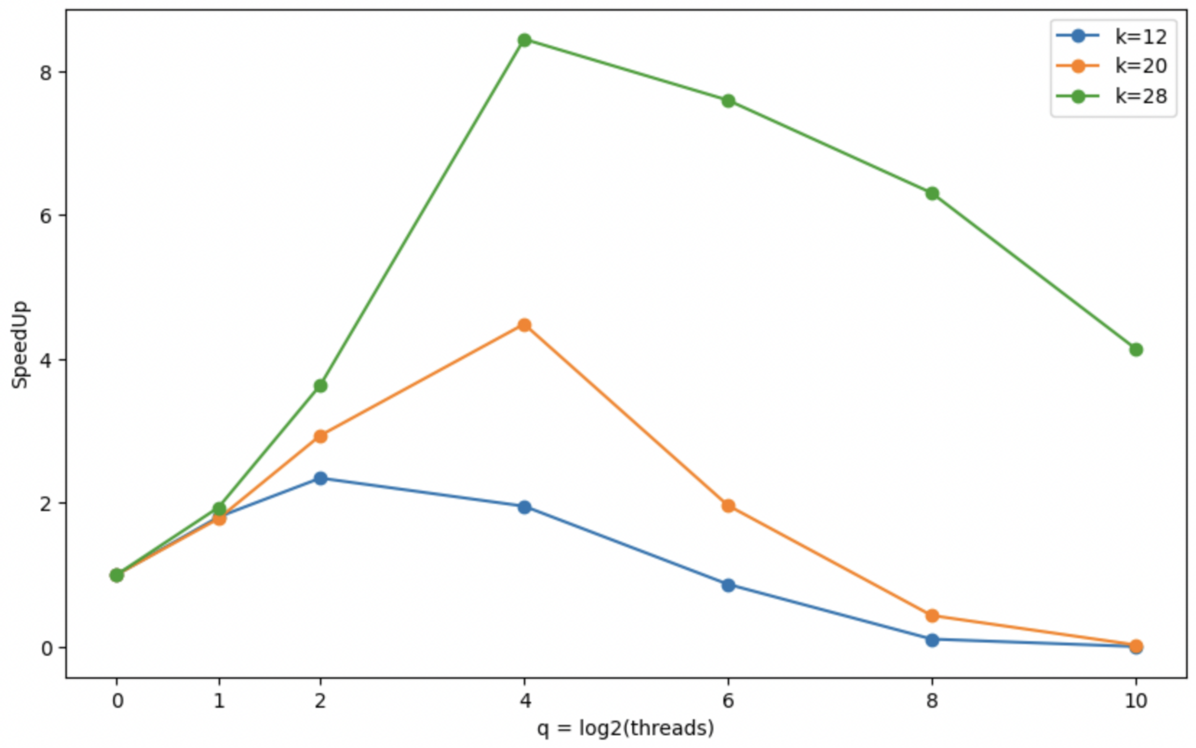
CSCE 735 Parallel Computing

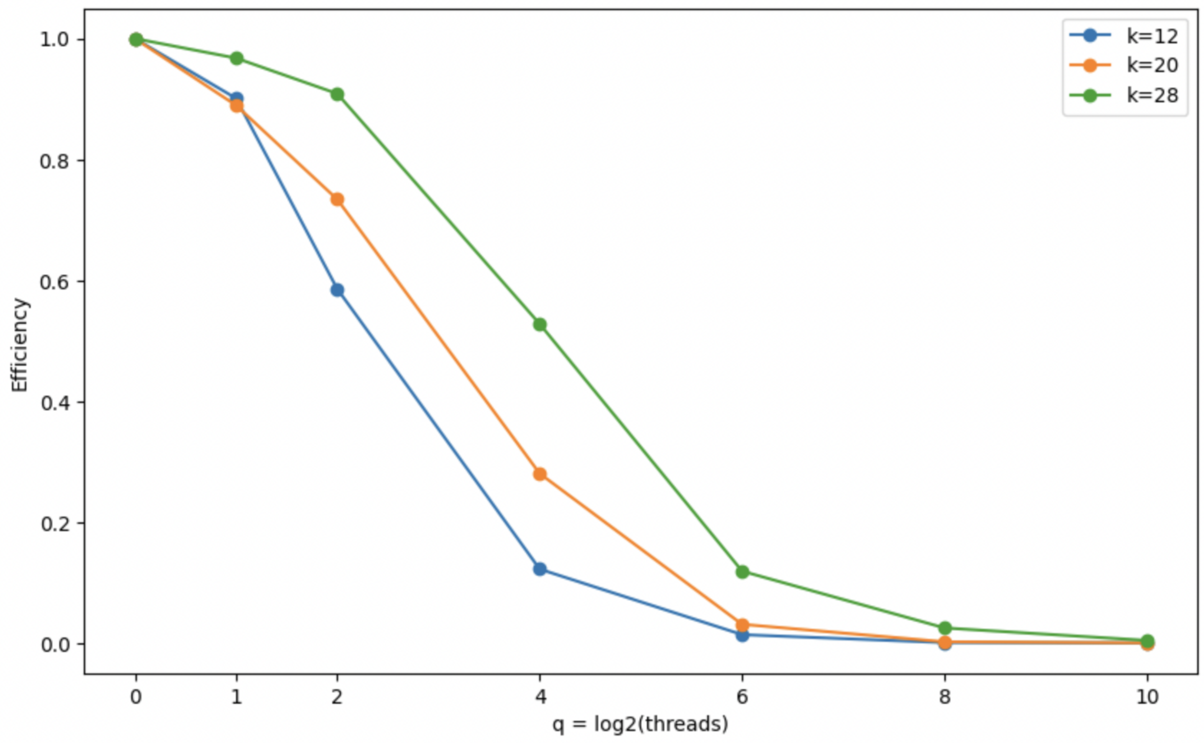
HW 3: Parallel Merge Sort Using OpenMP

1. **OpenMP sort:**
   The code was edited to support OpenMP. Below is the output of the program for the given samples.

List Size (k)	Threads (q)	Error	Time (sec)	QSort Time (sec)
16 (4)	2 (1)	0	0.0327	0.0000
16 (4)	4 (2)	0	0.0065	0.0000
16 (4)	8 (3)	0	0.0141	0.0000
1048576 (20)	16 (4)	0	0.0398	0.1711
16777216 (24)	256 (8)	0	1.0749	3.4745

2. **Efficiency and SpeedUp plots for k = 12,20,28 and q = 0,1,2,4,6,8,10**





1. From the speedUp graph, we can infer that there is not much improvement in speedup for k=12. This might be because the thread creation overhead takes more time than the task itself. This can be also observed when we run the program without multithreading(see below). The execution time is lesser for non-parallel code for smaller values of k.
2. The efficiency of execution does not improve much after q=6. As we can see, speedup and efficiency at q=8 and q=10 is almost the same for all k. This shows that, even if we increase it to execute in more threads, the speed up is not significant.
3. When the thread size is constant, we observe that execution time increases as the size of the list increases which is the expected behavior.

Log for the experiments:

```
List Size = 4096, Threads = 1, error = 0, time (sec) =   0.0164, qsort_time =   0.0010
List Size = 4096, Threads = 2, error = 0, time (sec) =   0.0091, qsort_time =   0.0009
List Size = 4096, Threads = 4, error = 0, time (sec) =   0.0070, qsort_time =   0.0010
List Size = 4096, Threads = 16, error = 0, time (sec) =   0.0084, qsort_time =   0.0007
List Size = 4096, Threads = 64, error = 0, time (sec) =   0.0189, qsort_time =   0.0004
List Size = 4096, Threads = 256, error = 0, time (sec) =   0.1555, qsort_time =   0.0004
List Size = 4096, Threads = 1024, error = 0, time (sec) =   7.9191, qsort_time =   0.0005

List Size = 1048576, Threads = 1, error = 0, time (sec) =   0.1820, qsort_time =   0.1706
List Size = 1048576, Threads = 2, error = 0, time (sec) =   0.1023, qsort_time =   0.1723
List Size = 1048576, Threads = 4, error = 0, time (sec) =   0.0620, qsort_time =   0.1721
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0406, qsort_time =   0.1718
List Size = 1048576, Threads = 64, error = 0, time (sec) =   0.0927, qsort_time =   0.1727
List Size = 1048576, Threads = 256, error = 0, time (sec) =   0.4195, qsort_time =   0.1723
List Size = 1048576, Threads = 1024, error = 0, time (sec) =   7.3237, qsort_time =   0.2400

List Size = 268435456, Threads = 1, error = 0, time (sec) =  62.5765, qsort_time =  62.5069
List Size = 268435456, Threads = 2, error = 0, time (sec) =  32.3344, qsort_time =  62.5484
List Size = 268435456, Threads = 4, error = 0, time (sec) =  17.2146, qsort_time =  62.7187
List Size = 268435456, Threads = 16, error = 0, time (sec) =   7.4115, qsort_time =  62.7101
List Size = 268435456, Threads = 64, error = 0, time (sec) =   8.2399, qsort_time =  62.5878
List Size = 268435456, Threads = 256, error = 0, time (sec) =   9.9235, qsort_time =  62.8006
List Size = 268435456, Threads = 1024, error = 0, time (sec) =  15.1031, qsort_time =  62.6101
```

3. k = 28 and q = 5, altering OMP_PLACES and OMP_PROC_BIND

OMP_PLACES	**OMP_PROC_BIND**	**Performance**
threads	spread	time (sec) = 7.2675, qsort_time = 62.7480
threads	master	time (sec) = 65.8533, qsort_time = 62.2383
threads	close	time (sec) = 6.1644, qsort_time = 62.6294
cores	spread	time (sec) = 6.6068, qsort_time = 62.5885
cores	master	time (sec) = 66.1125, qsort_time = 62.7802
cores	close	time (sec) = 6.1610, qsort_time = 62.6075
sockets	spread	time (sec) = 6.9923, qsort_time = 62.6025
sockets	master	time (sec) = 7.0568, qsort_time = 62.6571
sockets	close	time (sec) = 6.9809, qsort_time = 62.5711

For OMP_PLACES as cores and threads there isn't much difference in performance. This could be because many threads cannot be configured in a single core in Grace.

The execution time is lower in case of OMP_PLACES=sockets with OMP_PROC_BIND as master compared to others. The sockets can accommodate up to 32 threads or tasks. Even if all the threads are assigned to the master thread, it can still manage 32 tasks concurrently. By parallelizing, we can achieve a quick execution time of 7.0568 seconds.