Report Oblig 1 IN3030

Haiyuec

Processor:

Intel I7-8550u

Base frequency: 1.80 GHz

Max Turbo frequency: 4.00 GHz

4 cores 8 Threads

How to run the program:

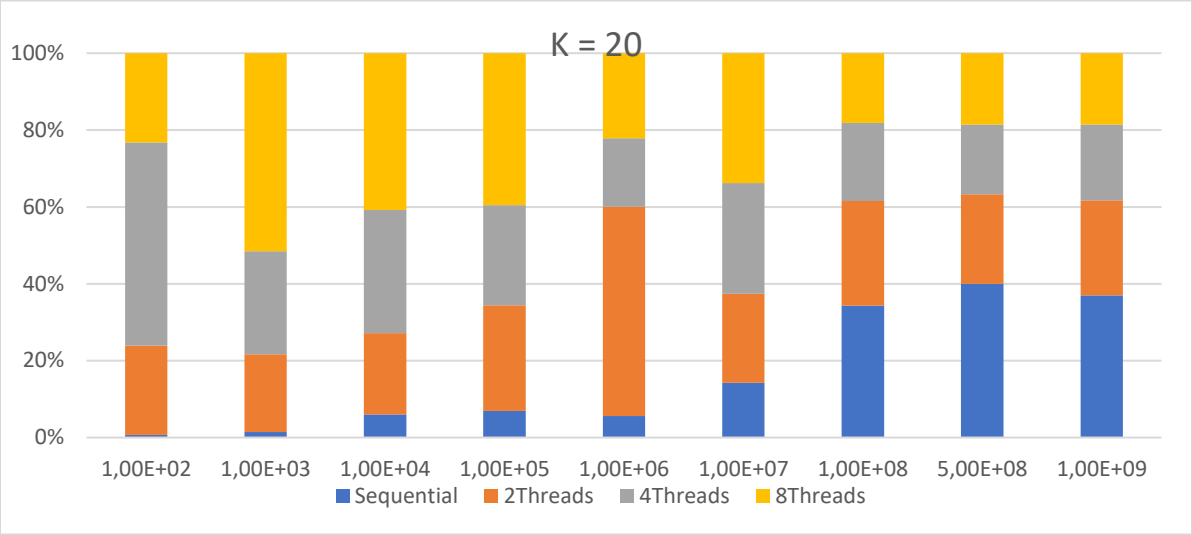
javac *.java

java -Xmx***ram***m Main n k (for n = 1* 10 ^ 9 11GB ram is required)

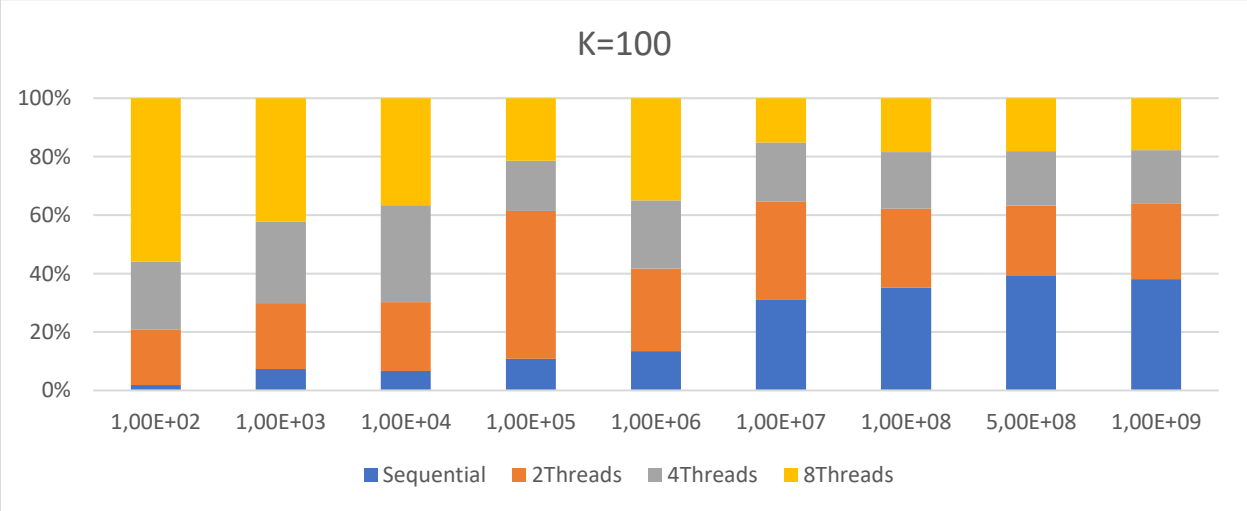
Runtime data:

PS: Graphs show the ratio between the different versions. E.g. If yellow block is half the size of the blue block, that means the 8 threads parallelized version is double the speed of the sequential version.

k	n	Arrays.sort	Sequential	2Threads	4Threads	8Threads
20	1,00E+02	0,029	0,015	0,546	1,240	0,546
20	1,00E+03	0,230	0,041	0,575	0,762	1,464
20	1,00E+04	0,687	0,221	0,783	1,183	1,506
20	1,00E+05	1,060	0,303	1,203	1,145	1,732
20	1,00E+06	8,550	0,309	3,010	0,986	1,226
20	1,00E+07	68,578	0,647	1,050	1,310	1,534
20	1,00E+08	810,798	7,060	5,598	4,183	3,733
20	5,00E+08	#	368,796	214,852	167,300	171,197
20	1,00E+09	#	663,942	443,290	354,283	332,844



n	k	Arrays.sort	Sequential	2Threads	4Threads	8Threads
1,00E+02	100,0	0,019	0,065	0,661	0,806	1,938
1,00E+03	100,0	0,231	0,356	1,103	1,361	2,058
1,00E+04	100,0	0,928	0,438	1,535	2,169	2,398
1,00E+05	100,0	9,165	0,804	3,739	1,275	1,585
1,00E+06	100,0	67,536	0,739	1,557	1,282	1,921
1,00E+07	100,0	842,912	8,048	8,789	5,236	3,938
1,00E+08	100,0	9745,309	73,533	56,552	40,511	38,624
5,00E+08	100,0	#	353,251	214,081	167,69	161,96
1,00E+09	100,0	#	692,063	472,27	329,557	322,933



For k = 20:

We can see that the parallel version is faster than the sequential version when n is greater than 1 * 10^8. I think this is because that the sequential version it only sorts the first 20 elements, and it only makes a maximum of 20 swaps for any element that is found later in the array that is greater than the 20^{th} greatest. The parallel version has to find the greatest 20 elements in each of the threads, and uses a max-heap to sort the answers from the threads, which requires additional time. The cost of initializing the threads would also add cost on the runtime of the parallel version.

For k = 100:

We can see here that the parallel version is faster than the sequential version already for n = 1 * 10^7. I believe that this difference is caused by the increase in k from 20 to 100. Because now the program has to make a maximum of 100 swaps for every element in the second half of the array, the cause has increased significantly. Therefore, it is now more beneficial to perform several insertions simultaneously.

Edge case:

In my implementation, if the number of elements in the unsorted array assigned to a thread is lower than k, it would just return an new array consisting of that section, which would be sorted by the main-thread with heap. This is essentially sequential heap-sort with extra steps.

However, if the number of elements assigned to a thread is close to k, such as the case where k = 20 and the program ran with 4 threads to sort 100 elements. In this situation, the program is essentially sorting 4 arrays of size 25 sequentially, and then sorting 80 elements with heap sort, which of course if highly in-efficient.