CmpE 322 Project2

Zeynep Buse Aydın - 2019400066

Input - Output

The project has a one source .c file which can be compiled with the Makefile. Then the user must enter ***./myprogram.o "size_of_array" "number_of_threads"***.

If there will be no multithreading, the user must give 0 or 1 as the ***number_of_threads***. If the ***number_of_threads*** is not a divider of 10, then the program doesn't execute at all since the purpose is to divide the number of tasks equally.

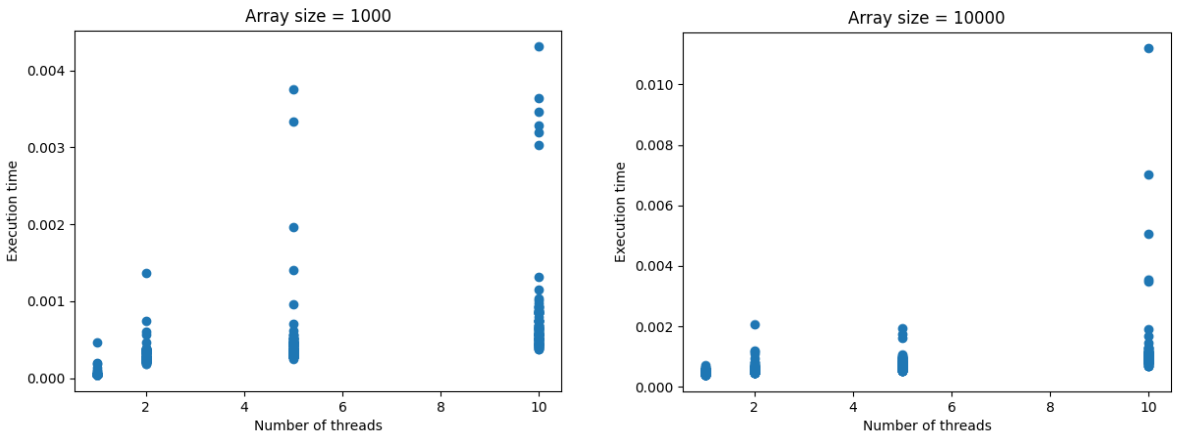
The results are written on the files with names "output{***number_of_threads***}.txt" for simplicity.

Implementation

There are 10 different functions which calculate (or find) the desired statistics of the array that consists of ***size_of_array*** integers. When the value is found, it is added to the global variable ***resultArray*** in the order they will be written on the output file. (resultArray[0] is the minimum element, resultArray[1] is the maximum element, etc.)

These functions are divided equally to the threads. Threads are created in the for loop and their start routine is the ***threadRoutine*** function. This ***threadRoutine*** function takes the for loop variable ***i*** as the parameter, calculates the number of functions (***funcPerThread***) that a thread must execute and starting from the ***i*funcPerThread*** index executes the next ***funcPerThread*** number of functions from the function array.

Comments on Execution Times



These plots are generated by executing the program 30 times in a row each time. The first 30 executions plotted in the left graph is with ***size_of_array*** = 1000 and the right one is again 30 executions but with ***size_of_array*** = 10000.

As it can be seen in the plots, there is a tendency to have a larger execution time as the number of threads are increasing. This might be surprising -it was definitely for me- since multithreading is generally expected to improve performance. I believe in our case, the arithmetic operations that are performed are not very complicated or time consuming. The workload of thread creations, synchronization and also communication is heavier than the actual arithmetic operations. Therefore multithreading doesn't really help with the performance.