Samira Mantri

NetID: sm6412

Lab 2

Table 1 (N=10,000)

Threads	**2**	**5**	**10**	**20**	**100**
Speedup	2	4.5	7.4	8.4	0.3

Table 2 (N=100,000)

Threads	**2**	**5**	**10**	**20**	**100**
Speedup	2	5	9.6	17.8	2.5

Explanation of Behavior for Tables:

As you can see the performance of my code continues to increase before suddenly dropping at the 100 threads mark for both tables. One reason this happens is because openMP is a shared memory system. With more threads, more time is spent alternating between them as they try to read and update data. Another reason for the behavior in the table is that I use "#pragma omp for" within the inner loop of the for loop that determines the prime numbers. After every "#pragma omp for" block exists an implicit barrier. As more threads are generated the program slows down when they hit the synchronization point because synchronization points increase the chance of load imbalance. Therefore, these factors affirm the behavior seen in the table because after a certain point adding more threads will make the parallel code slower than the sequential. Though both the tables follow the mentioned behavior, there is a noticeable difference. The speedup increases at a faster rate in the table with the larger data set

(N=100,000). This is because the larger data set allows for the parallelism to be more useful. The greater amount of work done in the sequential code, the better that amount of work was divided and worked on by numerous threads to the point that the extra overhead to create them was outweighed by the fact that parallelism was more efficient. At least until a certain point as I have discussed.