Description Code Questions Attempts History

Image Convolution

The coding deadline is 11 days from now (2015-07-10T23:00:00Z) .

Objective

The lab's objective is to implement a tiled image convolution using both shared and constant memory as discussed in class. Like what's discussed in class, we will have a constant 5x5 convolution mask, but will have arbitrarily sized image (We will assume the image dimensions are greater than 5x5 in this Lab).

To use the constant memory for the convolution mask, you can first transfer the mask data to the device. Assume you decided to name the pointer to the device array for the mask M. As described in Lecture 3-5, you can use `const float * __restrict__ M` as one of the parameters during your kernel launch. This informs the compiler that the contents of the mask array are constants and will only be accessed through pointer variable `M`. This will enable the compiler to place the data into constant memory and allow the SM hardware to aggressively cache the mask data at runtime.

Convolution is used in many fields, such as image processing where it is used for image filtering. A standard image convolution formula for a 5x5 convolution filter `M` with an Image `I` is:

$$P_{i,j,c} = \sum_{x=0}^{4}\sum_{y=0}^{4} I_{i+x-2,j+y-2,c} M_{x,y}$$

where `P_{i,j,c}` is the output pixel at position `i,j` in channel `c`, `I_{i,j,c}` is the input pixel at `i,j` in channel `c` (the number of channels will always be 3 for this MP corresponding to the RGB values), and `M_{x,y}` is the mask at position `x,y`.

Prerequisites

Before starting this lab, make sure that:

- You have completed all week 3 lecture videos

Input Data

The input is an interleaved image of `height x width x channels`. By interleaved, we mean that the the element `I[y][x]` contains three values representing the RGB channels. This means that to index a particular element's value, you will have to do something like:

```
index = (yIndex*width + xIndex)*channels + channelIndex;
```

For this assignment, the channel index is 0 for R, 1 for G, and 2 for B. So, to access the G value of `I[y][x]`, you should use the linearized expression `I[(yIndex*width+xIndex)*channels + 1]`.

For simplicity, you can assume that `channels` is always set to `3`.

Instruction

Edit the code in the code tab to perform the following:

- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- implement the tiled 2D convolution kernel with adjustments for channels
- use shared memory to reduce the number of global accesses, handle the boundary conditions in when loading input list elements into the shared memory

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

Pseudo Code

Your sequential pseudo code would look something like

```
maskWidth := 5
maskRadius := maskWidth/2 # this is integer division, so the result is 2
for i from 0 to height do
  for j from 0 to width do
    for k from 0 to channels
      accum := 0
      for y from -maskRadius to maskRadius do
        for x from -maskRadius to maskRadius do
          xOffset := j + x
          yOffset := i + y
          if xOffset >= 0 && xOffset < width &&
             yOffset >= 0 && yOffset < height then
            imagePixel := I[(yOffset * width + xOffset) * channels + k]
            maskValue := K[(y+maskRadius)*maskWidth+x+maskRadius]
            accum += imagePixel * maskValue
          end
        end
      end
      # pixels are in the range of 0 to 1
      P[(i * width + j)*channels + k] = clamp(accum, 0, 1)
    end
  end
end
```

where `clamp` is defined as

```
def clamp(x, start, end)
  return min(max(x, start), end)
end
```

Input Format

For people who are developing on their own system. The images are stored in PPM (`P6`) format, this means that you can (if you want) create your own input images. The easiest way to create image is via external tools. You can use tools such as `bmptoppm` . The masks are stored in a CSV format. Since the input is small, it is best to edit it by hand.

Suggestions

- The system's autosave feature is not an excuse to not backup your code and answers to your questions regularly.
- If you have not done so already, read the tutorial (/help)
- Do not modify the template code provided – only insert code where the `//@@` demarcation is placed
- Develop your solution incrementally and test each version thoroughly before moving on to the next version
- Do not wait until the last minute to attempt the lab.
- If you get stuck with boundary conditions, grab a pen and paper. It is much easier to figure out the boundary conditions there.
- Implement the serial CPU version first, this will give you an understanding of the loops
- Get the first dataset working first. The datasets are ordered so the first one is the easiest to handle
- Make sure that your algorithm handles non-regular dimensional inputs (not square or multiples of 2). The slides may present the algorithm with nice inputs, since it minimizes the conditions. The datasets reflect different sizes of input that you are expected to handle
- Make sure that you test your program using all the datasets provided (the datasets can be selected using the dropdown next to the submission button)
- Check for errors: for example, when developing CUDA code, one can check for if the function call succeeded and print an error if not via the following macro:

```
#define wbCheck(stmt) do {                                                    \
        cudaError_t err = stmt;                                               \
        if (err != cudaSuccess) {                                             \
            wbLog(ERROR, "Failed to run stmt ", #stmt);                       \
            wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));    \
            return -1;                                                        \
        }                                                                     \
    } while(0)
```

An example usage is `wbCheck(cudaMalloc(...))` . A similar macro can be developed while programming OpenCL code.

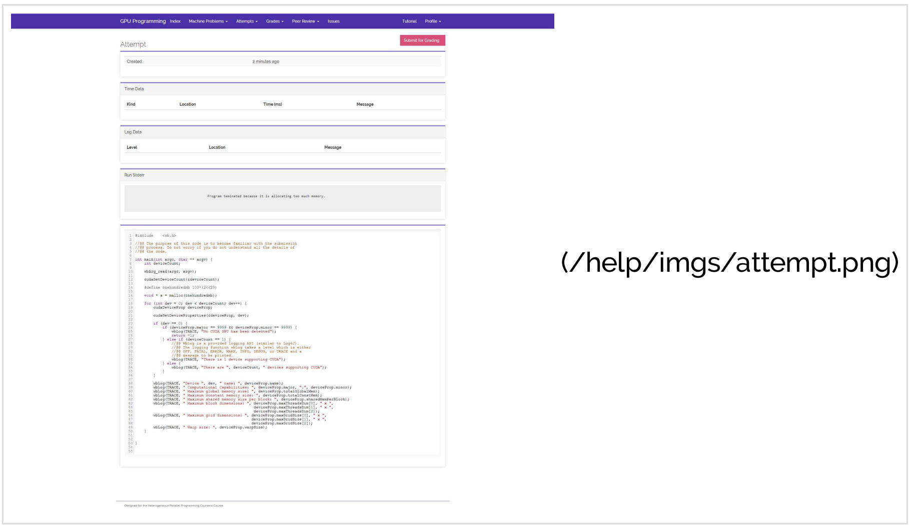
Plagiarism

Plagiarism will not be tolerated. The first offense will result in the two parties getting a 0 for the machine problem. Second offense results in a 0 for the course.

Grading

Grading is performed based on criteria that are specific for each lab.

For each attempt in the attempts tab, you'll see a *submit for grading* button.



Attempt

Created:	2 minutes ago

Time Data

Kind	Location	Time (ms)	Message

Log Data

Level	Location	Message

Run Stderr

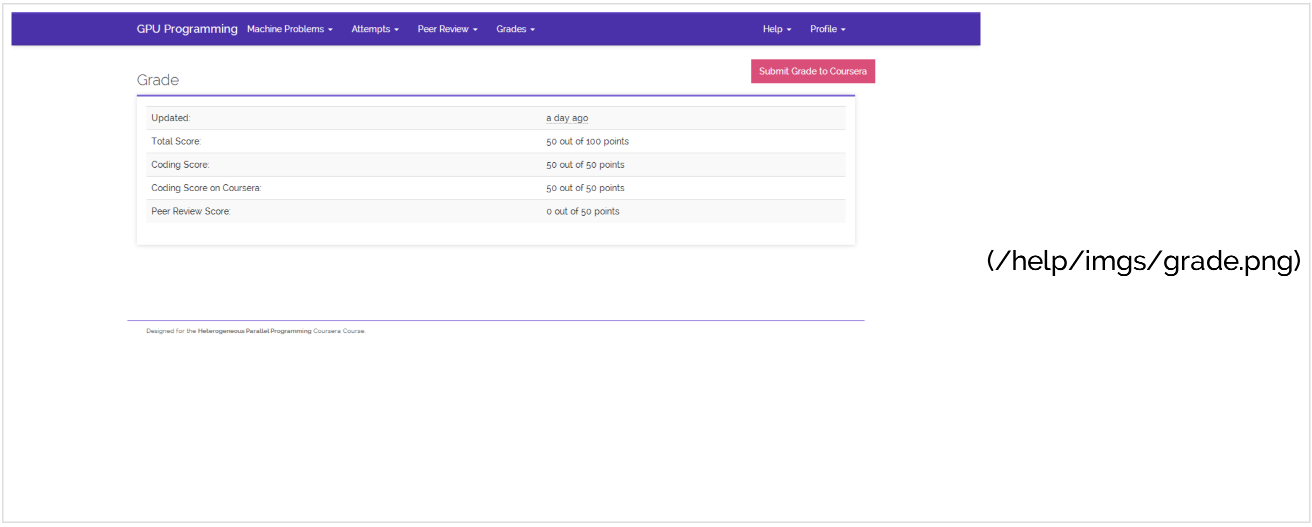
```
Program teminated because it is allocating too much memory.
```

```
#include    <wb.h>

//@@ The purpose of this code is to become familiar with the submission
//@@ process. Do not worry if you do not understand all the details of
//@@ the code.

int main(int argc, char ** argv) {
    int deviceCount;

    wbArg_read(argc, argv);

    cudaGetDeviceCount(&deviceCount);

    #define onehundredmb 100*(2<<20)

    void * s = malloc(onehundredmb);

    for (int dev = 0; dev < deviceCount; dev++) {
        cudaDeviceProp deviceProp;

        cudaGetDeviceProperties(&deviceProp, dev);

        if (dev == 0) {
            if (deviceProp.major == 9999 && deviceProp.minor == 9999) {
                wbLog(TRACE, "No CUDA GPU has been detected");
                return -1;
            } else if (deviceCount == 1) {
                //@@ WbLog is a provided logging API (similar to Log4J).
                //@@ The logging function wbLog takes a level which is either
                //@@ OFF, FATAL, ERROR, WARN, INFO, DEBUG, or TRACE and a
                //@@ message to be printed.
                wbLog(TRACE, "There is 1 device supporting CUDA");
            } else {
                wbLog(TRACE, "There are ", deviceCount, " devices supporting CUDA");
            }
        }

        wbLog(TRACE, "Device ", dev, " name: ", deviceProp.name);
        wbLog(TRACE, " Computational Capabilities: ", deviceProp.major, ".", deviceProp.minor);
        wbLog(TRACE, " Maximum global memory size: ", deviceProp.totalGlobalMem);
        wbLog(TRACE, " Maximum constant memory size: ", deviceProp.totalConstMem);
        wbLog(TRACE, " Maximum shared memory size per block: ", deviceProp.sharedMemPerBlock);
        wbLog(TRACE, " Maximum block dimensions: ", deviceProp.maxThreadsDim[0], " x ",
                                                   deviceProp.maxThreadsDim[1], " x ",
                                                   deviceProp.maxThreadsDim[2]);
        wbLog(TRACE, " Maximum grid dimensions: ", deviceProp.maxGridSize[0], " x ",
                                                  deviceProp.maxGridSize[1], " x ",
                                                  deviceProp.maxGridSize[2]);
        wbLog(TRACE, " Warp size: ", deviceProp.warpSize);
    }


}
```

Designed for the Heterogeneous Parallel Programming Coursera Course

(/help/imgs/attempt.png)

Clicking on that would tell the system to grade that attempt and redirect you to the grade report page.



Grade

Updated:	a day ago
Total Score:	50 out of 100 points
Coding Score:	50 out of 50 points
Coding Score on Coursera:	50 out of 50 points
Peer Review Score:	0 out of 50 points

Designed for the **Heterogeneous Parallel Programming** Coursera Course

(/help/imgs/grade.png)

The grades are automatically sent to Coursera. You will have to submit your grade back to Coursera by clicking the button submit to Coursera button.

Note: you cannot submit code for grading after the corresponding deadlines have passed.

Local Development

While not required, the library used throughout the course can be downloaded from Github (https://github.com/abduld/libwb). The library does not depend on any external library (and should be cross platform), you can use `make` to generate the shared object file (further instructions are found on the

Github page). Linking against the library would allow you to get similar behavior to the web interface (minus the imposed limitations). Once linked against the library, you can launch your program as follows:

```
./program -e <expected_output_file> -i <input_file_1>,<input_file_2> -o <output_file> -t <t
ype>
```

The `<expected_output_file>` and `<input_file_n>` are the input and output files provided in the dataset. The `<output_file>` is the location you'd like to place the output from your program. The `<type>` is the output file type: `vector` , `matrix` , or `image` . If an MP does not expect an input or output, then pass `none` as the parameter.

In case of issues or suggestions with the library, please report them through the issue tracker (https://github.com/abduld/libwb/issues) on Github.

Issues/Questions/Suggestions

In case of questions, please post them to the forums. For issues or suggestions, please use the issue tracker (https://github.com/abduld/wb/issues).

Deadline

The coding deadline is 11 days from now (2015-07-10T23:00:00Z) .

Note: you cannot submit code and answer questions for grading nor perform a peer review after the corresponding deadlines have passed.

Grading Rubric

100 out of the 100 possible points is reserved for code development.

Dataset

We provide the dataset (/dataset/6/dataset.zip) to allow you to develop the machine problem on your own system or to examine the data. This is purely optional however. Consult the tutorial (/help) for information about local development.