CS 677: Homework Assignment 1
Due: February 8, 6:15pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
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 prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

Submission Format. Electronic submission on Canvas is mandatory.

Problem 1. (50 points) Download vecAdd.cu from `https://github.com/olcf/vector_addition_tutorials/blob/master/CUDA/vecAdd.cu`. Compile and execute the code. This can be done using:

```
nvcc -o vecAdd  vecAdd.cu
```

or the makefile on the repository above.

Now, modify the code to perform addition of two 1024×1024 matrices generated in a similar way as the vectors a and b in the example. Show the top-left 5×5 submatrices of the input and output. **Submit the source code for this part without changing the filename.**

Write three kernel functions:

(a) One that has each thread producing one element of the output matrix.

(b) One that has each thread producing one row of the output matrix.

(c) One that has each thread producing one column of the output matrix.

There is no need to submit printouts for all three kernels. They should produce the same answer.

Problem 2. Kirk & Hwu problem 3.6. (10 points) We want to use each thread to calculate two (adjacent) elements of a vector addition. Assume that variable i should be the index for the first element to be processed by a thread. What should be the expression for mapping the thread/block indexes to the index of the first of the two elements? Why?

(a) `i = blockIdx.x*blockDim.x+threadIdx.x+2;`

(b) `i = blockIdx.x*threadIdx.x*2;`

(c) `i = (blockIdx.x*blockDim.x+threadIdx.x)*2;`

(d) `i = blockIdx.x*blockDim.x*2+threadIdx.x;`

Problem 3. (10 points) For a vector addition, assume that the vector length is 2000, each thread calculates one output element, the thread block size is 512 threads and the entire computation is carried out by one grid. How many threads will be in the grid? How many warps do you expect to have divergence due to the boundary check on the vector length?

Problem 4. Kirk & Hwu problem 5.1. (10 points) Consider matrix addition where each element of the output matrix is the sum of the corresponding elements of the two input matrices. Can one use shared memory to reduce the global memory bandwidth consumption? Hint: analyze the elements accessed by each thread and see if there is any commonality between threads.

Problem 5. Kirk & Hwu problem 5.3. (10 points) What type of incorrect execution behavior can happen if one forgets to use the `__syncthreads()` function in the kernel of slide 59, Lecture 2? Note that there are two calls to `__syncthreads()` each for a different purpose.

Problem 6. Kirk & Hwu problem 5.4. (10 points) Assuming that capacity were not an issue for registers or shared memory, give one case that it would be valuable to use shared memory instead of registers to hold values fetched from global memory. Explain your answer.