CSCE 735 Fall 2022

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

   Soln: Code is archived in the zip file. The following are the results obtained in the dedicated mode:

List Size	16	Threads	2	error	0	time (sec)	0.0004	qsort_time	0
List Size	16	Threads	4	error	0	time (sec)	0.0005	qsort_time	0
List Size	16	Threads	8	error	0	time (sec)	0.0008	qsort_time	0
List Size	1048576	Threads	16	error	0	time (sec)	0.023	qsort_time	0.1773
List Size	16777216	Threads	256	error	0	time (sec)	0.2314	qsort_time	3.3283

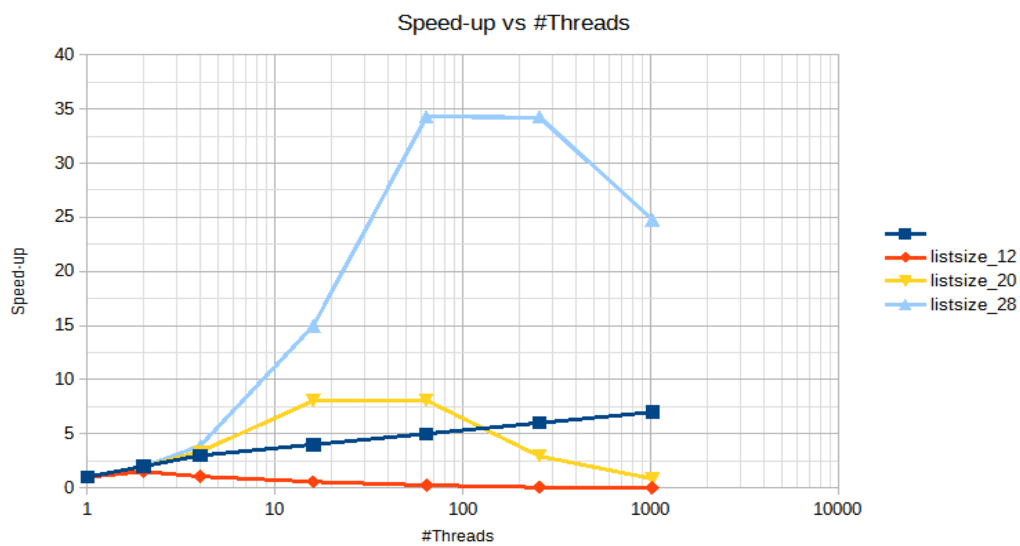
   As can be seen, the error returned is 0 for all the above tests.

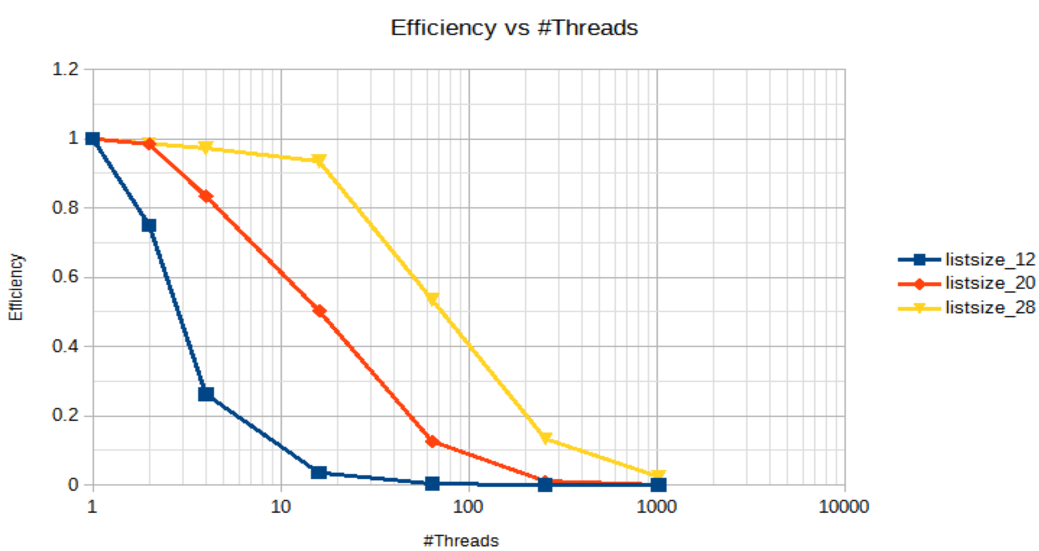
2. **(20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.**

   Soln: When the data in the speed up chart are examined in more detail, the blue, red, and yellow lines represent the speed up in relation to the number of threads across various values of k, or the size of the list to be sorted. In compared to k = 20, or k = 28, the speed improvement appears to be essentially nonexistent at the lowest k number, i.e. k = 12. This is to be expected since using many threads to process little amounts of data or input may increase the execution time due to higher thread management overhead than single threaded execution.
   When this occurs, the task's execution time in a single threaded environment is similar to the thread management overhead time. However, as k increases, single-threaded execution becomes slower and slower, therefore attempts at parallelization won't be able to catch up to the parallelized execution time with simply thread management overhead timings. It is clear from k = 20 and k = 28 that as the number of threads increases, performance improves up to a point and then tends to degrade as the number of threads increases owing to the cost associated with thread management.
   Another pattern that emerges from this is that when list size, or k value, increases, the limit is achieved at larger thread counts. This may be illustrated using work division principles.

There is a sweet spot where the number of work units that can be processed by a thread is optimal, below which the CPU will spend more time on thread handling than actually performing the task for the thread and above which it may require multiple context switches or extensive memory accesses to process its workload leading to high execution time.
On the basis of this knowledge, it is anticipated that greater thread counts would result in the best performance for larger list sizes.

Finally, it can be observed from the efficiency vs. thread count graph that efficiency declines as thread count rises. Due to thread synchronization and competition for processing resources, this has occurred.





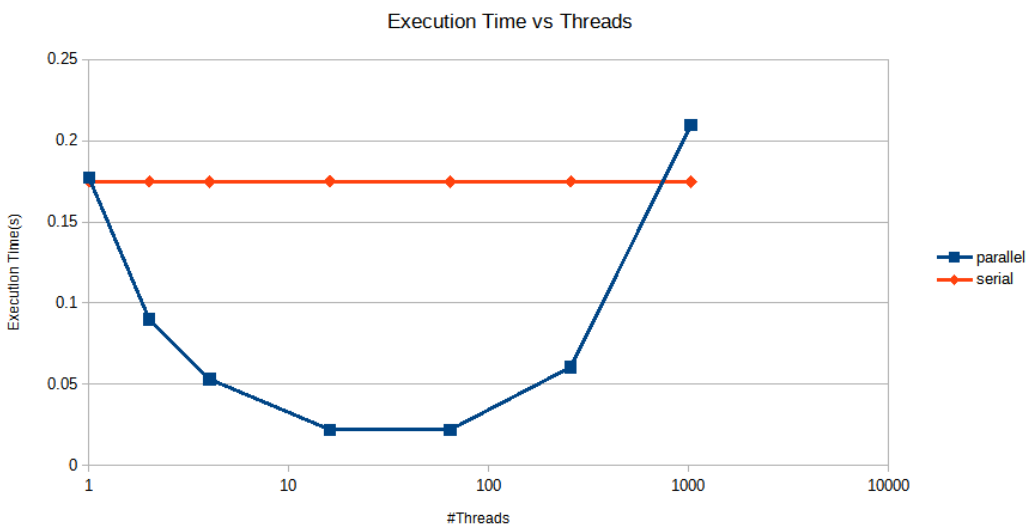
3. **(10 points) Your code should demonstrate speedup when sorting lists of appropriate sizes. Determine two values of k for which your code shows speedup as q is varied. Present the timing results for your code along with speedup and efficiency obtained to convince the reader that you have a well-designed 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.**

Soln: Two k values for speedup was observed as q was varied are : 20, 28.

k=20

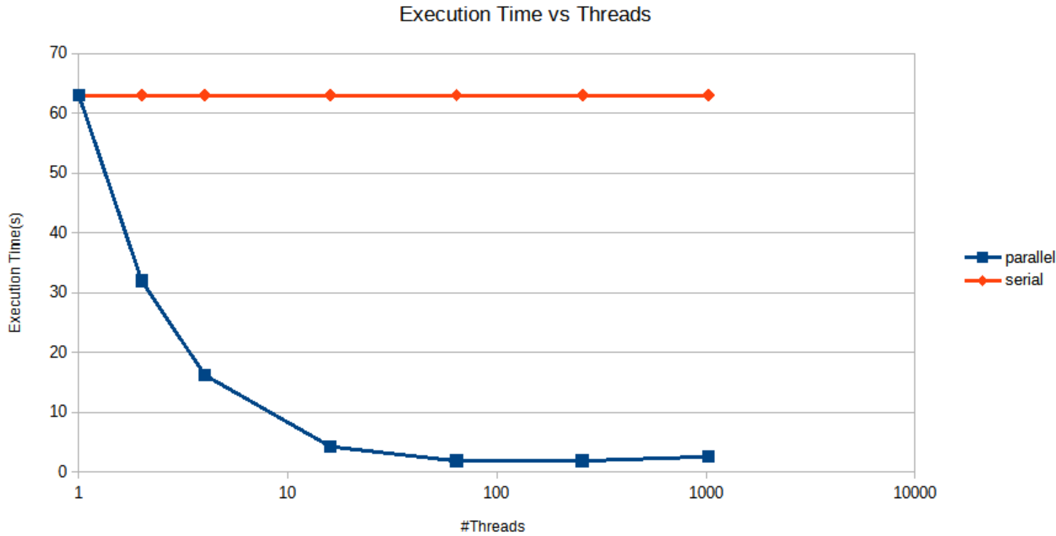
List Size	1048576	Threads	1	error	0	time (sec)	0.1772	qsort_time	0.1748	Speedup	1	Efficiency	1
List Size	1048576	Threads	2	error	0	time (sec)	0.09	qsort_time	0.173	Speedup	1.96888889	Efficiency	0.98444444
List Size	1048576	Threads	4	error	0	time (sec)	0.0531	qsort_time	0.1752	Speedup	3.33709981	Efficiency	0.83427495
List Size	1048576	Threads	16	error	0	time (sec)	0.022	qsort_time	0.178	Speedup	8.05454545	Efficiency	0.50340909
List Size	1048576	Threads	64	error	0	time (sec)	0.0219	qsort_time	0.1743	Speedup	8.0913242	Efficiency	0.12642694
List Size	1048576	Threads	256	error	0	time (sec)	0.0605	qsort_time	0.1724	Speedup	2.92892562	Efficiency	0.01144112
List Size	1048576	Threads	1024	error	0	time (sec)	0.2097	qsort_time	0.1718	Speedup	0.84501669	Efficiency	0.00082521

The chart and table above show that, for k = 20, execution time lowers with parallelization as the number of threads increases, until it approaches the ideal configuration (q = 6), where the lowest execution time, i.e., 0.0219 sec, is shown. After this point, adding more threads has no further effect on the execution time. Within a range of q values, or the number of threads employed, the chart compares execution durations in parallel mode (blue line) and single threaded mode (red line).



k=28

List Size	268435456	Threads	1	error	0	time (sec)	63.0536	qsort_time	62.6226	Speedup	1	Efficiency	1
List Size	268435456	Threads	2	error	0	time (sec)	31.9707	qsort_time	62.5856	Speedup	1.97223082	Efficiency	0.98611541
List Size	268435456	Threads	4	error	0	time (sec)	16.2081	qsort_time	62.555	Speedup	3.8902524	Efficiency	0.9725631
List Size	268435456	Threads	16	error	0	time (sec)	4.2148	qsort_time	62.5659	Speedup	14.9600456	Efficiency	0.93500285
List Size	268435456	Threads	64	error	0	time (sec)	1.8405	qsort_time	62.5531	Speedup	34.2589514	Efficiency	0.53529612
List Size	268435456	Threads	256	error	0	time (sec)	1.841	qsort_time	62.5433	Speedup	34.2496469	Efficiency	0.13378768
List Size	268435456	Threads	1024	error	0	time (sec)	2.5459	qsort_time	63.0577	Speedup	24.766723	Efficiency	0.02418625



The chart and table above show that for k = 28, execution time lowers with parallelization as the number of threads is raised until it approaches the ideal configuration (q = 6), where the lowest execution time, i.e., 1.8405 sec, is seen. The progressive rise in execution durations beyond q = 6 shows that after this point, adding threads doesn't help reduce execution times. Within a range of q values, or the number of threads employed, the chart compares execution durations in parallel mode (blue line) and single threaded mode (red line).