CS4345 General Purpose Computation on GPU (2016/2017 Semester 1)

Programming Assignment 3

Release Date: 13 October 2016, Thursday
Submission Deadline: 30 October 2016, Sunday, 11:59 PM

LEARNING OBJECTIVES

Writing CUDA programs using CUDA C to perform general-purpose computation on the GPU. After completing the programming assignment, you should have

- learned how to write simple CUDA C programs,
- learned how to build CUDA C programs in Microsoft Visual Studio,
- learned how to divide your solution into parallel threads,
- learned how to optimize CUDA kernels by using shared memory, by reducing uncoalesced global memory accesses, and by reducing shared memory conflicts.

TASKS

There are **two tasks** in this programming assignment. Both tasks require you to write a CUDA kernel function.

Please download the ZIP file **cs4345_assign3_2016_todo.zip** from the **Assignment** folder in the IVLE Workbin. Your task is to complete the source file **convolution.cu** by implementing the CUDA kernel functions `GPU_Convolve1()` for **Task 1** and `GPU_Convolve2()` for **Task 2** according to the following requirements.

Each kernel function is to compute the same result. Given two 1-D arrays of floating-point numbers, where one is called the **data** and the other the **filter**, each kernel is to compute the **discrete convolution** of the two. A CPU version of the function, `CPU_Convolve()`, is given and please refer to it for the exact specification of the discrete convolution operation. Note that this CPU function **works correctly for any positive number of elements in the data array, and any positive odd number of elements in the filter array**. The output array always has the same size as the input data array.

The file **cs4345_assign3_2016_todo.zip** contains Visual Studio 2010 Solution and Project files (can be loaded and converted by VS2012 and later). To build the project, you need to have **CUDA Toolkit 6.5** and the corresponding **CUDA Samples 6.5** already installed on your computer, and the environment variables `CUDA_PATH` and `NVCUDASAMPLES_ROOT` set to the correct directories.

An executable of the sample completed program, **convolution_done.exe**, is also provided in the ZIP file. You are welcome to run it to get a sense of the speed-up you can achieve by using your GPU to perform the computations.

TASK 1: Without Using Shared Memory

In this task, you are to complete the CUDA kernel function **`GPU_Convolve1()`**. In this version, each thread is responsible for computing the result of one element in the output array. Each thread should read the elements of the data and filter arrays directly from the device global memory without staging them in the shared memory. You are free to add new helper functions to the program.

NOTE: For Task 1, you must make sure that your CUDA kernel works correctly for any positive number of elements in the data array, and any positive odd number of elements in the filter array. You cannot assume that the number of elements of the data array is a multiple of the thread block size.

TASK 2: Using Shared Memory

In this task, you are to complete the CUDA kernel function **`GPU_Convolve2()`**. Again, in this version, each thread is responsible for computing the result of one element in the output array. Threads within a block cooperate with each other by sharing data through the shared memory.

Each block of threads together read the entire filter array from the global memory to the shared memory. They also collaborate to transfer to the shared memory the part of the data array that will be used by some threads in the block. To make the programming simpler, you can let the threads in the block transfer three consecutive full "tiles" of the data array to the shared memory, where the number of elements of a tile equals the thread block size. Of the three tiles, the middle tile's position in the data array corresponds to the position of the tile to be output by the thread block to the output array. You should be careful about the boundary conditions—for some thread blocks, only two full tiles can be transferred to the shared memory.

You should also avoid **uncoalesced global memory accesses** and **shared memory conflicts** as much as possible. If this is implemented correctly, typically, it would be a few times faster than the version without using shared memory. You are free to add new helper functions to the program. **Your above optimizations should target CUDA hardware with Compute Capability 1.0 and 1.1.**

NOTE: For Task 2, the filter width can be any positive odd integer smaller than the thread block size, and the number of elements of the data array is always a multiple of the thread block size. The thread block size to be used is always a multiple of 32.

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **8%** of your total marks for CS4345.

Program **correctness** constitutes **90 marks** while **design/style** constitutes **10 marks**. Note that if your program cannot be compiled and linked, you get 0 (zero) mark for program correctness.

Good coding style. Comment your code adequately, use meaningful names for functions and variables, and indent your code properly. You must fill in your **name**, **matriculation number**, and **NUS email address** in the **header comment**.

Correctness. Marks are allocated as follows to the two tasks:

- **Task 1 — 30 marks**
- **Task 2 — 60 marks**

SUBMISSION

For this assignment, you need to **submit only** the completed source file **convolution.cu**.

You must put it in a ZIP file and name your ZIP file **<*matric_no.*>.zip**. For example, **A0123456X.zip**. All letters in your matric. number must be capitalized.

Submit your ZIP file to the **Assignment 3 Submission** folder in the IVLE Workbin. Before the submission deadline, you may upload your ZIP file as many times as you want to the correct folder. **We will take only your latest submission.** Once you have uploaded a new version to the folder, you **must delete the old versions**. Note that when your file is uploaded to the Workbin folder, the filename may be automatically appended with a number. This is fine, and there is no need to worry about it.

DEADLINE

Late submissions will NOT be accepted. The submission folder in the IVLE Workbin will automatically close at the deadline.

——— End of Document ———