This is a makeup assignment. I will take the best 4 of your assignments (A1–A5) as your score for the assignments contribution of your final grade. I will not substitute this makeup assignment for any penalties I may have imposed.

All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates (unless otherwise instructed), but you may not look at their code. If in doubt, ask the instructor.

Acknowledge all sources you found useful.

Your code should compute the correct results.

Partial credit is available, so attempt all exercises.

Submit your answers as a PDF file.

The compressed archive (e.g. ZIP) file you upload to Blackboard should have your name in the filename, e.g. `JRandomStudentA5.zip`

Exercise 1

(1D work distribution) Convert the supplied `vector_add.cpp` to a CUDA program. Modify the filename and the makefile as necessary. Your aim is to create a CUDA version of the `do_add` function.

Do NOT modify `vector_util.h`.

1. Write a CUDA kernel for `do_add` where consecutive elements of each array are assigned to consecutive threads. Call the resulting binary `vadd_coal`.
2. Write a CUDA kernel for `do_add` where the arrays are viewed as containing non-overlapping blocks of 32 elements (the last block can contain fewer elements). Each thread then adds the elements of each block. Call the resulting binary `vadd_block`.
3. Write a CUDA kernel for `do_add` where elements of the arrays are distributed to the threads in a way different from Q1 and Q2 above. Describe your scheme. Your scheme must not duplicate work. Call this `vadd_custom`.

Exercise 2

(Matrix flip) Write a program to flip a dense matrix (i.e. 2D array) in CUDA. I.e. location (x, y) in the input matrix will be at location $(n - x - 1, m - y - 1)$ in the output matrix, where n is the width of the matrix and m is the height of the matrix.

Your program will be invoked as: `matrix_flip input.txt output.txt`.

The file `input.txt` will be in the following format:

```
3 2
5 6
7 8
9 0
```

Here the first line contains the size of the matrix `m n`, where m is the number of rows, and n is the number of columns.

Your output (written to the supplied `output.txt`) must be in the same format:

```
3 2
0 9
8 7
6 5
```

Note that flipping a matrix twice is an identity transformation.

In both the questions below, make sure you can handle any matrix size upto 1024x1024 (upto 32-bits per element, i.e. integer).

Make sure your work distribution to the threads does not result in duplicate work.

1. Write a program `matrix_flip_1d` which uses a 1D launch configuration to achieve the matrix flip.
2. Write a program `matrix_flip_2d` which uses a 2D launch configuration (i.e. uses both `threadIdx.x` and `threadIdx.y`)

Exercise 3

(Matrix flip using textures) Modify `matrix_flip_2d` from Exercise 2 above to use a 2D *CUDA texture objects* for the input matrix. Call this `matrix_flip_2d_tex`. Compare the performance to `matrix_flip_2d`.

Exercise 4

(Parameter passing) Study the pieces of code given below and answer the following questions. You can implement them to figure out the answers.

You may not discuss these questions with anybody other than the instructors.

1. Given the following CUDA function:

   ```
   __global__ void test(int n) {
       tid = threadIdx.x + blockDim.x * blockIdx.x;
       n = tid;
       assert(n == tid);
       printf("%d: %d\n", tid, n);
   }
   ```

   Will the assertion ever be false (for any number of threads)? Do threads share one copy of `n` or do they each have an individual copy?

2. Consider the code below:

   ```
   __global__ void test2(int n) {
       tid = threadIdx.x + blockDim.x * blockIdx.x;
       n = tid;
   }
   ```

   And on the CPU:

   ```
   int n = 100;
   test2<<<1, 1>>>(n);
   cudaDeviceSynchronize();
   printf("%d\n", n);
   ```

   What will be the output of the `printf` statement?

END.