EE-193: Parallel Computing
Lab 4: Matrix multiplication with CUDA

In this lab, you will code a matrix multiplication using C++ and CUDA. Start with the file matrix_mpy.cu, which contains most of the support code, but leaves the interesting part to you. It reuses most of the routines from our C++-threads matrix multiply.

In particular, you have to write the host function Matrix::mpy1() and also write the CUDA kernel function that it calls.

In this assignment, we're worried less about getting the best performance and more about just getting everything to work. As we discussed in class, you should try to implement an algorithm that does not require a critical section. This requires that, for any single entry in the result matrix, only one thread writes that entry.

Collect timing data for matrices of size 1Kx1K, 2Kx2K and 4Kx4K (i.e., for LOG2_N of 10, 11 and 12). You should stick with a block size of 16x16. Note that computing the reference solution on the host will take far longer than the GPU version; it may take roughly 1 minute for 2Kx2K and over 5 minutes for 4Kx4K.

If you're ambitious, you can also break down the timing data to the next level: how much time is spent in getting data from host→GPU, in doing computation on the GPU, and in copying data back to the host.

Provide a short report discussing your timing results. You should compare them not only to each other, but also to the C++ threads code from HW #3.

Debugging on a GPU can be done in many ways Perhaps the simplest is to use *printf*() statements. CUDA does support *printf*(). Note that printing is done from the CPU rather than the GPU; thus, *printf*() actually buffers its input and returns it to the CPU for printing. CUDA does not support C++ printing with **cout**; however, you can always use an ostrstream and then call *printf*() on its *str*() function if you like.

Logistics:

- The lab assignment is due ????? at midnight. You may work on this assignment in pairs.
- Submit your project via **provide**. You should submit a copy of your report as a PDF, as well as your version of histogram.cxx.

Logistics for GPUs

- First, you should tell the shell where nvcc is kept:
    - setenv PATH /usr/local/cuda-8.0/bin:${PATH}
    - setenv LD_LIBRARY_PATH /usr/local/cuda-8.0/lib64:${LD_LIBRARY_PATH}
- You can compile with the line **nvcc -std=c++11 -O2 -arch=compute_50 -I /usr/local/cuda/samples/common/inc matrix_mpy.cu ee193_utils.cxx**
- **nvcc** is the standard C++ compilation script for CUDA.
- **-std=c++11** does the same as it does for GCC.
- **-O2** ensures that the host code gets optimized (and the reference dumb matrix-mpy code lives on the host)

- **-arch=compute_50** tells **nvcc** to target its code at a particular "compute capability," which is how Nvidia differentiates newer GPUs from older ones. In particular, using *printf*() will not work unless you do this.

Resources:

- As usual, use any Linux box in labs 116 or 118. Before you collect benchmarking data, you should reboot the machine. All of the Linux-lab machines have GPUs and are enabled for CUDA.