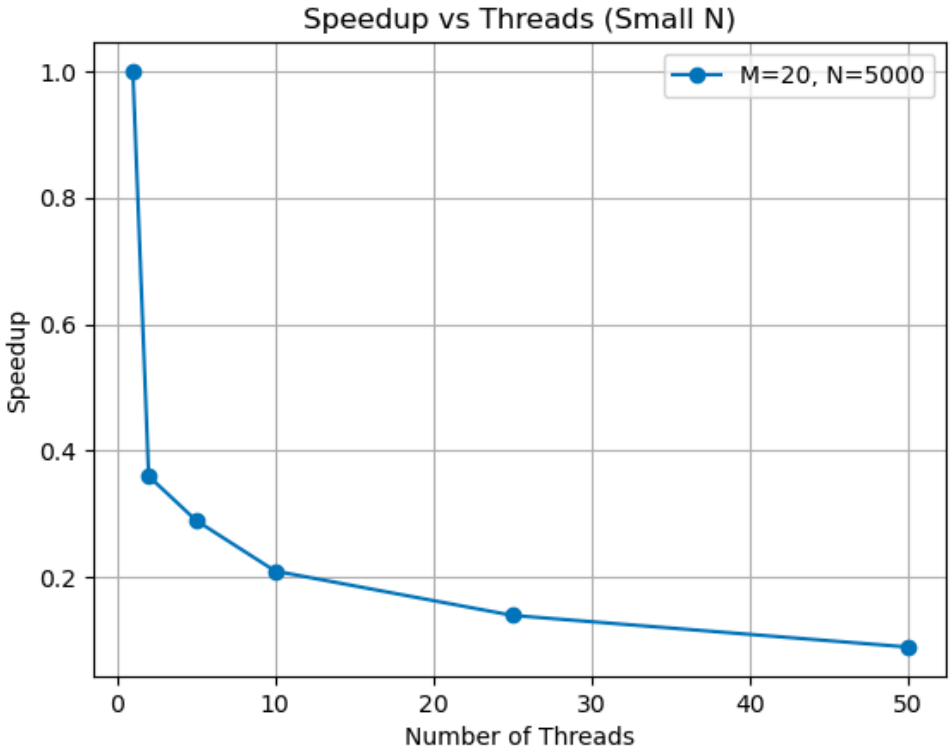
Graph 1

The formula for calculation speedup is given below where, T(n) is equal to time with t threads and T(1) is time with 1 thread:

Speedup(n) = T(1) / T(n)

Number of Threads	1	2	5	10	25	50
Time Taken (M= 20, N = 5000)	0.000149	0.000414	0.000512	0.000711	0.001078	0.001638
Speedup	1	0.36	0.29	0.21	0.14	0.09



For the smaller input size (M=20, N=5000), we observe that increasing the number of threads beyond a certain point does not improve performance. It worsens the execution time. This happens because the overhead of creating and managing multiple threads outweighs the benefit of parallelizing such a small workload. This behavior is expected because for small N=5000, adding more threads increases the overhead. The workload is small, so the communication time dominates over the actual computation. That's why the speedup is actually decreasing with more threads. The optimal performance was seen with 2-5 threads, after which performance degraded.

Graph 2

The formula for calculation speedup is given below where, T(n) is equal to time with t threads and T(1) is time with 1 thread:

Speedup(n) = T(1) / T(n)

Number of Threads	**1**	**2**	**5**	**10**	**25**	**50**
Time Taken (M= 20, N = 5,000,000)	1.224548	0.682533	0.308103	0.224178	0.197697	0.177881
Speedup	1	1.79	3.97	5.46	6.19	6.88

For the larger input size (M=20, N=5,000,000), we see a clear improvement in performance with an increasing number of threads. This is because the workload is large enough to effectively utilize multiple threads. The time taken reduces significantly as the number of threads increases. Although we still use a critical section to update the shared array of primes, its impact is negligible here because checking prime dominates the overall execution time. As a result, higher thread counts give better speedup.