COP4521 Programming Assignment 3: Compute Prime Triplet with Multiprocessing

Objectives:

- Practice parallel programming with multiprocessing
- Experience with performance optimization

Description:

A prime triplet is a set of three prime numbers where the smallest and the largest differ by 6. For example, (5, 7, 11) and (13, 17, 19) are prime triplets. In number theory, there is a prime triplet conjecture, which states that there are infinitely many prime triplets. In this assignment, you will write a multi-process Python program that prompts the user to input an integer number (positive or negative), and then display the smallest prime triplet whose smallest prime is no less than the number entered. Note that prime numbers are positive numbers (the smallest prime is 2). The multi-process program should also allow the user to specify the number of worker processes used in the program in the command line argument. For example, "python3 assignment3.py 16" would run the program with 16 worker processes.

Due time: October 14, 2024, 11:59pm.

Submission: Name your program lastname_firstinitial_mpprimetriplet.py and submit on Canvas.

Grading (70 points total):

- Programs with a runtime error for legitimate input (including 1000,000,000,000) will get no more than 5 points.
- Programs that do not use more than one process will get no more than 5 points.
- Include basic header (template at the course website) for assignment and name your program lastname_firstinitial_mpprimetriplet.py. In the header, summarize the program execution times for two inputs 1,000,000,000,000 and 3,000,000,000,000 with 1, 2, 4, 8 worker processes (10 points).
- Include correct time measurement code to measure and output program execution time excluding the user input time (5 points).
- Allow the user to specify the number of worker processes in the command line argument (5 pts)
- Pass correctness test cases for 1 and 2 worker processes. The program will be tested on randomly generated test cases from negative number to 4,000,000,000,000. (20 points)
- Pass correctness test cases for 1, 2, and 4 worker processes and **has at least a speedup of 1.2 for sufficiently large input** with multiprocessing (versus using one worker process). The program will be tested on randomly generated test cases from negative number to 4,000,000,000,000. (10 points)
- Pass correctness test cases for any number of worker processes between 1 and 16 (inclusive) and **has at least a speedup of 1.2 for sufficiently large input** with multiprocessing (versus using one worker process). The program will be tested on randomly generated test cases from negative number to 4,000,000,000,000. (5 points)

- Compute prime triplet using multiprocessing with no more than 16 worker processes in less than 4 seconds on linprog for N=1,000,000,000,000 and have similar speedups for other sufficiently large input values. Only programs that pass the correctness tests 100% can get the points in this item. (7 points)
- Compute prime triplet using multiprocessing with no more than 16 worker processes in less than 2 seconds on linprog for N=1,000,000,000,000 and have similar speedups for other sufficiently large input values. Only programs that pass the correctness tests 100% can get the points in this item. (3 points)
- Compute prime triplet using multiprocessing with no more than 16 worker processes in less than 1 second on linprog for N=1,000,000,000,000 and has similar speedups for other sufficiently large input values. Only programs that pass the correctness tests 100% can get the points in this item. (3 points)
- Compute prime triplet using multiprocessing with no more than 16 worker processes in less than 0.8 second on linprog for N=1,000,000,000,000 and has similar speedups for other sufficiently large input values. Only programs that pass the correctness tests 100% can get the points in this item. (2 points)
- +5 **extra** points for the first person to report an error in the provided code.

Notes:

- Writing efficient parallel code requires using efficient sequential code as the baseline. For example, if the sequential code to be parallelized runs for more than 30 seconds when N=1000000000000, it is then virtually impossible to reach 2 seconds with multiprocessing on linprog using 16 worker processes.
- A sequential program is provided, which runs for about 5 seconds when N=1000000000000. You are not required to use this program in any way. But you may borrow some ideas in this code to improve your baseline sequential code, or directly use this as your baseline if you choose to. Parallelizing your own program is always preferred. If your sequential baseline runs for more than 10 seconds on linprog for that input, it is unlikely that the corresponding multi-process code can run less than 1 second.
- The provided code has the timing logic for your reference.
- A reference implementation has about 180 lines of code. Parallel programming is inherently challenging. If you never wrote parallel programs before, you would encounter various obstacles doing the assignment. Start NOW and prepare to spend a lot of time on it.
- You can find the parallel programming design pattern in the example code that we discussed in class (pi_mw.py and primes_mw_sort.py) and apply the pattern in this assignment.