CSCE 735 Parallel Computing – HW2

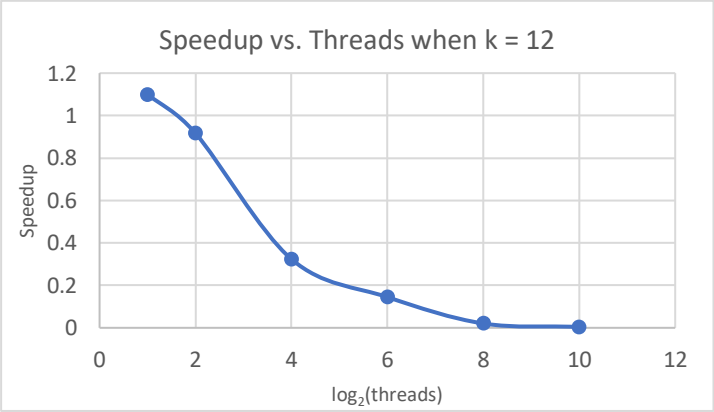
832001192
Chun-Sheng Wu

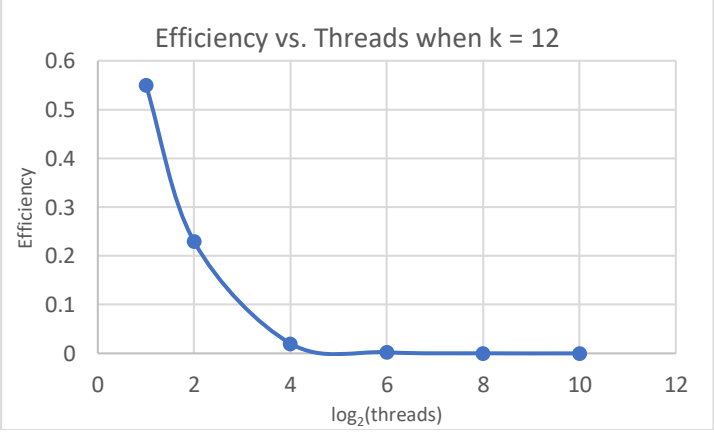
1. Revise the code to execute list sorting using threads without errors.

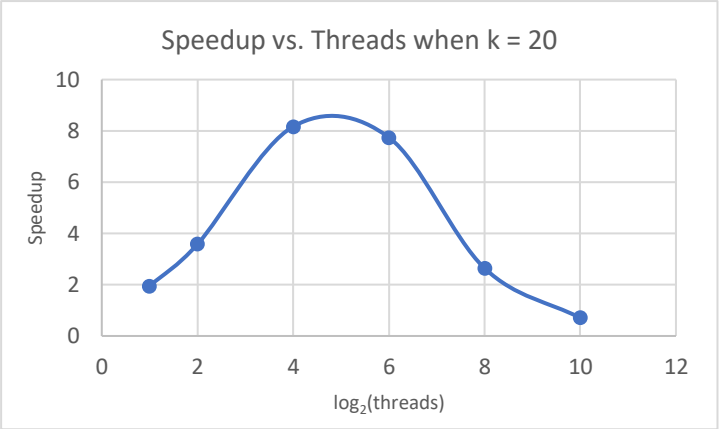
I additionally created a list to perform the execution in parallel since there happened an unknown problem that I could not use the "list" with identical logics. I also put several barriers in the loop to enable data synchronization between threads. Below is the result of my code with the conditions asked by the topic.

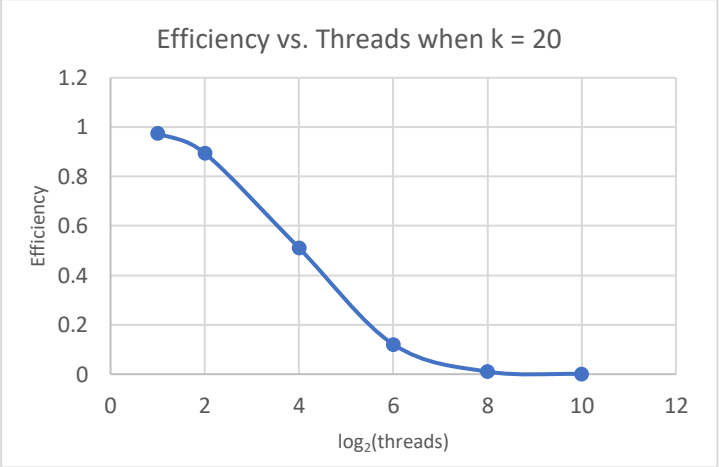
```
[jinsonwu@grace5 HW2]$ ./sort_list.exe 4 1
------------------------------------------------------------
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0004, qsort_time =   0.0000
------------------------------------------------------------
[jinsonwu@grace5 HW2]$ ./sort_list.exe 4 2
------------------------------------------------------------
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0006, qsort_time =   0.0000
------------------------------------------------------------
[jinsonwu@grace5 HW2]$ ./sort_list.exe 4 3
------------------------------------------------------------
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0011, qsort_time =   0.0000
------------------------------------------------------------
[jinsonwu@grace5 HW2]$ ./sort_list.exe 20 4
------------------------------------------------------------
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0298, qsort_time =   0.1445
------------------------------------------------------------
[jinsonwu@grace5 HW2]$ ./sort_list.exe 24 8
------------------------------------------------------------
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.2777, qsort_time =   2.5007
------------------------------------------------------------
```

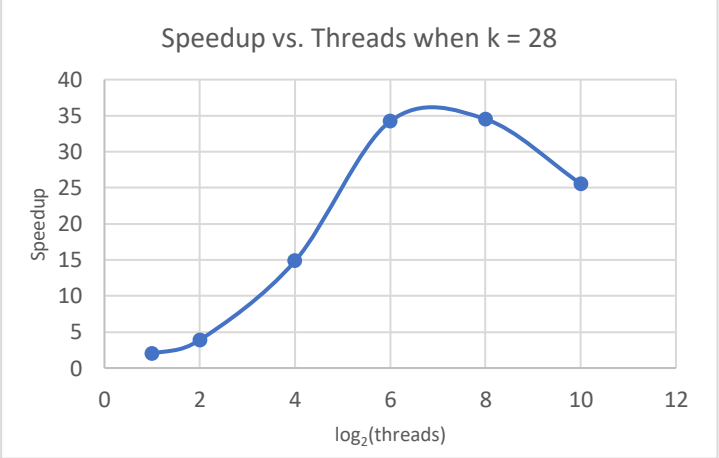
2. Plot speedup and efficiency for various k & q. Comment about their behaviors.

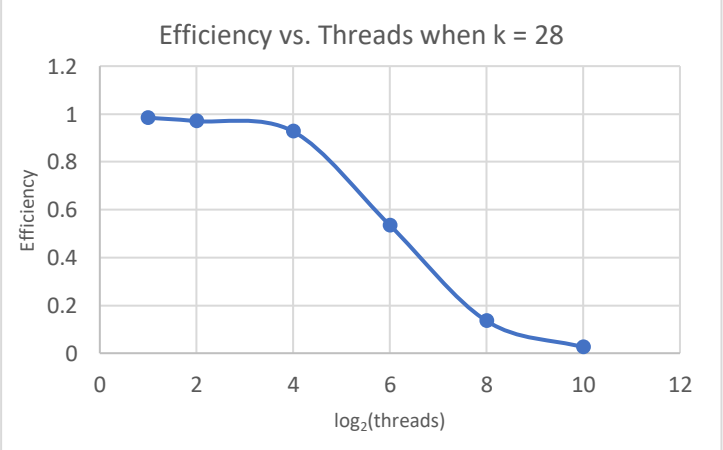





Speedup vs. Threads when k = 20
Speedup
10
8
6
4
2
0
0
2
4
6
8
10
12
log2(threads)


Efficiency vs. Threads when k = 20
Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
log2(threads)


Speedup vs. Threads when k = 28
Speedup
40
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
log2(threads)

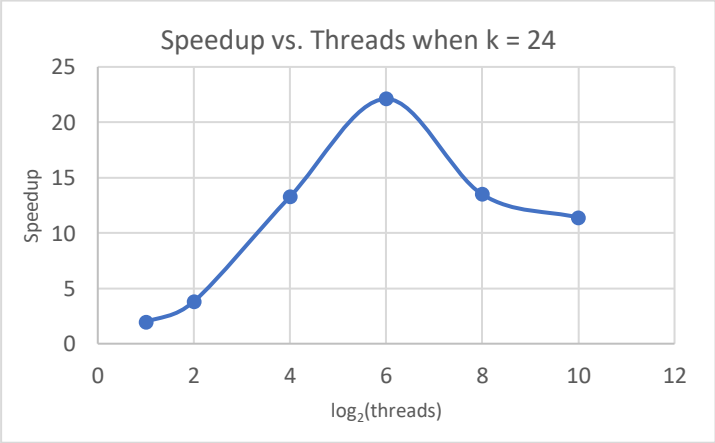

Efficiency vs. Threads when k = 28
Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
log2(threads)

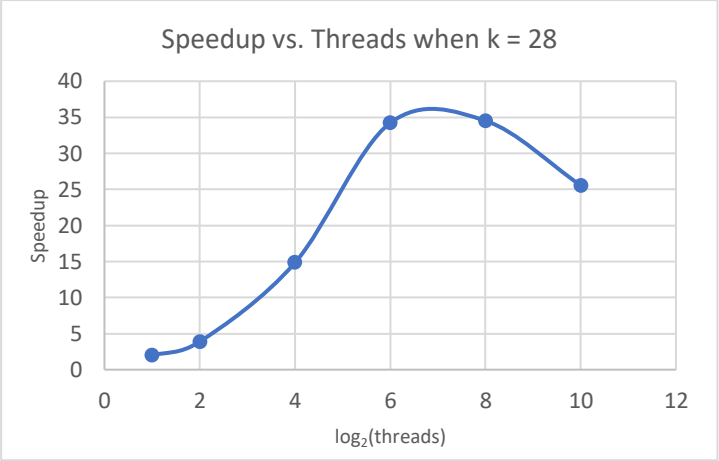
It is out of surprise that only performing with 2 threads had speedup when execution list sorting at k = 12. Guess that the task loading was light that revoking multiple threads to operate this code was waste of time. The time to call the hardware resource took more time than directly sorting the list conventionally. Correspondingly, the efficiency performance was not close to 1 in case that it did not reach desired speedup.

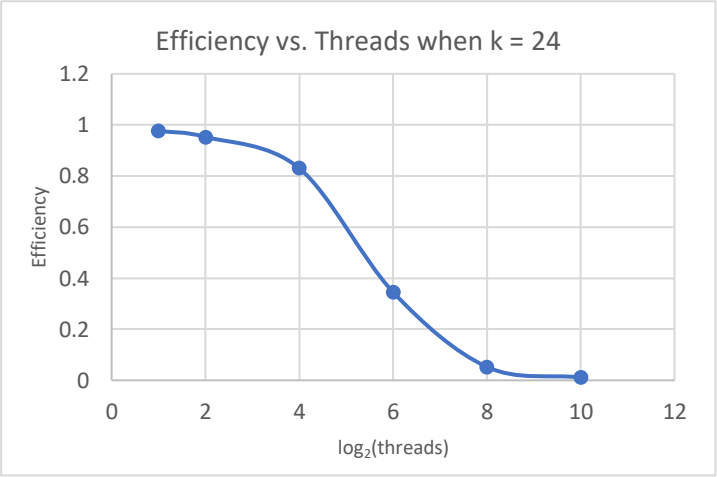
As for k = 20, the speedup slightly caught the trend of the increase of threads, but still not able to utilize the threads to the fullest. Ideally, the peak should be at 48 or more threads representing all the threads (cores) on the board was executing, not idling. Its efficiency dropped linearly with more threads involved and down to almost 0 after 256 threads.

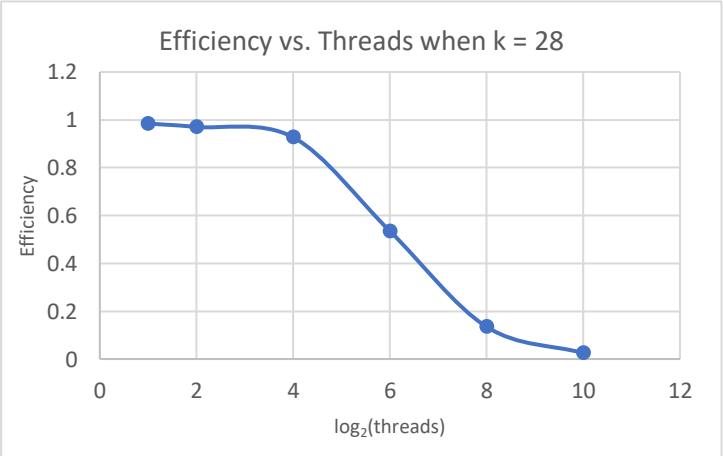
The speedup and efficiency curves best aligned with our expectation with k = 28. The maximum speedup (minimum execution time) was achieved when q = 8. This result was in conformity with the consequence in HW1. But the curve told us that the optimal speedup in this condition might occur at q ~= 7. I did a further experiment about this in the following section. Efficiency performed well and was nearly flat at 1 with relatively fewer threads (q = 1, 2, 4). However, it decreased when obtaining the maximum speedup. Whether to exploit threads more than intrinsic threads (48) would be a lesson if taking tradeoff into consideration.

3. **Determine two values of k for which your code shows speedup as q is varied. Present and persuade the reader that your code is well-designed with speedup and efficiency plots.**









I decided to select k = 24 & 28 to demonstrate the parallelization status of my code because huge amount of computing could better exhibit the utilization of threads. I also did the experiment with k = 32, but the process was killed in the same conditions of previous settings. Both speedup and efficiency graphs showed that this code was under adequate exploitation of threads with threads below 64, which indicated that it employed 48 cores (threads) on the node efficiently. In this point, this code was well-designed with comparable threads included.