CSCE 735 Fall 2022

HW 3: Parallel Merge Sort Using OpenMP

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

Soln: 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 2^20 and 2^24, the projected performance increase over single threaded processing is visible.

List Size	16	Threads	2	error	0	time (sec)	0.0068	qsort_time	0
List Size	16	Threads	4	error	0	time (sec)	0.0064	qsort_time	0
List Size	16	Threads	8	error	0	time (sec)	0.006	qsort_time	0
List Size	1048576	Threads	16	error	0	time (sec)	0.0281	qsort_time	0.1718
List Size	16777216	Threads	256	error	0	time (sec)	0.6992	qsort_time	3.4767

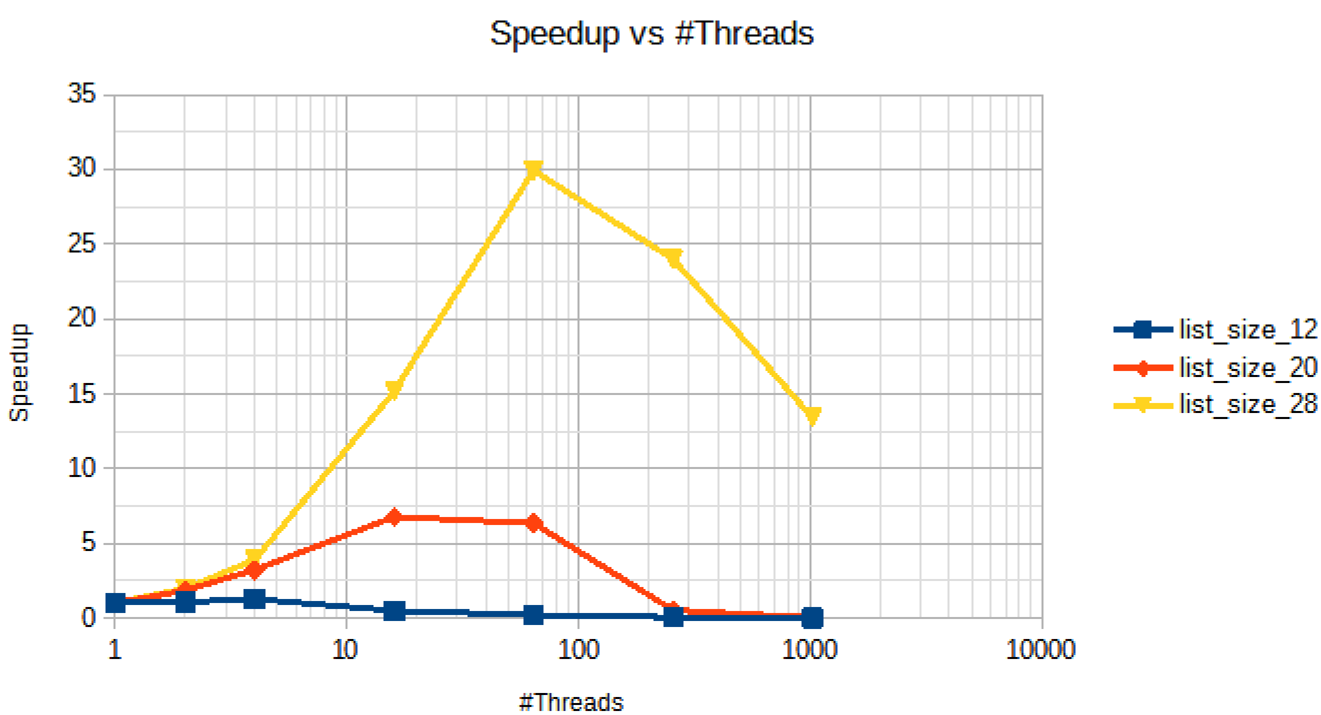
2. (20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.

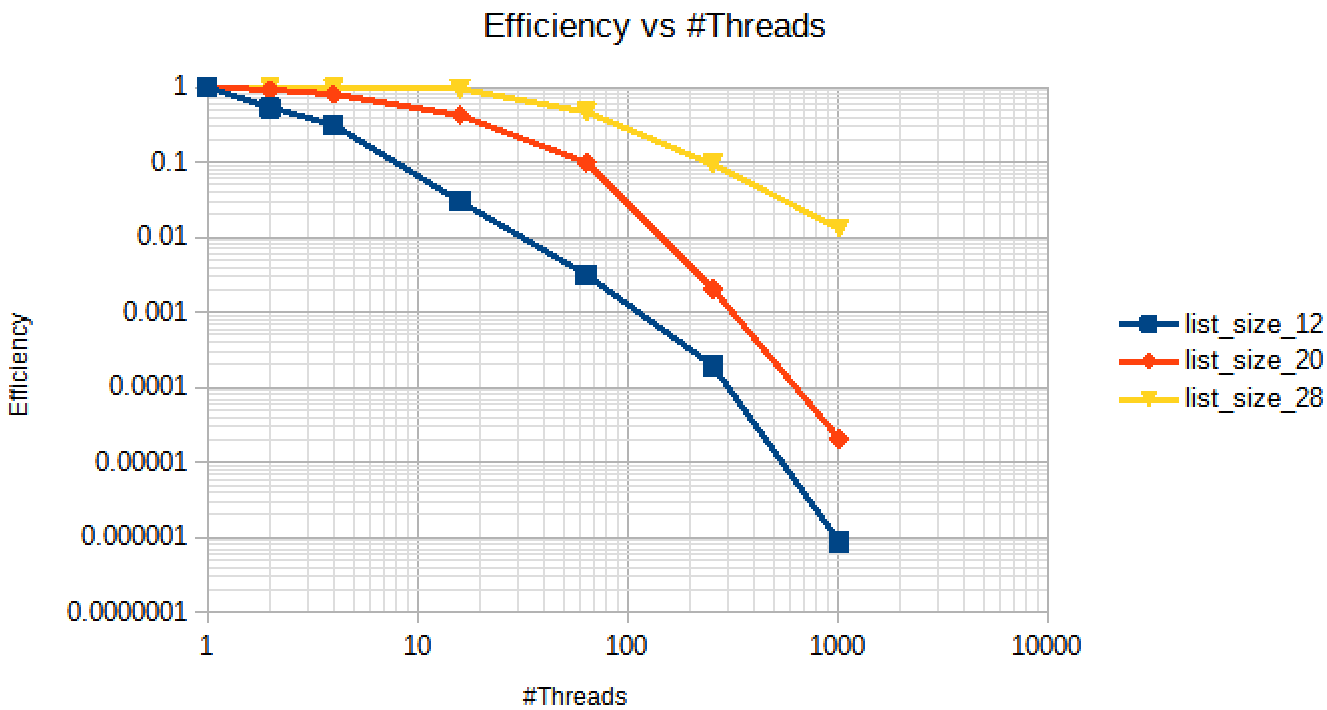
Soln:

List Size	4096	Threads	1	error	0	time (sec)	0.0077	qsort_time	0.001	Speedup	1	Efficiency	1
List Size	4096	Threads	2	error	0	time (sec)	0.0072	qsort_time	0.001	Speedup	1.0694444444	Efficiency	0.534722222
List Size	4096	Threads	4	error	0	time (sec)	0.0061	qsort_time	0.0007	Speedup	1.262295082	Efficiency	0.31557377
List Size	4096	Threads	16	error	0	time (sec)	0.016	qsort_time	0.0007	Speedup	0.48125	Efficiency	0.030078125
List Size	4096	Threads	64	error	0	time (sec)	0.038	qsort_time	0.0004	Speedup	0.2026315789	Efficiency	0.003166118
List Size	4096	Threads	256	error	0	time (sec)	0.1547	qsort_time	0.0004	Speedup	0.0497737557	Efficiency	0.000194429
List Size	4096	Threads	1024	error	0	time (sec)	8.5961	qsort_time	0.0004	Speedup	0.0008957551	Efficiency	8.74761E-07

List Size	1048576	Threads	1	error	0	time (sec)	0.1846	qsort_time	0.1717	Speedup	1	Efficiency	1
List Size	1048576	Threads	2	error	0	time (sec)	0.1003	qsort_time	0.1711	Speedup	1.8404785643	Efficiency	0.920239282
List Size	1048576	Threads	4	error	0	time (sec)	0.0572	qsort_time	0.172	Speedup	3.2272727273	Efficiency	0.806818182
List Size	1048576	Threads	16	error	0	time (sec)	0.0273	qsort_time	0.1707	Speedup	6.7619047619	Efficiency	0.422619048
List Size	1048576	Threads	64	error	0	time (sec)	0.0291	qsort_time	0.1723	Speedup	6.3436426117	Efficiency	0.099119416
List Size	1048576	Threads	256	error	0	time (sec)	0.3519	qsort_time	0.2622	Speedup	0.5245808468	Efficiency	0.002049144
List Size	1048576	Threads	1024	error	0	time (sec)	8.7011	qsort_time	0.2256	Speedup	0.0212157084	Efficiency	2.07185E-05

List Size	268435456	Threads	1	error	0	time (sec)	62.5417	qsort_time	62.488	Speedup	1	Efficiency	1
List Size	268435456	Threads	2	error	0	time (sec)	31.8405	qsort_time	62.5202	Speedup	1.9642185267	Efficiency	0.982109263
List Size	268435456	Threads	4	error	0	time (sec)	16.0605	qsort_time	62.6563	Speedup	3.894131565	Efficiency	0.973532891
List Size	268435456	Threads	16	error	0	time (sec)	4.1302	qsort_time	62.6372	Speedup	15.14253547	Efficiency	0.946408467
List Size	268435456	Threads	64	error	0	time (sec)	2.0907	qsort_time	62.653	Speedup	29.91423925	Efficiency	0.467409988
List Size	268435456	Threads	256	error	0	time (sec)	2.6074	qsort_time	63.2016	Speedup	23.986231495	Efficiency	0.093696217
List Size	268435456	Threads	1024	error	0	time (sec)	4.6621	qsort_time	62.8635	Speedup	13.414920315	Efficiency	0.013100508



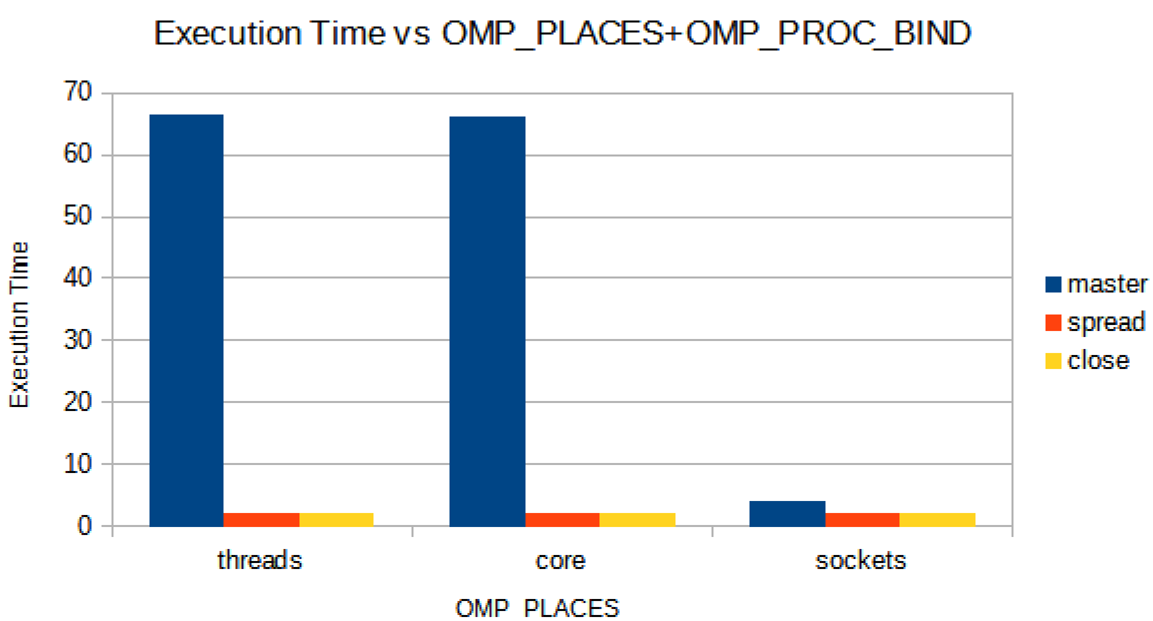


As compared to k = 20, or k = 28, the speed up appears to be essentially non-existent at the lowest k number, i.e., k = 12. This is to be expected since using many threads when the amount of input or data to be processed is modest would increase the execution time due to higher thread management cost than using a single thread. When the thread management overhead time is equivalent to the time required to complete the operation in a single threaded environment, this is the condition. However, as k increases, single threaded execution takes substantially longer, and hence attempts at parallelization will not eclipse the parallelized execution time with only thread management overhead timings. This is clear for k = 20 and k = 28, where increasing the number of threads improves performance until a maximum is reached, beyond which performance tends to decrease due to the previously noted thread management overheads. Another trend seen here is that as the size of the list, or k value, increases, the maximum is achieved at increasing thread counts. Work division notions can be used to illustrate this. There is a sweet spot where the number of work units that a thread can process is optimal, above which it may require multiple context switches or extensive memory accesses to process its workload, resulting in a long execution time, and below which the CPU will spend more time on thread handling rather than executing the thread's task. Based on this concept, greater list sizes are predicted to provide optimal performance at higher thread counts. Last but not least, the efficiency vs. thread count graph demonstrates that efficiency declines as thread count rises. This results from thread synchronization and competition for computing resources.

3. (10 points) For the instance with k = 28 and q = 5 experiment with different choices for OMP_PLACES and OMP_PROC_BIND to see how the parallel performance of the code is impacted. Explain your observations.

Soln:

List Size	268435456	Threads	32	error	0	time (sec)	2.1354	qsort_time	62.9506	FALSE	FALSE
List Size	268435456	Threads	32	error	0	time (sec)	66.411	qsort_time	63.2341	master	threads
List Size	268435456	Threads	32	error	0	time (sec)	66.0739	qsort_time	62.8416	master	core
List Size	268435456	Threads	32	error	0	time (sec)	3.9601	qsort_time	62.6805	master	sockets
List Size	268435456	Threads	32	error	0	time (sec)	2.1308	qsort_time	63.0451	spread	threads
List Size	268435456	Threads	32	error	0	time (sec)	2.1305	qsort_time	63.0183	spread	core
List Size	268435456	Threads	32	error	0	time (sec)	2.1528	qsort_time	62.5315	spread	sockets
List Size	268435456	Threads	32	error	0	time (sec)	2.1699	qsort_time	62.6303	close	threads
List Size	268435456	Threads	32	error	0	time (sec)	2.1743	qsort_time	62.9999	close	core
List Size	268435456	Threads	32	error	0	time (sec)	2.1518	qsort_time	62.5315	close	sockets



When OMP_PLACES is set to "Threads," as is the case on the first cluster from the left, a noticeably longer execution time is seen when OMP_PROC_BIND is set to "Master." For the identical pair of variables in the second cluster from the left, the same observation is found with the "Cores" and "Master" combination. According to definition, the full team of logical threads will be scheduled at the same place where the master thread is present when OMP_PROC_BIND is set to "Master." The performance will be the same whether OMP_PLACES is set to "Cores" or "Threads" with OMP_PROC_BIND set to "Master" because a Grace core only contains one hardware thread. The execution time for the same OMP_PROC_BIND configuration is significantly reduced when OMP_PLACES is set to "Sockets," which may be attributed to the fact that a socket in Grace has 24 hardware threads available, allowing it to execute numerous logical threads in parallel and enhance performance.

As neither a common collection of data nor an independent set of data are being used exclusively by the application, due to this lack of distinction, both "Spread" and "Close" would produce similar results. It is because each configuration will help either stage of the merge process, i.e., the first stage where each thread works on its own smaller subset of the main list, or the last stage where each thread works on its shared set of larger subsets of the main list.