CSCE 735 Homework 2

Name: Pranav Anantharam
UIN: 734003886

1. The revised code (thread-based parallel merge sort implementation) produced the below output for varying list sizes and thread counts:

Spreadsheet Image:

List Size	Number of Threads	Error	Time (in seconds)	Qsort Time (in seconds)
16	2	0	0.0004	0
16	4	0	0.0006	0
16	8	0	0.0009	0
1048576	16	0	0.0226	0.1803
16777216	256	0	0.2189	3.4486

The error count was reported zero for all scenarios tested.

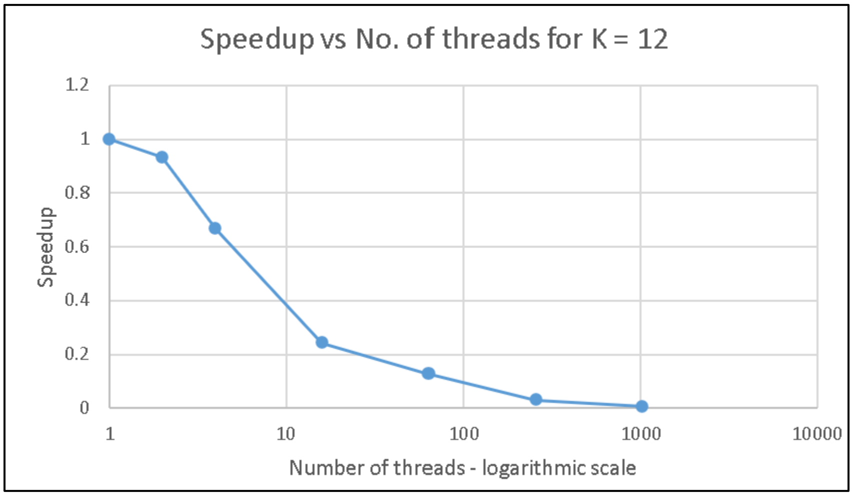
---

2. The plots for speedup and efficiency for varying values of k and q are given below:

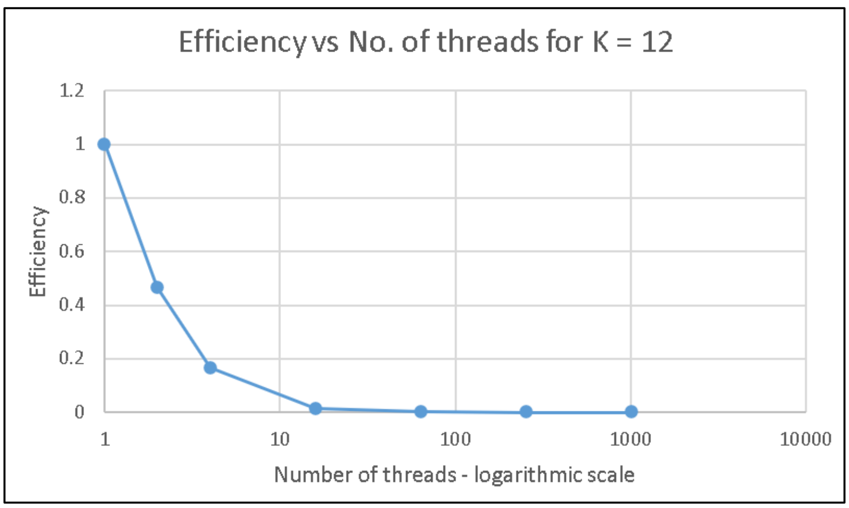
Spreadsheet Calculations:

List Size	Number of Threads	Error	Time (in seconds)	Qsort Time (in seconds)	Speedup	Efficiency
4096	1	0	0.0014	0	1	1
4096	2	0	0.0015	0	0.93333333	0.46666667
4096	4	0	0.0021	0	0.66666667	0.16666667
4096	16	0	0.0058	0	0.24137931	0.01508621
4096	64	0	0.011	0	0.12727273	0.00198864
4096	256	0	0.0463	0	0.03023758	0.00011812
4096	1024	0	0.2468	0	0.00567261	5.5397E-06
1048576	1	0	0.1846	0	1	1
1048576	2	0	0.0984	0	1.87601626	0.93800813
1048576	4	0	0.0558	0	3.30824373	0.82706093
1048576	16	0	0.0221	0	8.35294118	0.52205882
1048576	64	0	0.0279	0	6.61648746	0.10338262
1048576	256	0	0.059	0	3.12881356	0.01222193
1048576	1024	0	0.2742	0	0.67323122	0.00065745
268435456	1	0	65.2685	0	1	1
268435456	2	0	33.083	0	1.97287126	0.98643563
268435456	4	0	16.7397	0	3.89902447	0.97475612
268435456	16	0	4.3435	0	15.0267066	0.93916916
268435456	64	0	1.9684	0	33.1581488	0.51809607
268435456	256	0	1.8213	0	35.8362159	0.13998522
268435456	1024	0	2.4773	0	26.3466274	0.02572913

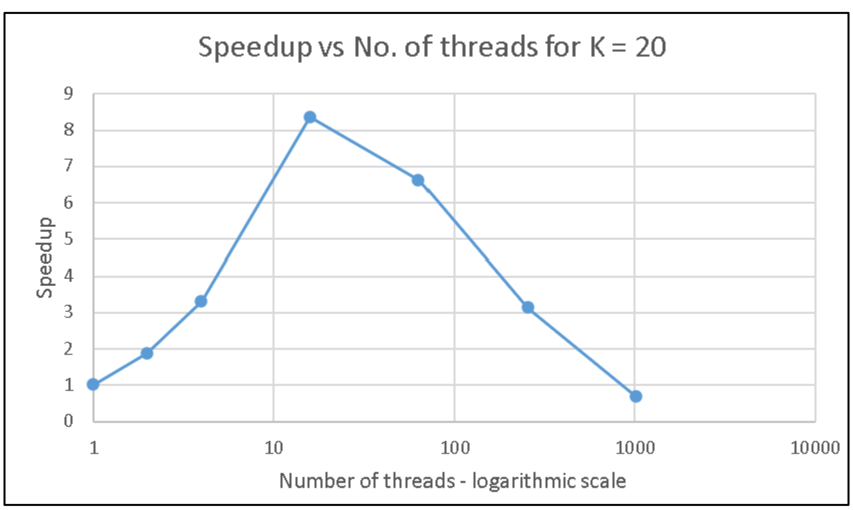
Speedup vs No. of threads for List Size = 4096 elements



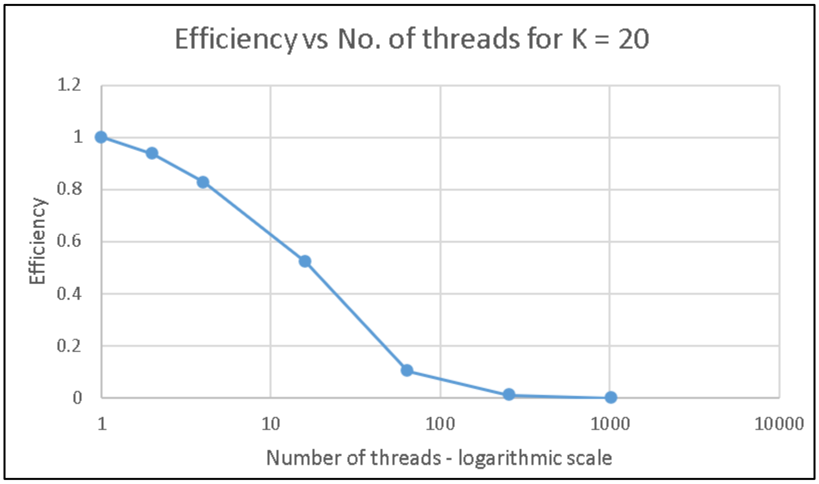
Efficiency vs No. of threads for List Size = 4096 elements



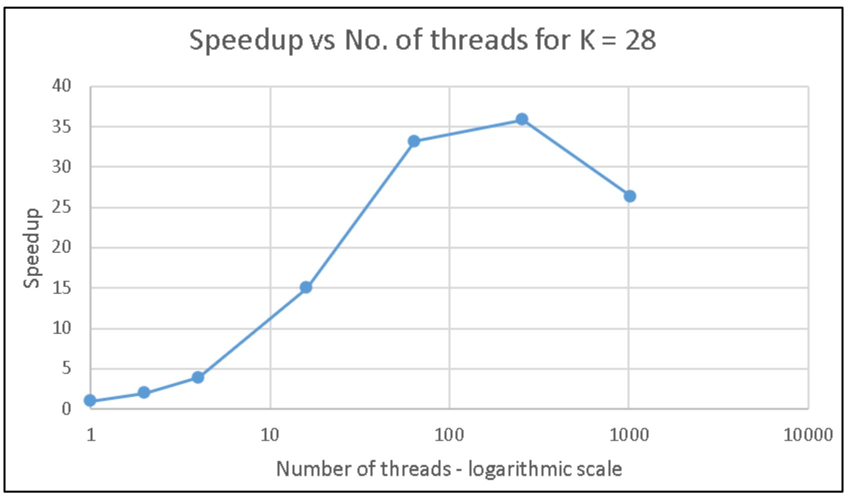
Speedup vs No. of threads for List Size = 1048576 elements



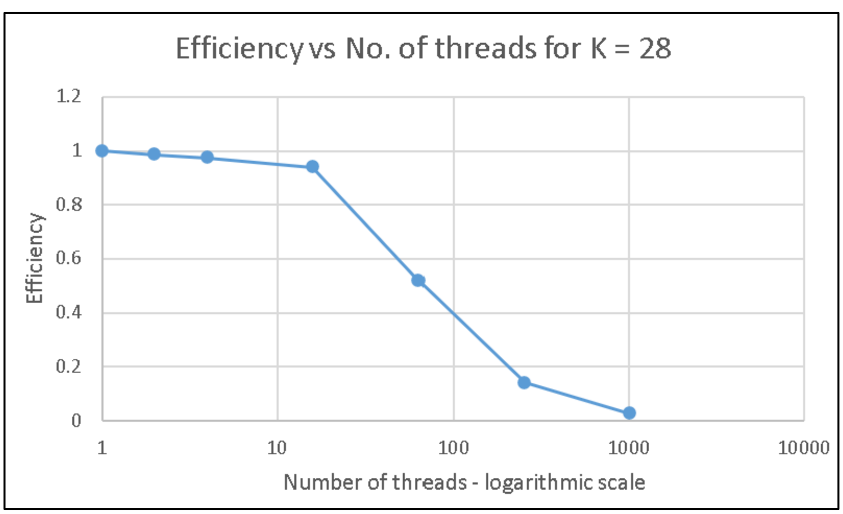
Efficiency vs No. of threads for List Size = 1048576 elements



Speedup vs No. of threads for List Size = 268435456 elements



Efficiency vs No. of threads for List Size = 268435456 elements



The results of the experiment align with my understanding of the expected behavior of the parallelized code.

For small values of K (K=12), the parallel merge sort implementation may take longer to perform sorting compared to the serial merge sort implementation. This can be attributed to the significant overhead involved in scheduling threads in comparison to the time duration taken to perform the sorting. And so, the serial merge sort implementation may be more efficient for the smaller list sizes.

For larger values of K (K = 20, 28), speedup increases the number of threads increases. This is expected to happen as more threads will be able to complete more work in parallel and the large list size allows for more parallelism. But the speedup decreases beyond the optimal number of threads. This occurs since the overhead associated with managing threads becomes significant as the number of threads increases.

Moreover, the increase in the number of threads for a constant list size displays a decrease in efficiency. This is since the amount of work assigned to each thread reduces with an increase in the number of threads. And so, the overhead involved in switching threads becomes significant, which results in a decrease in efficiency.

---

3. Consider the results of my experiments for K = 20 and K = 28, to demonstrate the variation of speedup and efficiency with list sizes and number of threads.

Variation of Speedup and Efficiency with Number of threads for K = 20

List Size	Number of Threads	Error	Time (in seconds)	Qsort Time (in seconds)	Speedup	Efficiency
1048576	1	0	0.1846	0	1	1
1048576	2	0	0.0984	0	1.876016	0.938008
1048576	4	0	0.0558	0	3.308244	0.827061
1048576	16	0	0.0221	0	8.352941	0.522059
1048576	64	0	0.0279	0	6.616487	0.103383
1048576	256	0	0.059	0	3.128814	0.012222
1048576	1024	0	0.2742	0	0.673231	0.000657

Variation of Speedup and Efficiency with Number of threads for K = 28

List Size	Number of Threads	Error	Time (in seconds)	Qsort Time (in seconds)	Speedup	Efficiency
268435456	1	0	65.2685	0	1	1
268435456	2	0	33.083	0	1.972871	0.986436
268435456	4	0	16.7397	0	3.899024	0.974756
268435456	16	0	4.3435	0	15.02671	0.939169
268435456	64	0	1.9684	0	33.15815	0.518096
268435456	256	0	1.8213	0	35.83622	0.139985
268435456	1024	0	2.4773	0	26.34663	0.025729

Consider the case where K = 20, with an increase in number of threads executing in parallel, the execution times decrease, and hence speedup increases. Specifically, when the number of threads is **16**, we observe a speedup of **8.35 (around 8 times faster than serial execution)** due to parallel execution while maintaining an efficiency of **52.20 %**. However, beyond a certain number of threads, the speedup decreases, and execution times increase due to thread management and context switch overheads.

A similar behavior is observed for K = 28, where the execution times decrease with an increase in number of threads. For example, when the number of threads is **256**, we observe a speedup of **35.83 (almost 36 times faster than serial execution)**. Beyond this point, the speedup decreases due to significant thread scheduling overheads.

From the above results, we can conclude that we have a well-designed parallel merge-sort implementation.
