the compilation of the program, you can use the ./mysort instruction on the command line, to execute the program. There are just 3 arguments you can give to the ./mysort instruction (you must at least give 1, the input file for successful execution of the program).
- These are --name, input file (without any argument variable), output file (with variable -o), number of threads(with variable -t) and algorithm(with varible --arg) required to sort.
- The --name argument will print out the name and that’s it. But it will also exit the file.
- The input file doesn't require a seperate variable, but without an input file, the code will not run and will exit the program.
- The output file uses the variable –o. If Output file is provided then it will write to it, if not it will print on the console.

Extant Bugs

1. If you provide more command line variables than the required, then the input file might not be taken by the program, and there won't be a sort program execution.
2. If values in the input file is greater than INT_MAX(+2147483647), then the program will not be able to apprehend them and will give wrong answers.

Version Control Link:

https://github.com/IMNG7/Concurrent_Lab3