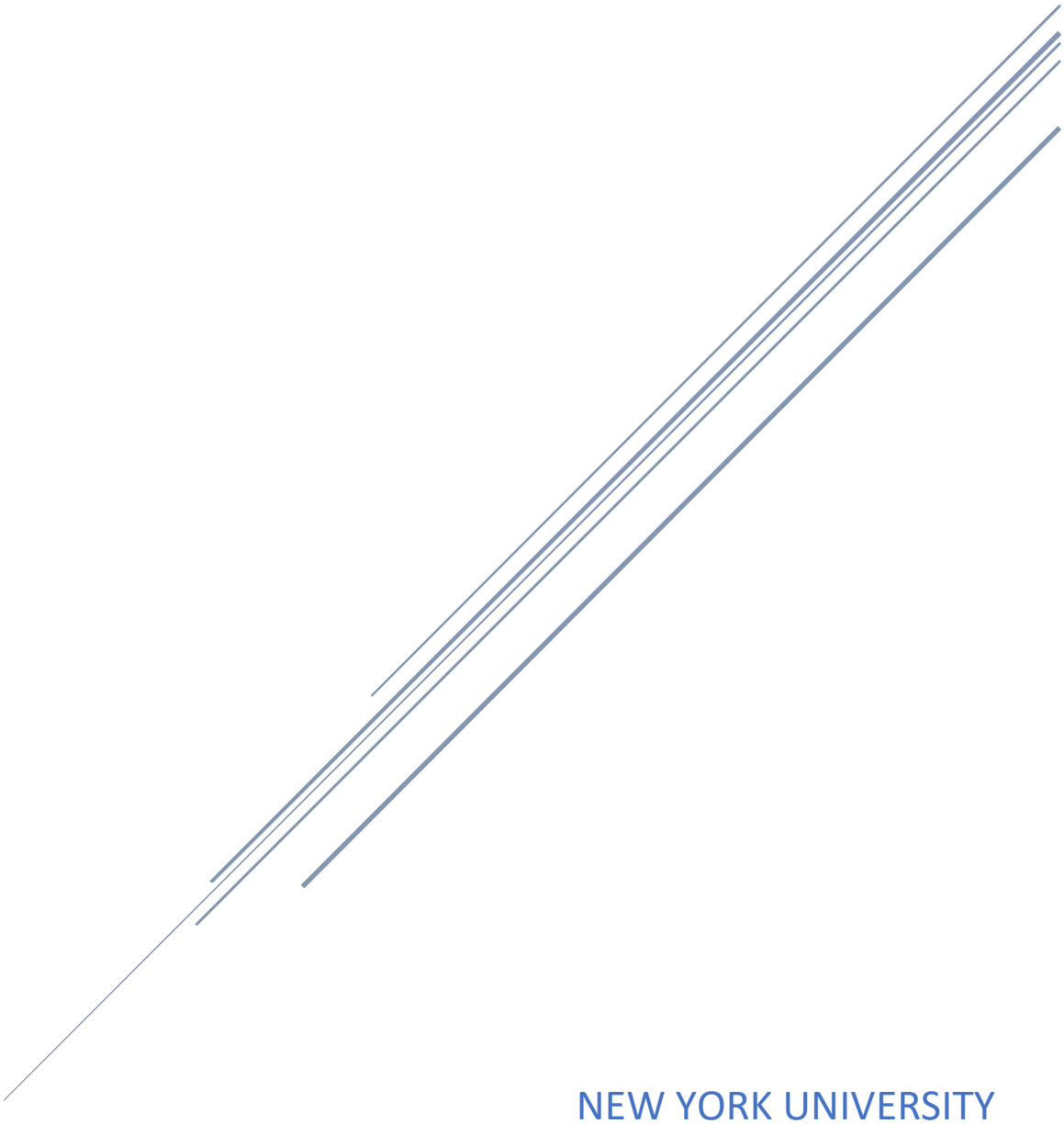
LAB 02
GENERATE PRIME NUMBER

MEISI LI

ml6095



NEW YORK UNIVERSITY

CSCI-UA.0480-003 PARALLEL COMPUTING

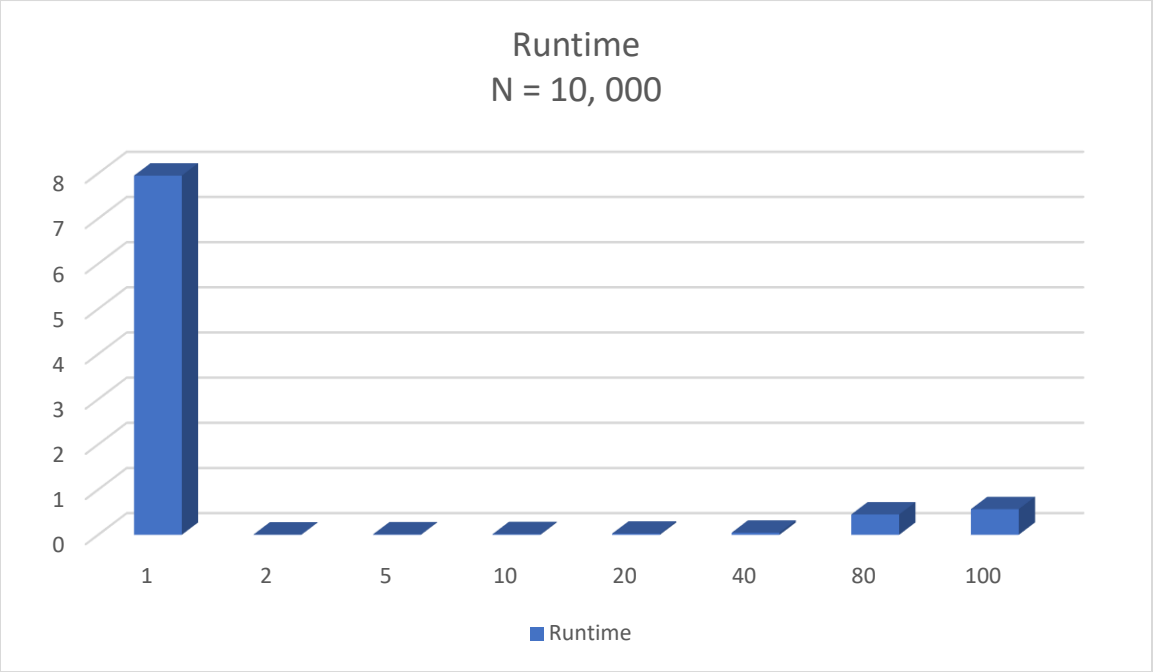
Tables:

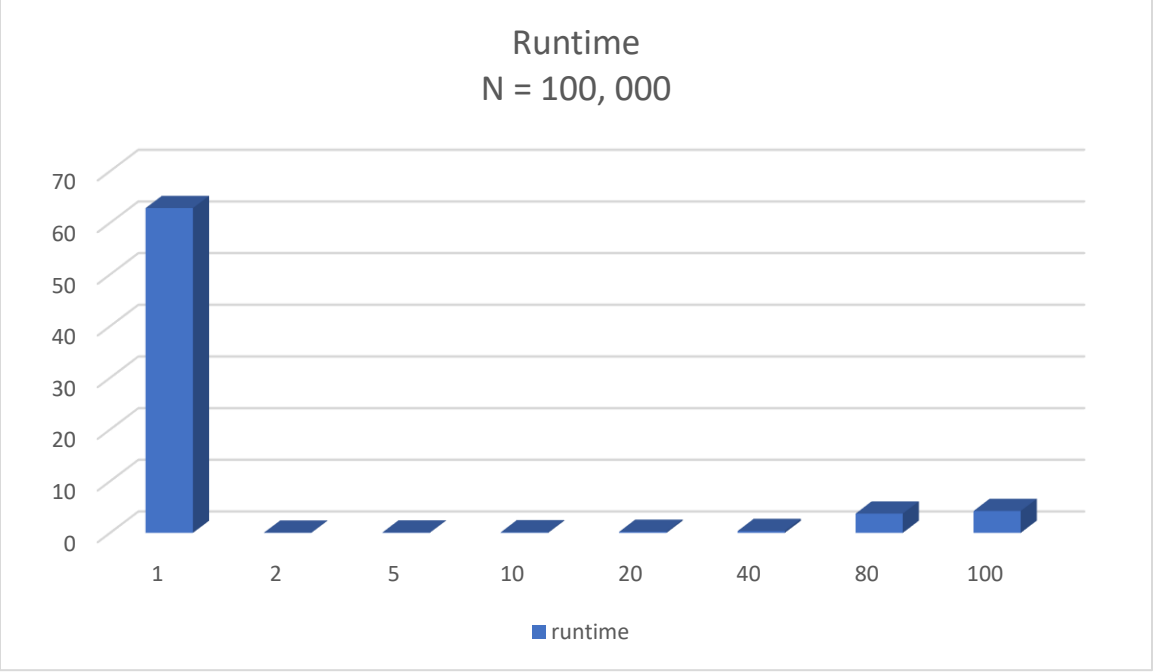
table 1(N = 10, 000)	
threads	runtime
1	7.95138
2	0.005883
5	0.009461
10	0.014354
20	0.025642
40	0.046569
80	0.4502
100	0.568555

table 2(N = 100,000)	
threads	runtime
1	62.85972
2	0.047069
5	0.076087
10	0.121951
20	0.217063
40	0.358614
80	3.819656
100	4.235837

From above table, the trend are similar. The largest runtime is when the thread is 1. From threads 2 to 100, the runtime is increasing but is smaller than serial type. This is because in my code, I made a the for loop in parallel function in '#pragma omp parallel for'. This could fork a team of threads to execute the following for loop and then diving the iterations of the loop among the thread.

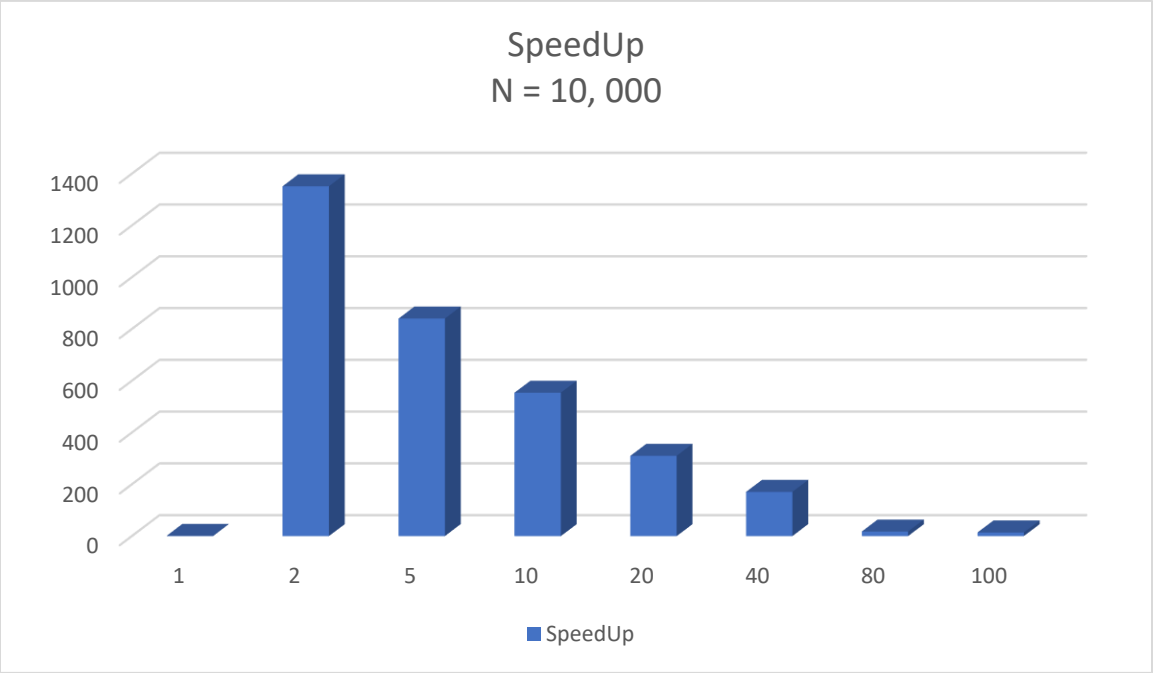
The graphs of runtime:

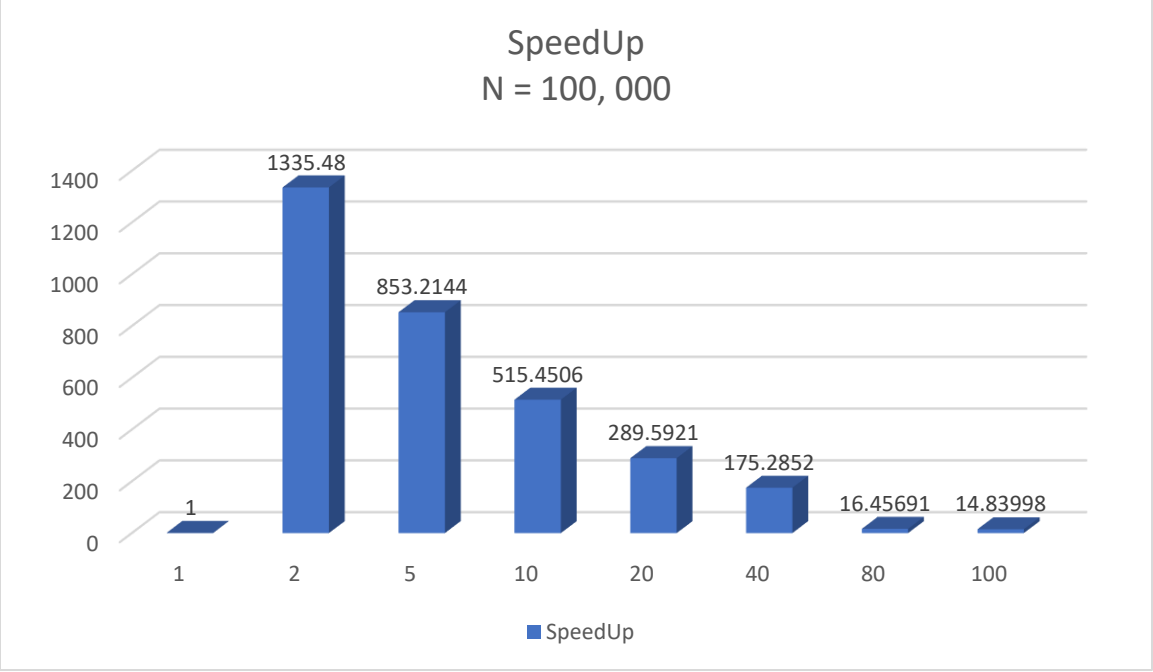


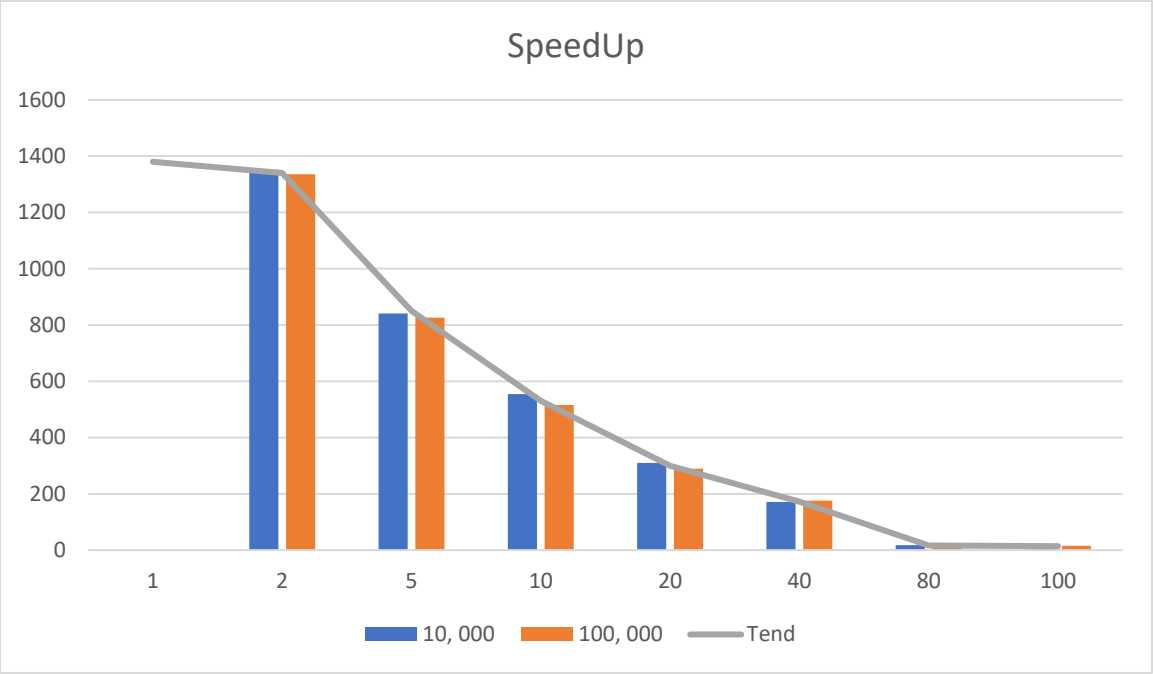


The graph of SpeedUp:

$$Speedup = \frac{Time_{Serial}}{Time_{Parallel}}$$







Conclusion:

The speedup distributions of N=10,000 and N=100,000 are similar. As we can see from above, the number of threads increases, the speedup is decreasing exponentially. It might because the performance cost of fork() and join() increases and then let the overhead increase.

Also, the speedup decreases when the N increase. Each thread will be distributed more data to work, it requires more time to execute.