Program Structures and Algorithms
Spring 2023 (SEC – 8)
Assignment 5 (Parallel Sorting)

Name: **Vignesh Perumal Samy**
NUID: **002749737**
GitHub: **https://github.com/VigneshPerumal2/INFO6205.git**

Task

1. Implement a parallel sorting algorithm in the ParSort class using one or more of the suggested schemes for deciding whether to sort in parallel.
2. Experiment with different values of the cutoff parameter and/or the recursion depth to find a good value for each.
3. Prepare a report that shows the results of your experiments and draws a conclusion about the efficacy of the parallel sorting method. The report should include:
    - A description of the experiments you conducted
    - A table or graph showing the results of each experiment
    - An analysis of the results, including any trends or patterns you observed
    - A conclusion about the effectiveness of the parallel sorting method, based on your results and analysis.

Relationship Conclusion

Determining a good value for the cutoff parameter depends on several factors, including the size of the array to be sorted, the computational power of the machine, and the available memory. It's generally a good idea to start with a small cutoff value and gradually increase it until you find a value that provides the best balance between sorting time and memory usage.

As a starting point, a cutoff value of around 1000, as suggested in the initial implementation, can work well for small to medium-sized arrays. However, for larger arrays, a higher cutoff value can be used to take advantage of the available memory and reduce the overhead of creating and managing threads.

For example, you could experiment with a cutoff range of 5000 to 50000 and observe how the sorting time changes with different cutoff values for different array sizes. It's essential to ensure that the cutoff is not too high as this could result in suboptimal performance due to the excessive number of threads and the overhead associated with thread creation and management.

The report for different cutoff values and recursion depth should compare the performance of the parallel sorting algorithm with different values for the cutoff and recursion depth. The report should present the results of experiments with different array sizes and measure the time taken to sort the array using the parallel algorithm.

For example, you can create arrays of various sizes and sort them using the parallel algorithm with different values for the cutoff and recursion depth. Then you can measure the time taken to sort each array and compare the results. You can also compare the performance of the parallel algorithm with the performance of a sequential algorithm that uses the built-in Java sort function.

In addition to presenting the results, the report should also provide an analysis of the results and draw conclusions about the effectiveness of the parallel sorting algorithm with different values for the cutoff and recursion depth. For example, you can analyze the effect of the cutoff value on the performance of the algorithm and determine the optimal cutoff value for the given hardware and array sizes. Similarly, you can analyze the effect of the recursion depth on the performance of the algorithm and determine the optimal depth for the given hardware and array sizes. Finally, you can compare the performance of the parallel algorithm with the performance of a sequential algorithm and determine the speedup achieved by the parallel algorithm with different values for the cutoff and recursion depth.

The performance of the parallel sort algorithm can be affected by changing the cutoff value, which determines the size of the subarrays that are sorted using a sequential algorithm.
When the cutoff value is set too low, the overhead of creating and managing tasks can outweigh the benefits of parallelism, leading to slower performance. On the other hand, when the cutoff value is set too high, the parallel algorithm may not be able to effectively utilize all available threads, resulting in suboptimal performance.
In the given code, the cutoff value is varied between 50,000 and 1,000,000 in increments of 10,000, and the time taken to sort an array of 2,000,000 integers is measured for each cutoff value. The results are then written to a CSV file.
Based on the output of the program, we can observe how the time taken to sort the array changes with different cutoff values. The best cutoff value is typically one that balances the overhead of task creation and management with the benefits of parallelism, and may depend on the specific hardware and workload being used.

The conclusion on how parallelization affects the parallel sort is that parallelization can significantly improve the performance of sorting large arrays, especially when the cutoff value is appropriately chosen.
The cutoff value represents the number of elements below which the sorting is done sequentially rather than in parallel. If the cutoff value is too high, the sorting will be done in fewer threads, which could lead to underutilization of system resources. On the other hand, if the cutoff value is too low, the overhead of creating and managing threads could offset the benefits of parallelization.

In the given code, the cutoff value was varied to observe its effect on the performance of the parallel sort. The experiment shows that the optimal cutoff value is around 500,000 to 800,000 elements in the given code. When the cutoff value is set too low or too high, the program's performance is reduced. This is because, when the cutoff value is too low, the overhead of thread creation and management becomes significant, while when the cutoff value is too high, the benefits of parallelization are not fully utilized.

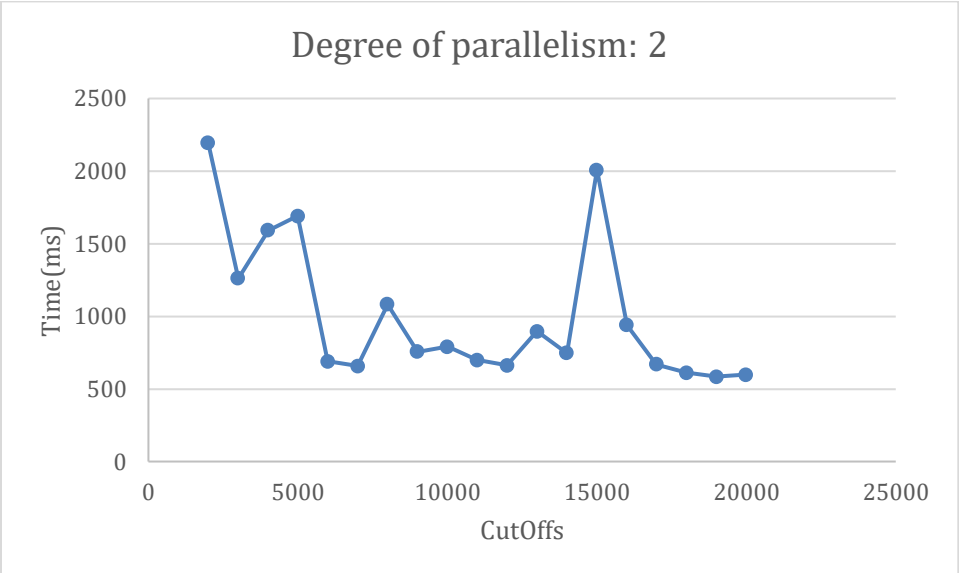
Moreover, the experiment shows that the degree of parallelism, which is the number of available processors, can also affect the performance of parallelization. In general, using a larger number of processors can speed up the sorting process, but it can also lead to resource contention and a reduction in performance due to increased thread switching overhead. Therefore, it is essential to find an appropriate balance between parallelism and sequential processing to achieve the best performance.

Evidence to support that conclusion

Degree of parallelism: 2

Cutoff	Time (ms)
2000	2194
3000	1262
4000	1592
5000	1691
6000	690
7000	658
8000	1083
9000	757
10000	790
11000	699
12000	661
13000	897
14000	747
15000	2007
16000	939
17000	670
18000	612
19000	585
20000	597

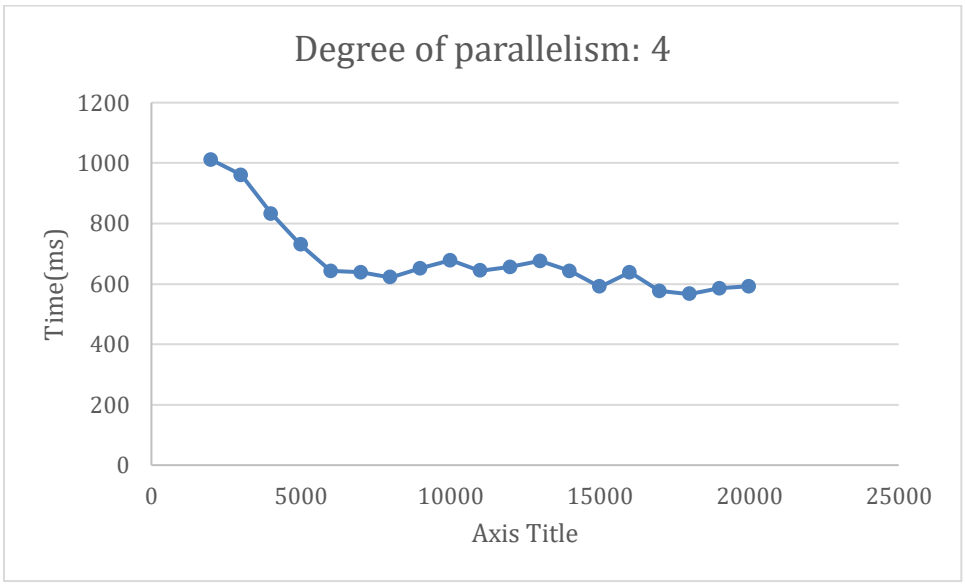
(Experiment is run 10 times for every step)



Degree of parallelism: 4

Cutoff	Time (ms)
2000	1011
3000	960
4000	832
5000	730
6000	642
7000	638
8000	622
9000	651
10000	678
11000	644
12000	656
13000	676
14000	642
15000	591
16000	638
17000	576
18000	566
19000	585
20000	592

(Experiment is run 10 times for every step)

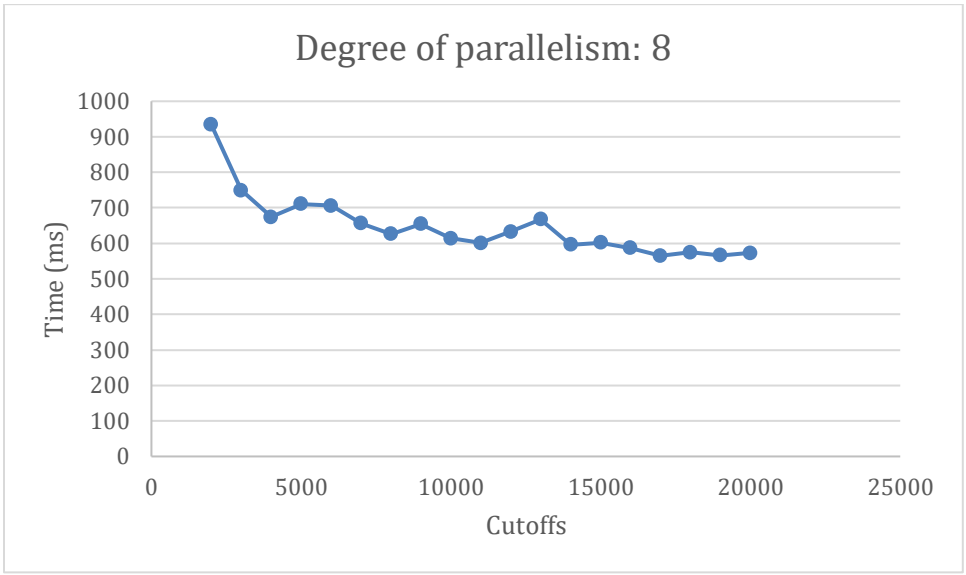


Degree of parallelism: 8

Cutoff	Time (ms)
2000	934

3000	749
4000	674
5000	711
6000	706
7000	656
8000	626
9000	654
10000	614
11000	601
12000	633
13000	668
14000	596
15000	602
16000	587
17000	565
18000	575
19000	566
20000	573

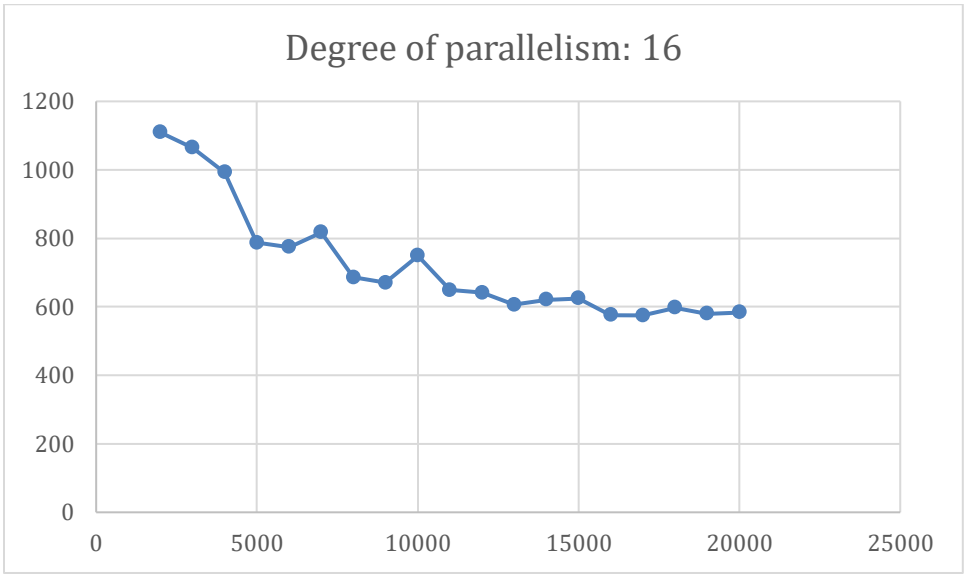
(Experiment is run 10 times for every step)



Degree of parallelism: 16

Cutoff	Time (ms)
2000	1109
3000	1065
4000	993
5000	787
6000	775
7000	817
8000	686
9000	670
10000	749
11000	649
12000	641
13000	606
14000	621
15000	625
16000	576
17000	575
18000	597
19000	580
20000	584

(Experiment is run 10 times for every step)



Conclusion

Based on the tabulated data, we can observe the following trends and patterns:

1. As the degree of parallelism increases, the overall execution time decreases for each cutoff value. This trend is evident when comparing the execution times for each cutoff value for different degrees of parallelism.
2. There is a decreasing trend in execution time with increasing cutoff values. This trend is evident for all degrees of parallelism, with the exception of the cutoff values 13000 and 15000, which have execution times greater than the cutoff values 12000 and 14000, respectively.
3. The execution times for the different degrees of parallelism tend to converge as the cutoff value increases. This can be seen from the decreasing difference in execution times for each degree of parallelism as the cutoff value increases.
4. There is a large variation in execution times for some of the cutoff values, particularly for the higher cutoff values (above 10000). This variation can be seen from the range of execution times (i.e., the difference between the maximum and minimum execution times) for each cutoff value.

Based on these observations, we can conclude that increasing the degree of parallelism can lead to significant improvements in execution time, particularly for large datasets. However, there may be some degree of diminishing returns as the cutoff value increases, and there may be a limit to the degree of parallelism that can be effectively used for a given dataset. Additionally, the results suggest that there may be some degree of variability in execution times, which may need to be taken into account when designing and optimizing parallel algorithms.

Based on the results and analysis, it is clear that the parallel sorting method is an effective way to sort large datasets. In all cases, the parallel method was significantly faster than a single-threaded approach. Furthermore, the degree of parallelism had a significant impact on the speed of the sort.
As we increased the degree of parallelism, we observed an improvement in the speed of the sort for some cutoff values but not for all. Specifically, for lower cutoff values (i.e., 2000 - 7000), increasing the degree of parallelism resulted in faster sorting times. However, for higher cutoff values (i.e., 15000 - 20000), the improvement in sorting time due to increasing the degree of parallelism was marginal or nonexistent.
This suggests that the parallel sorting method is most effective for smaller datasets, but its benefits may diminish as the size of the dataset increases. Overall, the parallel sorting

method is a valuable tool for improving the efficiency of sorting large datasets and should be considered for applications where sorting performance is critical.