Name: Jingfei Shao
Net id: js9950

Table1 N = 10,000						
Num of threads	1	2	5	10	20	100
Speedup	**1**	1.77	3.35	4.44	3.65	0.11

Table2 N = 100,000						
Num of threads	1	2	5	10	20	100
Speedup	1	1.93	4.58	8.37	12.87	0.92

Report:
In my application, there are two parts to generate the prime numbers: to create the original array and to cross out the non-prime numbers. I have entirely parallelized the first part, and for the second part— the nested for loop, I parallelized the inner for loop, and left the outer loop sequential, because I cannot control the rank of the threads and the loops are not independent. In other words, race condition occurs. So I left the outer loop sequential, which could harm the performance a lot. Also, for the parallelized inner loop, there are many synchronization points (say, 1 for each i, so total i), which can also affect the performance negatively. In addition, initially I found that I have repeatedly create threads that can be used again, and this worsened the performance as well. So I am just initializing threads once at the beginning of nested for loop and then reuse the threads now.

By increasing the number of threads gradually, some speedup can be achieved, which proves that the parallelization works. Yet, after increasing the number of threads to 100, the performance decreases, and the speedup goes below 1. The possible reason is that the problem size is not big enough, so the additional speedup achieved (probably no speedup is achieved because each thread is not assigned and doing enough work) is not strong enough to cover the additional overheads caused by creating more threads and have more synchronization points. Here, to have a sense of it, the times for creating 1, 10 and 100 threads are 0.000003s, 0.000009s and 0.001011s separately. Thus, for this particular problem, we can tell that the optimal number of threads for n = 10,000 and n = 100,000 would be somewhere around 10 to 20 — 100 threads are way too much for problems of this size.

Some addition notes: before coming up with this final solution, I have tried different ways to parallelize my code. One problem I encounter is race condition as mentioned above, and I found it quite hard to solve, so I left that part of code sequential. Another one is that by adding more loops, I need to create more synchronization points, which make the code slower as well. I also thought of another algorithm which does not use % to check if one number can be divided by another, (by crossing out all the numbers whose indexes is multiple of the current one), but no matter what I tried I cannot achieve any speedup. One reason is the problem size is not big enough, so not much space is left for

the performance improvement. Another one is that by implementing this algorithm, the runtime of sequential code would be something like 0.00004, which way too fast already.