Necdet Can Uzduran - 2019400195

First of all I want to mention about the which file is which and corresponding executable files. Even though it can be seen in Makefile I want to clarify.

main1.cpp is the source file for **1** thread execution -> create executable named **program1**
main1.cpp -> writes the results in output1.txt

main5.cpp is the source file for **5** thread execution -> create executable named **program5**
main5.cpp -> writes the results in output2.txt

main10.cpp is the source file for **10** thread execution -> create executable named **program10**
main10.cpp -> writes the results in output3.txt

main_opt.cpp is the source file for optional part of the project that accepts arbitrary number of threads as a second parameter -> create executable named **program_opt**
main_opt.cpp -> writes the result in output4.txt

Approach

My approach is similar for 1 and 10 thread of execution parts of the project. For 1 thread execution, I basically coded down all 10 tasks and assign the corresponding value to a global variable, let's say min, max, mode etc. For 10 thread of execution case I hardcoded 10 threads and assign 1 task each.

For 5 thread of execution case I simply combined given 10 task into 5 tasks by basically grouping 2 of the task into one function (For example findMin + findMax turns into findMinMax) and assign these tasks to 5 hardcoded threads. (By hardcoded I mean that, I create these threads line by line and joined them line by line, instead of using a for loop which will result in the same thing at the end.)

For optional case I created an array of threads with the length of given parameter (second parameter which is the # of threads). Additionally, I created an array of function pointers and assign the given tasks to these threads one by one. For a concrete example let's say given # of threads is 4. I assign 4 tasks to 4 different threads and wait for their execution (via pthread_join()). Then do the same thing again. After these two assignment and execution sequence we still have two remaining tasks (10-4*2). I distrbuted these last two tasks to another two threads which are not in use and finalize the whole process. (Used a status array to check if a thread is assignable in the thread array.)

Execution Times

I want to compare the 1-5-10 first and then talk about optional case. Let's say execution times are defined as $E_1 E_5, E_{10}$ for 1 thread of execution, 5 thread of execution and 10 thread of execution respectively. For my case, I can almost clearly say that $E_1 > E_5 > E_{10}$. Hence, parallelism benefits to execution for these case.In orde to be more concrete, $E_1 = 3.98$, $E_5 = 3.49$ and $E_{10} = 3.09$ where N = 2500000 and time units are in seconds. (Of course the calculated execution times may change since they are not deterministic but the hierarchy would remain). For optional case the even though execution time decreases when # of threads increases, that is not the case for all cases. What I mean by that is, sometimes execution time of 2 threads may be smaller than execution time of 4 threads for example. In my opinion, the sole reason behind this outcome is thread creating overhead. Additionally, in my implementation I wait all working threads until their termination (exit) hence the task assigning is not that dynamic (even if your job is finished early you need to wait for another one to get assigned with another job) , so it is hard to observe the benefit of more thread of execution.