Blocked 2D Convolution
(Download ZipFile)

This MP is a blocked implementation of a matrix convolution. This assignment will have a constant 5x5 convolution kernel, but will have arbitrarily sizes "images".

Matrix convolution is primarily used in image processing for tasks such as image enhancing, blurring, etc. A standard image convolution formula for a 5x5 convolution kernel A with matrix B is

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

where $0 <= i < B.height$ and $0 <= j < B.width$

Elements that are "outside" the matrix B, for this exercise, are treated as if they had value zero.

1) Unzip the lab template into your SDK projects directory.

2) Edit the source files 2Dconvolution.cu and 2Dconvolution_kernel.cu to complete the functionality of the matrix convolution on the device.

3) The modes of operation for the application are described here.

   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 programs output. If it matches (within a certain tolerance), if will print out "Test PASSED" to the screen before exiting.

   One argument: The application will use the random initialization to create the input matrices, and write the device-computed output to the file specified by the argument.

   Three arguments: The application will read input matrices from provided files. The first argument should be a file containing two integers. The first and second integers will be used as N.height and N.width, respectively. The second and third function arguments will be expected to be files which have exactly enough entries to fill matrices M and N respectively. No output is written to file.

   Four arguments: The application will read its inputs using the files provided

by the first three arguments, and write its output to the file provided in the fourth.

Note that if you wish to use the output of one run of the application as an input, you must delete the first line in the output file, which displays the accuracy of the values within the file. The value is not relevant for this application.

3) Report.

It's time to do some performance testing and analysis. Included in the MP3-convolution_block folder is a folder called "test", which contains two test case input sets. Using these test cases, and any others that you wish to create to support your findings, provide answers to the following questions, along with a short description of how you arrived at those answers.

1. 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?
2. 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.

Grading:

Your submission will be graded on the following parameters.

Demo/knowledge: 25%
- Computation runs on a G80, producing correct outputs files.

Functionality: 40%
- Correct handling of boundary conditions
- Uses shared, constant or texture memory to cover global memory latency.

Report: 35%