Parallel and Distributed Computing: Homework 3

Objectives: This is the first assignment to illustrate and practice multithreaded programming. You will bi able to put to practice what was presented in class regarding protected data structures, in this particular case a common counter, and the creation of threads capable of running simultaneously in a multi-core system. The prime number printing program is important in its own right and is simple enough for all to understand. A testing for prime addendum is given.

Consider the computer program below that finds all primes between 1 and 10^7.

Algorithm 1: Prime Finder

```
1  Function prime_finder():
2      i ← 1;
3      limit ← 10^7;
4      while i ≤ limit do
5          if isPrime(i) then
6              primes_found.append(i)
7          end
8          i ← i+1;
9      end
10 return
```

Assignment

Utilizing the provided template code, your task is to implement two versions of the prime finder function in C. One version is a sequential function implemented in `primes_sequential()`, while the other `primes_parallel()` is a concurrent implementation that uses multiple threads and a concurrent data structure. You are to finally provide a comparative analysis.

Program specification

Your programming task is to implement the functions `primes.c:primes_sequential()` and `primes.c:primes_parallel()`.

Both functions receive a first parameter `primes_list`. This is a dynamic concurrent array of integers that you will use to store all the primes found. This data structure has its own functions to operate it. The definitions of such functions and their documentation can be found in `common/conc_darray.h`.

The second argument is the last number in the testing range. Note that in both, the sequential and the parallel implementation, the minimum number in the rage is always 2.

The third argument is only used in the parallel implementation of the function. This specifies the number of threads that will work together in the search for prime numbers.

```
primes_sequential(carr_d_t *primes_list,
                  unsigned int max);
primes_parallel(carr_d_t *primes_list,
                  unsigned int max,
                  unsigned int thr);
```

As for debugging purposes, you are highly encouraged to use the macro `common/util_common.h:DEBUG` , and to change the content of any given source code to fit your needs. However, consider that:

- The only source file to be submitted is `primes.c`. So declare and define all data structures, static functions, contstants, etc., that your solution needs within that file.
- Your implementation must match the provided function signatures.
- Only standard libraries and the ones defined in the template code can used.

Besides these restrictions, you are free to design your solution in any which way you like. The file `primes.c` is used to build the program prime numbers.bin . This program will call your function with different parameters. It already handles and check the correctness of the command line arguments that it receives. You can check details on how to pass arguments to it by running `./prime_numbers.bin help`.

The computation Both functions will find and store the prime numbers from 2 to the given maximum, which will be always less or equal than 10^7. After storing the prime numbers found, the program will print the time taken to completion (already in the `main.c` template file. Your function should NOT print anything).

For the sequential 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 interval to distribute the input set evenly. For example, if 10 threads are used to find primes up to $10,000,000$, 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 10^0 and 10^6, but hardly any between $9 \cdot 10^6$ and 10^7. In addition, the time complexity of the prime checker may be proportional to the number tested. Therefore, the larger the numbers in the interval, the slower the prime test may become.

A more promising way to split the work among the threads is to assign to each thread successive integers one at at a time. 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. This is the approach you will use in your implementation.

To achieve scalability in the experimental results, a computer with a multi-core processor should be used. A four-core processor is strongly recommended.

The output Both functions have to store all the prime numbers between 2 and the given maximum in the `primes_list` data structure. The numbers don't need to be in any particular order. At the end of the execution, `main.c` prints the total execution time in seconds with 8 decimal positions. Use this value in your preformance analysis as you vary the number of threads and maximum number to test.

Report

Write a report of a maximum of three pages considering the following points:

- A description of the system where the programs were executed: Processor, RAM, and Operative System.
- A comparative table showing, on the search up to 10^7, the execution times of the single-thread version, and the multi-thread with 1, 2, 4, 6, and 8 threads.

- An analysis of the performance of the programs with emphasis on correctness and speedup.
- An analysis of the sequential version versus the multi-thread version with only one thread. Is there a difference? How do you explain the results? Add plots and figures as necessary to explain your results.

What happens if Simultaneous Multi-Threading (or "Hyper-Threading") is enabled or disabled? You may find the article in logicalincrements.com useful.

Submission

Submit **one** zip file named `hw3.zip`, containing **exactly 3 files**. Make sure the zip file does **not** include sub-directories (as Mac's default compression tool does) or extra files:

1. `primes.c`: The well commented C source code of your sequential and multi-threaded implementations of the functions here described. **Do not submit any other source code file. The signature of the functions must be exactly as it is given to you in the template**.
2. `report.pdf`: The digitally produced report of a maximum of three pages.
3. `team.txt`: Text file with two lines containing the UCINetID and name of each student in your team:
   `<UCINetID>, <firstName>, <lastName>`

Addendum: Checking if a number is prime.

This article was copied from the Internet and converted into a Latex document. The original Internet article is no longer available.

You may remember that a prime number is one whose only factors are itself and 1. Other numbers are called oblong numbers (they can be represented as an oblong of dots) or composite numbers. Today we examine some shortcuts for finding the prime numbers.

Let's take 37 for example. Is it prime? One way is to try each number in turn, from 2 to 37 to see if any of them divide exactly in to 37. On your calculator this is easy, because the answer to this division calculation would end with ".0" or just show as a whole number. If it shows anything after a decimal point, then it didn't divide into 37 exactly.

A Shortcut

One shortcut is to realise that if, say, 15 did divide into the number we are testing, then so would 3 and also 5 since 15=3x5. Similarly, if 18 was a factor of the number being tested, since 18=2x9=2x3x3 then 2 would also be a factor and so would 3 (and also 2x3=6 and 3x3=9).

So one shortcut is to only test **prime numbers** smaller than the number you are testing as possible factors. So, instead of checking 2,3,4,5,6,7,... all the way up to 36 to see if 37 is prime, we need only test 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 and 31. But there is another shortcut too:

Another shortcut!

Since 3 is a factor of 18 and 18/3=6, then 18=3x6. We need only test the smaller number, 3, to see if it is a factor of 18 and not 6. Similarly, 5 is a factor 20 and 20=5x4 so we need only test the smaller number 4 as a factor. But what if we don't know the factors of a number? So, testing a number to see if it is prime means we need only test the "smaller" factors.

But where do smaller factors stop and larger factors start? The principle here is: Suppose one number is a factor of N and that it is smaller than the square-root of the number N. Then the second factor must be larger than the square-root.

We can see this with the examples above: For 18, we need only test numbers up to the square-root of 18 which is 4.243, i.e. up to 4! This is much quicker than testing all the numbers up to 17!! For 25, we need to test numbers up to and including the square-root of 25, which is 5. And for 37, we need only go up to 6 (since 6x6=36 so the square-root of 37 will be only a little bit larger).

Which numbers to test as factors

So, putting these two shortcuts together, we need only test those prime numbers up to 6 to see if they are factors of 37. If any are, the number is not prime (it is composite) and if none of them are, then the only factors would be 1 and 37 and 37 would be prime. The numbers to test are therefore:

- **Test for 2:** 37 is not even, so 2 is not a factor of 37.
- **Test for 3:** 37/3 is 12.3333 so 3 does not divide exactly into 37 either.
- **Test for 5:** 37 does not end with 0 or 5 so 5 is not a factor of 37.
- **Test for 7:** 7 is the next prime to test, but it is bigger than the square-root of 36, so we can stop now.

So 37 has no factors (except 1 and 37 of course) and therefore 37 is a prime number.