A2 – Thrust Library, Image Processing – BLUR

Deadline: 9AM, August 6, 2018.
These assignment questions are courtesy the GPU Accelerated Computing kit by UIUC and NVIDIA. Dataset generators and the template CUDA code may have errors. The image processing programs in this assignment use image read/write code from libwb (https://github.com/abduld/libwb).

Q1. Implement vector addition using Thrust. Thrust is a Standard Template Library for CUDA that contains a Collection of data parallel primitives (eg. vectors) and implementations (eg. Sort, Scan, saxpy) that can be used in writing high performance CUDA code. Checkout all the libraries at: http://developer.nvidia.com/technologies/libraries. Refs for Thrust: http://docs.nvidia.com/cuda/thrust/index.html, http://www.mariomulansky.de/data/uploads/cuda_thrust.pdf, https://www.bu.edu/pasi/files/2011/07/Lecture6.pdf. Edit the template code to perform the following:

1. Generate a `thrust::dev_ptr<float>` for host input arrays
2. Copy host memory to device
3. Invoke thrust::transform()
4. Copy results from device to host

Instructions about where to place each part of the code is demarcated by the //@@ comment lines. The executable generated as a result of compiling the lab can be run using the following command:

```
./ThrustVectorAdd_Template -e <expected.raw> -i <input0.raw>,<input1.raw> -o <output.raw>
-t vector
```

where `<expected.raw>` is the expected output, `<input0.raw>`,`<input1.raw>` is the input dataset, and `<output.raw>` is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process. Answer the following questions.

1. How many floating operations are being performed in your vector add kernel?
2. How many global memory reads are being performed by your kernel?
3. How many global memory writes are being performed by your kernel?
4. In what ways did Thrust make developing a functional vector addition code easier or harder?

Q2. Implement an efficient image blurring algorithm for an input image. An image is represented as `RGB float` values. You will operate directly on the RGB float values and use a 3x3 Box Filter to blur the original image to produce the blurred image (Gaussian Blur). Edit the code in the template to perform the following:

1. allocate device memory
2. copy host memory to device
3. initialize thread block and kernel grid dimensions
4. invoke CUDA kernel
5. copy results from device to host
6. deallocate device memory

The executable generated as a result of compiling the lab can be run using the following command:

```
./ImageBlur_Template -e <expected.ppm> -i <input.ppm> -o <output.ppm> -t image
```

where <expected.ppm> is the expected output, <input.ppm> is the input dataset, and <output.ppm> is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process.

A pseudo-code version of the algorithm is shown below:

```
BLUR_SIZE = 1 // gives a 3x3 BLUR window
foreach pixel in the image; do
  get pixel values of all valid pixels under the BLUR_SIZE x BLUR_SIZE window;
  sum the pixel values of all valid pixels under the window;
  new pixel value = (sum of pixel values) ÷ (no. of valid pixels) // average of the window pixel values
done
```

Answer the following questions.

1. How many floating operations are being performed in your color conversion kernel?
2. How many global memory reads are being performed by your kernel?
3. How many global memory writes are being performed by your kernel?
4. Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.