Version: 9/16/2015

ME759
High Performance Computing for Engineering Applications
University of Wisconsin-Madison
Assignment 3

1. Write a CUDA application that does the following:
   - It allocates on the device memory to hold an array of 16 integers.
   - It launches a kernel on two blocks, each block having eight threads. The kernel is set up such that each thread computes the sum of its thread index and block index, and then saves the result (an integer) into the array for which memory was allocated on the device. Subsequently, the kernel uses a `printf` statement to print out what it saved into the device array.
   - The host copies back the data stored in the device array into a host array of size 16.
   - The host uses a loop to print to screen the 16 values stored in the host array.

NOTE: pass as an argument to the kernel the pointer to device memory where you are to save the information generated by each thread.

2. This exercise will provide an opportunity to understand how a problem can be solved using different execution configurations; i.e., number of blocks and threads in an execution grid. To this end, assume that you start with two arrays of random numbers (double precision) between -10 and 10. You are supposed to add these two arrays of length N on the GPU and eventually get the result into a host array of length N. You will have to run a loop to investigate what happens when the value of N increases by factors of 2 from $N = 2^{10}$ to $N = 2^{20}$. Specifically, report in a plot the amount of time it took for the program to add the two arrays as a function of the array size. Plot the inclusive time using a blue line. Plot the exclusive time using a red line. Here's how this would work:
   - Allocate space on the host for arrays `hA`, `hB`, and `hC`, `refC`, and then populate `hA` and `hB` with random numbers between -10 and 10. Each of these arrays is of size N.
   - Store the result of `hA`+`hB` into `refC`
   - Allocate space on the device for `dA`, `dB`, and `dC`
   - [For inclusive timing, start the timing now]
   - Copy content of `hA` and `hB` into `dA` and `dB`, respectively.
   - [For exclusive timing, only start the timing now]
   - Invoke kernel that sums up the two arrays
   - [For exclusive timing, stop the timing now]

- Copy the content of `dC` back into `hC`
- [For inclusive timing, stop the timing now]
- Report the amount of time required to complete the job
- Confirm that the numbers in `refC` and `hC` are identical within 1E-12

Here's an important point. You'll have to do the sequence of steps above twice (plot all the result on the same plot though). The first time, the number of threads in a block is going to be 32. As the value of N increases, you'll have to adjust the number of blocks you launch to get the job done. The second time around, you are going to use 1024 threads in a block (this is currently the largest number of threads that you can have in a block). Again, as the value of N increases, you'll have to adjust the number of blocks you launch to get the job done.
NOTE: Use a significantly thicker line to plot the 32-thread case.

FURTHER REMARKS ON YOUR HOMEWORK:
- Use CMake for both problems.