GPU Programming Basics

Northeastern University
NUCAR Laboratory

for all

Julian Gutierrez
David Kaeli

Hands-on Lab #4

Objective

- Calculate the best operating point for a simple image processing algorithm.
- Study different approaches for a convolutional filter on an image and its performance impact.

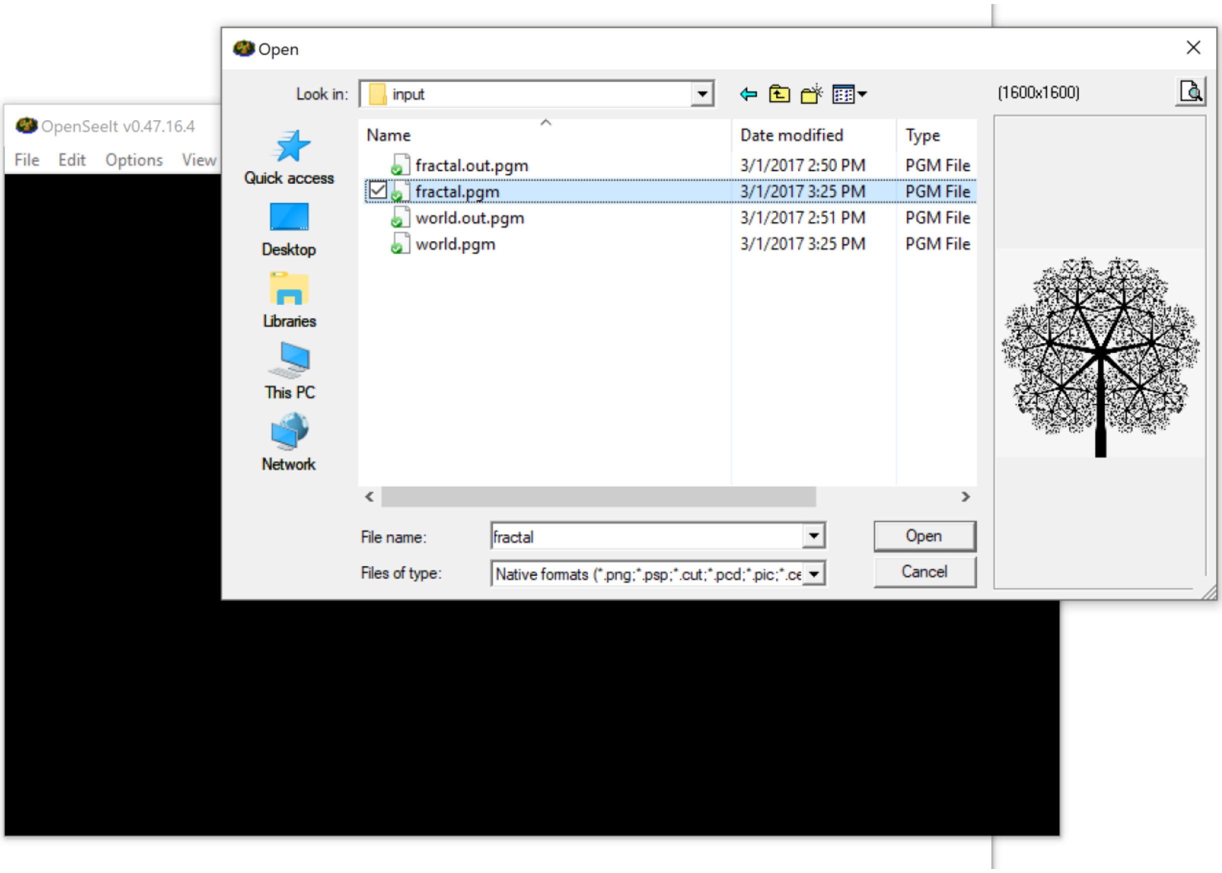
NOTE: Unless stated otherwise, work in pairs.

Part 1: Brightness

In this lab, we will be modifying images. To visualize the images, we need a program that can view them. Given our connection to the server is through an ssh terminal which doesn't support window displaying, we need to copy the images back to our computer to be able to visualize them (using FileZilla for example). We are using ppm and pgm formatting which means, conventional image tools on Windows don't support this file format. To view them, we will download an open source image viewer called OpenSeeIt.

https://sourceforge.net/projects/openseeit/

Using it is straight forward, but here is an image to demonstrate the usage:



Look in: input

(1600x1600)

Name	Date modified	Type
fractal.out.pgm	3/1/2017 2:50 PM	PGM File
☑ fractal.pgm	3/1/2017 3:25 PM	PGM File
world.out.pgm	3/1/2017 2:51 PM	PGM File
world.pgm	3/1/2017 3:25 PM	PGM File

File name: fractal

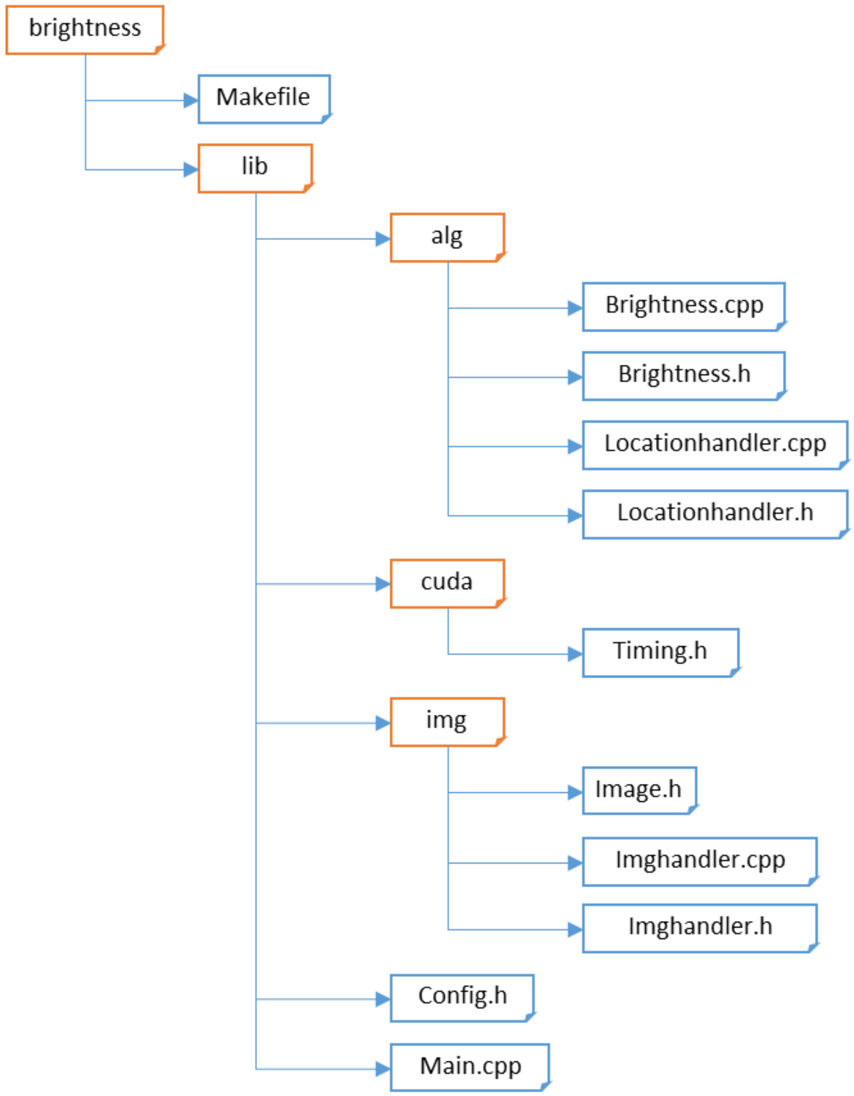
Files of type: Native formats (*.png;*.psp;*.cut;*.pcd;*.pic;*.ce

All the files needed can be found in the following folder on the server:

```
/scratch/gutierrez.jul/GPUClass/HOL4/DELIVERABLE/
```

Copy the folder into your home directory to be able to modify the contents.

File structure:



Read the following files and understand how they work and answer the questions.

- Makefile.
  - Helps automate the process of compiling the code for the project.
  - What could you add in this file that might improve the performance of your code? Look at how the files are compiled.
- Config.h
  - This file oversees the definition of the different variables used in the execution of the code.
  - What changes in this file could significantly affect the performance of the algorithm?

- Main.cpp
  - Main function that calls the function to execute the algorithm on the gpu.
  - Why do we run a warmup function?
  - Extra: How do we read the input image and write the output image? Tip: Look at the files under the img folder.
- Brightness.cu
  - This is the function that controls the GPU and executes the kernel.
  - Look at how the kernel is executed. Grid size, block size, kernel code.
  - Understand how the brightness function and the kernel work.

First, let's run the code to see what it does. To compile the code run the following command:

```
make all
```

To run the code:

```
./brightness --image ../input/fractal.pgm
```

It will produce an output image called:

```
result.ppm
```

Copy this file with FileZilla to your computer. Use the previous program to open the image.

To compare the input and output, open both inputs. Do you see any differences?

You can change the brightness level by using --brightness <value> in the execution of the program. Try writing a negative value. Remember pixel values go from 0 to 255.

```
./brightness --image ../input/fractal.pgm --brightness -100
```

Your job is to find which block size performs best by modifying it on the `config.h` file. Test it with the `fractal.pgm` image and the `world.pgm` separately. NOTE: Sometimes execution time might vary a lot, so doing a couple of test runs and averaging the result is encouraged (only if we have enough time).

- Which block size is the best one for each input?
- Could these values be different if the size of the image varies significantly?
- How do you think it affects the performance?
- Why do we use only block sizes with 2^n form?

Block Size	**Kernel Execution Time (fractal)**	**Kernel Execution Time (world)**
2		
4		
8		
16		
32		

- What optimization could be done to improve this code?

Run nvprof the same way we did for the second lab. Use the best configuration for the block size based on the smallest kernel execution time for the fractal image. MAKE SURE YOU COMPILE THE CODE AGAIN. NOTE: Also remember that we are running a warmup code which executes the same instructions. You can remove this by setting the WARMUP flag to 0 in the config file.

```
nvprof ./brightness --image ../input/fractal.pgm
```

Look at the percentages and decide, is it worth optimizing this code? If yes, where would you optimize?

Now let's look at some of the metrics by running the following command:

```
nvprof -m all  ./brightness --image ../input/fractal.pgm &> nvprof.log
```

This command will show all metrics and save the output into nvprof.log. Open the file and look at the results, is there anywhere you think we could further improve the kernel?

Extra: ONLY IF YOU FINISHED THE NEXT PART AS WELL. Test one of your ideas to improve the performance. Did it work?

Part 2: Sobel Algorithm

As shown in class, Sobel algorithm is an edge detection algorithm. It applies a convolution filter on the image and outputs another image highlighting the pixels that are located on borders based on a certain threshold.

In this part, we are going to analyze different implementations of the same algorithm and compare their performance. The structure of the folders is identical to the brightness algorithm we saw on the previous part. Compilation and execution is the same as well.

To compile the codes run the following command inside their respective folders:

```
make all
```

To run the code:

```
./sobel --image ../../input/fractal.pgm
```

It will produce an output image called:

```
result.ppm
```

We have 5 different versions of the code:

1. Basic
2. Opt1
3. Opt2
4. Opt3
5. Opt4

“Basic” is a naïve implementation of the Sobel algorithm. The other versions have different improvements done to the code to try and achieve better performance. The idea is to fill out the following table with the execution times by running each code on both inputs (no need to change the block size, though this might give you an idea).

Version	**Kernel Execution Time (fractal)**	**Total Execution Time (fractal)**	**Kernel Execution Time (world)**	**Total Execution Time (world)**
Basic				
Opt1				
Opt2				
Opt3				
Opt4				

- Note: To understand what they’re doing, look at the sobel.cu files inside the alg folder.

After recollecting the data, answer these questions:

- Explain what each optimization version does.
- Which one works best and why?
- Which one is the easiest to implement?
- Extra: Choose one of the optimized versions and run nvprof. What metrics change compared to the basic implementation?
- Extra: Change the algorithm by changing the filter to something you want and don’t use a threshold for the value. How does the output look?