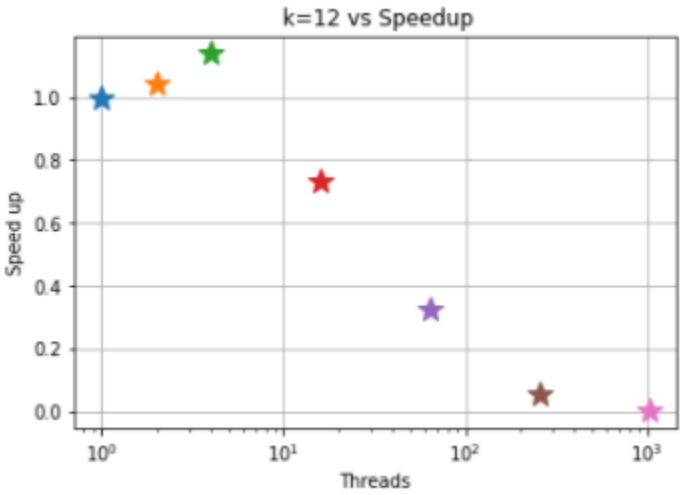
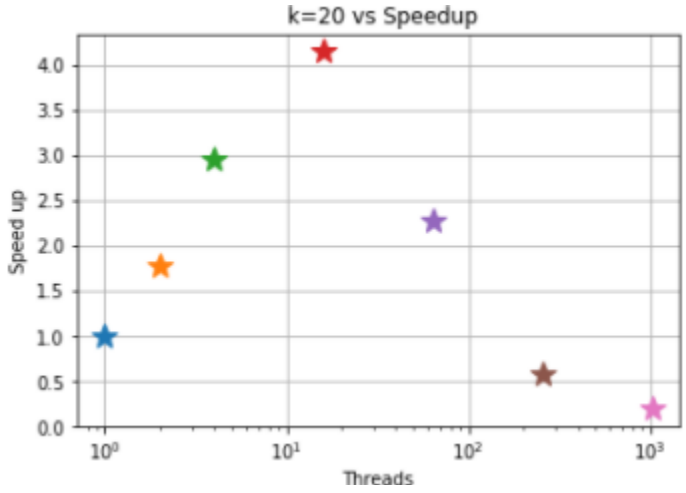
HW3: Parallel Merge Sort Using OpenMP

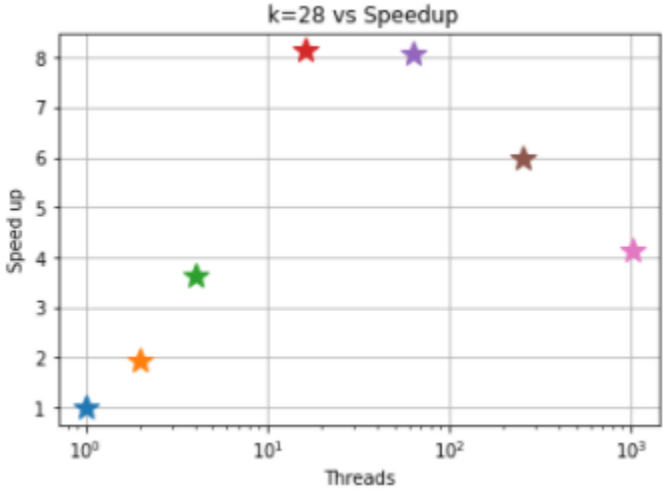
1) The code is Revised to implement parallel merge sort via OpenMP. The code compiled successfully and reported error=0 for the following instances:

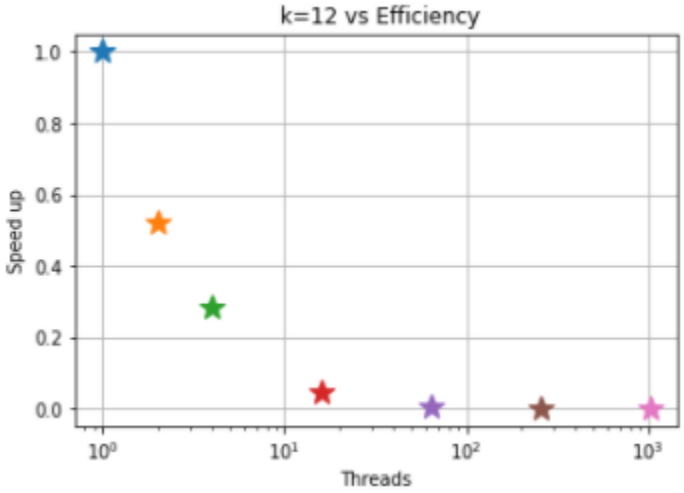
```
[nimoshika@grace1 HW3-735]$ ./sort_list_openmp.exe 4 1
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0056, qsort_time =   0.0001
[nimoshika@grace1 HW3-735]$ ./sort_list_openmp.exe 4 2
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0064, qsort_time =   0.0000
[nimoshika@grace1 HW3-735]$ ./sort_list_openmp.exe 4 3
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0081, qsort_time =   0.0000
[nimoshika@grace1 HW3-735]$ ./sort_list_openmp.exe 20 4
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0456, qsort_time =   0.1370
[nimoshika@grace1 HW3-735]$ ./sort_list_openmp.exe 24 8
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.8925, qsort_time =   2.6408
[nimoshika@grace1 HW3-735]$
```

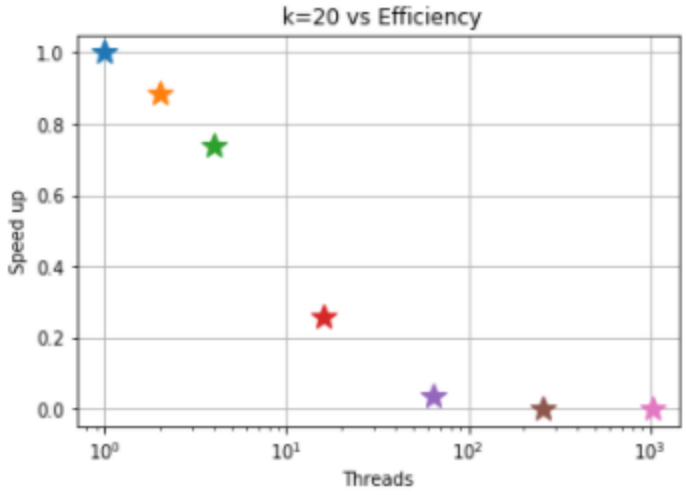
2) speedup and efficiency for all combinations of k and q are plotted.

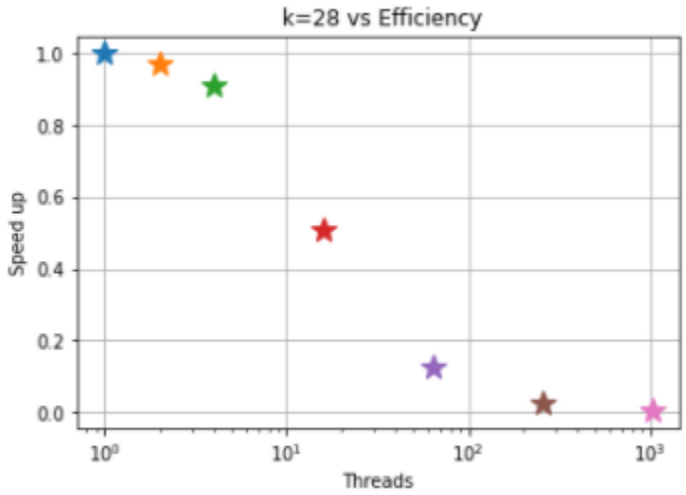



k=20 vs Speedup
Speed up
Threads
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
10^0
10^1
10^2
10^3


k=28 vs Speedup
Speed up
Threads
1
2
3
4
5
6
7
8
10^0
10^1
10^2
10^3


k=12 vs Efficiency
Speed up
Threads
0.0
0.2
0.4
0.6
0.8
1.0
10^0
10^1
10^2
10^3





The result of this experiment aligns with the expected behavior of the parallelized code. As the number of threads increases, the speedup also increases. After utilizing the number of cores (48 in grace) the speedup reduces. Efficiency decreases as the number of threads increases which is the expected behavior in parallelization.

3)

```
places = cores
master
List Size = 268435456, Threads = 32, error = 0, time (sec) =  66.3000, qsort_time =  62.9860
close
List Size = 268435456, Threads = 32, error = 0, time (sec) =   6.0957, qsort_time =  62.6963
spread
List Size = 268435456, Threads = 32, error = 0, time (sec) =   7.0993, qsort_time =  62.6735
places = threads
master
List Size = 268435456, Threads = 32, error = 0, time (sec) =  66.4127, qsort_time =  63.2096
close
List Size = 268435456, Threads = 32, error = 0, time (sec) =   6.3822, qsort_time =  62.6327
spread
List Size = 268435456, Threads = 32, error = 0, time (sec) =   6.6244, qsort_time =  62.3400
places = sockets
master
List Size = 268435456, Threads = 32, error = 0, time (sec) =   6.9345, qsort_time =  62.5882
close
List Size = 268435456, Threads = 32, error = 0, time (sec) =   6.8886, qsort_time =  62.5462
spread
List Size = 268435456, Threads = 32, error = 0, time (sec) =   7.0906, qsort_time =  62.5795
```

The above is the observed values for OMP_PLACES and OMP_PROC_BIND. In PLACES, as socket is specified, the task gets distributed among the **sockets** which means each socket has almost half of the task. This makes it run in **parallel** and the time is almost similar for master, spread and close. When in **cores and threads**, the **master** assigns every thread in the team to the same place as the master thread which makes it queued up and execute it in **serial**. So, the time for master execution is almost like qsort time which is running in serial. Generally, master's execution time is more compared to close and spread as it assigns every thread in the team to the same place as the master thread. Close has less execution time (1 sec less) when compared to spread as it assigns threads to places closest to the place of the parent thread in order of thread ids; threads in the team execute on places that are consecutive starting from the parent's position, with wrap around. Sparse distribution is not very effective in this case, so the spread time is more compared to close.