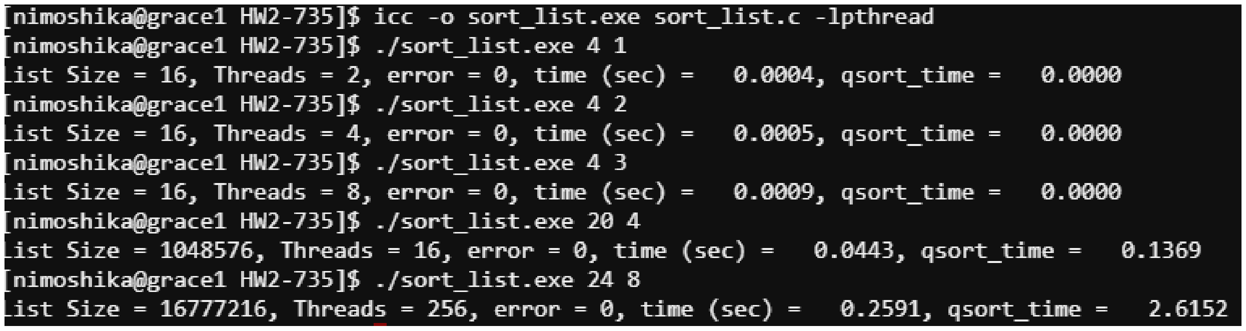
HW 2: Parallel Merge Sort Using Threads

Nimoshika Jayaraman

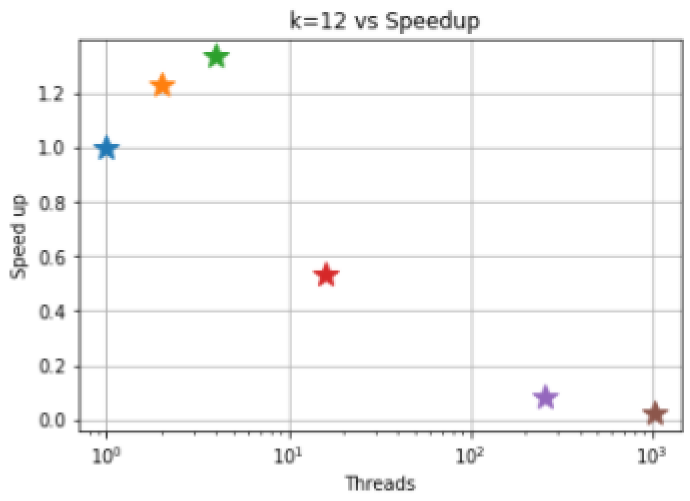
UIN:631000852

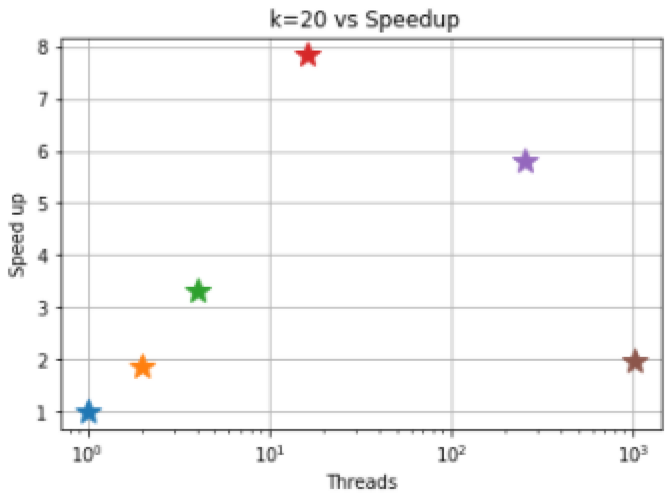
1. The code was revised to implement a thread-based parallel merge sort. The code compiled successfully and reported `error=0. Screenshot attached.`

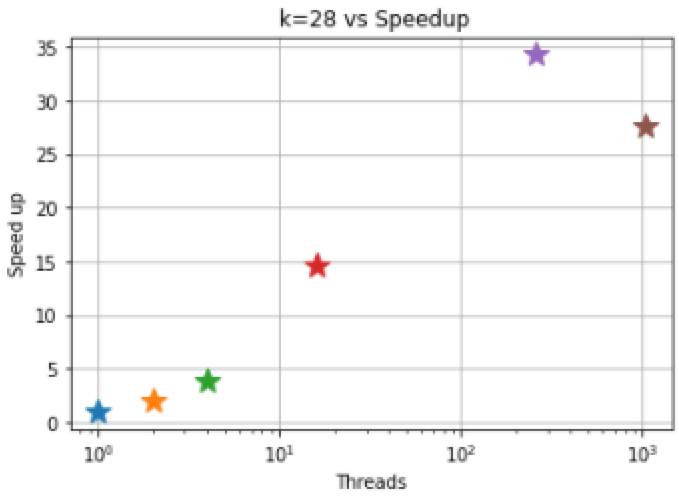


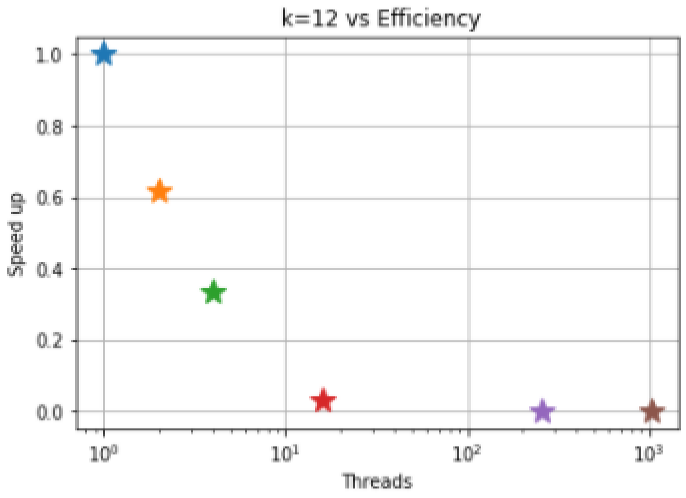
```
[nimoshika@grace1 HW2-735]$ icc -o sort_list.exe sort_list.c -lpthread
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 1
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0004, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 2
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0005, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 3
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0009, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 20 4
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0443, qsort_time =   0.1369
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 24 8
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.2591, qsort_time =   2.6152
```

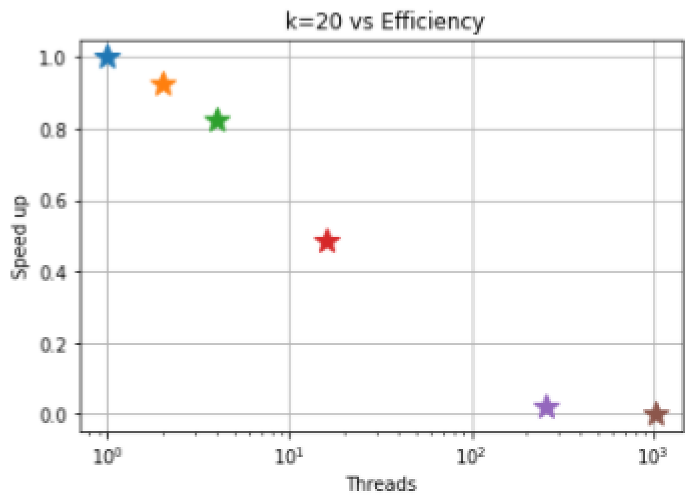
2. Plot of speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28; q = 0, 1, 2, 4, 6, 8, 10.

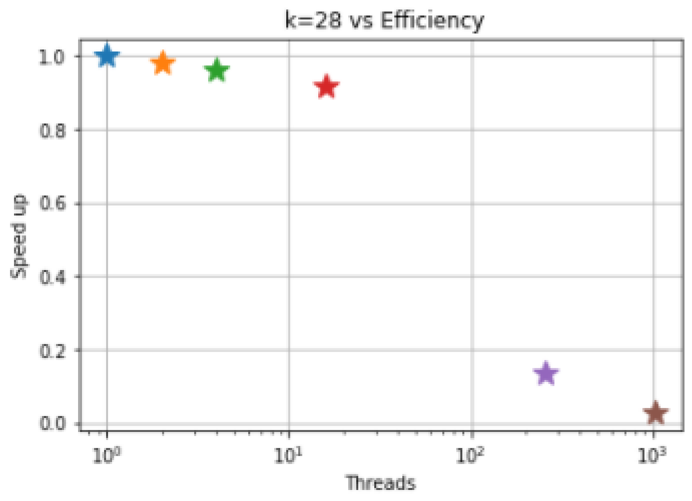



k=20 vs Speedup
Speed up
Threads


k=28 vs Speedup
Speed up
Threads


k=12 vs Efficiency
Speed up
Threads





The results of the experiments align with my understanding and behavior of 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. My code demonstrates the speedup when sorting lists of appropriate sizes. Two values of K (20, 12) for which the time is decreased after parallelization is shown below

<u>Before Parallelization:</u>

```
./sort_list.exe 20 4
```

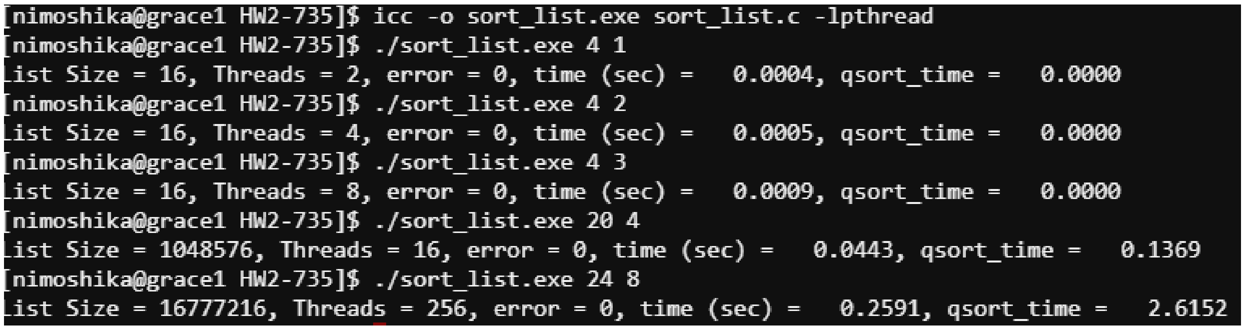
List Size = 1048576, Threads = 16, error = 0, **time (sec) = 0.1414**, qsort_time = 0.1260

After Parallelization:

```
./sort_list.exe 20 4
```

List Size = 1048576, Threads = 16, error = 0, **time (sec) =   0.0225**, qsort_time =   0.1775 (From Sbatch)

Screenshot from terminal:



```
[nimoshika@grace1 HW2-735]$ icc -o sort_list.exe sort_list.c -lpthread
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 1
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0004, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 2
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0005, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 4 3
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0009, qsort_time =   0.0000
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 20 4
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0443, qsort_time =   0.1369
[nimoshika@grace1 HW2-735]$ ./sort_list.exe 24 8
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.2591, qsort_time =   2.6152
```

Before parallelization:

./sort_list.exe 12 4

List Size = 4096, Threads = 16, error = 0, **time (sec) =   0.0015**, qsort_time =   0.0010

After Parallelization:

./sort_list.exe 12 4

List Size = 4096, Threads = 4, error = 0, **time (sec) =   0.0012**, qsort_time =   0.0009

Thus, the parallel sort code is well designed to minimize the execution time. It's efficiency and speedup are represented above.