25.11.2022

Project 2: Paula Vondrlik - 2022690390

File description

The list is created with random seed zero, so the array is the same for every call of the program. This is to increase comparability. The list is sorted in the main function regardless of the number of threads, since several function needs a sorted list and performing the sort in the math-functions could lead to race conditions or requires a local copy for every thread.

The main file “thrNthr”(threads no threads)takes two arguments and creates the given amount of arrays, each array executes 10/number of array (lower bound) tasks while the last thread executes all leftover threads additionally(e.g. 4 threads: 2-,2-,2-,4– tasks). The operations are always the same and do not take into account, that the thread might have already performed the operation (e.g. first “sum”, then do “arithmetic_mean” by calculating sum again).
The second file “noThread” takes one argument and creates no thread. The calculations build on each other (e.g. use the already calculated min and max to calculate range).

Time measuring

number of elements in list is 10,000,000

n_threads	**Processor time in s**	**Wall-clock time in s**
“noThread”	0.135242	0.134135
1	0.227333	0.227398
2	0.233846	0.139881
5	0.277873	0.084195
10	0.318689	0.071446

*exluding the serial qsort +0.87s and list construction
+0.05s for processor and wall-clock time

Analysis

<u>Processor time:</u>
Processor times increases with the number of threads, around 0.01 per thread. This is caused by additional computation for creating and managing the threads and the increased number of context-switches between threads since the threads share the CPU time of the main thread (system-contention-scope).
<u>Wall clock time</u>
By doubling the number of threads, we get half of the wall clock time plus the 0.01s per additional thread. The speed increases because the program makes use of multicore architecture (scalability). However, we cannot see a great advantage from 5 to 10 threads because of the extra computation and overhead.
<u>Comparison of “noThread” and “thrNthr”</u>
Comparing between threads 1 and “noThread” emphasize that multithreading is not always a better solution, especially if task can make use of former results (e.g. reusing the already calculated sum). This shows the difficulties of multithreading to divide activities and balancing work.