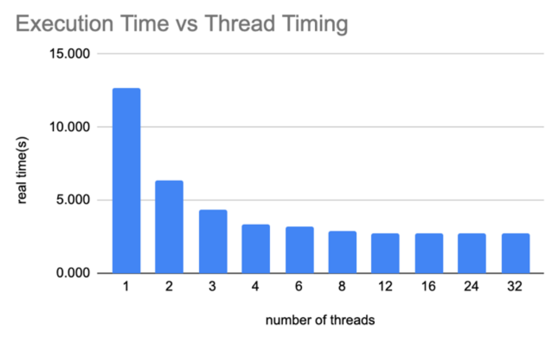
CPSC 457 Assignment 3

Question 2:

Table:

Threads	Timing (s)
1 (original)	12.585
1	12.704
2	6.369
3	4.341
4	3.324
6	3.176
8	2.884
12	2.760
16	2.747
24	2.719
32	2.710



a) We see n-times speed for approximately up to 4 threads, however, when the number of threads is increasing passed 4, the program speed flatlines and stays consistent.

b) Since threads and process can concurrently run on multiple cores, as we run the first 4 threads they each run on a core. Even though the number of threads is increased, the cores are occupied by other thread processes. The addition threads cant run at the same time and therefore the time is not N-times faster for n > 4.

Question 4:

Test File: medium.txt			
# threads	**Observed Timing (s)**	**Actual Speedup**	**Expected Speedup**
Original program	20.126	1.0	1.0
1	21.371	0.94	1.0
2	12.619	1.59	2.0
3	8.693	2.31	3.0
4	6.808	2.96	4.0
8	5.497	3.66	8.0
16	5.868	3.43	16.0

Test File: hard.txt			
# threads	**Observed Timing (s)**	**Actual Speedup**	**Expected Speedup**
Original program	6.878	1.0	1.0
1	7.283	0.94	1.0
2	3.745	1.84	2.0
3	2.584	2.66	3.0
4	2.036	3.38	4.0
8	1.647	4.18	8.0
16	1.749	3.93	16.0

Test File: hard2.txt			
# threads	**Observed Timing (s)**	**Actual Speedup**	**Expected Speedup**
Original program	6.878	1.0	1.0
1	7.277	0.95	1.0
2	3.719	1.85	2.0
3	2.586	2.66	3.0
4	2.031	3.39	4.0
8	1.646	4.18	8.0
16	1.816	3.79	16.0

The results for using 1-4 threads are as expected, the speedup is very close to the expected value, since cores are free to use up till 4 threads the speedup is approximately originaltime/n_threads. It is expected there is some discrepency due to the length of the code and other factors. Using 8 and 16 threads does not produce a speedup close to the expected value because use of too many threads (more than number of available cores) will cause the program to slow down relatively. Threads spend lots of time waiting for a core to free up so that they can run thir program.