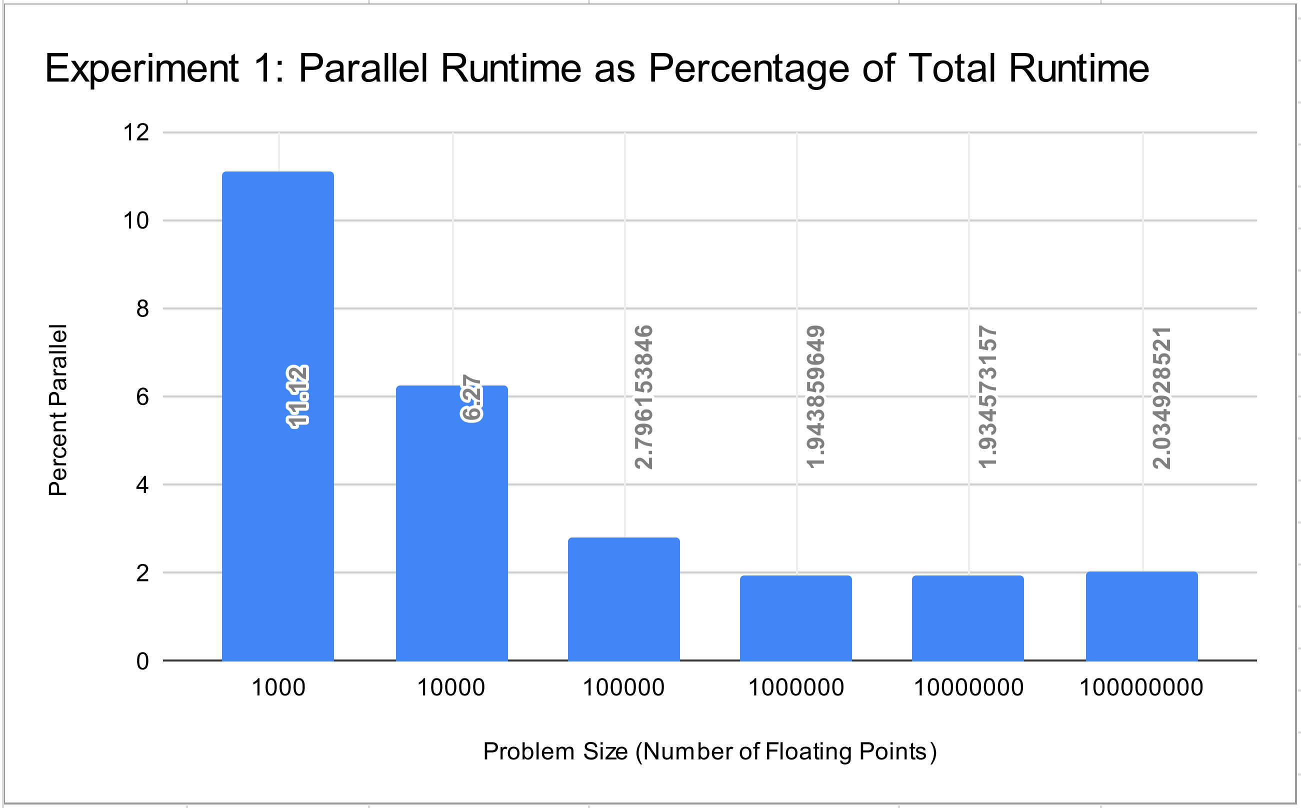
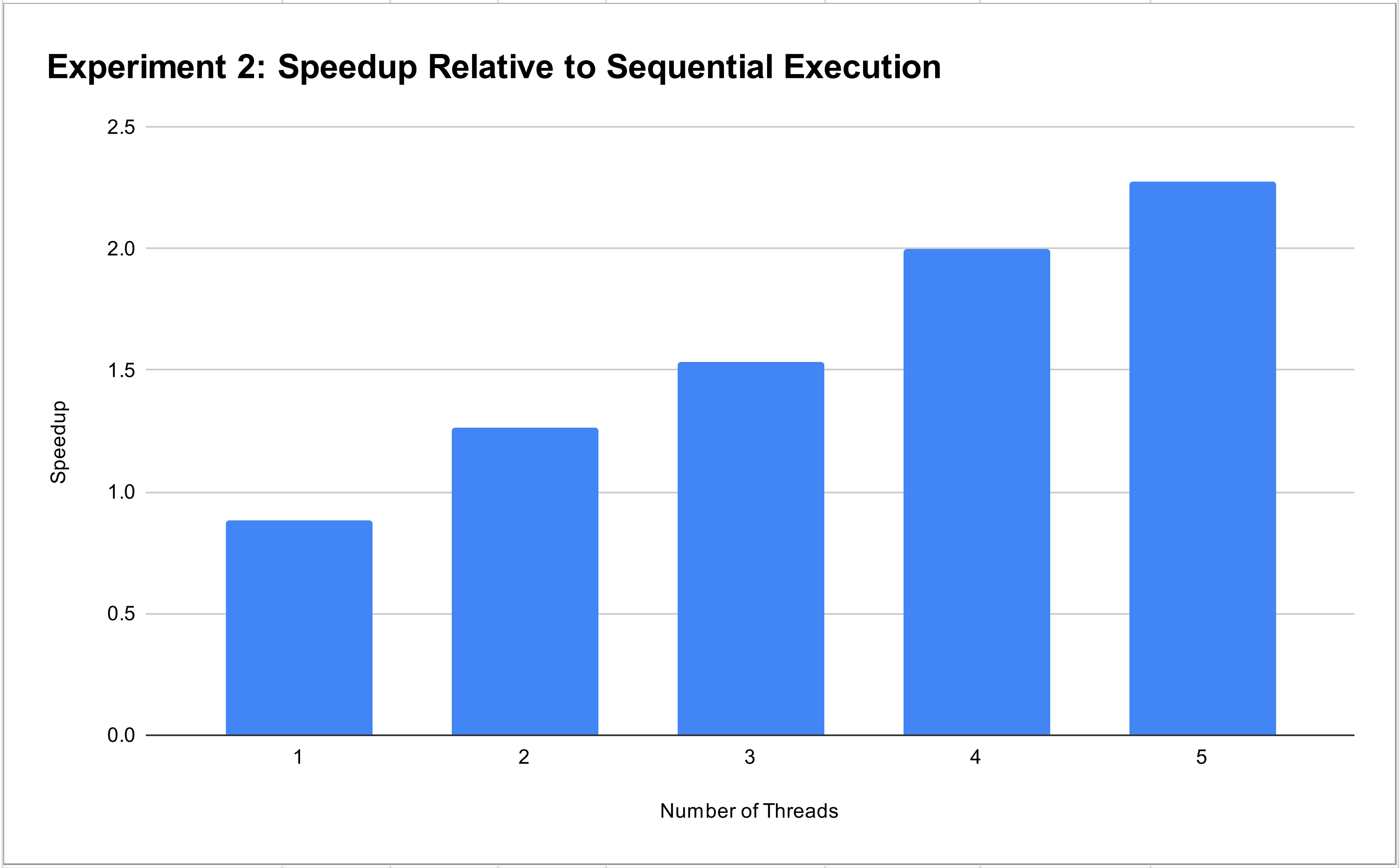
Problem Size	Real Time (s)	Parallel Time (s)	Percentage Parallel (%)
1000	0.005	0.000556	11.12
10000	0.01	0.000627	6.27
100000	0.052	0.001454	2.796153846
1000000	0.456	0.008864	1.943859649
10000000	4.463	0.08634	1.934573157
100000000	44.279	0.901046	2.034928521

Experiment Boundaries
Bins = 20
Threads = 10
Machine: crunchy1


Experiment 1: Parallel Runtime as Percentage of Total Runtime

Number of Threads	Trial 1	Trial 2	Trial 3	Average Runtime (s)	Speedup
1	0.033679	0.034443	0.034015	0.03404566667	0.8811694097
2	0.024052	0.023386	0.023752	0.02373	1.264222503
3	0.018802	0.019808	0.020123	0.01957766667	1.532358299
4	0.014847	0.015482	0.014686	0.015005	1.999333555
5	0.01286	0.01326	0.013418	0.01317933333	2.276291163
Sequential Implementation	0.03	0.03	0.03	0.03	

Experiment Boundaries
Numbers = 1,000,000
Bins = 50
Machine: crunchy1



Experiment 2: Speedup Relative to Sequential Execution

Experiment #3 - Efficiency Table

	1000	10000	100000	1000000	10000000
1	1	1	1	1	1
2	0.08836206897	0.3171296296	0.4748309542	0.5287885948	0.5296423166
3	0.04696449026	0.1790849673	0.4339169241	0.4809269673	0.5031653114
4	0.03213166144	0.1442105263	0.8106028217	0.4809097421	0.5137862817
5	0.0222826087	0.1033962264	0.532285233	0.4826962253	0.4700539674

Parallel Timing Table

	1000	10000	100000	1000000	10000000
1	0.000041	0.000274	0.003792	0.040281	0.398916
2	0.000232	0.000432	0.003993	0.038088	0.37659
3	0.000291	0.00051	0.002913	0.027919	0.264271
4	0.000319	0.000475	0.002339	0.02094	0.194106
5	0.000368	0.00053	0.001781	0.01669	0.169732

Efficiency(x) = (1_thread_runtime/x_thread_runtime) / x

Bins = 20

Question	Assumption	Analysis
Relationship between overall execution time and problem size	This question refers to the sequential time to read the data from the file and allocated all memory, in addition to handling resources (e.g. closing file pointers).	As the problem size increases, the amount of time required to read the file data increases, likewise, we require more memory allocation which increases the overall execution time. This analysis is supported by a general (trend) **increase** in execution time regardless of thread count for an increasing problem size.
Relationship between speedup and thread count	We only observe behavior from 1 to 5 threads and cannot make determinations as to edge cases that have not been tested by this lab assignment (e.g. using 100 threads).	We see consistent **positive correlation** between the number of threads and speedup. This makes sense ultimately because the problem size can be broken down into approximately equal components allowing for independent organization of the subproblems in each thread.
Did efficiency produce expected results	Efficiency is generated by observations of time for parallel components of the program only and does not include sequential code segments.	As efficiency is only a representation of the utilization of each core in an overall task, the efficiency results did match what I expected. Although speedup will gradually increase as the number of threads increase (for the values we tested), I did expect that there would be a point where efficiency would peak for a problem and then reduce as cores would generally not equally be utilized to solve a problem. In other words, speedup could be higher with 4 cores than 3 cores, but the 3 cores may have greater utilization and each may have more work to do than in the 4 core implementation.