Prime Threading

CSE 1325 - Fall 2019 - Homework #13 (Lab) - 1

Due Tuesday, November 12 at End of Lecture

Assignment Overview

To date, we've been content to use only one of the processing cores in our machines. But those other cores have become lonely! It's time to give them something useful to do.

You MAY use the example provided with Lecture 22, discussions with other students / SI / TAs / professor, Internet searches, or any other resource to complete this lab, EXCEPT to copy code from another student!

Full Credit

Using your Linux-based development environment, and in your GitHub-managed working directory **cse1325/P13/full_credit**, expand the provided ZIP file. You'll find the following files:

- **primes.cpp** - A non-threaded command line program (no gtkmm) that searches for all prime numbers between two integers, inclusive. You will change *only one integer* in this file.
- **threads.cpp** - Identical to primes.cpp. This assignment is to add threading to this version to speed up the search! **You will modify the code in this file**
- **Makefile** - Builds primes.cpp (`make primes`) and threads.cpp (`make threads`), and times their execution (`make timep` and `make timet`, respectively). You may, but do **not** need to, modify this file.
- **results.txt** - You will document the runtimes for primes.cpp and for threads.cpp with 2, 4, 8, 16, and 32 threads *on your machine*. This, plus changes to threads.cpp, will determine your grade for this assignment.

Step 1

Compile and time the execution of primes.cpp *on your machine* using `make timep` (the trailing p is for primes.cpp). The list of primes found will be in the file primes.txt, if you'd like to verify correct operation.

The command line parameters for primes are

1. the number of threads (default 1), which is ignored in this version, and
2. the maximum integer (default 10,000,000) to search for prime numbers (the main function always starts searching at 2, though class Prime_numbers is more flexible)

The parameters may be passed directly to primes, e.g., `primes 1 100000`, or via the Makefile, e.g., `make timep 1 100000`.

The output of the time command offers 3 times:

- **real** - This is the "wall time", that is, how long it took the program to run as experienced by the human at the console (that's you!).
- **user** - This is the total time spent by all cores in "user" mode, running your application code.
- **sys** - This is the total time spent by all cores in "system" mode, running kernel calls from your application code.

So the time as experienced by the computer (equivalent to the "system load") is **user+sys**, while the time experienced by the user is **real**.

Adjust the value of max_int in the main function of primes.cpp until it runs in between 20 and 30 seconds on your machine. Record this number in results.txt.

Step 2

Edit max_int in threads.cpp to match the value you selected in Step 1. Verify (almost) identical execution time using `make timet` (the trailing t is for threads.cpp). The list of primes found by this program will be in the file primes_threads.txt, and you can verify them using `diff primes.txt primes_threads.txt`.

Step 3

Modify threads.cpp to search for the prime numbers using two or more threads. `git add -u ; git commit` often!

- All of your changes will be to class Prime_numbers.
- You will need to refactor method **find_primes**, which currently searchs linearly for primes between lower and upper, into two methods: **find_primes_thread**, which performs the same search but *safely* updating the primes vector (HINT: You'll need a new class attribute for this!), and **find_primes,** which in the new version just creates and manages threads running find_primes_thread.
- The number of integers to be searched per thread is simply the difference between upper and lower, divided by the number of threads specified as the constructor parameter. Each thread should search a DIFFERENT range of integers, obviously.
- The default number of threads is set to 4, but this can be changed in your code and via the first parameter on the command line in threads.cpp.

You can compile and run the program using `make timet`. The list of primes will be in the file primes_threads.txt.

You can determine if the order of primes found in primes_threads.txt is different from the order of primes found by the non-threaded version in primes.txt using `diff primes.txt primes_threads.txt`. You can verify that both threaded and untreaded programs found the *same* prime numbers, even if in a different order, using `sort -n primes_threads.txt | diff primes.txt -` (note the trailing dash).

Step 4

Time your threaded version of threads.cpp running each different number of threads as listed in results.txt, recording your time results in results.txt. Answer the questions in results.txt as well.

Deliverables

Ensure that you have committed *and pushed* (at least) **primes.cpp**, **threads.cpp**, and **results.txt** to GitHub before you leave class today.