Assignment 5 Report

Hanzhang Wang

001010009

- Task

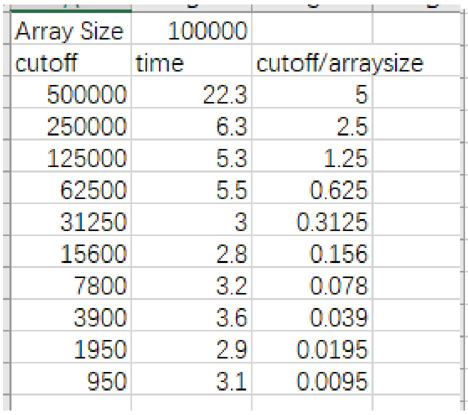
Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

To implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. Consider two different schemes for deciding whether to sort in parallel.

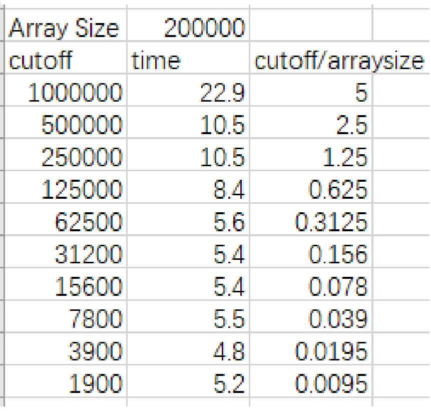
1. A cutoff (defaults to, say, 1000) which will be updated according to the first argument in the command line when running. To experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, decide on an ideal number (t) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of lg t is reached).

- Output

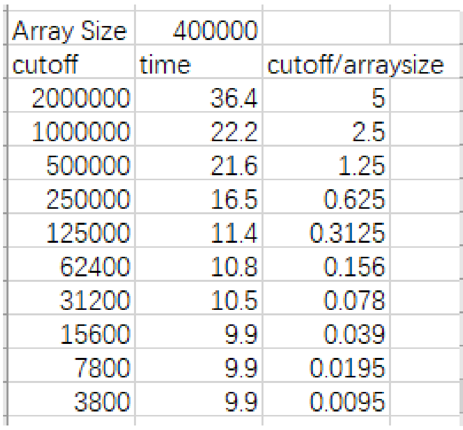
I set the array size from 100000 to 200000, 400000 and 80000 with degree of parallelism at 256 constant. Here are some screenshots of getting from program. And the data generated to the result.csv is stored together in the data.csv.



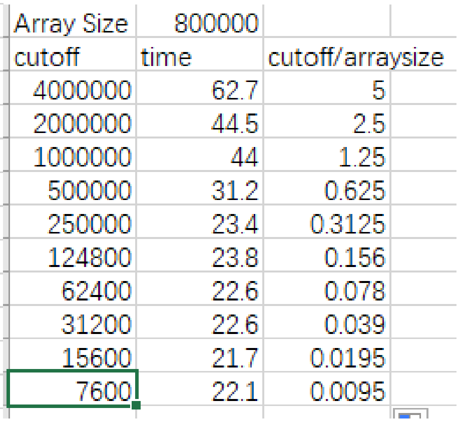
Array Size	100000	
cutoff	time	cutoff/arraysize
500000	22.3	5
250000	6.3	2.5
125000	5.3	1.25
62500	5.5	0.625
31250	3	0.3125
15600	2.8	0.156
7800	3.2	0.078
3900	3.6	0.039
1950	2.9	0.0195
950	3.1	0.0095



Array Size	200000	
cutoff	time	cutoff/arraysize
1000000	22.9	5
500000	10.5	2.5
250000	10.5	1.25
125000	8.4	0.625
62500	5.6	0.3125
31200	5.4	0.156
15600	5.4	0.078
7800	5.5	0.039
3900	4.8	0.0195
1900	5.2	0.0095



Array Size	400000	
cutoff	time	cutoff/arraysize
2000000	36.4	5
1000000	22.2	2.5
500000	21.6	1.25
250000	16.5	0.625
125000	11.4	0.3125
62400	10.8	0.156
31200	10.5	0.078
15600	9.9	0.039
7800	9.9	0.0195
3800	9.9	0.0095



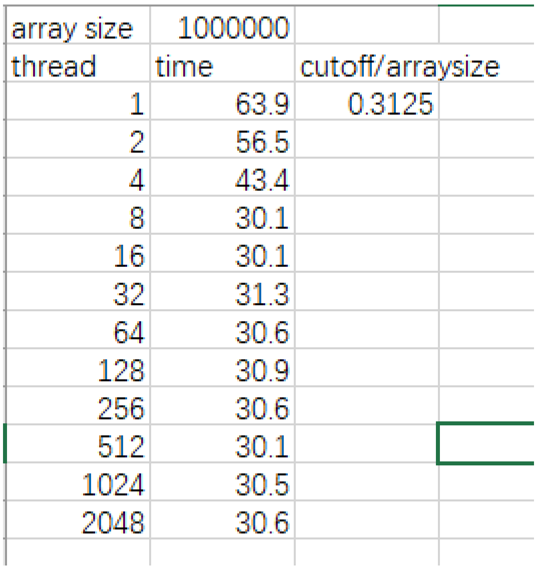
Array Size	800000	
cutoff	time	cutoff/arraysize
4000000	62.7	5
2000000	44.5	2.5
1000000	44	1.25
500000	31.2	0.625
250000	23.4	0.3125
124800	23.8	0.156
62400	22.6	0.078
31200	22.6	0.039
15600	21.7	0.0195
7600	22.1	0.0095

Based on data above, we can see that the time barely changed after the 5th cutoff and the cutoff/array size is 0.3125. So we can say that 0.3125 is a good cutoff in this case.

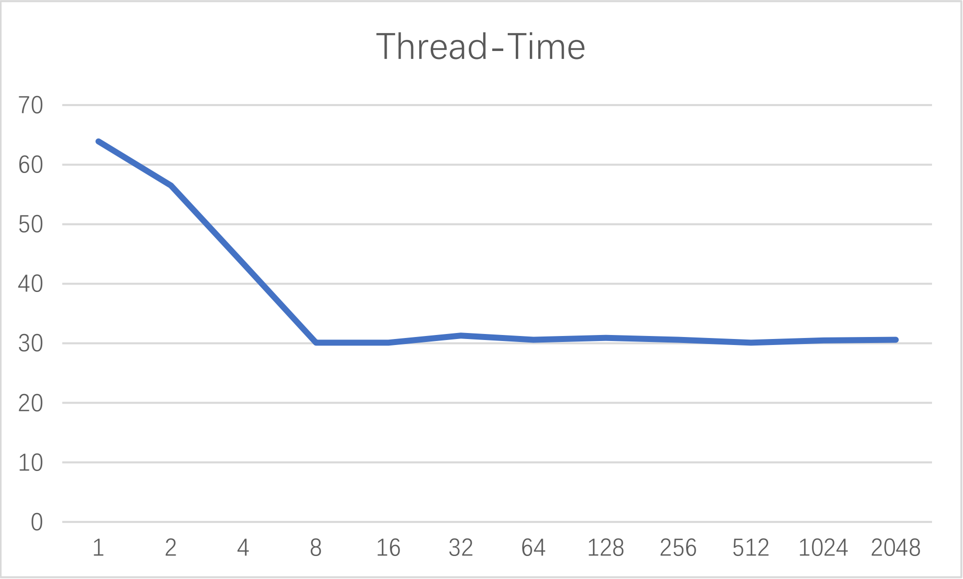
Next, let's make cutoff/array size constant at 0.3125 and vary threads.

I use different power of 2 for threads change since it is recursion involved.

Here is my data:



array size	1000000	
thread	time	cutoff/arraysize
1	63.9	0.3125
2	56.5	
4	43.4	
8	30.1	
16	30.1	
32	31.3	
64	30.6	
128	30.9	
256	30.6	
512	30.1	
1024	30.5	
2048	30.6	



It doesn’t change much after threads 8.

- Conclusion:

It performs most efficient at thread 8 with 0.3125 of array size based on my test data