Gaurav Thorat

002957343

INFO – 6205 Section 6

ASSIGNEMENT 5

Task: - To implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. Consider the following while parallel sorting

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 cutoff, then you should use the system sort instead.
2) Recursion depth or the 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 the depth of lg t is reached).
3) An appropriate combination of these.

Observations: -

1) For arrays of same size and same cutoff we can see that there is a performance difference when using less threads. If we use a smaller number of threads the sorting is slower.
2) Similarly, we can also conclude that after a certain number of threads there is no performance gain by having a greater number of threads.
3) After this observation about threads, the other thing that **significantly affects** the performance of parallel sort is the **cutoff** value. Cutoff value for smaller is inefficient or not that fast, and the performance while having a high cutoff is also not that efficient.
4) When the **cutoff value** is between **25-30% percent** of the **array size** we see a considerable performance difference. As this where it takes the **least** amount of time to sort irrespective of the number threads but by having the same array size and cutoff values.
5) In the below fig the above observation is highlighted in yellow.

Array Size	Cutoff Value	Time Taken	Threads
2000000	510000	3078	6
2000000	520000	2578	7
2000000	530000	1969	8
2000000	540000	1781	9
2000000	550000	1844	10
2000000	560000	1593	11
2000000	570000	1610	12
2000000	580000	1906	13
2000000	590000	1625	14
2000000	600000	1938	15

Array Size	Cutoff Value	Time Taken	Threads
2000000	510000	3047	3
2000000	520000	1953	4
2000000	530000	1860	5
2000000	540000	1781	6
2000000	550000	1812	7
2000000	560000	1782	8
2000000	570000	1781	9
2000000	580000	1797	10
2000000	590000	1797	11
2000000	600000	1796	12

Array Size	Cutoff Value	Time Taken	Threads
20000000	5100000	20968	6
20000000	5200000	19578	7
20000000	5300000	18454	8
20000000	5400000	18125	9
20000000	5500000	18015	10
20000000	5600000	17688	11
20000000	5700000	17656	12
20000000	5800000	17937	13
20000000	5900000	17797	14
20000000	6000000	18250	15

Array Size	Cutoff Value	Time Taken	Threads
20000000	5100000	25643	3
20000000	5200000	24075	4
20000000	5300000	23444	5
20000000	5400000	22689	6
20000000	5500000	23508	7
20000000	5600000	23580	8
20000000	5700000	21041	9
20000000	5800000	20594	10
20000000	5900000	20813	11
20000000	6000000	22609	12

Index	
Red Color	Not optimal
Yellow Color	Optimal

For arrays of same size and same cutoff we can see that there is a performance difference when using less threads. If we use a smaller number of threads the sorting is slower.
Similarly, we can also conclude that after a certain number of threads there is no performance gain by having a greater number of threads.
After this observation about threads, the other thing that **significantly affects** the performance of parallel sort is the **cutoff** value. Cutoff value for smaller is inefficient or not that fast, and the performance while having a high cutoff is also not that efficient.
When the **cutoff value** is between **25-30% percent** of the **array size** we see a considerable performance difference. As this where it takes the **least** amount of time to sort irrespective of the number threads but by having the same array size and cutoff values.
In the below fig the above observation is highlighted in yellow.