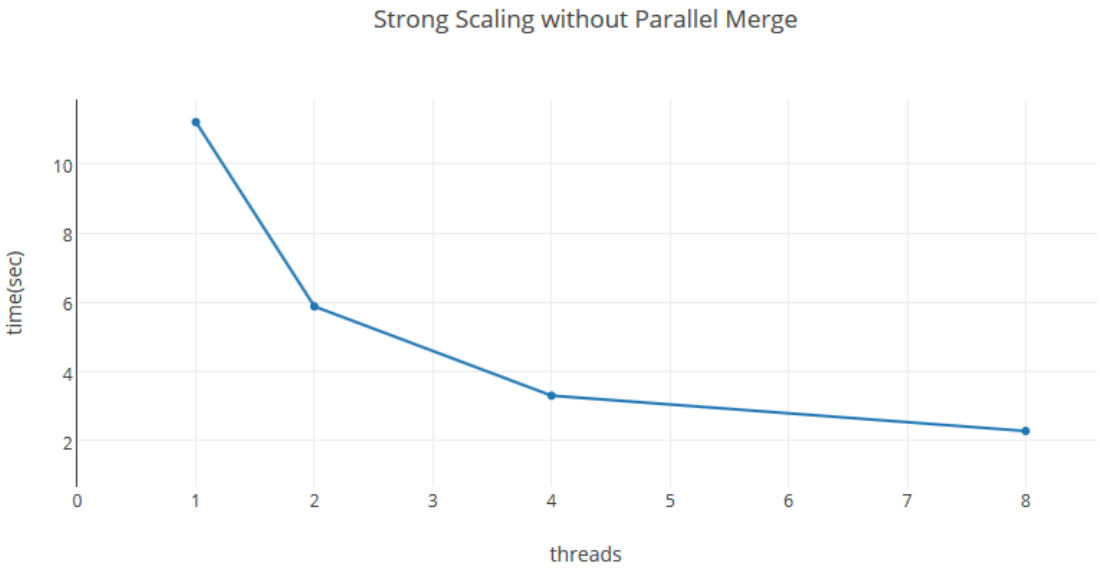
Christopher Almajose
A10463176
Jame Kephavone
A05265684
CSE 160

Programming Assignment 2

2.3.2: (Strong Scaling Study)

1) Varying Threads (no Parallel Merge)

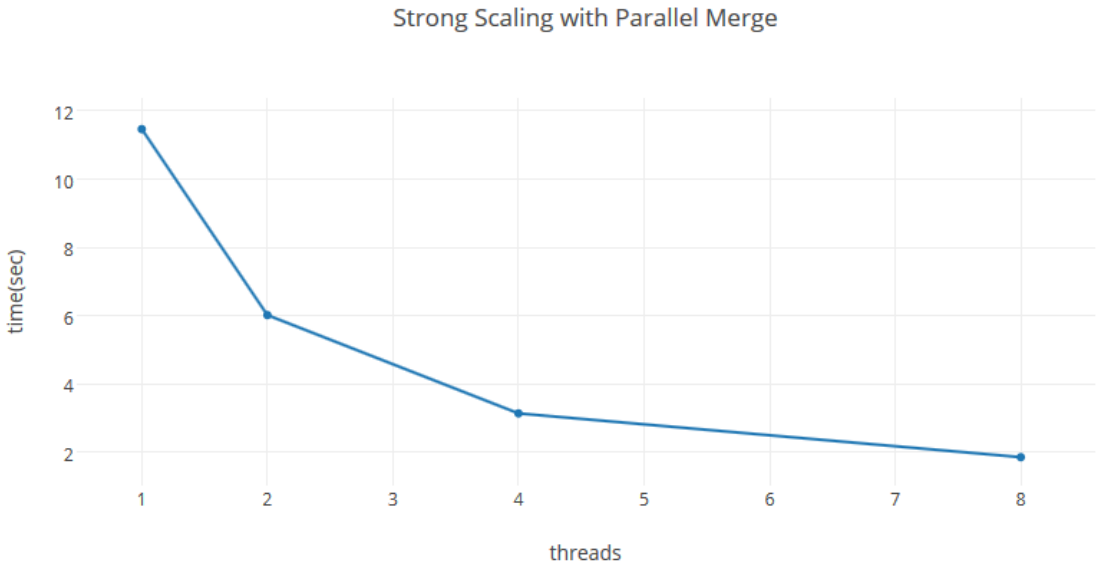


time(sec)	threads
11.202	1
5.877	2
3.302	4
2.280	8

Link: https://plot.ly/~calmajos/28/strong-scaling-without-parallel-merge/

Evaluation:
A logarithmic speedup as the number of the threads increase over time.

2) Varying Threads (with Parallel Merge)



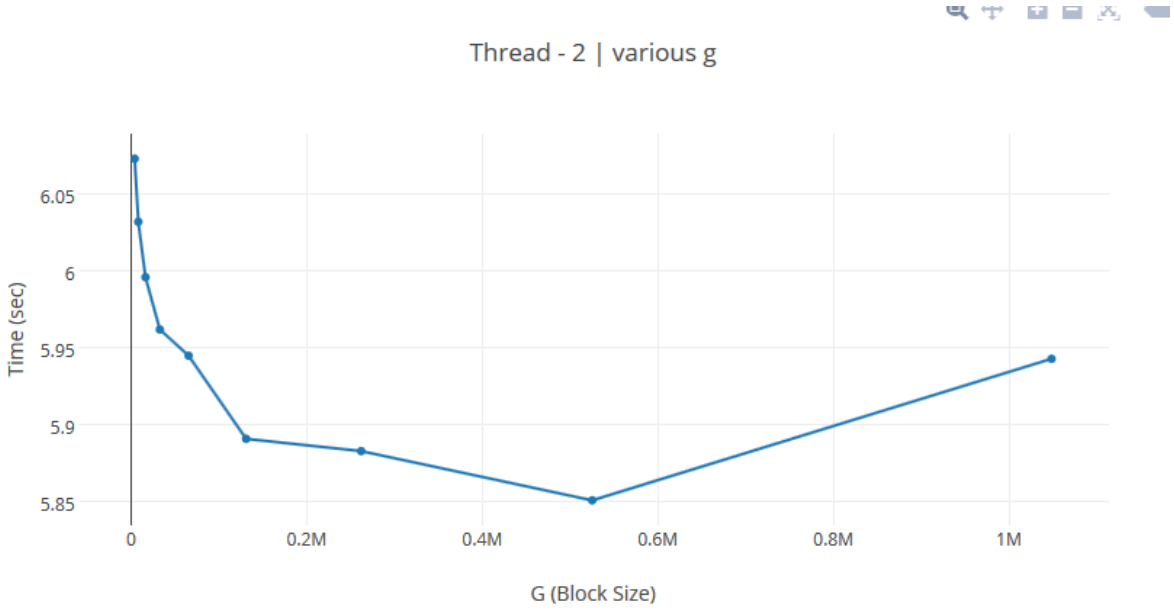
time(sec)	threads
11.464	1
6.021	2
3.145	4
1.863	8

Link: https://plot.ly/~calmajos/29/strong-scaling-with-parallel-merge/

Evaluation:
There is some speedup in the algorithm in parallelizing the merges, however, it is not as substantial than parallelizing the ‘divide and conquer’.

2.3.3

Two Threads:

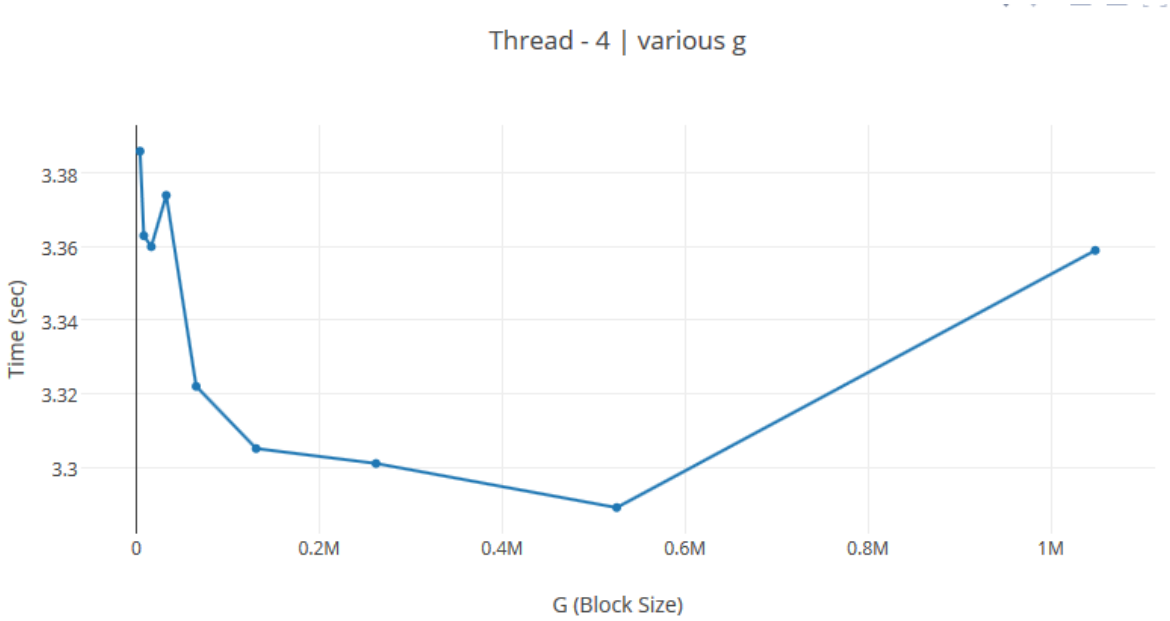


Time (sec)	G (Block Size)
6.073	4096
6.032	8192
5.996	16384
5.962	32768
5.945	65536
5.891	131072
5.883	262144
5.851	525288
5.943	1048576

Link: https://plot.ly/~calmajos/34/thread-2-various-g/

The optimal value of G on 2 Threads is around block sizes of G 525288 - 1048576.

Four Threads

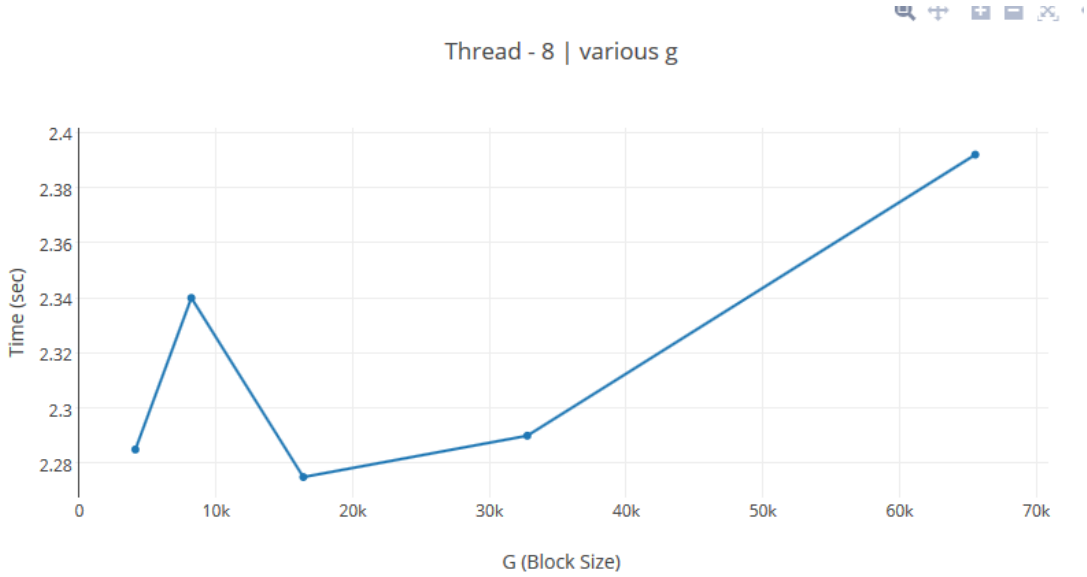


Time (sec)	G (Block Size)
3.386	4096
3.363	8192
3.36	16384
3.374	32768
3.322	65536
3.30507	131072
3.30102	262144
3.289	525288
3.359	1048576

Link: https://plot.ly/~calmajos/35/thread-4-various-g/

The optimal value of G on 4 Threads is around block sizes of G 525288 - 1048576.

Eight Threads



Time (sec)	G (Block Size)
2.285	4096
2.34	8192
2.275	16384
2.29	32768
2.392	65536

Link: https://plot.ly/~calmajos/36/thread-8-various-g/

The optimal value of G on 8 Threads is around block sizes of G 32768 - 65536.

Answer to question 2.3

A: The reason why the program slows down when g=n is the minimum block size is already met, so regardless of the number of threads, we cannot break down the problem and it must sort the whole block ‘n’ in an inefficient way. To get the benefit of speedup, the block sizes should be smaller than ‘n’. There is an optimal values for ‘g’ for various lengths ‘n’ to maximize speedup of random organized lists of values.

2.4 Thoughts

Because of the nature of merge sort, we are reducing the problem by log(2) for every iteration. With the power of more threads to work on the algorithm, we can reduce the problem even faster, as noticed in times in tables 2.3.2 versus a single core. However, the parallel merge give only limited speedup as it still is required to touch all work O(n) times, regardless of parallelizing the work.