Reah Rajmangal, rr2886
Parallel Computing: Lab 2, OpenMP: Results

Table 1: N=10,000

Table 2: N=100,000

N=10000	1	2	5	10	20	40	80	100
Speedup	1	1.303218	2.741979	5.199204	9.330677	17.59511	12.47506	12.31504
Time (s)	0.607735	0.466334	0.221641	0.11689	0.065133	0.03454	0.048716	0.049349
N=100000	1	2	5	10	20	40	80	100
Speedup	1	1.334244	2.742508	5.255664	9.929207	18.76418	28.2086	31.23361
Time (s)	63.22331	47.38511	23.0531	12.02956	6.367408	3.369362	2.241278	2.024208

In both cases (N=10,000 and N=100,000), speedup increases by almost half the number of threads going up to 40 threads. For instance, when N=100,000 and number of threads is 40, the speedup is almost 20x relative to number of threads=1. This is likely the case because I tried to reduce as much sequential coding as possible to increase performance as the number of threads increases—I focused on parallelizing the main algorithm with the for-loop and even parallelizing the initialization of the values in the array to "true" values beforehand. Hence my algorithm was mostly parallelized, with reading input and writing output the only sequential part of my code. There was no need to schedule dynamically the for-loops as my code seemed to be cache-friendly based on the speedup performance. It is interesting to note that speedup when the number of threads=2, 5, 10, 20 and 40 is about half the number of threads being used to run in each instance, which may indicate that the number of forking() and joining() threads increasing lead to an increase in overhead almost proportionately.

	1	2	5	10	20	40	80	100
Efficiency (N=10000)	0	0.303218	1.438761	2.457225	4.131473	8.26443	-5.12005	-0.16002
Efficiency (N=100000)	0	0.334244	1.408264	2.513157	4.673543	8.834972	9.44442	3.025006

There is speedup when N=10,000 and number of threads = 80 or 100, but it is not what we expect from the speedup beforehand. In other words, when looking at the efficiency of this program (the slope of speedup as we increase number of threads), efficiency increases until 40 threads when N=10,000. The decrease after (when threads=80 or 100) is likely because the overhead to create, fork and join 80 or 100 threads for each parallel part of the code costs more than parallelizing the code. The amount of data that we need to parallelize (10,000 numbers) is not enough to overload the amount of overhead the program will face. Efficiency also decreases when N=100,000 and number of threads is 80 or 100. However, unlike when N=10,000, efficiency remains positive but it is not as big as the previous efficiency points. This is likely due to the same overhead of creating threads, forking and joining them for each parallel part. If we were to increase the number of threads > 100 when N=100,000, it would be safe to predict that efficiency would soon become negative.