Project 2

Programming Project:

Parallel Implementation of Image Blurring (Filtering) on GPU

Instructions:

In this programming assignment, students are asked to write a program and a report on the program they developed, explaining their approach and results. The project involves taking a gray scale image and blurring it using a technique known as low pass filtering.

Please read the instructions here carefully and start working on your project as soon as possible to give yourself time to debug and complete the program.

1- Overview:

In image processing, the process of taking an input image and performing a spatial operation on its pixels is called filtering. There are two main filters; low-pass and high-pass filters. Low-pass filters perform a blurring operation on the image. The process involves taking a neighborhood around each pixel and replacing the pixel's value with the average value of its neighbors. In contrast, high-pass filers perform a subtraction operation as opposed to the averaging operation performed in low-pass filters, resulting in the detection of sharp changes in the image.

In this project you are asked to perform the low-pass (averaging/blurring) operation on an input image. The operation takes a 3x3 neighborhood around each pixel and calculates the average of the pixels in the neighborhood according to the following equation:

$$O(i,j) = \frac{1}{9} \sum_{m=-1,n=-1}^{m=1,n=1} In(i+m, j+n)$$

Where, *In* is the input image, *O* is the output image and (*i*,*j*) is the pixel location.

2- Tasks:

a. Project Setup:

To set up your project, go to the Microsoft Visual Studio and create a new project. Base your new project on CUDA 4.x+ template. Then add a .cpp file to the project and from the project properties add your OpenCV libraries, include headers, and binaries to your project.

Remove the unnecessary items from the kernel.cu file. You will write your code into the kerne.cu file. NOTE: Please refer to the video lectures to see how to set up your code.

b. Write a serial version of the program

In the first part of the project you are asked to write a serialized code that takes as input an image (in gray-scale) and call a function to perform the image blurring process on the input image.

c. Accelerate your code by parallelizing the program with CUDA using global memory

Write a kernel in CUDA that takes as input two pointers on the device to the input data (which comes from the input gray scale image data) and the output data to perform the blurring. Note that each thread will have an index and will be processing 9 pixels from the input data to generate one pixel for the output data (a gather operation).

You should particularly note that both the input and output image are on the CPU (host). Make sure that you define two pointers (one for the input and one for output) on your GPU (device). NOTE: the size of both GPU pointer must match the size of their CPU counterparts. Transfer the data from the input image from CPU to the GPU input pointer before invoking the kernel. After the Kernel runs to completion, use cudaDeviceSynchronize();. Then transfer the data from the output pointer on the GPU to the image data structure of the output image on the CPU.

d. Accelerate your code by parallelizing the program with CUDA using shared memory

Write another kernel in CUDA that takes as input two pointers on the device to the input data (which comes from the input gray scale image data) and the output data to perform the blurring operation. In this kernel, allocate a shared memory block. Then transfer the data for each block from the global memory pointer to its shared memory. Perform the pixel calculations on the values from the shared memory to speed up the process. Note that each thread will have an index and will be processing 9 pixels from the input data. However, you will have to treat the pixels in each block differently depending on where they are located. Please see Figure 1.

In Figure 1, each square is associated with one thread. Therefore, each block will be made of NxN threads. In this figure the top 4 blocks are shown (two from the first row and two from the second row. Notice that there are three kind of threads that you should account for:

1- The green-colored threads are those that are located on the boundary of the image. These threads will not have all of their 8 neighbors to calculate the average. For example, the thread on the top left corner –(0,0)- will only have 3 neighbors – (0,1), (1,0), and (1,1).

2- The blue-colored threads are the ones that have all of their 8 neighbors. Furthermore, when you copy the data to the shared memory, these threads can used the shared memory data to calculate the average.

3- The orange-colored threads have some of their neighbors in their own block and some of their neighbors in their adjacent block(s). For these threads you must use the global memory to calculate the average (as some of their neighbor's data will not be in the shared memory).

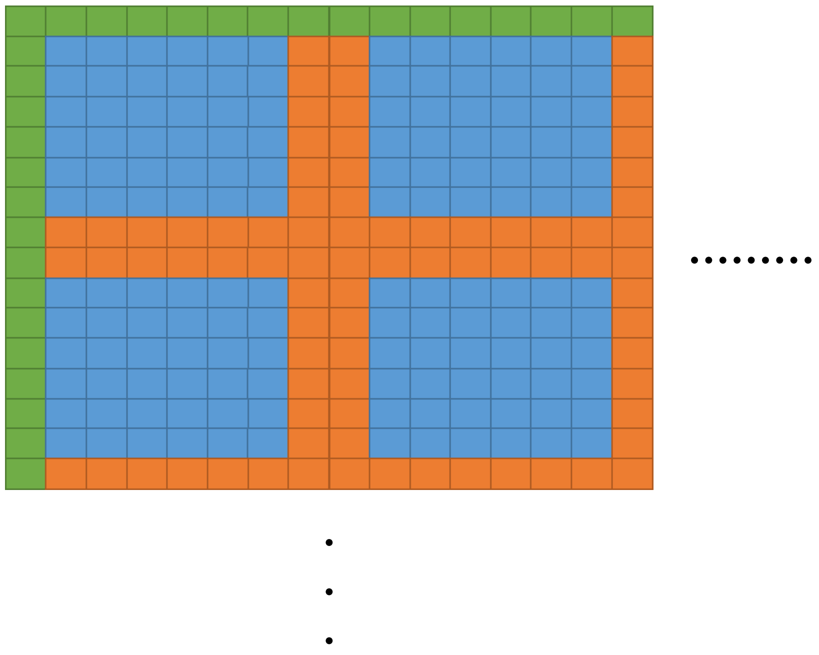

Figure 1. The breaking of an image into blocks of threads. Each thread (shown here in a square shape) is in charge of calculating the average over itself and its 8 neighbors and storing the result in the output image location.

e. Performance evaluation

Using CUDA events, set up start and stop events and calculate the elapsed time of your CUDA kernel with global memory and the one with shared memory. Also, use the time functions on CPU to calculate the time it takes for the CPU code to perform the image blurring. Create a table comparing your results.

3- Results:

a. Serial Results

Use the images available in the attachments of this project description from the blackboard to run your CPU side code. For each image use the imshow() statement to show the input image and the result of your serial CPU side code. Calculate the performance of you CPU code with CPU time functions and find out in milliseconds what is the time it takes for the CPU to blur an image.

b. GPU Results

Use the images available in the attachments of this project description from the blackboard to run your GPU side kernels. For each image use the imshow() statement to show the input image and the result of your parallel GPU side kernels. Use the CUDA Event functions to calculate the time in millisecond that it takes for the CUDA kernel using only global memory and the one using shared memory to blur an image.

4- Deliverables

a. Code

Submit your kernel.cu code as an attachment to your submission. Please make sure you document your code.

b. Report

Write a report on your project. The report should be about 3-5 pages long and should contain your own description of both your CPU and GPU side code. The report should contain the following items:

- Write a short section on the project summary.
- Your implementation of both the CPU and GPU code. This section should include any parameters you used on the CPU side code as well as your GPU side code, such as; kernel size, block and grid sizes, conditions, loops, etc.
- Any issues encountered while working on this project, and your attempts to address the issues.
- Results of your program both on the CPU and the GPU.
- Your conclusions about the implementation and the techniques you used and how you see them suitable for a real-life application.