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 2 (10pts) Due: Oct 4, 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. You are welcome to use the Amazon Web Services (AWS) machines that will be the shown to you in Lab 3.
- Late penalty : Normally, there is a 2 point per day penalty. It will not be waived. Please upload your submission through Blackboard and it will be time-stamped by Blackboard.

PART A . 3 points

- Look at the imflipPM.c code. Focus on the horizontal flip using the "I" option (memory-friendly horizontal flip). In the book, the performance results are given in Table 3.4. While the local buffering using memcpy() helps the performance by about 1.6x average (as shown in Table 3.5), this code is ***not*** core-friendly due to the major number of the byte accesses in the MTFlipHM() function.
- Your goal in this part is to make this function core-friendly. Design a version called MTFlipHMC() and put this code in a new source file called imflipPMC.c.
- You will be submitting this file through Blackboard, along with all other necessary files to be able to compile your code. Do not assume that, we will be adding the ImageStuff.c and ImageStuff.h to your submission. You must include those files and make your submission completely standalone and compile-able.
- Hint : Each pixel is 3 bytes containing RGB. However, the CPU's native data size is 64 bits (i.e., 8 bytes).The common denominator is 24 bytes (i.e., 3 unsigned long int). Read 24 bytes at a time into unsigned long int (64-bit) variables, which will be a single memory read and will be super efficient. This contains 8 pixels' values. Then, you have to use shifts and MASKs AND, OR, whatever you need. Shifts and Masks are all core instructions that are executed inside the ALU and are super efficient.
- MASKing is the process of clearing certain bits. Example : If a is a 98 decimal (01100010 binary), when you AND a with a mask=11110000 , you get 01100000. The last 4 bits are wiped out. This is how you clean bits ...
- SHIFTing is good to change the positions of bits. For example, in the previous example, a=01100010 binary. When you shift a to the right by 4 bits (a>>4) , you get 00000110. The high 4 bits replaced and wiped out the low 4 bits.
- As you can see, when you have BGR, BGR, BGR , ... stored in 24 byte consecutive memory locations, thereby containing the values of 8 pixels, you need to do quite a few of these SHIFT, AND tricks to end up with the pixels in places you want. However, this will eliminate consecutive inefficient memory accesses from memory, and all of the SHIFT, AND operations are handled in the ALU, thereby balancing the memory/core ratio.
- Report the speed up from imflipPM.c to imflipPMC.c exactly the way it is done in the book using a table for 1, 2, 3, 4, 5, 6, 7, 8 threads.
- Your program should run like: **./imflipPMC infile.bmp outfile.bmp numThreads [version]**, where [version] may be 0 or 1 for MTFlipHM() or MTFlipHMC().

UNIVERSITY of ROCHESTER

PART B . 1 points

- I've been asked by many of you folks whether it would make any difference to reduce the number of memcpy() calls from 4 to 3 in CODE 3.2 in the book. I want you to try this and at least one other possible idea to make the MTFlipHM() memory-friendlier. Name your version MTFlipHM2().
- Return your updated code MTFlipHM2() inside the same imflipPMC.c program. MTFlipHM2() is the plain simple " I reduced the memcpy()'s from 4 to 3" version.
- Come up with another potential idea to make the "W" flipper even more memory friendly. For example, you can try increasing/decreasing the number of rows you are transferring with each memcpy(). Make this version MTFlipHM3(). This program should run with the same arguments as Part A, but [version] may now be 2 or 3 for MTFlipHM2() or MTFlipHM3().
- Report the speedup from the initial to the second to the third option in a table for 1, 2, 3, 4, 5, 6, 7 and 8 threads, exactly the way it is done in the book.

PART C . 3 points

- Make sure to understand the improvements in the Chapter 4 of the book by continuously improving the expensive core operations. But, none of these functions Rotate(), Rotate2(), ... Rotate7() make any attempt to be memory-friendly. Each version makes it even more core-friendly, but, memory accesses are still pretty hectic.
- Your goal is to write a memory friendly version of Rotate7() shown in Code 4.11. The code was made so core-friendly after all of the improvements that, the inefficiencies in memory accesses became dominant and are exposed in the runtime.
- Study how memory accesses can be improved. Submit your code as a new function Rotate8() which adds memory-friendliness to Rotate7(). Submitted your code in a new source file called imrotateMMC.c.
- We should be able to compile it exactly the way the original code works, except, we will be able to run Rotate8() with the "8" option in the command line. i.e. the syntax is: **./imrotateMMC infile.bmp outfile.bmp angle numThreads 8**

PART D . 3 points

- In this part, you will design, yet another version of the same function called Rotate9() and include it in imrotateMMC.c and we should be able to run it with the command line option "9" as the function.
- Rotate9() will incorporate asynchronous multi-threading to process the rotated rows. Define Mutex variables similar to the ones shown in Chapter 5 of the book and launch a number of threads defined by the command line. Each thread will check to see what the next row-to-process is and will process it.
- When threads are done with their work, they will join, exactly the way it is done in Chapter 5.

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

SUBMISSION POLICY FOR ALL FOUR 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.
- Although you are welcome to use ImageStuff.c and ImageStuff.h from the book, your submission should include every necessary file and we shouldn't have to add any other file, even the ones given in the class, such as ImageStuff.c and ImageStuff.h. Make a big zip file that includes everything and upload it to the Blackboard.
- 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 imflipPMC or make imrotateMMC, 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 "Project2 Submission Area." If you pass the deadline, the Blackboard will report it.
- There is a 2 point per day late submission penalty.
- These files need to be submitted: imflipPMC.c, imrotateMMC.c, proj2report.pdf along with the Makefile and any dependencies.

BONUS POINTS:

- The fastest imrotateMMC code gets 3 bonus points.
- Runner up (second fastest) gets 2 bonus points.
- The next 5 fastest students get 1 bonus point.