PP-f22

This site uses Just the Docs, a documentation theme for Jekyll.

Programming Assignment VI: OpenCL Programming

Parallel Programming by Prof. Yi-Ping You

Due date: 23:59, Jan 5, Thursday, 2023

The purpose of this assignment is to familiarize yourself with OpenCL programming.

TABLE OF CONTENTS

1Programming Assignment VI: OpenCL Programming

a1. Image convolution using OpenCL

b2. Requirements

c3. Grading Policy

d4. Evaluation Platform

e5. Submission

f6. References

Get the source code:

```
$ wget http://nycu-sslab.github.io/PP-f22/assignments/HW6/HW6.zip
$ unzip HW6.zip -d HW6
$ cd HW6
```

1. Image convolution using OpenCL

Convolution is a common operation in image processing. It is used for blurring, sharpening, embossing, edge detection, and more. The image convolution process is accomplished by doing a convolution between a small matrix (which is called a *filter kernel* in image processing) and an image. You may learn more about the convolution process at Wikipedia: Convolution.

Figure 1 shows an illustration of the concept of applying a convolution filter to a specific pixel, value of which is `3`. After the convolution process, the value of the pixel becomes `7`—how the resulting value is computed is illustrated on the right of the figure.

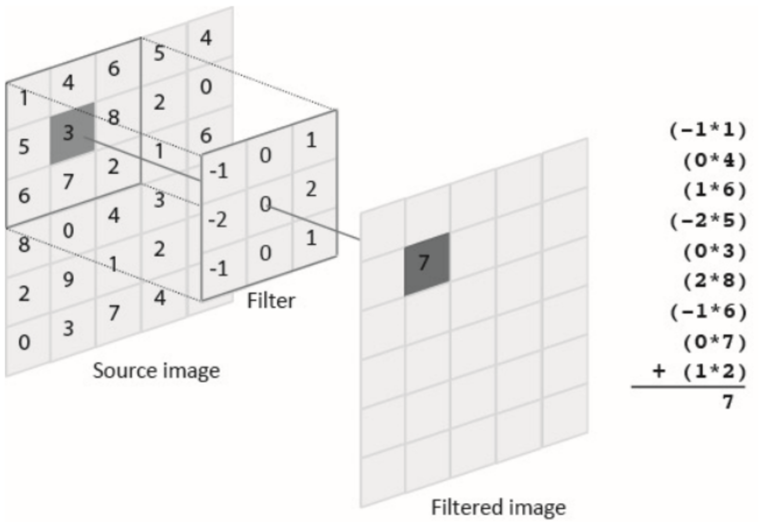

Figure 1. Applying a convolution filter to the dark gray pixel of the source image (value of which is `3`).

In this assignment, you will need to implement a GPU kernel function for convolution in OpenCL by using the zero-padding method. A serial implementation of convolution can be found in `serialConv()` in `serialConv.c`. You can refer to the implementation to port it to OpenCL. You may refer to this article to learn about the zero-padding method. Figure 2 shows an example of applying the zero-padding method to the source image (on the left) and thereby resulting a same-size, filtered output image (on the right).

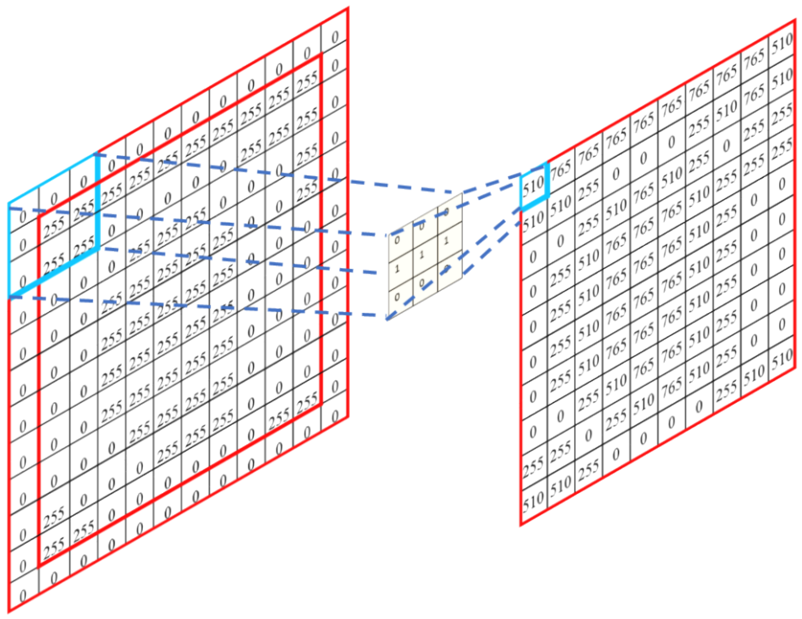

Figure 2. Applying the zero-padding method to a source image.

Your job is to parallelize the computation of the convolution using OpenCL. A starter code that spawns OpenCL threads is provided in function `hostFE()`, which is located in `hostFE.c`. `hostFE()` is the host front-end function that allocates memories and launches a GPU kernel, called `convolution()`, which is located in `kernel.cl`.

Currently `hostFE()` and `convolution()` do not do any computation and return immediately. You should complete these two functions to accomplish this assignment.

You can build the program by typing `make`, and run the program via `./conv`. Your program should read an input image from `input.bmp`, perform image convolution, and output the result image into `output.bmp`.

You can use the following command to test your own image:

```
ffmpeg -i source.png -pix_fmt gray -vf scale=600:400 destination.bmp
```

You can use a different filter kernel by adding option `-f N` when running the program (i.e., `./conv -f N`), where N is either 1 (by default), 2, or 3, and indicates which filter kernel is used. Each filter kernel is defined in a CSV file (`filter1.csv`, `filter2.csv`, or `filter3.csv`). The first line of the CSV file defines the width (or height) of the filter kernel, and the remaining lines define the values of the filter kernel.

2. Requirements

You will modify only `hostFE.c` and `kernel.cl`.

Q1 (*5 points*): Explain your implementation. How do you optimize the performance of convolution?

[Bonus] Q2 (*10 points*): Rewrite the program using CUDA. (1) Explain your CUDA implementation, (2) plot a chart to show the performance difference between using OpenCL and CUDA, and (3) explain the result.

Answer the questions marked with Q1 (and Q2) in a REPORT using HackMD. Notice that in this assignment a higher standard will be applied when grading the quality of your report.

Note: You cannot print any message in your program.

3. Grading Policy

NO CHEATING!! You will receive no credit if you are found cheating.

Total of 100+10%:

- Correctness (75%): 25% for each of the three filters (`filter1.csv`, `filter2.csv`, and `filter3.csv`). The breakdown of the 25%:
  - 10%: Your parallelized program passes the verification.
  - 15%: The speedup over the serial version should be greater than 5.0 for filter1 and 3, 4.0 for filter2.
- Performance (20%): Compete with your classmates. See the metric below.
- Questions (5+10%).

Metric:

T −Y ⎩ { T −F ×60%,if Y <T + ⎛
40%,if Y <F ×2 20%,else

$$\frac{T-Y}{T-F} \times 60\%, \text{if} Y < T + \begin{cases} 40\%, \text{if} Y < F \times 2 \\ 20\%, \text{else} \end{cases}$$

where Y

Y

and F

F

indicate the execution time of your program and the fastest program, respectively, and T =F ×1.5

T = F × 1.5

.

4. Evaluation Platform

Your program should be able to run on UNIX-like OS platforms. We will evaluate your programs on the workstations dedicated for this course. You can access these workstations by `ssh` with the following information.

The workstations are based on Ubuntu 20.04 with Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz processors and GeForce GTX 1060 6GB. `g++-10`, `clang++-11`, `cuda11.5`, and `OpenCL3.0` have been installed.

IP	Port	User Name	Password
140.113.215.197	10002-10010	{student_id}	{your_passwd}

Login example:

```
$ ssh <student_id>@140.113.215.197 -p <port>
```

You can use the testing script `test_hw6` to check your answer *for reference only*. Run `test_hw6` in a directory that contains your `HW6_XXXXXXX.zip` file on the workstation.

5. Submission

All your files should be organized in the following hierarchy and zipped into a `.zip` file, named `HW6_xxxxxxx.zip`, where `xxxxxxx` is your student ID.

Directory structure inside the zipped file:

- `HW6_xxxxxxx.zip` (root)
  - `kernel.cl`
  - `hostFE.c`
  - `url.txt`

Notice that you just need to provide the URL of your HackMD report in `url.txt`, and enable the write permission for someone who knows the URL so that TAs can give you feedback directly in your report.

Zip the file:

```
$ zip HW6_XXXXXXX.zip kernel.cl hostFE.c url.txt
```

Be sure to upload your zipped file to new E3 e-Campus system by the due date.

You will get *NO POINT* if your ZIP's name is wrong or the ZIP hierarchy is incorrect.

You will get a 5-point penalty if you hand out unnecessary files (e.g., obj files, .vscode, .__MACOSX).

6. References

- NVIDIA OpenCL SDK Code Samples
- Getting started with OpenCL and GPU Computing
