CO332 – Heterogeneous Parallel Computing

A2 – Vector Addition, Image Processing, Tiled MM

Reference: Wen Mei Hwu, PMPP, 2e. Modules 2, 3, 4.
Deadline: 9AM, August 16, 2017. Upload assignment here: https://cloud.nitk.ac.in/index.php/s/bPKdguGrNP7cRvf

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). Do understand what's happening under the hood to extract the max out of this assignment.

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.

Q3. [Ref. PMPP 3e. Chapter 3. Scalable Parallel Execution.]

The purpose of this lab is to convert an RGB image into a gray scale image. The input is an RGB triple of float values. You have to convert the triplet to a single float grayscale intensity value. A pseudo-code version of the algorithm is shown below:

```
for ii from 0 to height do
  for jj from 0 to width do
    idx = ii * width + jj
    # here channels is 3
    r = input[3*idx]
    g = input[3*idx + 1]
    b = input[3*idx + 2]
    grayImage[idx] = (0.21*r + 0.71*g + 0.07*b)  // converts 3 r g b values to a single grayscale value.
  end
end
```

Image Format

The input image is in PPM P6 format while the output grayscale image is to be stored in PPM P5 format. You can create your own input images by exporting your favorite image into a PPM image. On Unix, **bmptoppm** converts BMP images to PPM images (you could use **gimp** or similar tools too).

Run command:

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

where `<expected.pbm>` is the expected output, `<input.ppm>` is the input dataset, and `<output.pbm>` is an optional path to store the results. Questions.

1. How many floating operations are being performed in your color conversion kernel?
2. Which format would be more efficient for color conversion: a 2D matrix where each entry is an RGB value or a 3D matrix where each slice in the Z axis represents a color. I.e. is it better to have color interleaved in this application? can you name an application where the oposite is true?
3. How many global memory reads are being performed by your kernel?
4. How many global memory writes are being performed by your kernel?
5. Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

Q4. Perform Matrix Multiplication of two large integer matrices in CUDA. Answer the following questions.

1. How many floating operations are being performed in your matrix multiply 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.

Q5. [Ref. Chapter 4. Memory and Data Locality.]

Implement a tiled dense matrix multiplication routine using shared memory. Use the template code. Run command:

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

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.

1. How many floating operations are being performed in your matrix multiply kernel? explain.
2. How many global memory reads are being performed by your kernel? explain.
3. How many global memory writes are being performed by your kernel? explain.
4. Describe what further optimizations can be implemented to your kernel to achieve a performance speedup.
5. Compare the implementation difficulty of this kernel compared to the previous MP. What difficulties did you have with this implementation?
6. Suppose you have matrices with dimensions bigger than the max thread dimensions. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication in this case.
7. Suppose you have matrices that would not fit in global memory. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication out of place.