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

Assignment 5 (15 pts) Due : Nov 23, 2015 11:59PM

Assignment 5 Description

- **You can develop your code on a CIRC node or your own GPU. It must be written in CUDA and developed on a Fermi, Kepler, or Maxwell engine (GF, GK, or GM, respectively). It will be graded on BlueHive, though, so you should test it there before handing it in.**
- **If you do not allocate/de-allocate memory properly, points will be taken off your program. So, your malloc() and cudaMalloc()'s must be used properly in your code.**
- **Each late day will subtract 2 points from your assignment points**

- In this project, you will use the imedge.c program that was in Chapter 5 of the book.
- You will take the Astronaut.bmp picture, and create an edge-detected version of it (Fig. 5.3).
- You will try to run this program with 16 different threshold values and save these images under 16 different names, and make an animated GIF out of them.
- The big deal in this program is to use the CPU and GPU together to achieve the fastest runtime from start to finish (of 16 different edge-detected images). Use threshold values 0 through 128 in 16 increments, which will create an animation that has more edges showing up as the threshold decreases.

PART A (5 points):

- Design a pure-GPU version of the program without optimizations on the data transfer.
  - Report the run times of transferring the data entirely to the GPU $T_{H \rightarrow D}$
  - Report the execution time of running the kernels in the GPU T_{Exec}
  - Report the run times of transferring the data entirely back from the GPU $T_{D \rightarrow H}$
- Based on the ideas you get from execution times, now, you are ready to optimize it.
- The program is launched with the following command line parameters:
  - **./imedgeG Infile Outfile**
  - For example: **./imedgeG Astronaut.bmp AEDGE.bmp** will read the Astronaut.bmp file, and write 16 different output files named AEDGE1.bmp, AEDGE2.bmp, ... AEDGE16.bmp.
  - You can then look at the animation of it using ImageMagick as you saw in Proj4.
- Submit your code as **imedgeG.cu** through Blackboard, along with everything needed to compile it (ImageStuff etc.). Call it **Proj5PartA.zip**.

PART B (5 points):

- Now, design a streamed version of it.
- Create a pinned memory to store the image, transfer it, and execute it in small chunks.
- Vary the chunk size to optimize the performance. You should optimize it on BlueHive, and report if the best chunk size was different on your personal computer (if you ran it there too).
  - Since this should almost eliminate the transfer times (i.e., coalesce), report only the start-to-finish time.
  - Compare this to the sum of $\mathbf{T_{H\rightarrow D} + T_{Exec} + T_{D\rightarrow H}}$ in Part A.
- The program is launched with the following command line parameters:
  - **./imedgeGStr Infile Outfile**
  - It does exactly the same as before, except, it is streamed inside.
- Submit your code as **imedgeGStr.cu** through Blackboard, along with everything needed to compile it (ImageStuff etc.). Call it **Proj5PartB.zip**.

PART C (5 points):

- So far, you have been using the CPU as a mere data-shuttle. Now it is time to make it do useful work. In this part, you will use multiple CPU threads to do useful work.
- What is this useful work ? This is your decision to make. You will definitely launch multiple CPU threads to do the work.
- Ideas are: one thread could be solely responsible for sending the data to/from the GPU. The same thread could also be responsible to do this in a streamed fashion, or should a different thread be responsible for incoming vs. outgoing transfers ?
- Multiple other threads could be doing the thresholding (Chapter 5 of the book) at 16 different values.
- Which devices (CPU/GPU) should be involved in each stage (Gaussian, thresholding, etc.)? For example, should the GPU handle 90% of the Gaussian stage while the CPU does the other 10%? Or should you let the GPU handle the whole thing?
- Try at least 3 different task-breakdown scenarios like this. Your goal is to do whatever you have to do, and beat the runtime result in PART B.
- The program is launched with the following command line parameters:
  - **./imedgeGStrC Infile Outfile**
  - It does exactly the same as before, except it should take advantage of CPU threads as well.
- Submit your code as **imedgeGStrC.cu** through Blackboard, along with everything needed to compile it (ImageStuff etc.). Call it **Proj5PartC.zip**.
-

You are required to turn in a report that has screen shots and ideas you tried with their results.
At the end, make a table that lists different ideas, and the runtimes (with the transfer vs. GPU execution vs. CPU execution time breakdown).

Report all of your results in an organized fashion in a file named **Proj5Report.pdf**. Upload this file along with your other three files.