Lab 5 Report

Programming and Architecture of Computing Systems

César Borja and Nerea Gallego

December 10, 2023

1 Introduction

This report is about OpenCL programming. The aim of the assignment is to understand the basic fundamentals of OpenCL API [2] and basic kernel programming. We have implemented the three different OpenCL kernels from the "Basic kernel programming" section of the lab guide. These three kernels are:

- **Image flip**, where the goal is to transform an image by flipping it over a central vertical axis.
- **Histogram**, where the goal is to calculate the histogram for each color channel (red, green and blue).
- **Image rotation**, where the objective is to transform an image by rotating it by a certain angle.

2 Environment

The tests done in this assignment have been executed in the 10.3.17.177 machine from lab 0.06a.

We have used the image "Lena" (see Fig.1) of size 512×512 pixels to test the kernels.

Figure 1: Lena test image.

3 Image flip

We have decided to represent the input and output images as character buffers in the device memory, while in the host memory we use the CImg [1] library. Thus, the kernel receives 4 arguments: two pointers to image buffers (input and output) and the width and height of the image.

The kernel has been programmed so every work item handles the rotation of a pixel, i.e. the rotation of the three RGB channels. Because of this, the number of dimensions used to specify the and work-items in the work-group (*work_dim* parameter of *clEnqueueNDRangeKernel*) is 2. We also tested to use dimension 3, so each work item only handles the rotation of one channel of one pixel but the kernel execution time increased by 50%. So for simplification and time efficiency, we decided to use dimension of 2.

Metrics The values of the metrics for this exercise are the following:

- **Execution time of the overall program** = 0.281246 sec
- **Execution time of the kernel** = 0.001710 sec
- **Bandwith to/from memory to/from kernel (dBW)** = 1577542.36 [dBytes/sec]
- **Throughput**: As we can not know the number of instructions made by the OpenCL kernel, we are not elaborating the throughput metric in MIPS. Instead, we are using the number of "effective work operations". In this case, the effective work operation is to flip a pixel. So the **number of pixels flipped per second** is 153300584.79.
- **Memory footprint**= 3145744 Bytes. It has been calculated by the sum of the sizes of the input and output images at the host memory ($512 * 512 * 3 * 2$ bytes, 3 RGB channels, 2 images, 1 byte per pixel) plus the size of two integers ($4 * 2$ bytes, width and height parameters), plus the sum of the sizes of the input and output images at the device memory ($512 * 512 * 3 * 2$ bytes) plus the width and height parameters at the device memory ($4 * 2$ bytes).

Fig.2 shows the resulting image after the image flip kernel execution.

Figure 2: Lena test image flipped after the execution of the image flip kernel.

4 Histogram

Our OpenCL kernel implementation receives an input image (as a character buffer) and stores the resulting histograms in three global memory buffers (256 elements each) for the red, green and blue histograms. To do so, for each pixel (each work item) we obtain the red, green and blue components and then we atomically increment the corresponding histogram components to avoid data races. This data races may occur if one of the channels of two different pixels are equal, then two work-items will both increment the same histogram component. Using *atomic_inc* we ensure that the histogram is calculated correctly. As it is said, one pixel is handled by one work item, so the work dimension (*work_dim*) is 2.

Once we have the histogram representing the distribution of pixel intensities, we plot the histogram in the main program.

Metrics The values of the metrics for this exercise are the following:

- **Execution time of the overall program** = 0.198432 sec
- **Execution time of the kernel** = 0.008464 sec
- **Bandwith to/from memory to/from kernel (dBW)** = 790449.18 [dBytes/sec]
- **Throughput**: Similarly to the previous case, the "effective work operation" refers to a processed pixel, i.e for a pixel, incrementing the corresponding bin in each of the three histograms. So the **number of pixels processed per second** is 30971644.61.

- **Memory footprint**= 1579024 Bytes. It has been calculated by the sum of the sizes of the input image ($512 * 512 * 3 * 2$ bytes, 3 RGBA channels, 1 byte per pixel) plus the size of two integers ($4 * 2$ bytes, width and height parameters) plus the output histogram arrays ($256 * 3 * 4$ bytes, 3 histograms, 4 bytes per integer) at the host memory, plus the sum of the sizes of the input image ($512 * 512 * 4 * 2$ bytes) and output histogram arrays ($256 * 3 * 4$ bytes) and the width and height parameters ($4 * 2$ bytes) at the device memory.

Fig.3 shows the resulting histogram of the Lena test image. The RGB channels are superposed.

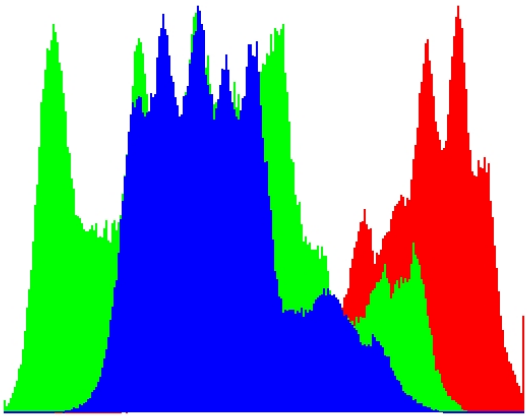

Figure 3: Lena test image histogram. The three RGB channels are superposed.

5 Image rotation

Similarly to the image flip kernel, the image rotation kernel receives two pointers to character buffers (input and output images) and the width and height of the image. Additionally, the kernel also receives the angle of rotation in radians as a float. The work dimension is also 2, so every work item handles the rotation of a pixel.

The coordinates of the rotated pixel are calculated following the equations given in the laboratory guide. Once we get the new coordinates, we do bilinear interpolation based on nearest neighbors.

Bilinear interpolation The reason why we do bilinear interpolation is to not lose smoothness of color gradations. When an image is rotated, the pixel grid of the output image doesn't align with the pixel grid of the input image. This means that the color of a pixel in the output image often needs to be a combination of the colors of multiple pixels from the input image.

What bilinear interpolation does is to make this combination. It takes the color of the 4 pixels in the input image that are closest to the rotated point, and combines them in a way that takes into account the relative distances from these 4 pixels to the mapped point. This results in a smooth transition of colors.

Metrics The values of the metrics for this exercise are the following:

- **Execution time of the overall program** = 0.353145 sec
- **Execution time of the kernel** = 0.009384 sec
- **Bandwith to/from memory to/from kernel (dBW)** = 1573720.52 [dBytes/sec]
- **Throughput**: In this case, the "effective work operation" refers to rotate a given pixel considering as center or rotation the center of the image. So the **number of pixels rotated per seconds** is 27935208.87.

- **Memory footprint** = 3145752 Bytes. It has been calculated by the sum of the sizes of the input and output images at the host memory ($512 * 512 * 3 * 2$ bytes, 3 RGB channels, 2 images, 1 byte per pixel) plus the size of two integers ($4 * 2$ bytes, width and height parameters), plus the sum of the sizes of the input and output images at the device memory ($512 * 512 * 3 * 2$ bytes) plus the width and height parameters at the device memory ($4 * 2$ bytes).

Figures 4 and 5 shows the Lena test image rotated 45^{o} and 270^{o}.

Figure 4: Lena test image rotated 45^{o}.

Figure 5: Lena test image rotated 270^{o}.

6 Conclusions

In conclusion, in this report we have explored various aspects of OpenCL programming through the implementation of three distinct kernels: image flip, histogram calculation, and image rotation. The successful execution of the kernels on the test image Lena, demonstrates the applicability of OpenCL in parallel computing for image transformations and analysis.

The numerical values obtained for execution time, bandwidth, throughput, and memory footprint align with expectations for efficient OpenCL implementations in image processing tasks. The execution times for the image flip, histogram, and image rotation kernels are notably low, indicating well-optimized algorithms. High bandwidth values suggest efficient data transfer between host and device memories. The throughput values obtained demonstrate effective parallelization and pixel processing rates. Memory footprint values are reasonable, considering both host and device memory.

References

[1] *Cimg*. `https://cimg.eu/links.html`.

[2] *OpenCL*. `https://man.opencl.org/`.