Nikhil Gosike

Parallel Computing

Professor Mohamed Zahran

4/17/19

Introduction

This is a program finds and outputs into an output file every prime number from 2 to N with t number of threads. N and t are specified by the user in command line when running the program.

Tables

Below are two tables to show and understand the speedup and behavior of my program for finding all primes between 2 and 10,000 as well as between 2 and 100,000. Each problem size is run with 1, 2, 5, 10, 20, and 100 threads. The speedup is measuring as S = T\_serial/T\_parallel, where T\_serial is taken as running this program with 1 thread. It should be noted that this program was run on the crunchy machines through my NYU CIMS account. This made for some inconsistent runs that were significantly different from the normal run time depending on which crunchy machine was being used and the time of day (e.g. many students using a crunchy system at once made for inconsistent runs).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 10000.txt | | | | |  |
| # of threads | 1 | 2 | 5 | 10 | 20 | 100 |
| time | 0.09656 | 0.006758 | 0.023286 | 0.035385 | 0.123501 | 0.319818 |
| speedup | 1.00 | 14.29 | 4.15 | 2.73 | 0.78 | 0.30 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 100000.txt | | | | |  |
| # of threads | 1 | 2 | 5 | 10 | 20 | 100 |
| time | 0.143312 | 0.063487 | 0.0638654 | 0.125106 | 0.195271 | 0.377197 |
| speedup | 1.00 | 2.26 | 2.24 | 1.15 | 0.73 | 0.38 |

For 10,000 primes, from the table, we see that 2, 5, and 10 threads give significant speedups greater than 2, which quickly decreases to speedups less than 1. There also a clear downtrend in speedup as we go from 2 to 100 threads. As the number of threads increases, forks and joins of this many threads creates performance issues that makes the problem less than ideal compared to sequential performance. In addition, communication among the threads also becomes important and an expensive operation. Furthermore, there is a sort of “work imbalance” due to the inner for loop of my code. For example, when crossing out multiples of 2, the inner for loop iterates approximately 5,000 times, while for larger number such as 5,000 it only iterates 2 times. Since parallelizing the outer for loop splits segments from 2 to floor, we see that the work imbalance increases heavily as threads increase, leading to a decrease in performance. For 100,000 primes, we see the same trend, but the speedup is much less and falls off at a faster rate. This is mainly due to the work imbalance that becomes further emphasized with a larger problem size. Of course, there is more data to work with as each thread enters the parallel section which generally leads to decreased performance.