CSCE 735 Fall 2022

Name: Rohan Chaudhury
UIN: 432001358

HW 2: Parallel Merge Sort Using Threads

1. (70 points) Revise the code to implement a thread-based parallel merge sort. The code should compile successfully and should report error=0 for the following instances:
./sort_list.exe 4 1
./sort_list.exe 4 2
./sort_list.exe 4 3
./sort_list.exe 20 4
./sort_list.exe 24 8

ANSWER:

The parallelized code is attached in the zip file. The logs are as follows:

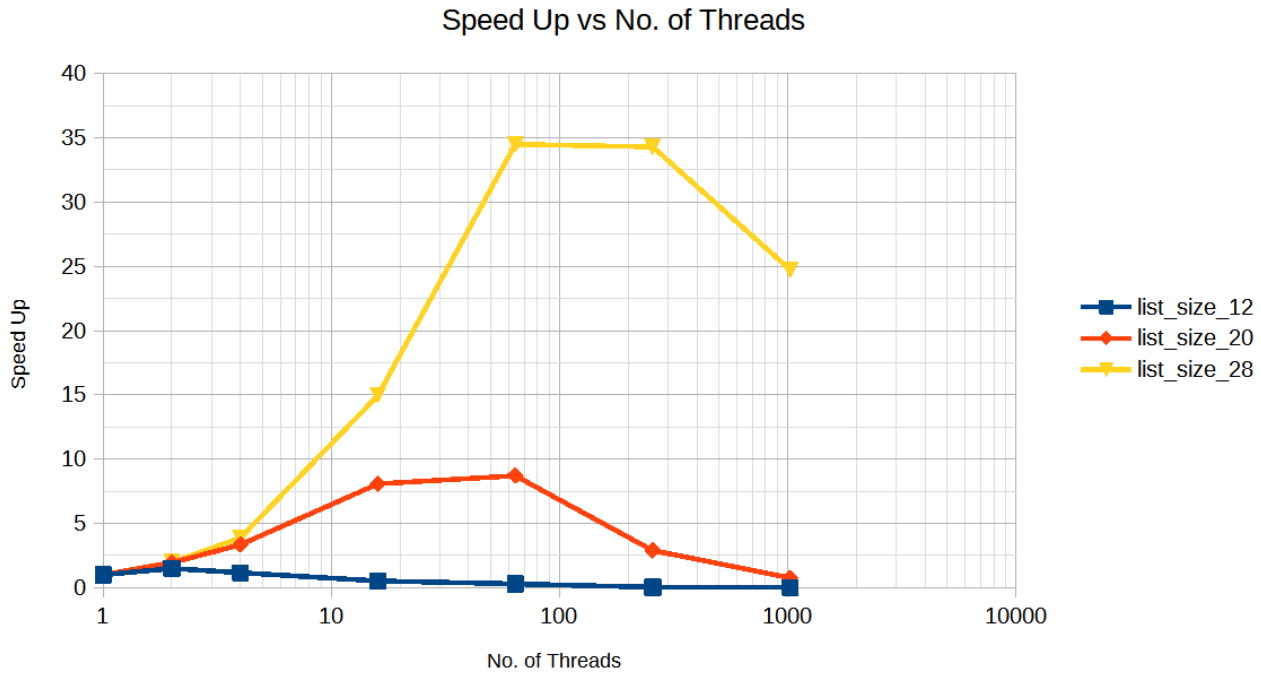
List Size = 16, Threads = 2, error = 0, time (sec) = 0.0005, qsort_time = 0.0000
List Size = 16, Threads = 4, error = 0, time (sec) = 0.0006, qsort_time = 0.0000
List Size = 16, Threads = 8, error = 0, time (sec) = 0.0009, qsort_time = 0.0000
List Size = 1048576, Threads = 16, error = 0, time (sec) = 0.0217, qsort_time = 0.1773
List Size = 16777216, Threads = 256, error = 0, time (sec) = 0.2168, qsort_time = 3.3305

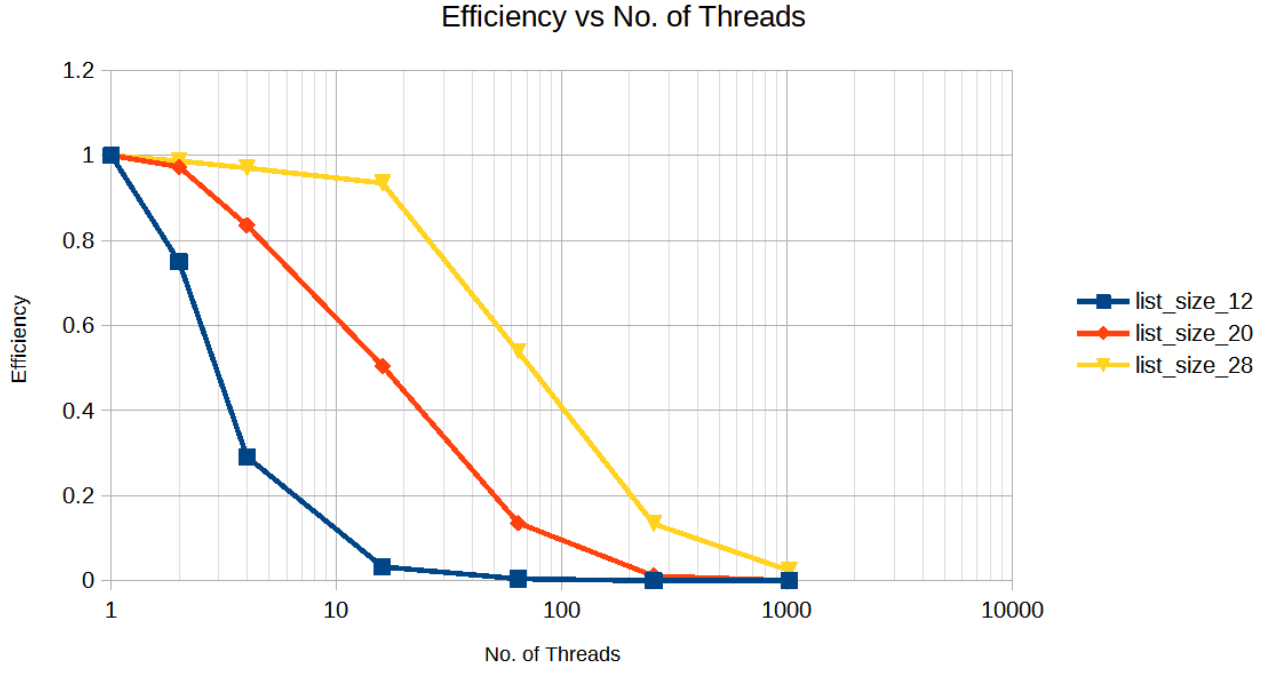
List Size	16	Threads	2	error	0	time (sec)	0.0005	qsort_time	0
List Size	16	Threads	4	error	0	time (sec)	0.0006	qsort_time	0
List Size	16	Threads	8	error	0	time (sec)	0.0009	qsort_time	0
List Size	1048576	Threads	16	error	0	time (sec)	0.0217	qsort_time	0.1773
List Size	16777216	Threads	256	error	0	time (sec)	0.2168	qsort_time	3.3305

As can be observed, the error returned for all of the above tests is 0. It should also be emphasized that for smaller list sizes, parallelization has no advantage over single threaded processing. For list sizes 20 and 24, the predicted performance increase over single threaded processing is seen.

2. (20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.

ANSWER:





We can see the following things from the two graphs above:

1. When k=12 (that is, list size = 2^12) is compared to k=20 or k=28, the speedup is quite small. Because employing many threads adds thread management cost when the input data size is comparatively small. When the time required to run the thread is similar to this cost, the efficiency of serial execution decreases.
2. The speedup increases till a point and then decreases for k=20 and k=28. This is due to the excessive number of threads, and the thread management cost begins to play a significant part in time consumed.

3. The number of threads required to achieve maximum speed is varied between k=20 and k=28. This is due to the work being divided into smaller jobs for the threads to do, resulting in a shorter overall time spent. However, after reaching the maximum speedup, raising the number of threads increases the time taken due to context switching across the threads.
4. The necessity for thread synchronization and resource management reduces efficiency as the number of threads increases.

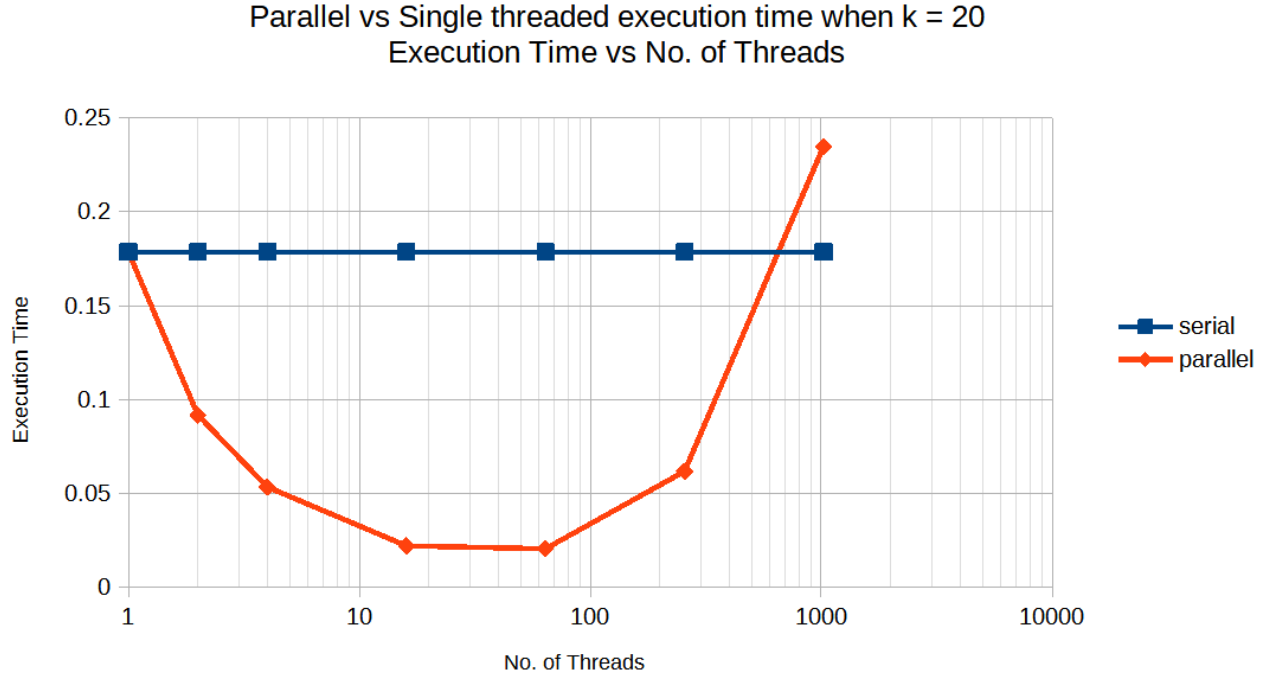
3. (10 points) Your code should demonstrate speedup when sorting lists of appropriate sizes. Determine two values of k for which your code shows speedup as q is varied. Present the timing results for your code along with speedup and efficiency obtained to convince the reader that you have a well-designed parallel merge sort. You may use results from experiments in previous problems or identify new values k and q to illustrate how well your code has been parallelized.

ANSWER:

As q was varied, two k values for speedup were observed: 20, 28.

Below is the table for k=20

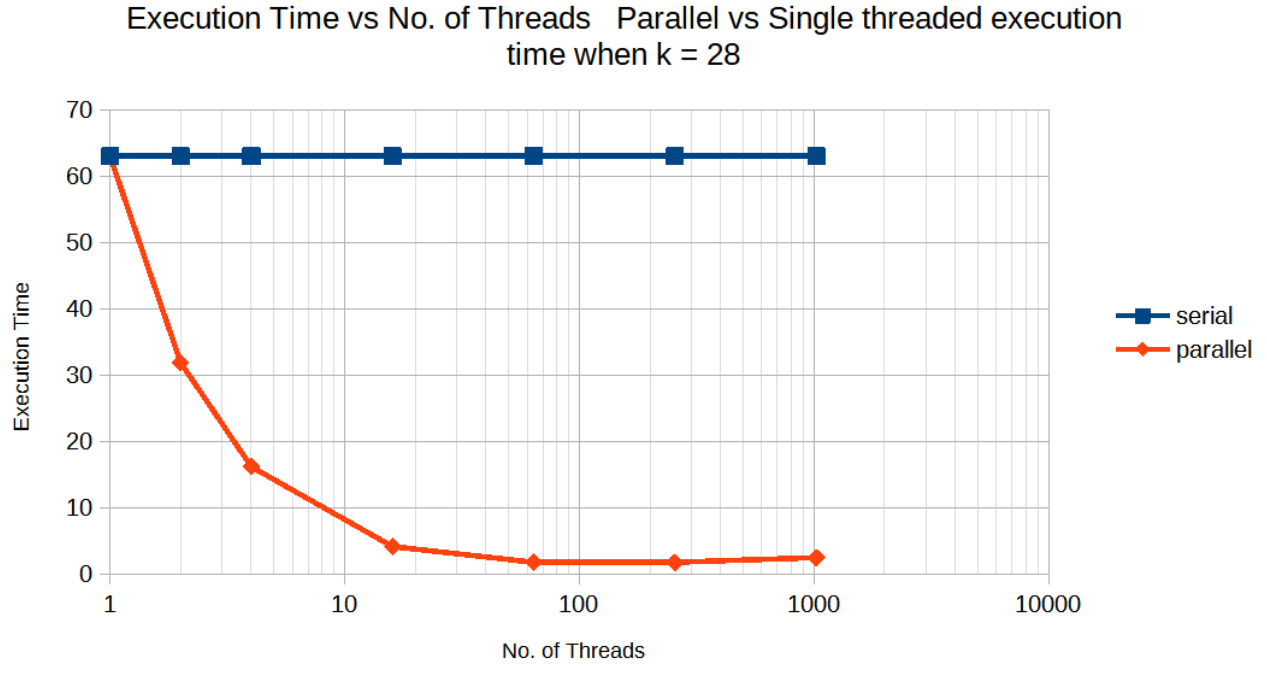
List Size	1048576	Threads	1	error	0	time (sec)	0.1785	qsort_time	0.1743	Speed up =	1	Efficiency =	1
List Size	1048576	Threads	2	error	0	time (sec)	0.0917	qsort_time	0.1769	Speed up =	1.946564885	Efficiency =	0.973282443
List Size	1048576	Threads	4	error	0	time (sec)	0.0534	qsort_time	0.1744	Speed up =	3.342696629	Efficiency =	0.835674157
List Size	1048576	Threads	16	error	0	time (sec)	0.0221	qsort_time	0.1767	Speed up =	8.076923077	Efficiency =	0.504807692
List Size	1048576	Threads	64	error	0	time (sec)	0.0205	qsort_time	0.1743	Speed up =	8.707317073	Efficiency =	0.136051829
List Size	1048576	Threads	256	error	0	time (sec)	0.0617	qsort_time	0.1747	Speed up =	2.893030794	Efficiency =	0.011300902
List Size	1048576	Threads	1024	error	0	time (sec)	0.2348	qsort_time	0.1729	Speed up =	0.760221465	Efficiency =	0.000742404



As demonstrated in the table and chart above, with k = 20, execution time falls with parallelization as the number of threads increases until it reaches the best option (at number of threads = 2^6 = 64) with the least runtime, 0.0205 seconds. Raising the number of threads has reduction effect on the execution time after this point. The graph compares execution durations in parallel mode (orange line) with single threaded mode (blue line) over q values, which represent the number of threads used.

Below is the table for k = 28

List Size	268435456	Threads	1	error	0	time (sec)	62.9848	qsort_time	62.5606	Speed up =	1	Efficiency = 1
List Size	268435456	Threads	2	error	0	time (sec)	31.9119	qsort_time	62.5535	Speed up =	1.973708867	Efficiency = 0.986854434
List Size	268435456	Threads	4	error	0	time (sec)	16.2256	qsort_time	62.5607	Speed up =	3.881816389	Efficiency = 0.970454097
List Size	268435456	Threads	16	error	0	time (sec)	4.2106	qsort_time	62.5397	Speed up =	14.95862822	Efficiency = 0.934914264
List Size	268435456	Threads	64	error	0	time (sec)	1.8261	qsort_time	62.891	Speed up =	34.49142982	Efficiency = 0.538928591
List Size	268435456	Threads	256	error	0	time (sec)	1.8367	qsort_time	62.5081	Speed up =	34.29237219	Efficiency = 0.133954579
List Size	268435456	Threads	1024	error	0	time (sec)	2.5465	qsort_time	63.1247	Speed up =	24.73387002	Efficiency = 0.02415417



As depicted in the table and graph above, for k = 28, execution time falls with parallelization as the number of threads increases until it reaches the best choice (at number of threads = 2^6 = 64) with the shortest runtime, i.e., 1.8261 sec. After this, increasing the number of threads has no effect on execution time reduction, as seen by the progressive increase in execution times beyond number of threads = 2^6 = 64. The graph compares execution durations in parallel mode (orange line) with single threaded mode (blue line) over q values, which represent the number of threads used.