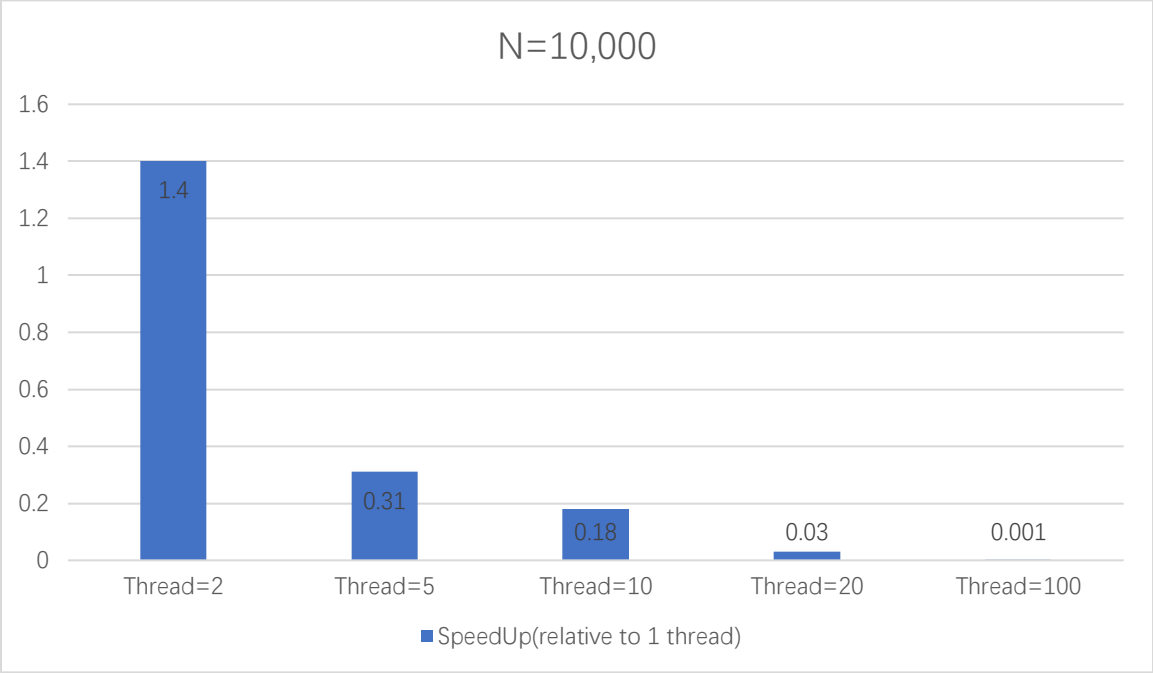
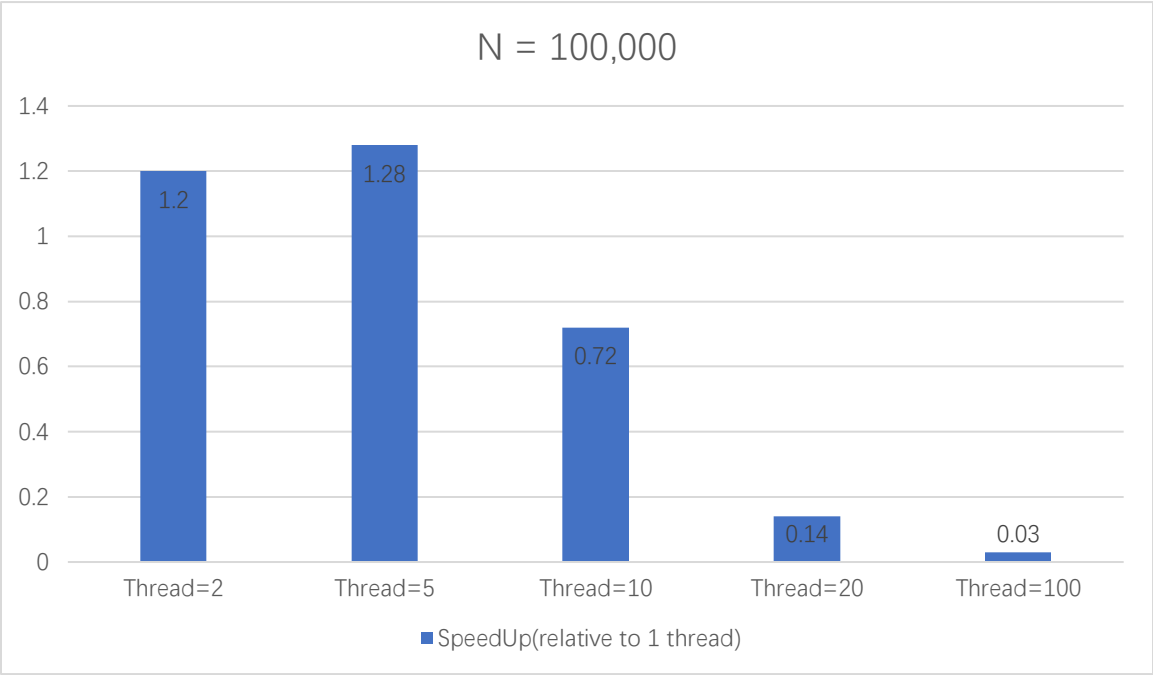
N=10,000	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
Speedup	1.4	0.31	0.18	0.03	0.001



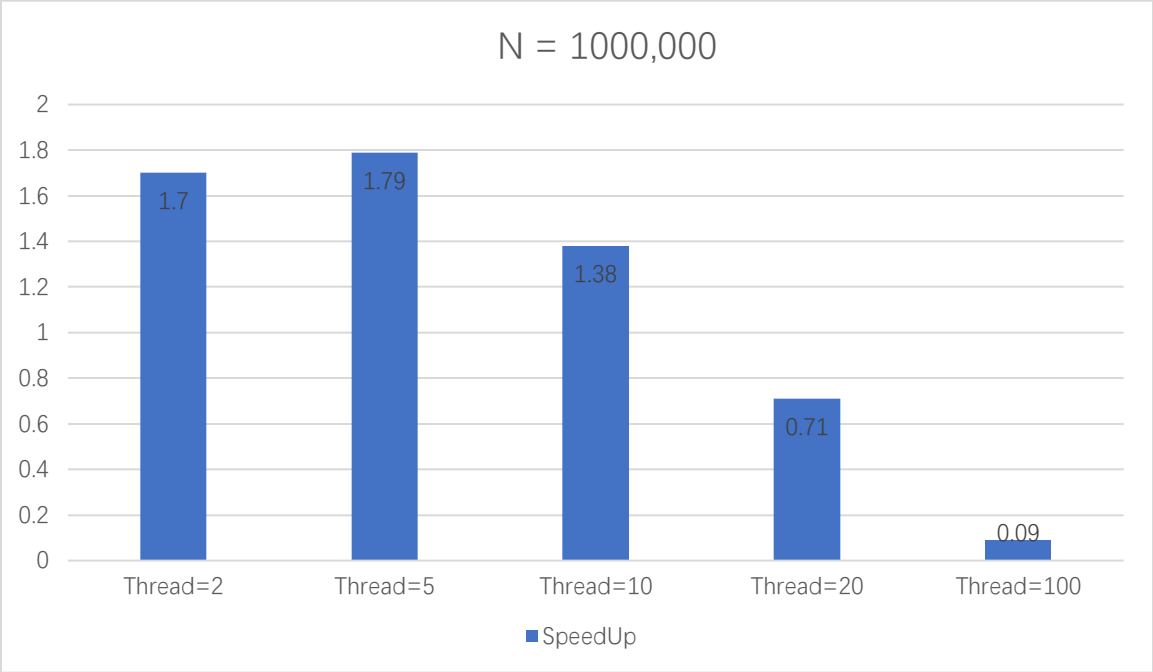
Observation: Thread = 2 have certain speedup, for Thread>2 they are all slower than sequential version. Mainly because the overhead in creating and managing the threads is much more consuming than doing the computation.

N=100,000	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
Speedup	1.2	1.28	0.72	0.14	0.03



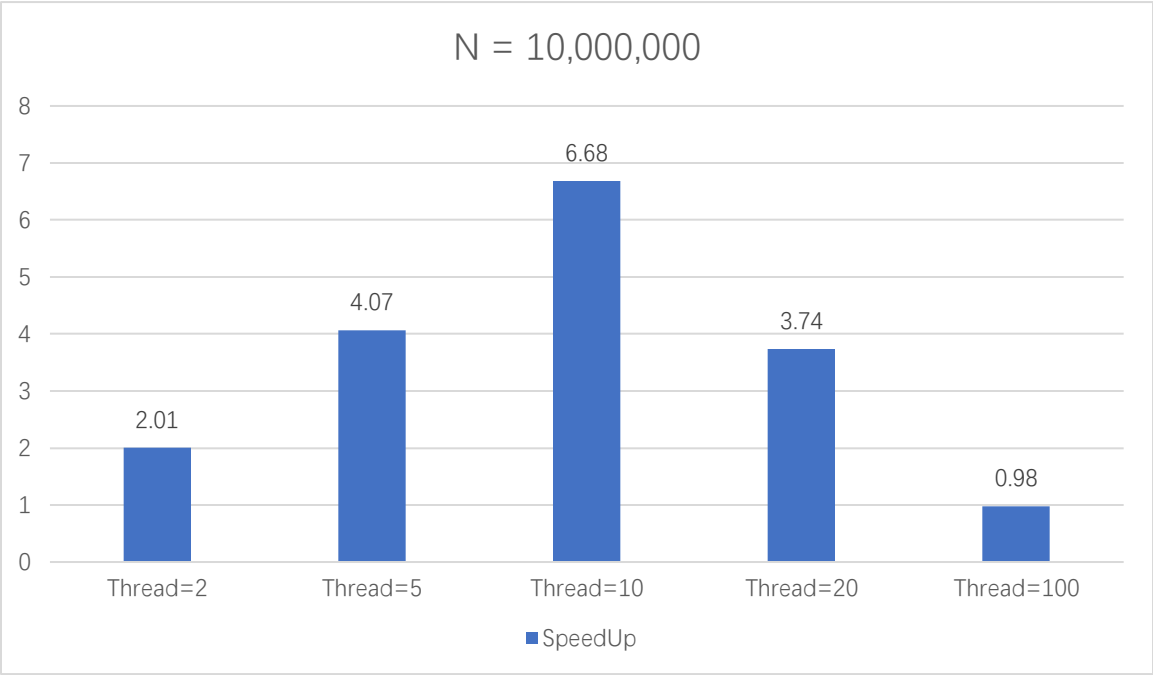
Observation: Thread = 2 & 5 have certain speedup, for Thread>5 they are all slower than sequential version. The reason is as above. But here Thread=5 also gain some speed up compared with sequential version, since a bit more work could be done in parallelism.

N=1000,000	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
Speedup	1.7	1.79	1.38	0.71	0.09



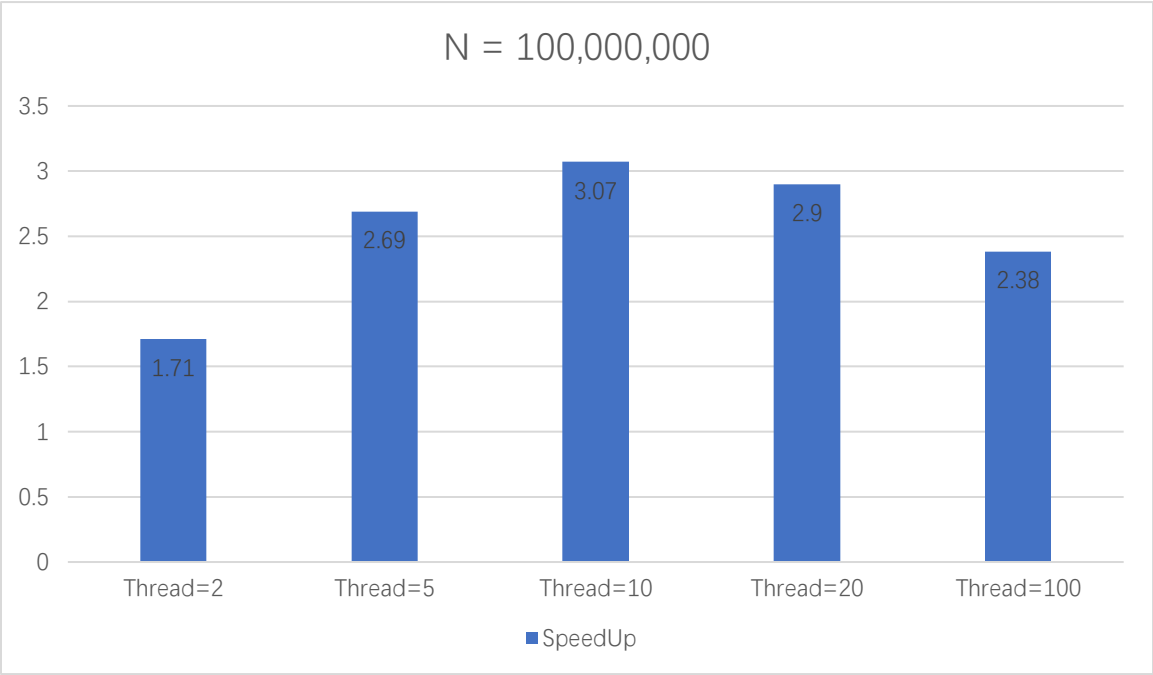
Observation: Thread = 2 & 5 &10 have certain speedup, for Thread>10 they are all slower than sequential version. We see gradually the speed up for more-thread-version starts to increase, it indicates that the work done in parallel is compensating the overhead in the initialization and management of threads.

N=10,000,000	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
Speedup	2.01	4.07	6.68	3.74	0.98



Observation: Thread = 2 & 5 & 10 & 20 have certain speedup, the version using 10 threads have highest speedup=6.68, the version using 100 threads now has very close performance compared to sequential version. In particular, Thread=5 and Thread=10 is enjoying a really high speed up and thus efficiency, which indicates that there are suitable for this problem for this particular size N=10,000,000.

N=100,000,000	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
Speedup	1.71	2.69	3.07	2.9	2.38



Observation: all versions have certain speed up, the highest speedup=3.07 occurs with using 10 threads. The N is so big that work done parallel, no matter how many threads you use (>=2), it could also make up for the time spent creating and managing the threads.

Analysis:

First of all, from the 5 tables we see the general pattern that when N is small only using 2 threads yield some speedup and using more threads the performances are worse than sequential version. As N goes big in magnitude, the version using more threads gradually gain some speedup accordingly.

The reasons behind these patterns are three-fold from my perspective.

First of all, the algorithm implemented is already quite efficient. The Sieve's algorithm is quite good with both time and space complexity O(n). Thus, we could believe that the sequential version is quite fast actually.

Secondly, the synchronization and thread management cost outweigh the performance gain by parallelism. As we know, creating and managing threads require operating system intervention which is costly in time. And the parallel for loop is implicit synchronization barrier and we must have that because the outer loop in the algorithm must be executed in sequential order, only the inner loop could exploit parallelism. When the problem size is not big enough, these overheads impact the performance and speedup.

Thirdly, the work imbalance problem. As we can investigate, when the outer loop

index gets big, there are only a few loops needed in the inner for loop because a few multiple of the loop index would exceed the N. Thus, maybe there are only 5 loops needed to be done while we are scheduling 100 threads to do this job, which is obviously very inefficient.

I also include the raw data statistics collected in the following table for easy reference.

	Thread=1	Thread=2	Thread=5	Thread=10	Thread=20	Thread=100
N=10,000	0.000151	0.000109	0.000696	0.000744	0.004326	0.017622
	0.000212	0.000113	0.000287	0.000921	0.007522	0.017438
	0.000144	0.000139	0.000903	0.001212	0.005438	0.017215
N=100,000	0.001545	0.001222	0.001277	0.002025	0.008480	0.054758
	0.001485	0.001332	0.001241	0.002222	0.015339	0.056442
	0.001728	0.001416	0.001193	0.002360	0.011660	0.041401
N=1000,000	0.018085	0.010642	0.010104	0.012308	0.028868	0.119461
	0.018132	0.011491	0.008940	0.011846	0.024931	0.102024
	0.017236	0.009458	0.010856	0.014454	0.021484	0.105376
N=10,000,000	0.255883	0.150954	0.080133	0.044310	0.093078	0.343904
	0.338197	0.157061	0.080465	0.056617	0.079131	0.335478
	0.391244	0.182340	0.081233	0.046570	0.091572	0.329871
N=100,000,000	3.687972	2.508753	1.587518	1.331571	1.355716	1.631962
	4.974683	2.547480	1.554420	1.463515	1.670205	1.964605
	4.193216	2.454245	1.626972	1.380171	1.403205	1.815574

(The above is raw statistics used in computing speed up. For each test I take it three times.

They are taken on NYU CIMS crunchy1 machine. I compute the speed up by taking the average of three records.)