Parallel Sorting

Vikram Bharadwaj Ramesh

Problem Statement:

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

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cut-off, then you should use the system sort instead.
2. Recursion depth or number of available threads. Using this determination, you might 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 depth of lg t is reached).

Considered conditions:

1. Large array sizes are considered so that the time taken for creating threads do not affect our performance calculations.
2. The array is shuffled for optimum average value
3. The cutoff value is calculated by trial method-running each method in increments
4. The sort method is run 100 times for each pair of array size and cut off
5. The number of threads that are created are calculated

Observations:

Note: The optimum values are highlighted

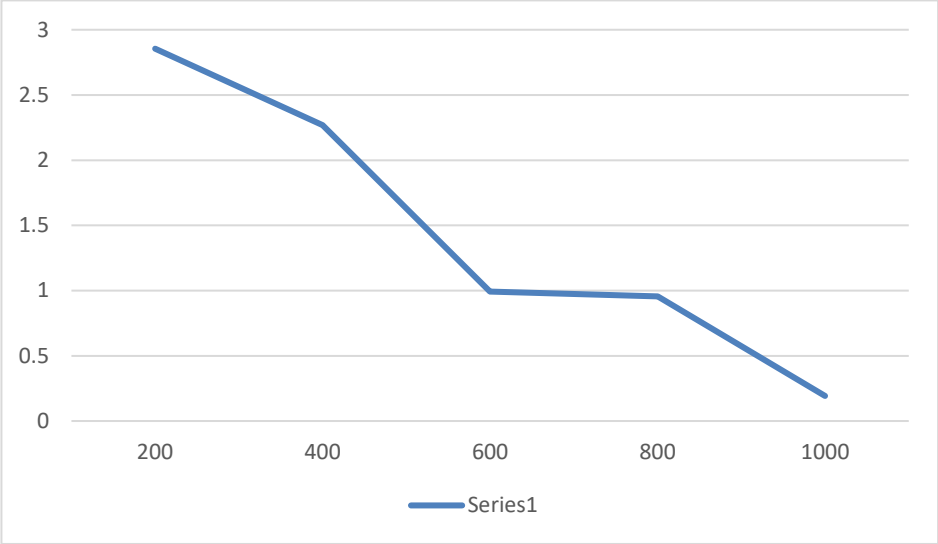
Cut-Off	Array Size	Time in milliseconds(ms)	Number of threads created
200	1000	2.85483	14
400	1000	2.26838	6
600	1000	0.99166	2
800	1000	0.95518	2
1000	1000	0.19227	0
1000	5000	5.53995	14
2000	5000	3.21356	6
3000	5000	1.88278	2
4000	5000	1.69801	2
5000	5000	0.60723	0
2000	10000	3.42575	14
4000	10000	2.95443	6
6000	10000	2.17216	2
8000	10000	2.23306	2
10000	10000	1.39344	0
10000	50000	6.56636	14
20000	50000	6.8675	6
30000	50000	4.99577	2
40000	50000	4.41307	2
50000	50000	4.44624	0

20000	100000	10.23424	14
40000	100000	12.34377	6
60000	100000	6.70389	2
80000	100000	6.48851	2
100000	100000	7.59948	0

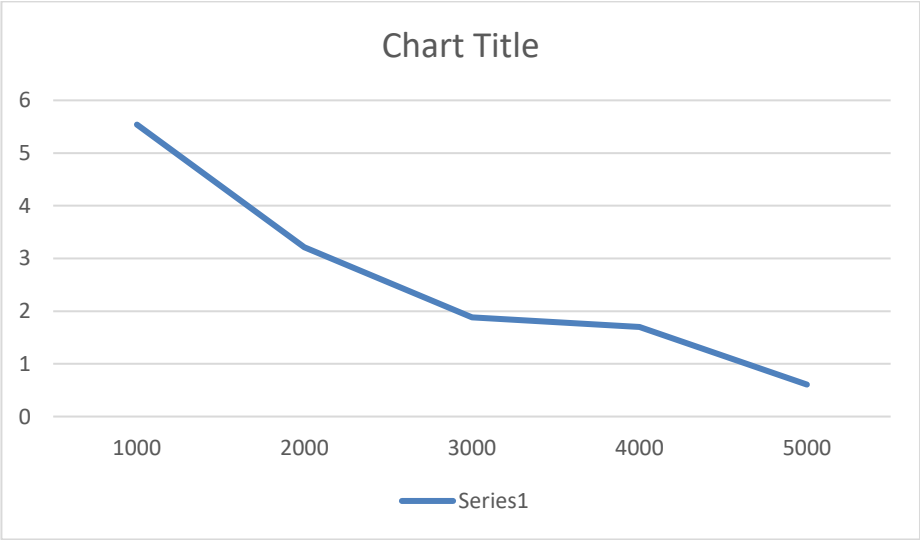
Graphical Representation of results:

Please Note:
X-Axis: cut-off value
Y-Axis: time taken

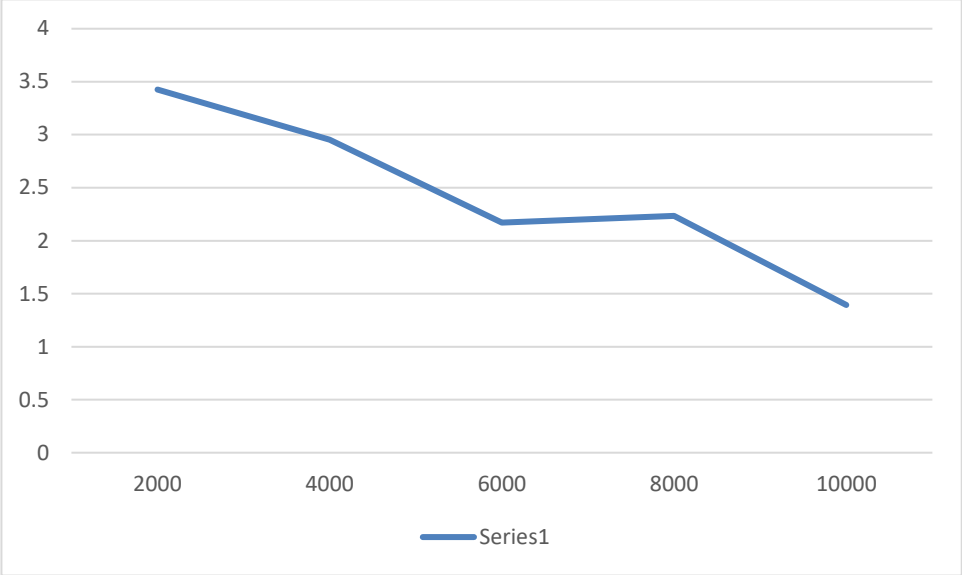
Array Size: 1000



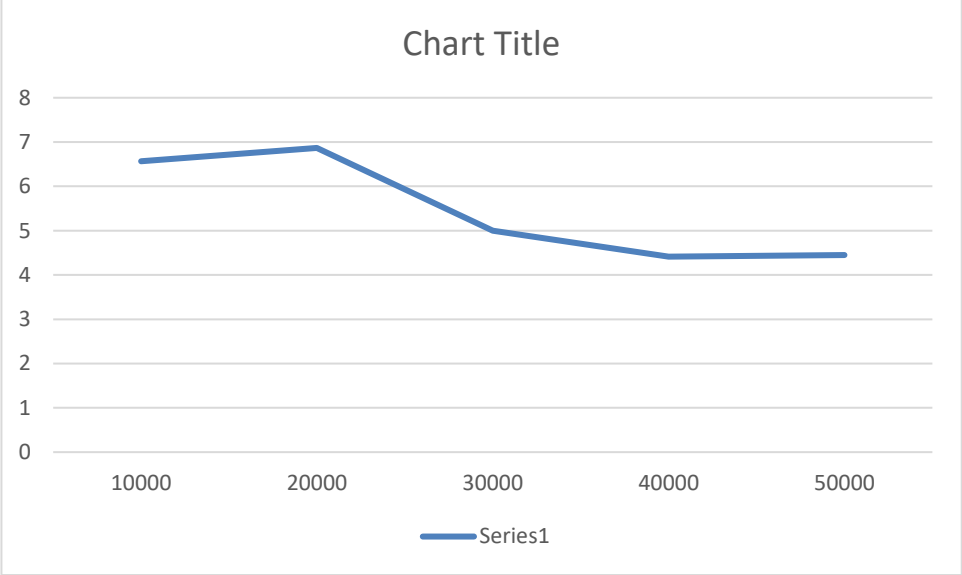
Array Size: 5000



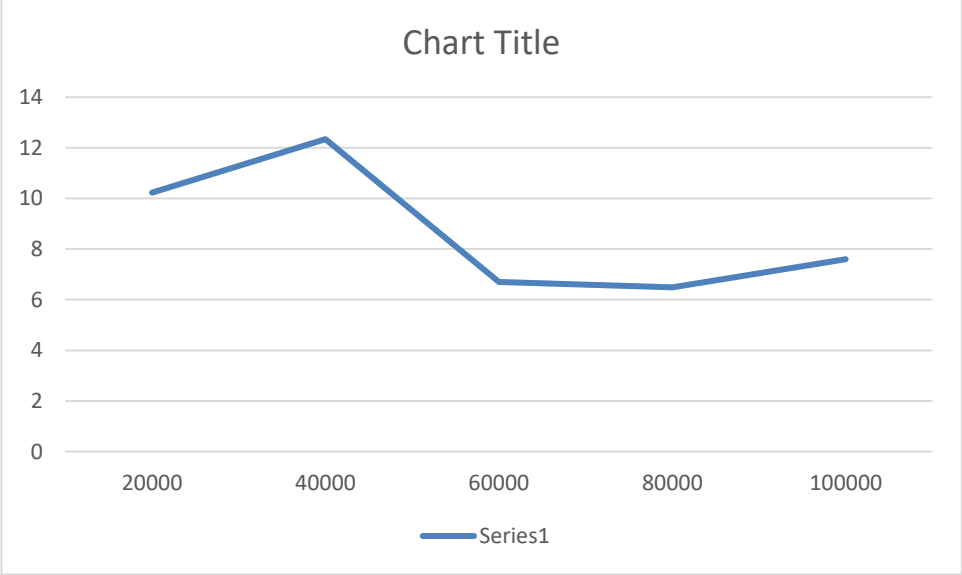
Array Size: 10000



Array Size: 50000



Array Size: 100000



Conclusion:

- In case of the smaller arrays (results are highlighted), the cut off is same as that of the array size, hence only system sort is called. In this case the parallel threads/depth is 0.
- It is observed that for smaller arrays, system sort is optimum since the thread creation time is skipped. For larger arrays it is not the same. The threads help in optimization.
- Recursion depth or the available threads is calculated based on the threads created during parallel sort and the ideal number for this parameter is calculated just before system sort is called.
- The optimum value is highlighted in the below table

The cutoff/ array size is calculated as below:

Cut-off	Array Size	Cut-off / Array Size
200	1000	0.2
400	1000	0.4
600	1000	0.6
800	1000	0.8
1000	1000	1.0
1000	5000	0.2
2000	5000	0.4
3000	5000	0.6
4000	5000	0.8
5000	5000	1.0
2000	10000	0.2
4000	10000	0.4
6000	10000	0.6
8000	10000	0.8
10000	10000	1.0
10000	50000	0.2
20000	50000	0.4
30000	50000	0.6
40000	50000	0.8
50000	50000	1.0
20000	100000	0.2
40000	100000	0.4
60000	100000	0.6
80000	100000	0.8
100000	100000	1.0

It is observed that 0.8 is the optimum value of threads/depth for cut-off/array size for larger arrays