Instructor: Dr. Tolga Soyata. *© U of R, Electrical and Computer Engineering,* *tolga.soyata@rochester.edu*

GPU Parallel Programming using C/C++
ECE206/406 CSC266/466

Project 1 (10pts) Due: Sep 22, 2015 11:59PM

- In this project, you will be developing a multi-threaded CPU program using pthreads (POSIX threads). No GPU will be used in this project. You will run your program in a computer that has an at least 4 core CPU. 4 core CPU means either a 4C/4T or a 4C/8T CPU.
- If you are not sure what kind of a CPU a computer has, find the INTEL code (e.g., i3-530) and go to Google and type "ARK i3 530" which will send you to INTEL's website to find the cores/threads.
- The object of this project is to develop a Prime Number Finder application.
- Late penalty : Normally, there is a 2 point per day penalty. I am waiving it for this project. But, get used to submitting on time. Late penalties will not be waived going forward, in Projects 2-5.

PART A . 2 points

- Write the purely serial version of **prime.c** which finds prime numbers in a large array of 8192 64-bit integer numbers. Submit this program on Blackboard in the "Project 1 Submission Area."
- You will design your code to run on a Unix system (Cygwin64 is fine on Windows machines). You will be designing your program to compile using the Makefile, identical to the projects shown in the lecture.
- Just add the Makefile entries to the one I have in the lecture notes. That's all you need.
- This program reads the file we created for you, **numbers.txt**, which has 8192 integer numbers in ASCII format. They are no more than 14 digits, so, they will fit in 64-bit integer format. Some of these numbers are prime numbers. Your program will determine which ones are prime.
-
- Your program will output a file called **primes.txt** which contains the (location, number) pairs in this format: **88 : 8191**. This means that, Line Number 88 (which we call Number 88) contains a prime number whose value is 8191. Your **primes.txt** will look like this:
- **50 : 31**
- **...**
- **88 : 8191**
- **...**
- In other words, each line in your output file will have one of these pairs we previously mentioned.
- Do not write the primes into the disk as they are found. Have a separate array called PrimeIndexes[] and Primes[] to record the line numbers (indexes) and the prime numbers.
- After this array has been completely written, and the program execution is done, write them onto the disk. That way, you are not incorrectly accounting for the disk IO time. You do not need to sort the output.
- Your program should be run like: **./prime [input.txt] [output.txt]**
- Time your program, to determine how fast it runs. Time only the execution portion. Do not time the IO read/write portion.

UNIVERSITY of ROCHESTER

PART B . 4 points

- Now that the serial version is working, you are ready to write the multi-threaded (MT) version.
- Call the MT version **primeP.c** ...Use a separate file for this program (just like imflip.c vs. imflipP.c) …
- Make the number of threads parametric (much like in imflipP.c), i.e.,

```
Int NumThreads=4;      // 4 is the default if no input is specified
```

- In **primeP.c**, each thread will be responsible for a portion of the numbers (i.e. ChunkSize). For example, in an 8-threaded version, each thread is responsible for determining 1024 numbers' primality.
- Threads will separately write their findings into the Primes[] and PrimeIndexes[] arrays, and at the very end, the results will be written to the disk.
- Time only the total execution time of the threaded version, not the disk IO.
- Your program should be run like: **./primeP [input.txt] [output.txt] [NumThreads]**
- **REPORT 1:** Plot the execution time in the y axis vs. the NumThreads in the x axis. Write a small one page report explaining the behavior. Vary NumThreads from 1 to 64, identical to **imflipP.c**.
- Upload your report to Blackboard as **proj1report1.pdf**.

PART C . 4 points

- Since each thread is responsible for a portion of the numbers (ChunkSize), let's play with this parameter.
- In a new program, **primePC.c**, define

```
int   ChunkSize=64;          // 64 elements per chunk
int   NumChunks=128;         // total number of chunks to process
```

-
- the product of the two should equal 8192, i.e., the total number of elements.
- Vary the chunk size and try to find the best split to determine which is better for multi-threading
- **REPORT 2:** Plot in 3D → the NumThreads (linear) in the x axis, ChunkSize (log) as y axis, and execution time as z axis. Write a small one page report explaining this behavior. Vary NumThreads from 1 to 64. Vary ChunkSize from 8 to 1024 (in 2x increments).
- Clearly, if you make the ChunkSize 1024, you can't go over 8 threads. So, that's where you stop for that ChunkSize. Your goal is to determine the best execution time point.
- Your program should be run like: **./primePC [input.txt] [output.txt] [NumThreads] [ChunkSize]**
- Upload your report to Blackboard as **proj1report2.pdf**.

SUBMISSION POLICY FOR ALL THREE PARTS:

- You need to include a Makefile along with your submission.
- Your program should be compiling and working without a problem with an entry in the Makefile. The entry will be to compile **prime.c, primeP.c**, and **primePC.c**, like the ones in the lecture examples.
- Do not have any exceptions such as "you had to comment out this and that to compile."
- If the T.A.'s cannot grade by a simple command line, like : **make prime**, you will lose all of the points from that PART. No exceptions. There is no way the T.A.'s can keep up with the exceptions. Students make their compilation as easy as the code that is given in the lectures.
- Emailing the code is NOT allowed. You must upload it on Blackboard under "Project1 Submission Area." If you pass the deadline, the Blackboard will report it. I am waiving the late penalties in this project, but, they will not be waived in the later projects. Get used to being on time please.
- These files need to be submitted: **prime.c, primeP.c, primePC.c, Makefile, proj1report1.pdf, proj1report2.pdf.**

UNIVERSITY of ROCHESTER