Parallel Computing - Lab 2

Vaibhav A Gadodia - vag273

April 18, 2019

1 Method

The way the program has been parallelized is that the 'for' loop that initializes the array of potential prime numbers is distributed among the team of threads. The outer 'for' loop in the code block removing the composite numbers is the other loop being distributed. Here, each thread checks if a number in the array has already been removed or not and removes the number's multiples, if not.

This doesn't affect the solution's correctness because if a number is actually a prime then the work flow remains same as in a sequential program. However, in case a number is actually a composite that has not been cancelled out already, then, treating it as a prime and cancelling out its multiples doesn't affect the final solution as they would have been cancelled out through a prime factor of the current number that is being treated as a prime. For example, removing the multiples of 4 is the same as removing the multiples of 2 from the list of potential prime numbers.

2 N = 10,000

2.1 Data

Threads	1	2	5	10	20	100
Time	0.0944934 s.	0.0587814 s.	0.0310582 s.	0.01724 s.	0.0118368 s.	0.0132724 s.
Speedup	1	1.61	3.04	5.48	7.98	7.12

2.2 Analysis

The speedup increases as we increase the number of threads, albeit with a diminishing rate, hitting the peak somewhere between 20 and 100 cores.

From the time statistics obtained during the program runs, it can be concluded that the reason for the speedup and its diminishing rate is that the prime number generation portion of the program spends the most time in cancelling out the composite numbers. Since, this particular process is one of the loops that has been parallelized, an increased number of threads saves a significant amount of time, thereby, producing appreciable speedups as the number of threads increases. The diminishing rate of speedup is explained by the thread creation overhead, which, significantly impacts the performance here as the problem size isn't big enough in comparison to the time lost to this overhead and the implicit synchronization point that shows up after the 'for' loop when the threads join. Since, threads have the potential of asynchronous cache misses, this synchronization point becomes worse as the number of threads increases.

3 N = 100,000

3.1 Data

Threads	1	2	5	10	20	100
Time	6.759140 s.	4.239036 s.	2.165987 s.	1.217675 s.	0.638645 s.	0.254488
Speedup	1	1.59	3.12	5.55	10.58	26.56

3.2 Analysis

The speedup increases as we increase the number of threads, albeit the rate of increase slowly begins decreasing as the number of threads used is increased. The speedup hits its peak at or somewhere after 100 threads.

The reason for the speedup is similar to the one given in the previous section for the smaller problems size. However, here, due to the larger problem size, the benefits of parallelization are even higher and this is evident in the higher speedups obtained here. However, the speedup doesn't exactly increase linearly with the increase in thread number because of the increased thread creation overhead and the increased potential of synchronization points slowing down the program.