Tennessee TECH

CSC1300: LAB 12

CONCEPTS:

- Arrays
- Parallel Processing
- Functions
- Separate files

PARALLEL PROCESSING

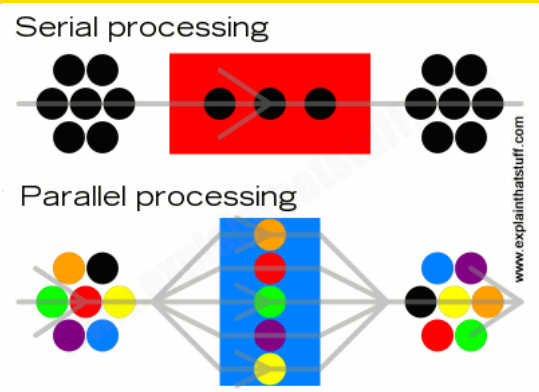


Image copied from http://www.explainthatstuff.com/how-supercomputers-work.html

BACKGROUND

Beginning in the early 2000's chip manufacturers realized that they were quickly reaching a critical point in the production of microprocessors. The amount of power required to keep increasing the clock speed of a single core processor was a power function meaning they were quickly reaching a point where very large increases in power consumption would result in only minimal speed gains. In addition, this increase in power was also causing extreme heat that would have to be dissipated or else risk melting important computer parts.

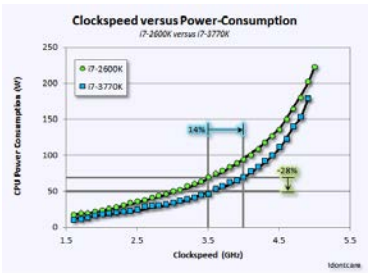


Figure 1 - Image downloaded from https://www.quora.com/Why-havent-CPU-clock-speeds-increased-in-the-last-5-years

Up until this point, chip manufacturers had held pretty close to "Moore's Law" and doubled the efficiency of their systems yearly. However, the rising energy needs and heat issues caused them to look into a new direction. Rather than placing one very large core in your system, they would place two smaller cores. Working independently of each other, they could handle more processes in the same clock cycle and still increase the efficiency of the system but would use less energy and produce less heat – and so the dual core was born. Today, almost all systems ship with at least a dual core processor, though many ship with a quad core which as the name suggests have four cores. In fact, most systems also utilize a special processor on your video card, the GPU, to improve video speed enabling those tasks to be offloaded freeing up even more clock cycles for your main processor.

The issue with parallel processing is that the computer has to know that multiple cores exist so that it can take advantage of them, and it, in most instances, it is not as simple as adding a single line of code to your program to make parallelism work. The programmer must plan for the software to act in parallel instead of sequentially.

DESCRIPTION / SPECIFICATIONS

In this program, we will be demonstrating the speedup that you can achieve by performing certain functions in parallel.

Your program should begin by reading in a range from the supplied **range.txt** file. This file will be formatted to have a lower bounds on the first line and an upper bounds on line 2. The program should include a menu that allows users to choose from two different methods of executing your program, using standard sequential summation and by using parallel summation. Once the user has selected a summation method, your program should begin a timer, using the supplied "timer.cpp" file below and call a function that will return your summation. Begin by finding all prime numbers that occur between the lower and upper bounds. As you loop through the number range, if the number is prime it should be added to an aggregator, otherwise it should be ignored. Once you have reached the upper limit, return the aggregator value to the main program and terminate the timer. Your program should output both the sum you calculated as well as the time necessary to accomplish the job.

PRIME NUMBERS

A number is considered prime if it is greater than 1 and has no factors between 2 and the square root of the number rounded up, inclusive. To accomplish this, your program should #include the cmath library to access the sqrt() function.

FUNCTIONS

Your header file (**functions.h**) should contain all necessary include files for your **functions.cpp** and **parallel_lab.cpp** source files. The header file should also contain all the necessary function prototypes.

ISPRIME FUNCTION

bool isPrime(int);
Given an integer n, returns true if n is prime and false if n is not prime. A number is considered prime if it is greater than 1 and has no factors between 2 and the square root of the number rounded up, inclusive. To accomplish this, your program should #include the cmath library to access the sqrt() function.

PARALLEL_NUM_PRIMES FUNCTION

int parallel_num_primes(int, int);
Given integers lower_bound and upper_bound, returns the number of primes in that range inclusive. Print message to user before calculation that tells the user that you are finding the number of primes between the lower & upper bound…and make sure to print out the actual numbers of the lower & upper bound. Calculation will be done in parallel.

NOTE: must place the code (in red) below BEFORE the `for` loop that goes through the upper & lower bounds.

```
#pragma omp parallel for reduction (+:counter)
```

SEQUENTIAL_NUM_PRIMES FUNCTION

int sequential_num_primes(int, int);
Given integers lower_bound and upper_bound, returns the number of primes in that range inclusive. Print message to user before calculation that tells the user that you are finding the number of primes between the lower & upper bound…and make sure to print out the actual numbers of the lower & upper bound. Calculation is done sequentially.

DISPLAYMENUGETCHOICE FUNCTION

int displayMenuGetChoice();
Displays a menu and returns the choice of the user. Validates input!

MAIN FUNCTION

1. Open input file called “range.txt” (only continue program if the file can be found).

2. Read in two integers….the first is the lower bound and the second is the upper bound.
3. Display a menu to the user with the following choices, get the user's choice & validate the user's choice. You will loop this program until the user chooses #3 (to quit).

```
Number of Primes Program
Choose from the options below
     1. Number of primes using a sequential function
     2. Number of primes using a parallel function
     3. Quit
Please enter your choice:
```

4. If the user chooses #1 then
    a. Start the timer (use **Timer.h**, **Timer.cpp** to help with this which is provided for you)
    b. Call the sequential_num_primes function and get the number of primes
    c. End the timer
    d. Get the total time
5. If the user chooses #2 then
    a. Start the timer
    b. Call the parallel_num_primes function and get the number of primes
    c. End the timer
    d. Get the total time
6. Display the number of primes & the total time like the sample output below:

```
Number of Primes: 664579
Total Time: 21
```

TIMER

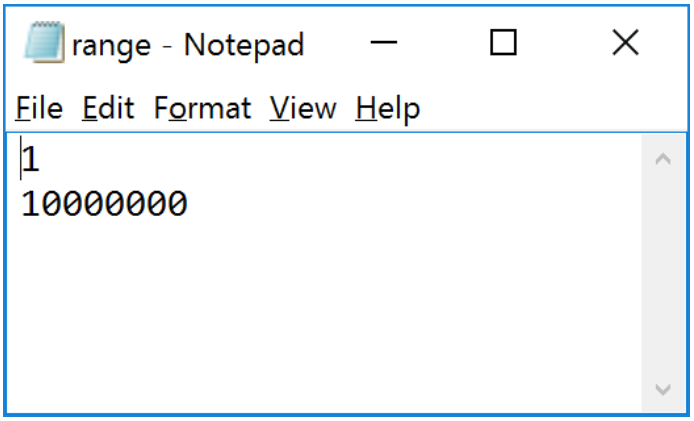
You have been provided with the library file "Timer.h" and the source file "Timer.cpp". Use these files in the creation of your program. You will need to **#include** this library in your parallel_lab.cpp. Timer.h contains functions for capturing the system time as well as for getting the total elapsed time in seconds between two time values.

COMPILING WITH OPENMP CODE

```
g++ -fopenmp –I ./ functions.cpp parallel_lab.cpp Timer.cpp –o
parallel_lab
```

SAMPLE OUTPUT

The sample output is on the next page. If you make your range in range.txt like mine below (1 to 10000000), then you should have similar run times to my sample output. If you do not, then it is highly possible that your isPrime function is not efficient.



```
Number of Primes Program
Choose from the options below
     1. Number of primes using a sequential function
     2. Number of primes using a parallel function
     3. Quit
Please enter your choice: 1


Finding the number of primes between 1 and 10000000 inclusive ...
Number of Primes: 664579
Total Time: 20


Number of Primes Program
Choose from the options below
     1. Number of primes using a sequential function
     2. Number of primes using a parallel function
     3. Quit
Please enter your choice: 2


Finding the number of primes between 1 and 10000000 inclusive ...
Number of Primes: 664579
Total Time: 10


Number of Primes Program
Choose from the options below
     1. Number of primes using a sequential function
     2. Number of primes using a parallel function
     3. Quit
Please enter your choice: 4
Please enter a valid choice: 3
```

HOW TO TURN IN

Upload **functions.h, functions.cpp, parallel_lab.cpp, range.txt, Timer.cpp**, and **Timer.h** to the ilearn dropbox named "EXTRA LAB".

HOW WILL I BE GRADED?

GRADE	DESCRIPTION
1	The lab is 100% complete and correct
0	No lab was submitted or it was incorrect.