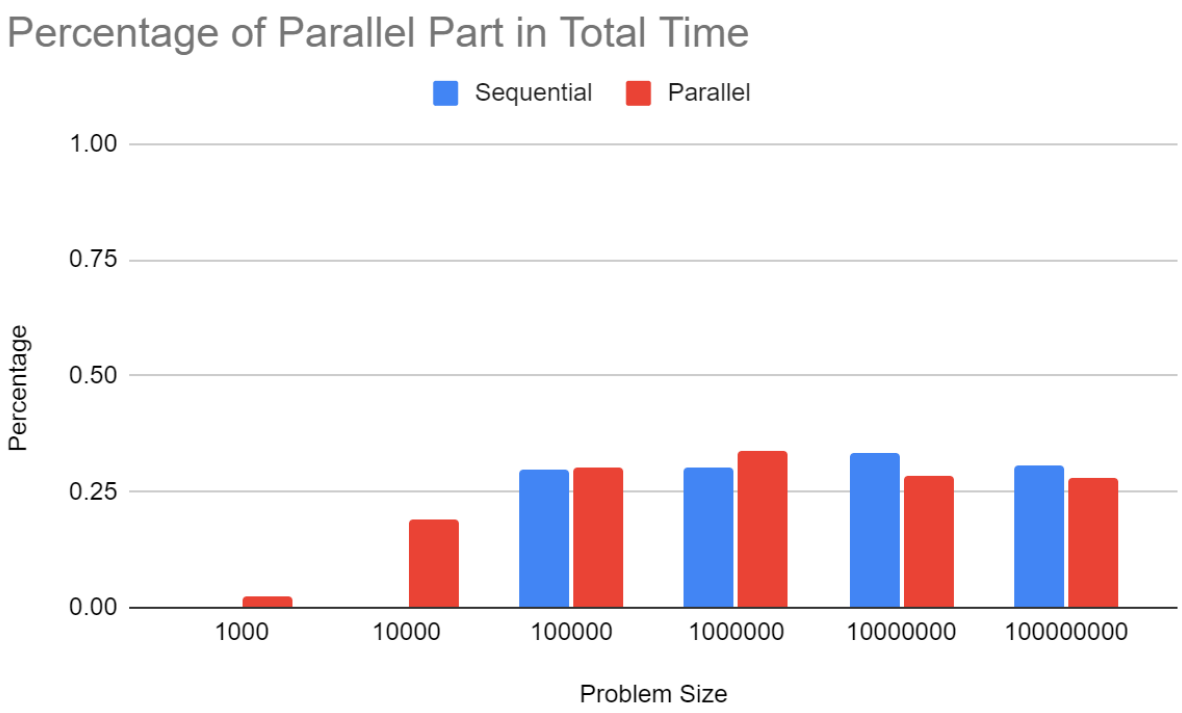
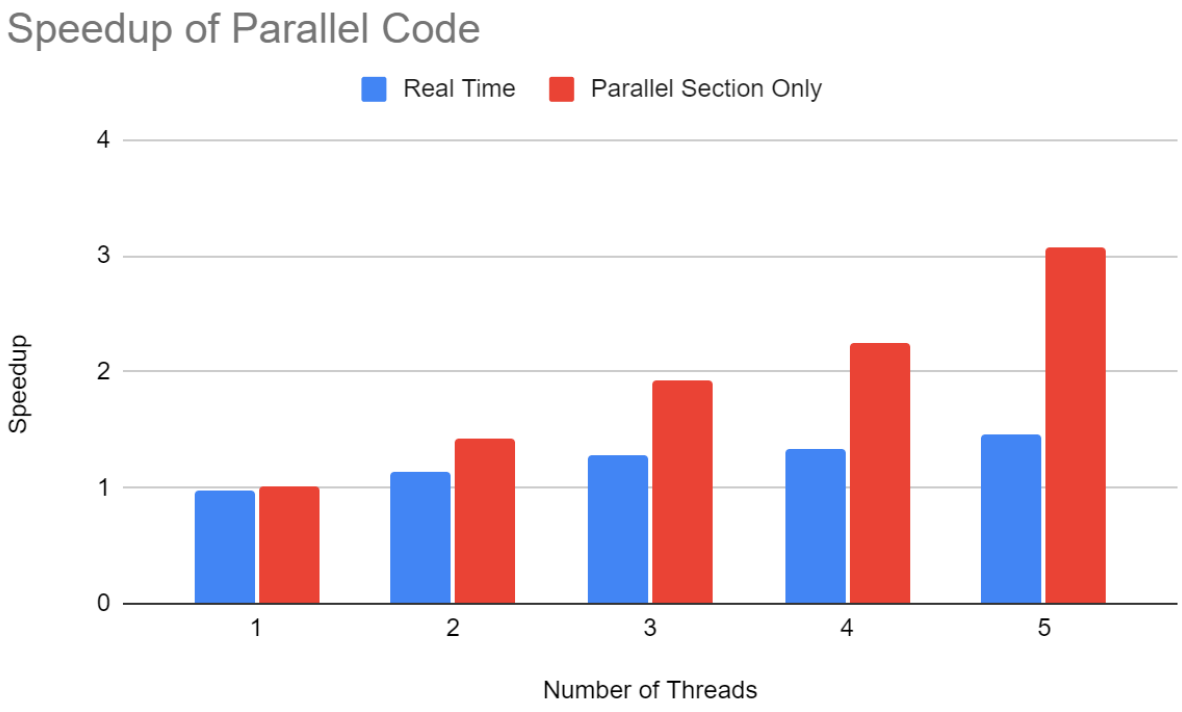
Experiment 1:



Experiment 2:



Experiment 3:

	1000	10000	100000	1000000	10000000	100000000
1	1.00	1.00	1.00	1.00	1.00	1.00
2	0.29	0.45	0.42	0.44	0.49	0.51
3	0.14	0.26	0.26	0.27	0.34	0.34
4	0.11	0.22	0.22	0.27	0.30	0.29
5	0.07	0.19	0.22	0.21	0.24	0.24

What can you say about:

1. The fraction of the sequential part from the overall execution time as the problem size increases.

According to my data, it seems that after a while the fraction of the sequential part of code from the overall execution time approaches around 70%. At maximum, only about 30% of the program is parallelized. This makes sense because reading a file is a majority of the non-parallel part of the program.

2. The speedup as the number of threads increases, both for the version of lab 1 and the version of this lab. Justify your conclusion.

As the number of threads increases, the computation speedup and the real time speedup also increase, at least for large problem sizes. In lab 2, there is an increase in speedup from increasing the thread count, however in lab 1, there was actually a slowdown for small problem sizes due to the overhead of communication between processes. Additionally, real time speedup is much less than computation time speedup (speedup of the parallel portion) because both lab 1 and lab 2 have a portion of the program that is not parallelizable.

3. Did the efficiency experiment produce the results you expected? Justify.

For the most part they met my expectations. Of course, efficiency for the smaller problem sizes are all quite terrible because of the overhead from creating threads, however I imagined for the larger sizes the efficiency would be better, especially for the largest problem size and 2 threads, especially since the best efficiency I get is 0.51. I imagine this is because of how my code is written and I am missing a key parallelizable element, resulting in rather poor efficiency.