HW-2 : Performance evaluation results

These tests were performed on one of KTH:s computers that has 11th Generation Intel® Core™ i7 Processors with 8 cores and up to 16 threads. The source code was compiled by the following terminal command *gcc -O -fopenmp [FILE_NAME].c*. Each program was run 5 times and the median value was used for execution time.

MatrixSum

SIZE of input data = 100

1 thread = 2.1365e-05
2 threads = 1.2415e-05 Speedup = 1.720902135
4 threads = 9.35901e-06 Speedup = 2.282826923
8 threads = 1.0664e-05 Speedup = 2.003469617
16 threads = 1.9482e-05 Speedup = 1.096653321

SIZE of input data = 1000

1 thread = 0.00171254
2 threads = 0.000858181 Speedup = 1.995546394
4 threads = 0.000433006 Speedup = 3.955002933
8 threads = 0.000435103 Speedup = 3.935941605
16 threads = 0.000996631 Speedup = 1.718329051

SIZE of input data = 10000

1 thread = 0.0537986
2 threads = 0.0276389 Speedup = 1.946481228
4 threads = 0.014008 Speedup = 3.840562536
8 threads = 0.0135063 Speedup = 3.9832226444
16 threads = 0.00938563 Speedup = 5.732017989

The program for MatrixSum doesn't scale particularly well with a doubled number of threads for each data set. This can be due to false-sharing, which is common with programs that work on matrices.

Quicksort

SIZE of input data = 100000

1 thread = 0.00948032
2 threads = 0.00673927 Speedup = 1.40673
4 threads = 0.00776651 Speedup = 1.22067
8 threads = 0.0111954 Speedup = 0.846807
16 threads = 0.00928086 Speedup = 1.02149

SIZE of input data = 1000000

1 thread = 0.0272947
2 threads = 0.0146981 Speedup = 1.85703
4 threads = 0.00954602 Speedup = 2.85927
8 threads = 0.0104012 Speedup = 2.62419
16 threads = 0.0153522 Speedup = 1.7779

SIZE of input data = 10000000

1 thread = 0.383805
2 threads = 0.194715 Speedup = 1.97111
4 threads = 0.105012 Speedup = 3.65487
8 threads = 0.0667353 Speedup = 5.75116
16 threads = 0.0679239 Speedup = 5.65051

The program for parallel quicksort doesn't scale well as the speedup numbers clearly show. This may be due to the fact that the number of work done in each recursive step is not the same/equal. This makes it difficult to divide the workload evenly among multiple threads. The last test with the biggest input data shows a better speedup than the other two and this may be due to data locality where a huge number of the elements of the array may have been relocated to the cache at the same time during the execution of the algorithm.