**CSCE 735 Fall 2022  
HW 2: Parallel Merge Sort Using Threads**

Name: Sai Namith Garapati

UIN: 832001176

1. (70 points) Revise the code to implement a thread-based parallel merge sort. The code should

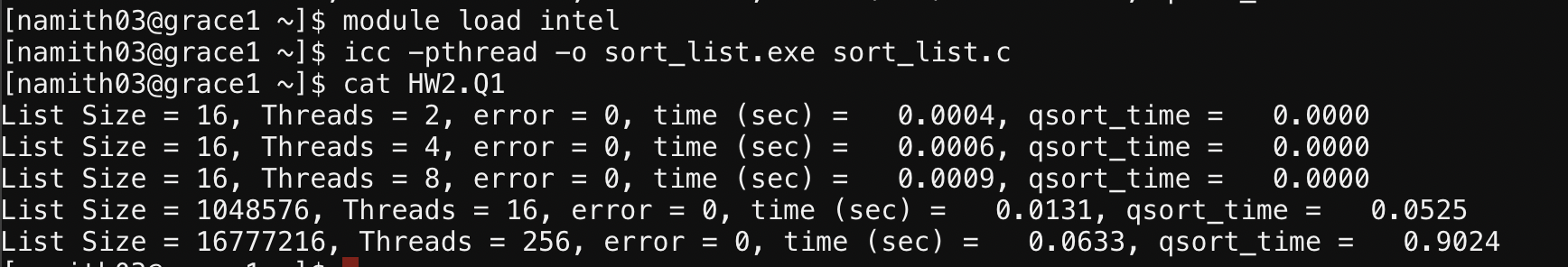
compile successfully and should report error=0 for the following instances: ./sort\_list.exe 4 1  
./sort\_list.exe 4 2  
./sort\_list.exe 4 3

./sort\_list.exe 20 4

./sort\_list.exe 24 8

Solution:

The code has been modified with additional functionality to implement the thread-based parallel merge sort. The results are as follows



The code compiled successfully and error = 0 was reported for all the given instances

(Note: compiled with icc -pthread -o sort\_list.exe sort\_list.c)

1. 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 behaviour of the parallelized code.

Solution:

The code is run for all possible combinations given in the question

For k = 12; q = 0,1,2,4,6,8,10

The results are as follows:

Text

Description automatically generated with low confidence

Plot for Speedup vs threads:

Chart, line chart

Description automatically generated

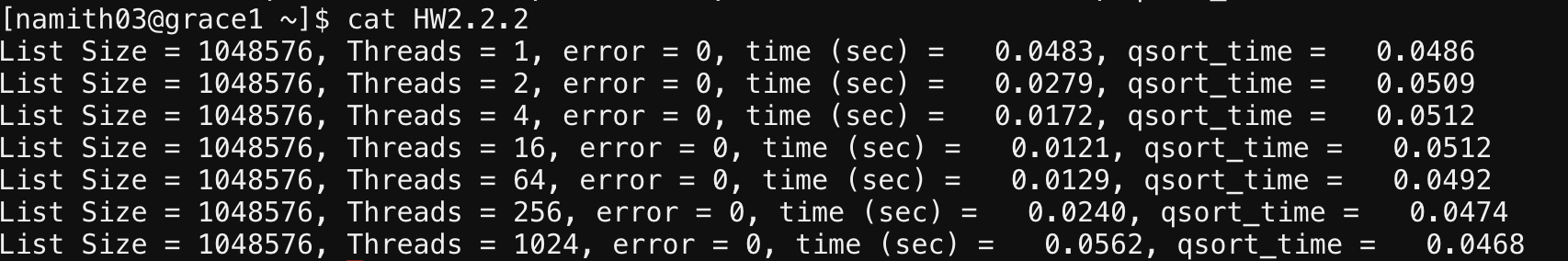
Plot for Efficiency vs threads:

Chart, line chart

Description automatically generated

For k = 20; q = 0,1,2,4,6,8,10

The results are as follows:



Plot for Speedup vs threads:

Chart, line chart

Description automatically generated

Plot for Efficiency vs threads:

Chart, line chart

Description automatically generated

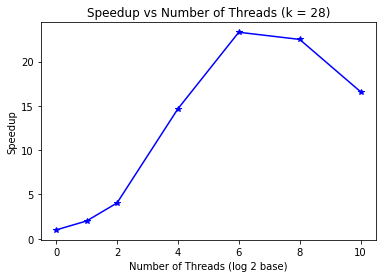
For k = 28; q = 0,1,2,4,6,8,10

The results are as follows:

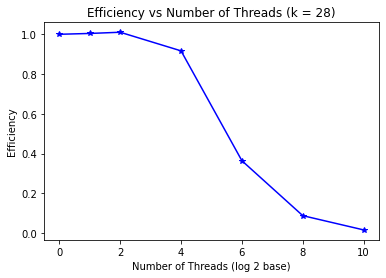
A screen shot of a computer

Description automatically generated with low confidence

Plot for Speedup vs threads:



Plot for Efficiency vs threads:



**Observation**: From the plots and results, it can be observed that speedup increases with increase in number of threads. But we can observe that after certain number of threads, the Speedup starts to drop with the increment in number of threads. The effect of parallelization in the code only be observed until a certain number of threads depending upon the size of the list. Beyond certain number of threads, the increase in number of threads results in overhead associated with it which results in the decrement in the Speedup.

1. (10 points) Your code should demonstrate speedup when sorting lists of appropriate sizes. Determine two values of k for which your code shows speedup as q is varied. Present the timing results for your code along with speedup and efficiency obtained to convince the reader that you have a well-designed parallel merge sort. You may use results from experiments in previous problems or identify new values k and q to illustrate how well your code has been parallelized.

Solution:

For k = 12, time vs threads:

Chart, line chart

Description automatically generated

For k = 20, time vs threads:

Chart, line chart

Description automatically generated

For k = 28, time vs threads:

Chart, line chart

Description automatically generated

**Observation:**

We can observe from the speedup graphs in question 2 that the when the list size is bigger, using higher number of threads resulted in a better speedup in comparison when the list size is small. It can be observed from the above timing graphs that with a larger list size (k = 28), with increase in number of threads, it takes a lot lesser time to execute compared to other values with comparatively smaller list sizes like k = 12. Hence, we can conclude from the above timing graphs and Speedup graphs (in question 2) that the parallel algorithm implemented for the merge sort works very well.