Cmpe 322, Project 2

Deniz Ünal, 2019400234

Output

Compilation creates 4 executable file no_thread.o for first part, five_threads.o and ten_threads.o for second part and optional_threads.o for optional part. They write outputs to output1.txt output2.txt output3.txt output4.txt respectively. Execution syntax for optional part:

./optional threads.o N <thread-count>

Implementation Details

In order to generate elements for the list I used the rand function and then sorted them for convenience and speed. This sorted array is saved in a global array (creation and sorting is not included in the printed time) Then created a different function for each statistical calculation which saves their results to some global variables.

In the first part the execution just consists of calling the methods one by one.

In the ten threads version of the second part every thread creation takes one statistical function as argument.

For five threads version and for the optional part there is an execute function which takes an void pointer. This method takes the pointer and converts it into func_list struct pointer. func_list struct has 2 members, a int which holds how many functions it will execute and an array of functions that will be executed. Execute function calls the functions in the struct one by one. Whenever a thread will be created, first an array of func_list structs will be created and functions will be distributed to this structs in the order given in the description and then the address of a member of the array will be given as an argument to the thread creation. (note that in the optional part if the thread count is 1 still one child will be created and if the thread count does not divide 10, number of functions they execute won't be equally distributed)

In all parts, time starts just before first thread creation and ends just after last thread completion.

Time Observations

N = 100 one thread: 0.00010s five threads: 0.00016s ten threads: 0.00027s
N = 350,000 one thread: 0.01487s five threads: 0.01115s ten threads: 0.01274s

When the N is low number of threads (or no threads) performs better because the overhead of thread creation exceeds the advantage of parallel execution.

When the N is large generally high number of threads perform better because the advantage of parallelism will be more likely to exceed the overhead.

However high number of threads does not equal to better performance. (In this example 5 threads is faster than 10) The reason for that is some of the calculations are slower than the others and the thread execution times are not balanced. When the number is high some unnecessary threads will be created which will only execute O(1) operations (such as median), and cause overhead for no performance gains.