Parallel and Distributed Computing: Homework 2

Due April 17, 2020, in pdf format on Canvas.

Consider the computer program `prime_print` that finds and prints all primes between 1 and 10^7:

```
void primePrint:
    long i = 0
    long limit = power(10,10)
    while i<limit do:
         if isPrime(i) do:
              printf(i)
         end if
         i = i+1
    end while
```

Assignment

Your task is to implement two versions of `primePrint` in C. One version is a sequential program (single thread) while the other is a concurrent implementation using threads and a concurrent data structure. You are to finally provide a comparative analysis.

The input The single-thread version will not receive any input on the command line. The multi-threaded version will receive the number of threads to be used as a command line argument (not as standard input). This number should be an integer greater than or equal to 1.

The computation Both programs will calculate and print the prime numbers from 1 to 10^7, and report their execution time.

For the sequential (single-threaded) version, the computation is straight forward.

For the parallel (multi-threaded) version, as a first attempt to parallelization, you might consider giving each thread an equal share of the input domain to distribute the input set evenly. For example, with 10 threads, each thread might check intervals of 10^6 consecutive numbers. However, this approach fails to distribute the load in a balanced manner because equal ranges of inputs do not necessarily produce equal amounts of work. Primes do not occur uniformly: there are many primes between 1 and 10^6, but hardly any between $9 \cdot 10^6$ and 10^7. In addition, the time complexity of the prime checker is proportional to the number tested. Therefore, the larger the numbers in the interval, the slower the prime test.

A more promising way to split the work among the threads is to assign to each thread successive integers, one at at a time to threads ready to run. When a thread is finished testing an integer, it asks for another from a central source (shared concurrent data structure.) In this particular case, the assignment of a single integer at a time is done using a shared counter accessible by each thread, one thread at a time. Each thread accessing the counter reads the counter and increases it to generate the next integer to be tested by the next thread requesting an integer. To safely increase the shared counter in this concurrent execution, a synchronization mechanism (such as mutex) must be used.

The output Both programs have to print on screen all the prime numbers between 1 and 10^7, one per line. In the last line, the program must print its total execution time in seconds, with 4 decimal positions. Every line must end with "`\n`".

Submission

Submit a zip file named `x.zip`, where `x` is your UCInetID, containing 3 files:

1. The commented C source code of the single-thread version of the program, called `single_prime_print.c`.
2. The commented C source code of the multi-thread version of the program, called `multi_prime_print.c`. It has to implement the shared counter approach, and determine the number of threads utilized based on the command line argumnts given by the user.
3. A report that includes:
    - A description of the system where the programs were executed: Processor, RAM Memory, and Operative System.
    - A comparative table showing the execution times of the single-thread version, the multi-thread with 1, 2, 4, 8, 16, 32, 64, and 128 threads.
    - An analysis of the performance of the programs with emphasis on correctness and speedup.
    - An analysis of the single-thread version versus the multi-thread version with only one thread. Is there a difference? How do you explain the results?

To achieve scalability in the experimental results, a computer with a multi-core processor should be used. A four-core processor is strongly recommended. What happens if *Simultaneous Multi-Threading* (click on emphasized text to open a web page on this subject) is enabled or disabled?