CSCE 735 Fall 2023 – HW3

Name: Ashutosh Chauhan
UIN: 232009024

1. **(70 points) Revise the code to implement parallel merge sort via OpenMP. The code should compile successfully and should report `error=0` for the following instances:**

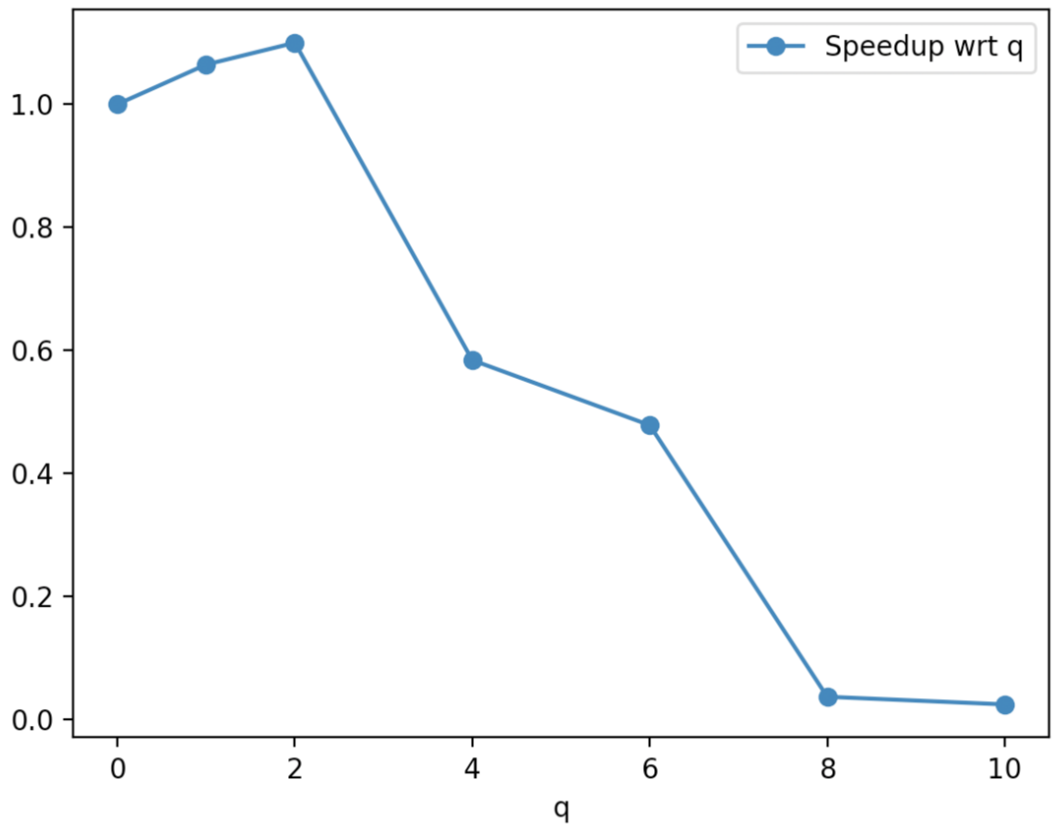
```
./sort_list_openmp.exe 4 1
./sort_list_openmp.exe 4 2
./sort_list_openmp.exe 4 3
./sort_list_openmp.exe 20 4
./sort_list_openmp.exe 24 8
```

	List Size	Threads	error	time	qsort_time
0	16	2	0.0	0.0622	0.0000
1	16	4	0.0	0.0057	0.0000
2	16	8	0.0	0.0067	0.0000
3	1048576	16	0.0	0.0266	0.1732
4	16777216	256	0.0	0.5348	3.4777

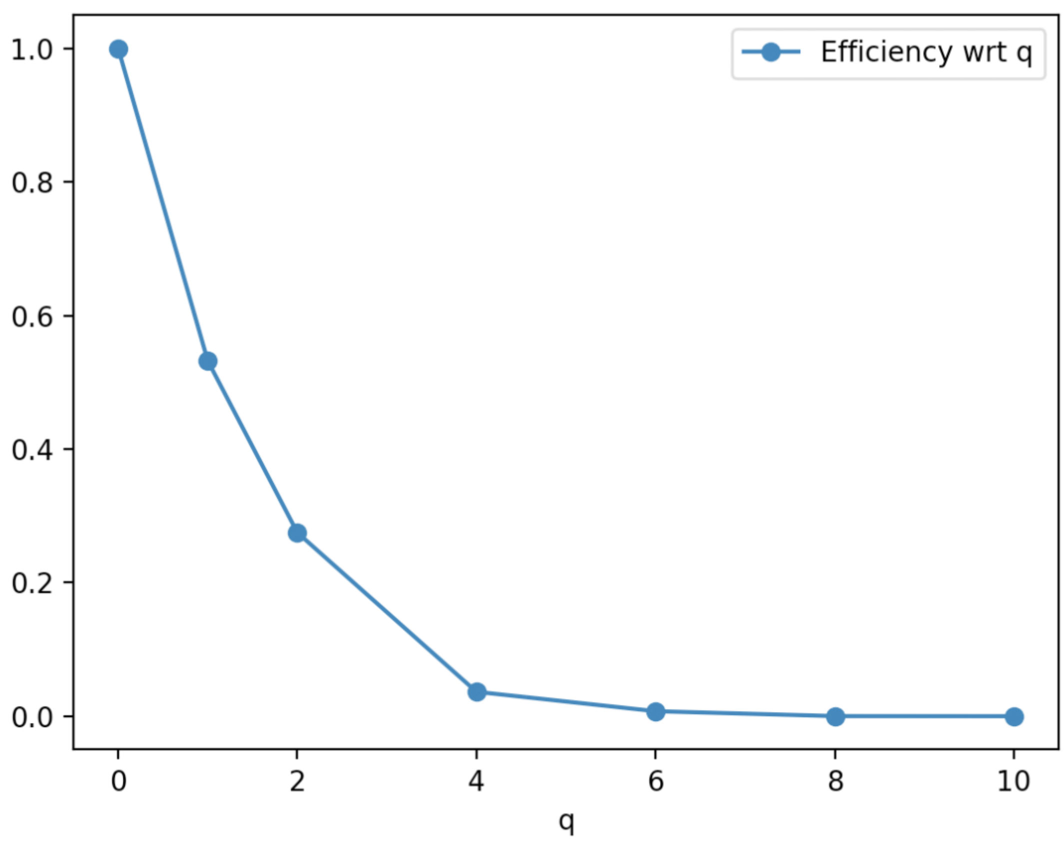
2. **(20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.**

<u>**K= 12**</u>

List Size	Threads	error	time	qsort_time	speedup	efficiency	q	k
4096	1	0.0	0.0066	0.0009	1.000000	1.000000	0.0	12.0
4096	2	0.0	0.0062	0.0008	1.064516	0.532258	1.0	12.0
4096	4	0.0	0.0060	0.0008	1.100000	0.275000	2.0	12.0
4096	16	0.0	0.0113	0.0007	0.584071	0.036504	4.0	12.0
4096	64	0.0	0.0138	0.0005	0.478261	0.007473	6.0	12.0
4096	256	0.0	0.1792	0.0004	0.036830	0.000144	8.0	12.0
4096	1024	0.0	0.2707	0.0004	0.024381	0.000024	10.0	12.0



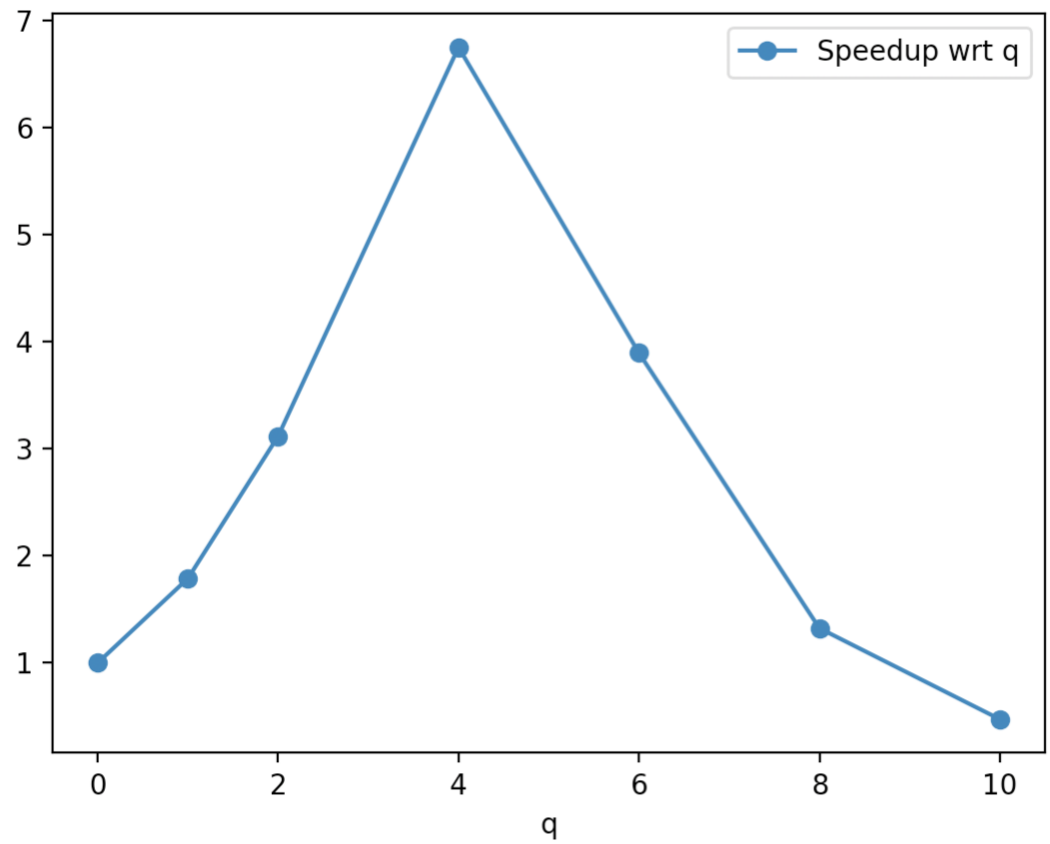
Speedup wrt q
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
q

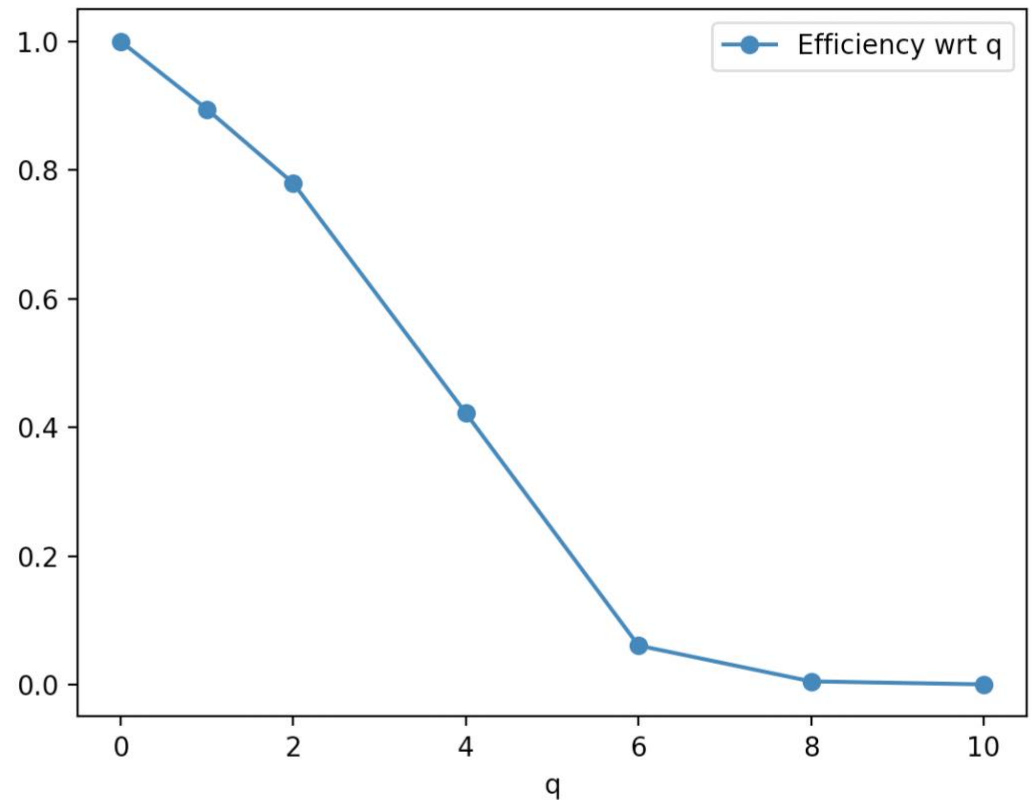


Efficiency wrt q
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
q

K = 20

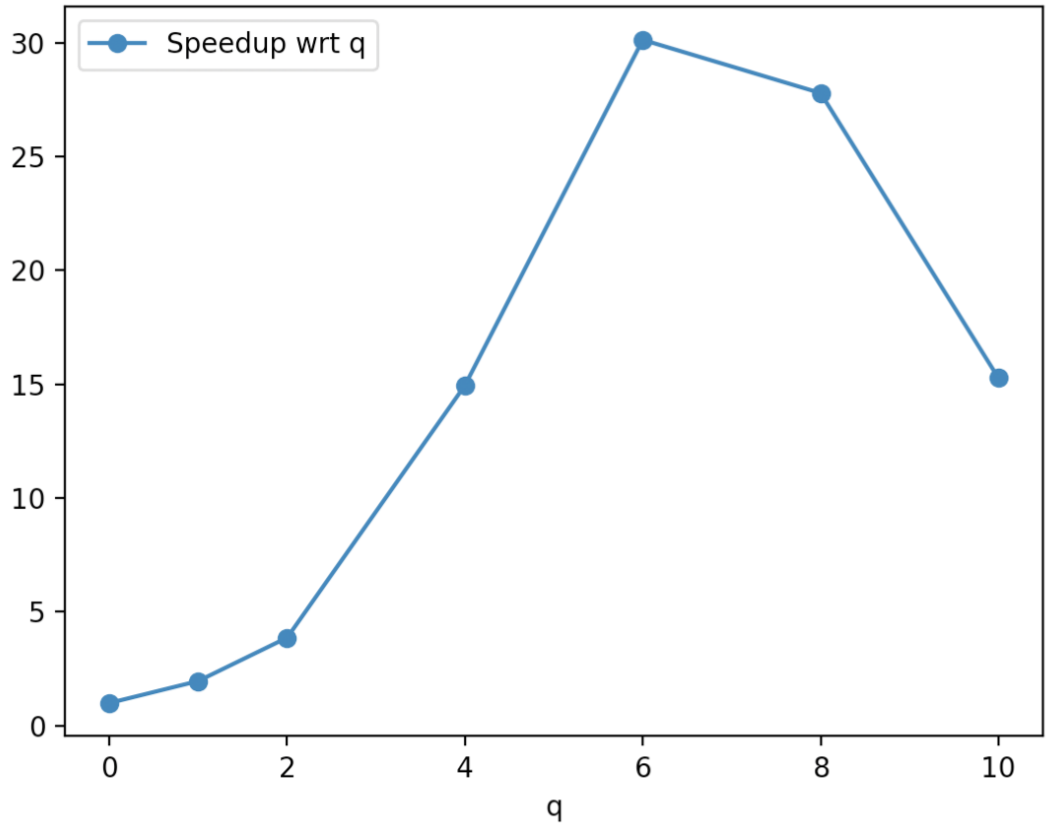
List Size	Threads	error	time	qsort_time	speedup	efficiency	q	k
1048576	1	0.0	0.1796	0.1715	1.000000	1.000000	0.0	20.0
1048576	2	0.0	0.1004	0.1719	1.788845	0.894422	1.0	20.0
1048576	4	0.0	0.0576	0.1724	3.118056	0.779514	2.0	20.0
1048576	16	0.0	0.0266	0.1726	6.751880	0.421992	4.0	20.0
1048576	64	0.0	0.0461	0.1732	3.895879	0.060873	6.0	20.0
1048576	256	0.0	0.1357	0.1733	1.323508	0.005170	8.0	20.0
1048576	1024	0.0	0.3797	0.2454	0.473005	0.000462	10.0	20.0

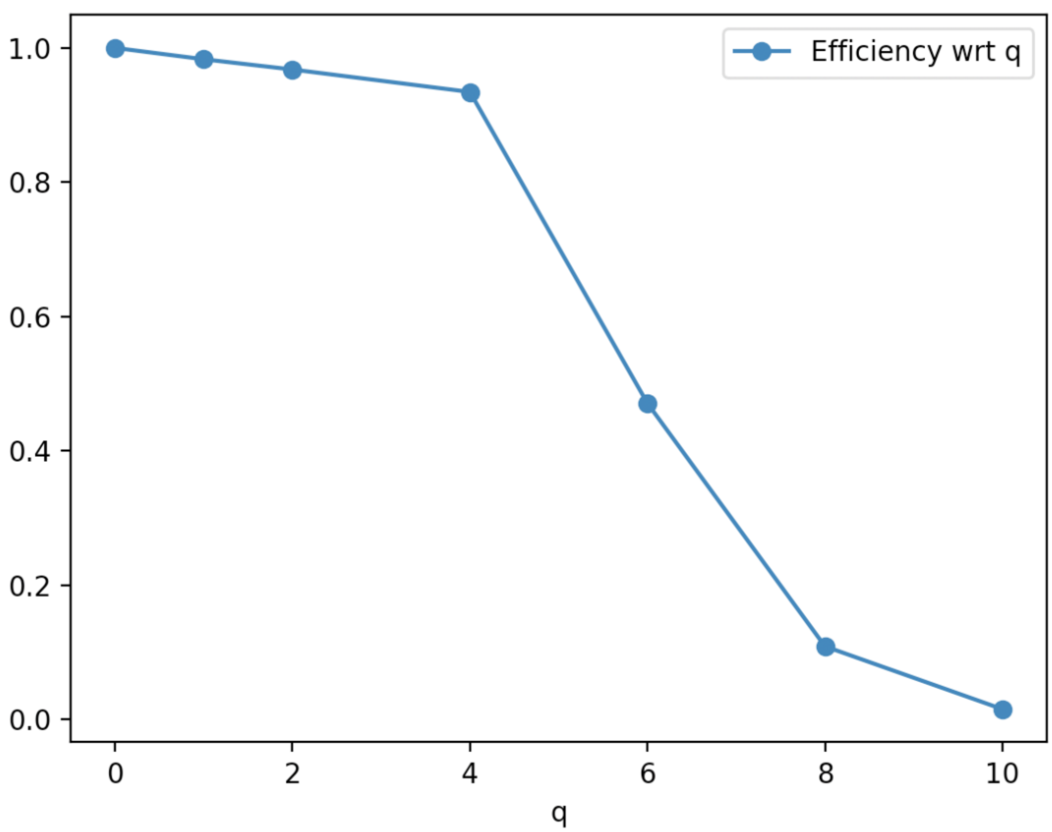




K=28

List Size	Threads	error	time	qsort_time	speedup	efficiency	q	k
268435456	1	0.0	62.8263	62.8271	1.000000	1.000000	0.0	28.0
268435456	2	0.0	31.9545	63.0195	1.966117	0.983059	1.0	28.0
268435456	4	0.0	16.2297	62.9980	3.871070	0.967767	2.0	28.0
268435456	16	0.0	4.2025	62.8149	14.949744	0.934359	4.0	28.0
268435456	64	0.0	2.0860	63.0342	30.118073	0.470595	6.0	28.0
268435456	256	0.0	2.2615	62.9614	27.780809	0.108519	8.0	28.0
268435456	1024	0.0	4.1145	63.5765	15.269486	0.014912	10.0	28.0


Speedup wrt q
30
25
20
15
10
5
0
0
2
4
6
8
10
q


Efficiency wrt q
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
q

Comments:

We can observe that in all the above cases speedup increases until it reaches a threshold and then starts decreasing after a certain value of threads. This is because every thread requires some system resources like stack, memory etc. for context switching. As we increase the number of threads the overhead associated with managing and switching between threads also increases. This outweighs the benefits that we get because of multiple threads after a certain value. Hence, we see this behavior. Also, we can observe that efficiency keeps on decreasing with an increase in the number of threads in all the above cases. This can be attributed to the growing number of underutilized threads. With the increase in the number of threads the amount of work shared by each thread is less compared to the overhead it takes to manage multiple threads.

3. **(10 points) For the instance with k = 28 and q = 5 experiment with different choices for OMP_PLACES and OMP_PROC_BIND to see how the parallel performance of the code is impacted. Explain your observations.**

Results for all the different choices for OMP_PLACES and OMP_PROC_BIND are as follows:

```
export OMP_PLACES="threads"
export OMP_PROC_BIND=close
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.2131, qsort_time =  62.8926

export OMP_PLACES="threads"
export OMP_PROC_BIND=spread
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.1810, qsort_time =  62.9885

export OMP_PLACES="threads"
export OMP_PROC_BIND=master
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =  67.1804, qsort_time =  62.7532

export OMP_PLACES="sockets"
export OMP_PROC_BIND=close
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.1994, qsort_time =  62.8824

export OMP_PLACES="sockets"
export OMP_PROC_BIND=spread
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.1783, qsort_time =  62.8608

export OMP_PLACES="sockets"
export OMP_PROC_BIND=master
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   3.9680, qsort_time =  62.9001

export OMP_PLACES="cores"
export OMP_PROC_BIND=close
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.2132, qsort_time =  62.8742

export OMP_PLACES="cores"
export OMP_PROC_BIND=spread
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =   2.1781, qsort_time =  62.8406

export OMP_PLACES="cores"
export OMP_PROC_BIND=master
./sort_list_openmp.exe 28 5

List Size = 268435456, Threads = 32, error = 0, time (sec) =  67.3733, qsort_time =  63.0034
```

Observation:

The way we arrange threads using OpenMP can affect how well a program runs, depending on what the program does and how the computer is built. There are two settings we can use to control how threads are arranged: OMP PLACES and OMP PROC BIND.
OMP PLACES offer three options to choose where threads are placed:

- Threads
- Sockets
- cores

OMP PROC BIND offers three ways to group threads. These policies decide how the threads are spread out based on the choices we made with OMP PLACES.

- Master
- Close
- Spread

Figuring out the best way to arrange and handle threads for a specific job and computer setup is super important. It can make a big difference in how fast things get done. If we put all the threads on one main part of the computer when we set the affinity to "MASTER," it can slow things down because that part might not have enough room or memory for everything.
On the other hand, if you spread out the threads evenly across the computer using "SPREAD" affinity, it makes it easier for them to share resources. The quickest way to get things done is when the threads are placed close to each other, which helps them talk and share information faster. To get the best performance, you need to try out different ways of arranging these threads.

This is clear from the above-observed data as well. Time taken in the Master thread policy is very high compared to the close and spread policy.