NCIT Summer School 2012: GPU Computing Workshop
University Politehnica of Bucharest
6 June – 8 June 2012

Assignment 2: 2D Convolution

In this problem, you have to implement a blocked matrix convolution. You will work with a constant 5x5 so called "convolution kernel matrix" (the **A** matrix), but will have arbitrarily sized "images" (the **B** matrix). 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

$$\mathbf{C}[i][j] = \sum_{m=0}^{4}\sum_{n=0}^{4}\mathbf{A}[m][n]\cdot\mathbf{B}[i+m-2][j+n-2]$$

where $0 \leq i < \mathbf{B}.\mathrm{height}$ and $0 \leq j < \mathbf{B}.\mathrm{width}$. Elements that are "outside" the matrix B, for this exercise, are treated as if they had value zero.

For this problem:

1) Edit the source file **`2Dconvolution.cu`** to complete the functionality of the matrix convolution on the device.

2) The modes of operation for the application are described as follows:
a) 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), it will print out "**`Test PASSED`**" to the screen before exiting.
b) 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.
c) 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. Note that for the purpose of this assignment, always **`N.height=N.width=5`**.
d) 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.

Dan Negrut, SBEL, University of Wisconsin - Madison

Report.
Included in the distribution folder is a subfolder called "test", which contains two test case input sets. Using these test cases, try to minimize the running time of the **ConvolutionKernel()** kernel. 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 or make use of **cudaEvents**, as discussed on Wednesday.

If time allows, also perform a scaling analysis using 2^4 to 2^{12} elements, comparing the running time of your GPU kernel with the provided CPU implementation (function **ComputeGold()**). Provide a plot of this analysis in your email.

All files should then be packaged (zip, tar, rar, etc) and emailed to:

ncit-hw@andrewseidl.com

Remember: the goal is for you to generate your own creative GPU implementation, not to copy code provided by outside sources.

If you run into issues compiling, try the following:

- add the 'common' include directory from the SDK:
  - -I$NVSDKCOMPUTE_ROOT/C/common/inc
- add the 'sdk' library directory:
  - **-L$NVSDKCOMPUTE_ROOT/C/lib**
- link against libcutil_x86_64:
  - -lcutil_x86_64