EC527: High Performance Programming with Multicore and GPUs
Programming Assignment 7

Objectives
Set up and test a GPU environment. Write and test a simple GPU program.

Prerequisites (covered in class or through examples in on-line documentation)

HW – GPU basics: accelerators, accelerator memory, SIMT support
SW – SIMT model, basic CUDA model
Programming – CUDA environment, CUDA/GPU set up and data transfer, simple GPU kernels

---

Assignment

Part 1:
Follow the directions in **CUDA-and-SCC-for-EC527.pdf** to set up your environment. Read `cuda_test.cu` and make sure you understand it. Compile and run the program.

Part 2: A simplified SOR
2a. Create a simplified SOR based on the sample code given in class (for the 1D averaging filter). It should do the following:

• On the host: Create and initialize a 2K×2K array of single precision floats. (Is "2k" 2000 or 2048? Either is okay)

• Transfer the array to the GPU
• Create a single block of 16x16 threads (extend the 1D example from the slides to 2D)
• Let each thread operate on a square patch of the input array (rather than a single output element). How big is each patch?

• Run 2000 iterations of SOR – the boundary should not change. You do not need to test for convergence.

• Transfer the output back to the CPU

2b. Write function(s) to perform the same calculation directly (i.e. on the CPU). Compare your CPU and GPU answers. Are they different? How?

2c. Time both codes (GPU and CPU). How long do they take?

2d. (*OPTIONAL*) In the work you did so far, the 256 threads (16x16) are each responsible for multiple elements. There are different ways of deciding what work is done by each thread — possibities include: 1D strip per thread, 2D tile per thread, interleaved in 1D (every 256th element), interleaved in 2D (every 16th element of every 16th row). Which one did you already do in part 2a? Try at least two others.

Part 3: Multiple blocks
Make the following changes to your GPU code. Check your answer and your timing.

3a. Let each thread operate on a single output array element (like the code in class)
3b. Let the block size be 16x16. Choose the grid dimensions accordingly.
3c. Modify the host code (kernel calls) so that the program executes correctly. That is, the iterations should now be controlled by the host not the GPU (why?).
3d. Check correctness and timing.
3e. (*OPTIONAL*) Try using thread control as in Part 2. For this to be interesting should also try different mappings of array elements to threads.

NOTE: Because of the difference in precision and various floating point technical issues, you will probably see some drift between the GPU version and the reference version even with correct code. You should be able to handle this by increasing the epsilon to, say, 5%.

NOTE: In past years many students have found **cuPrintf** useful; the relevant files are "`cuPrintf.cu`" and "`cuPrintf.cuh`", and are included with this assignment. See "Using cuPrintf.pdf" for instructions.