CS 677: Homework Assignment 3
Due: March 1, 6:00pm

Philippos Mordohai, Department of Computer Science
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment. I will assume that, as participants in a graduate course, you will be taking the responsibility of making sure that you personally understand the solution to any work arising from collaboration.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prohibit you from submitting a homework assignment in time, please e-mail me explaining the situation.

Submission Format. Electronic submission on Canvas is mandatory for this class. **Please submit a zip file containing a pdf file and your code.**

Problem 1. (100 points) For this problem, follow the steps below.

1. Download cs677_hw3.zip from `http://www.cs.stevens.edu/~mordohai/classes/cs677_s17/cs677s17_hw3.zip` and extract its contents into your SDK projects directory.

2. Edit the source files matrixmul.cu and matrixmul_kernel.cu to complete the functionality of the matrix multiplication on the device. The two matrices could be any size, but the resulting matrix should have less than 64,000 elements. (Note that this is not verified in the code.)

3. There are several modes of operation for the application.

    - No arguments: The application will create two randomly sized and initialized matrices such that the matrix operation M $\times$ N is valid, and P is properly sized to hold the result. After the device multiplication is invoked, it will compute the correct solution matrix using the CPU, and compare that solution with the device-computed solution. 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 three integers. The first, second and third integers will be used as M.height, M.width, and N.height. 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 from the files provided by the first three arguments as described above, 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.

In all cases, the program will print out the elements of P on which the CPU and GPU computations are inconsistent, and whether or not the comparison passed.

4. Use `nvcc -Xptxas -v matrixmul_kernel.cu` to see the resource usage of your kernel (although compilation will fail, it will only do so after compiling the kernel and displaying the relevant information. "smem" stands for shared memory and "cmem" stands for constant memory.) Then, answer the following questions for your implementation:

   (a) How much shared memory is used per block? What is the maximum number of blocks per SM, if the shared memory was the bottleneck?

   (b) How many registers are used per block? What is the maximum number of blocks per SM, if the number of registers was the bottleneck?

   (c) What is the maximum number of blocks per SM, if the number of thread contexts was the bottleneck?

   (d) For your kernel implementation and GPU, how many threads can be simultaneously executing?

Note: The output of running `nvcc -Xptxas -v matrixmul_kernel.cu` should look something like the following:

```
Used 4 registers, 60+56 bytes lmem, 44+40 bytes smem, 20 bytes
cmem[1], 12 bytes cmem[14]
```

As shown in the above example, the amounts of local and shared memory are listed using two numbers for each. The first number represents the total size of all variables declared in local or shared memory, respectively. The second number represents the amount of system-allocated data in these memory segments: device function parameter block (in shared memory) and thread/grid index information (in local memory). Used constant memory is partitioned in constant program "variables" (bank 1), plus compiler generated constants (bank 14).