ECEC 413: Introduction to Parallel Computer Architecture
CUDA Lab 4: Blocked 2D Convolution

Prof. Naga Kandasamy, ECE Department, Drexel University

March 1, 2010

The Lab is due March 19, 2010. You may work on the assignment in teams of up to two people.

Important: Before starting the lab assignment, please read the tutorial on getting the CUDA SDK working on the cluster. You will find the tutorial (written by Tony Shackleford) on webCT under the "Labs" folder.

Your lab assignment is as follows:

- Edit the source files `2Dconvolution.cu` and `2Dconvolution_kernel.cu` to complete the functionality of the matrix convolution on the GPU. Matrix convolution is primarily used in image processing for tasks such as image enhancing, blurring, etc. A standard image convolution formula for a 5 convolution kernel A with matrix B is

$$C(i,j) = \sum_{m=0}^{4}\sum_{n=0}^{4} A[m][n] * B[i+m-2][j+n-2]$$

  where $0 \le i <$ `B.height` and $0 \le j <$ `B.width`. Note that elements at the boundaries, that is, elements that are "outside" the matrix B, for this exercise, are treated as if they had value zero.

- This assignment will assume a constant 5×5 convolution kernel, but will have "images" or matrices of an arbitrary size. Your program should accept no arguments. The application will create a randomized kernel and image. A CPU implementation of the convolution algorithm will be used to generate a correct solution which will be compared with your GPU program's output. If the solutions match within a certain tolerance, if will print out "Test PASSED" to the screen before exiting.

- You must e-mail me all of the files needed to compile and run your code as a single zip file.

Also, once you have completed the assignment, answer the following questions in a separate lab report. Included within the source-code folder is a folder called "test", which contains two test case input sets. Using these test cases, provide answers to the following questions, along with a short description of how you arrived at those answers.

- What is the measured floating-point computation rate for the CPU and GPU kernels on this application? How do they each scale with the size of the input?

- How much time is spent as an overhead cost of using the GPU for computation? Consider all code executed within your host function, with the exception of the kernel itself, as overhead. How does the overhead scale with the size of the input? You are free to use any timing library you like, as long as it has a reasonable accuracy. Note that the CUDA utility library provides some timing functions which are modeled in the given test harness code, it you care to use those. Remember that kernel invocations are normally asynchronous, so if you want accurate timing of the kernel's running time, you need to insert a call to `cudaThreadSynchronize()` after the kernel invocation.

Your assignment will be graded on the following parameters:

- Correctness: 25%.
- Functionality: 40%, correct handling of boundary conditions, using shared, constant or texture memory to cover global memory latency.
- Report: 35%. Answer to question 1: 10%, answer to question 2: 25%