Programming Assignment 4 - CUDA

Due Date: Tues. 11/21

*** please get an early start on this! ***

Using CUDA at Ohio Supercomputer Center

The OSC Owens cluster is equipped with Tesla P100 GPUs. Some relevant stats for the P100:

```
Total amount of global memory:                 16 GB
Compute Capability:                            6.0
(56) Multiprocessors, (64) CUDA Cores/MP:     3584 CUDA Cores
GPU Clock rate:                                1.328 GHz
L2 Cache Size:                                 4.0 MB
Total amount of constant memory:               65536 bytes
Total amount of shared memory per block:       49152 bytes
Total number of registers available per block: 65536
Warp size:                                     32
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:           1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535
```

To use CUDA on the OSC cluster, you must allocate a node with an attached GPU. To interactively allocate such a node, use:

```
$ qsub -I -l walltime=0:59:00 -l nodes=1:gpus=1
```

(qsub -EYE -ell walltime ... -ell nodes ...).

To ensure the best resource availability for everyone, please only log on to a GPU host node when you are ready compile and run, then please exit when you are not actively testing.

To compile and test your programs you will need to load the CUDA environment:

```
$ module load cuda
```

and then use the Nvidia compilers. For example:

```
$ nvcc -O -o lab4p1 jones_jeffrey_lab4p1.cu
```

The "-O" flag sets the compiler to the default level (3), while the "-o lab4p1" flag, specifies the name for the the executable file, "lab4p1," which you can then execute by name:

```
$ lab4p1
```

Note that compilation (use the nvidia compiler, nvcc) can be performed on the login nodes, and does not require a node with a GPU. You will need to load the cuda module on the login node if you wish to do this. You will not be able to test your programs successfully on the login nodes, as they have no GPUs.

Nvidia CUDA drivers available free on-line

If your laptop/desktop has an Nvidia graphics card, you can download the CUDA drivers directly from Nvidia for your own local development and testing. Please see `https://developer.nvidia.com/cuda-downloads`. Nvidia's "CUDA Zone" also provides a wide array of tools and documentation: `https://developer.nvidia.com/cuda-zone`.

Part 1

Create both serial and CUDA parallel programs based on the following code segment, which multiplies the transpose of a matrix with itself:

```
double A[1024][1024], C[1024][1024];
// insert code to initialize matrix elements to random values between 1.0 and 2.0
for (i = 0; i < 1024; i++)
    for (j = 0; j < 1024; j++)
        for (k = 0; k < 1024; k++)
            C[ i ][ j ] += A[ k ][ i ] * A[ k ][ j ];
```

Use the code as above for your **serial** version on OSC using 1 node with 12 processors (full node). Use whatever techniques you feel appropriate to design a **CUDA parallel** version.
a) Report your results in estimated GFlops.
b) Measure both serial and parallel performance.
c) Report the CUDA compute structure (Grid, Block and Thread) you used and explain your results.

Part 2

Implement both serial and CUDA parallel programs to perform Sobel operator for edge detection on .bmp image files.

Background

The Sobel operator performs a 2-D spatial gradient measurement on images. The Sobel edge detector uses a pair of 3 x 3 stencils, or "convolution masks," one estimating gradient in the x-direction and the other estimating gradient in y-direction. The Sobel detector is incredibly sensitive to noise in pictures, it effectively highlights them as edges.

Sobel Operator Description

An image gradient is a change in intensity (or color) of an image. An edge in an image occurs when the gradient is greatest and the Sobel operator makes use of this fact to find the edges in an image. The Sobel operator calculates the approximate image gradient of each pixel by “convolution" of the image with a pair of 3x3 filters. These filters estimate the gradients in the horizontal (x) and vertical (y) directions. The magnitude of the gradient is simply the sum of these 2 gradients.

Gx:

-1	0	+1
-2	0	+2
-1	0	+1

Gy:

+1	+2	+1
0	0	0
-1	-2	-1

The magnitude of gradient is calculated using

$$G = \sqrt{Gx^2 + Gy^2}$$

Determining threshold for Pixel Classification

To perform Sobel edge detection the gradient magnitude is computed for each pixel (excluding the pixels in boundary), with the pixel being classified as white or black based on comparing the gradient to a threshold. Below are the steps to follow

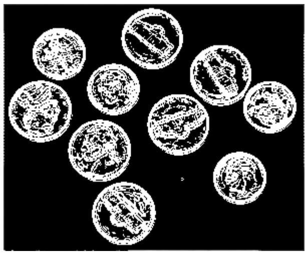
- Take the input image (N x M) pixels
- Iterate over each pixel (excluding the first/last row and first/last column pixels)
- Determine the Magnitude G of new pixel, from Gx and Gy where:
    - G$_x$ is the sum of the products of the G$_x$ stencil multiplied by the corresponding pixel values that align with the stencil when the stencil is centered on the current pixel (that is, the center element of the stencil corresponds to the current pixel).
    - G$_y$ is computed similarly by employing the G$_y$ stencil.
    - For each pixel, G(pixel) = $\sqrt{G_x^2 + G_y^2}$
- Use a threshold for classifying the pixel as black or white, if the magnitude is greater than threshold assign white(255), else black(0)
- The Resultant image (N x M) containing new magnitude will be a black & white image with explicit edges
- **Note:** Assume that the boundary pixels are simply copied to new image without any modifications

Our convergence criterion for this experiment is to achieve a image having **greater than 75% of percentage pixels being black**. We iterate over different threshold values starting at 0 and incrementing by 1%, until this convergence is achieved. Below is the logic flow

```
threshold = 0
while(black_cell_count < 0.75 * num_of_pixels)
{
     iterate over all pixels
    {
         G = sobel_output(pixel)
         if(G > threshold)
             sobel_image[pixel] = white (255)
         else
             sobel_image[pixel] = black (0)
             black_cell_count++
    }

    increment_threshold
}
```

Input Image:

Sobel Output Image: 

Input Images

Write your program to accept as input the 24-bit bmp image format.

Serial Code for Sobel operator with Classification convergence

Use the below code as a reference for your serial version of code

```
    //Read the binary bmp file into buffer


    //Allocate new output buffer of same size


    //Get any needed image attributes


    //Convergence loop
    threshold = 0;
    while(black_cell_count < (75*wd*ht/100))
    {

        black_cell_count = 0;
        threshold += 1;
        for(i=1; i < (ht-1); i++)
        {
            for(j=1; j < (wd-1); j++)
            {
                Gx = bmp_data[ (i-1)*wd + (j+1) ] - bmp_data[ (i-1)*wd + (j-1) ] \
                        + 2*bmp_data[ (i)*wd + (j+1) ] - 2*bmp_data[ (i)*wd + (j-1) ] \
                        + bmp_data[ (i+1)*wd + (j+1) ] - bmp_data[ (i+1)*wd + (j-1) ];

                Gy = bmp_data[ (i-1)*wd + (j-1) ] + 2*bmp_data[ (i-1)*wd + (j) ] \
                        + bmp_data[ (i-1)*wd + (j+1) ] - bmp_data[ (i+1)*wd + (j-1) ] \
                        - 2*bmp_data[ (i+1)*wd + (j) ] - bmp_data[ (i+1)*wd + (j+1) ];

                mag = sqrt(Gx * Gx + Gy * Gy);
                if(mag > threshold)
                {
                    new_bmp_img[ i*wd + j] = 255;
                }else{
                    new_bmp_img[ i*wd + j] = 0;
                    black_cell_count++;
                }
             }
         }
     }

//Write back the new bmp image into output file
```

Instrumentation

- Use the instructions and example code above to develop both serial and CUDA parallel versions.
  Note: In your CUDA version the entire convergence need not be parallel, try your best to optimize the sobel operator and parallelize as much as possible. You may implement multiple kernels if you wish. You may use serial code segments as necessary.
- A sample .bmp image is available in /class/cse5441 directory.
- Your program should take the following command line parameters:
  ./a.out <input image file.bmp> <serial processed image.bmp> <cuda processed image.bmp>
- Your program should output
  a) Time taken for serial execution
  b) Time taken for cuda execution
  c) Threshold obtained in serial vs cuda version
  A sample output is given below. You should report all requested information but are not restricted to this exact format.

```
./indresh_sira_lab4p2.out image_1.bmp serial_image.bmp cuda_image.bmp
********************************************************************
Image Info::
           Height=3658       Width=2962
Time taken for serial sobel operation: 12.0457 sec
Threshold during convergence: 69

Time taken for CUDA sobel operation: 1.0457 sec
Threshold during convergence: 69
********************************************************************
```

Reporting

Run your program against the sample image provided using instructions specified above. Provide the following:

- Provide a timing and threshold convergence summary for all images
- Explain your cuda organization (grid, block, thread) distribution
- Did you see any performance improvement in using GPU?
  Support your answer with numbers from your observation.

Submitting Results

Generally, follow the submission guidelines for the previous labs, with the following specifics:

- Create submission directory name "cse5441_lab4." and place your files in it.
- Submit your work from Owens using the OSC submit system:
  https://www.osc.edu/resources/getting_started/howto/howto_submit_homework_to_repository_at_osc.

important note: The submission system at OSC is new this semester. Please do not wait until the last minute to submit your programs, as unexpected issues could arise. Difficulty in submitting your work will not be grounds for a deadline extension.

- name your program files <lastname>_<firstname>_lab4p1.cu and <lastname>_<firstname>_lab4p2.cu.
- provide a single make file that will name your executables lab4p1 and lab4p2
- Upload your .pdf report to Carmen. Be sure to include in your report your name and section number as well as all relevant CUDA parameter settings you used.