# Assignment 3

Multiprocessor Programming 521288S

Miika Sikala 2520562, msikala18@student.oulu.fi

**Task:** OpenCL continuation.

1. Implement an OpenCL program that reads an image, converts to grey scale, applies a 5x5 filter to it and saves the final image
2. Profiling the implemented OpenCL program

Description of the tasks, necessary instructions and resources are all included as an attachment here. Please go through them.

**Expected Result:** A working implementation, saved final image and a 1-2 page brief report. Please don't forget to include your working hours and references (if any) in the report.

## Introduction

In this task, I gave a little bit more structure for the program to make development easier to manage. The implementation consists of few classes that are used (NOTE: all these classes are implemented by me).

| | |
|---|---|
| **PerfTimer:** | Wrapper for windows QPC to measure execution times on sub-microsecond scale. |
| **Image:** | Holds the image data and has an ability to load, save, convert to grayscale, and filter images. Loading and saving is done using *lodepng*. |
| **MiniOCL:** | A simple wrapper for OpenCL C library to make interfacing with it much easier. Used by Image class to perform parallel execution when target device is not `TARGET_NONE`. |

The program outputs execution times of each operation. The kernel execution time is measured separately using OpenCL events.

There is a `filter_t` struct that can be used to make custom filters. The struct requires the kernel (mask), mask size and the common divisor that is applied to the kernel. The implementation includes 5x5 mean filter (meanFilter) and 5x5 gaussian filter (gaussianFilter) which can be found in main.cpp.

After converting the image to grayscale, the program saves the grayscale image as “img/gray.png” and after filtering, “img/filtered.png”.

## Usage

**To change the target device**, you can change the `COMPUTE_DEVICE` flag in the beginning of main.cpp. The options are `TARGET_NONE` (no parallelization), `TARGET_GPU` (OpenCL on GPU) and `TARGET_CPU` (OpenCL on CPU). You must re-compile the program for the change to take effect.

The application takes one input parameter, which is the image file name. See the example below, which uses image file `img/im0.png`.

To change the filter to use, you can change the argument of `img.filter(yourFilter)`. The filter must be of type `filter_t`.

Compiling using *g++*:
```
g++ main.cpp MiniOCL.cpp lodepng.cpp %OCL_ROOT%/lib/x86_64/opencl.lib -Wall -I
%OCL_ROOT%\include -o image-filter.exe
```

Running:
```
image-filter.exe img/im0.png
```

## Testing and benchmarking the implementations

The correctness of the implementation was tested by checking the results manually. The screenshots of the outputs of the program are shown below. The resulting images are found in “img” folder and partly in Figure 4.

Figure 1 shows the execution on CPU without OpenCL. Figures 2 and 3 show the execution of the OpenCL programs on CPU and GPU, respectively. The outputs also show the execution time and device information. Figure 4 shows a detail of the processed image.

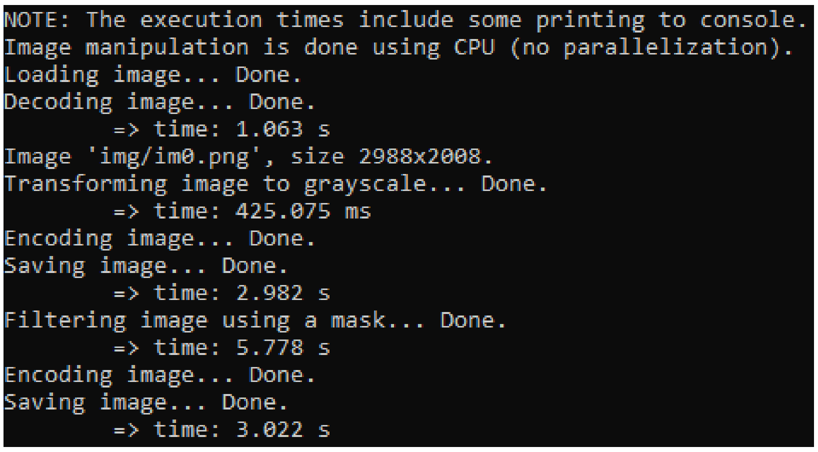

```
NOTE: The execution times include some printing to console.
Image manipulation is done using CPU (no parallelization).
Loading image... Done.
Decoding image... Done.
        => time: 1.063 s
Image 'img/im0.png', size 2988x2008.
Transforming image to grayscale... Done.
        => time: 425.075 ms
Encoding image... Done.
Saving image... Done.
        => time: 2.982 s
Filtering image using a mask... Done.
        => time: 5.778 s
Encoding image... Done.
Saving image... Done.
        => time: 3.022 s
```

*Figure 1. Output of the program without using OpenCL.*

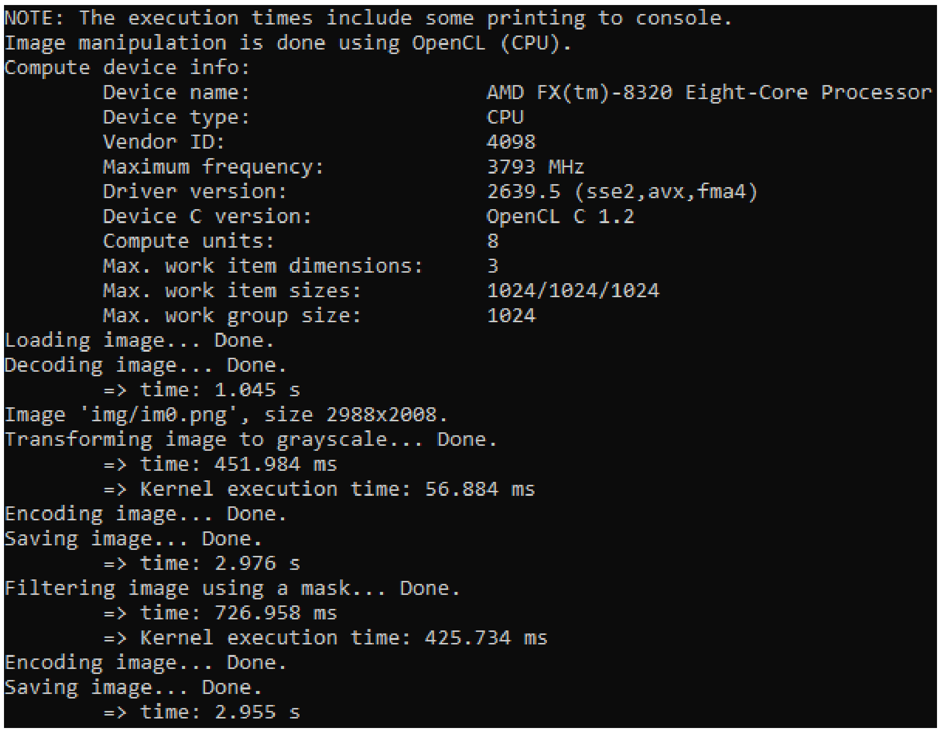
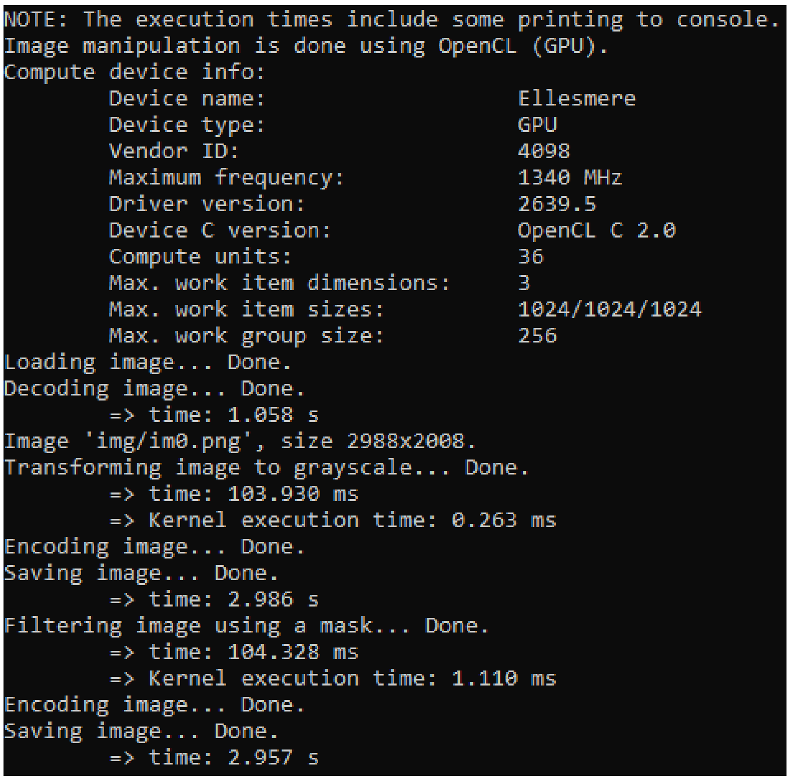

```
NOTE: The execution times include some printing to console.
Image manipulation is done using OpenCL (CPU).
Compute device info:
        Device name:                    AMD FX(tm)-8320 Eight-Core Processor
        Device type:                    CPU
        Vendor ID:                      4098
        Maximum frequency:              3793 MHz
        Driver version:                 2639.5 (sse2,avx,fma4)
        Device C version:               OpenCL C 1.2
        Compute units:                  8
        Max. work item dimensions:      3
        Max. work item sizes:           1024/1024/1024
        Max. work group size:           1024
Loading image... Done.
Decoding image... Done.
        => time: 1.045 s
Image 'img/im0.png', size 2988x2008.
Transforming image to grayscale... Done.
        => time: 451.984 ms
        => Kernel execution time: 56.884 ms
Encoding image... Done.
Saving image... Done.
        => time: 2.976 s
Filtering image using a mask... Done.
        => time: 726.958 ms
        => Kernel execution time: 425.734 ms
Encoding image... Done.
Saving image... Done.
        => time: 2.955 s
```

*Figure 2. Output of the program using OpenCL on CPU.*



```
NOTE: The execution times include some printing to console.
Image manipulation is done using OpenCL (GPU).
Compute device info:
        Device name:                    Ellesmere
        Device type:                    GPU
        Vendor ID:                      4098
        Maximum frequency:              1340 MHz
        Driver version:                 2639.5
        Device C version:               OpenCL C 2.0
        Compute units:                  36
        Max. work item dimensions:      3
        Max. work item sizes:           1024/1024/1024
        Max. work group size:           256
Loading image... Done.
Decoding image... Done.
        => time: 1.058 s
Image 'img/im0.png', size 2988x2008.
Transforming image to grayscale... Done.
        => time: 103.930 ms
        => Kernel execution time: 0.263 ms
Encoding image... Done.
Saving image... Done.
        => time: 2.986 s
Filtering image using a mask... Done.
        => time: 104.328 ms
        => Kernel execution time: 1.110 ms
Encoding image... Done.
Saving image... Done.
        => time: 2.957 s
```

*Figure 3. Output of the program using OpenCL on GPU.*

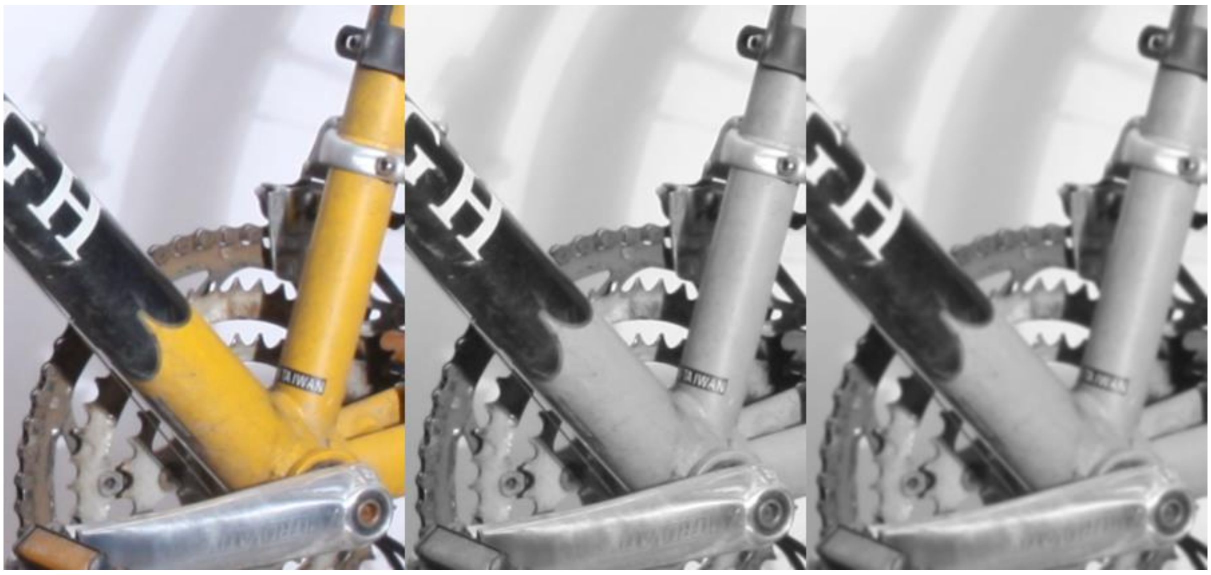


*Figure 4. A detail from the image. From left to right: original image, grayscale, filtered image (5x5 gaussian).*

Execution times:

| Operation | CPU (sequential) | CPU (OpenCL) | GPU (OpenCL) |
|---:|:---:|:---:|:---:|
| Load + decode | 1.068 s | 1.072 s | 1.064 s |
| Convert to grayscale | 425.390 ms | 79.090 ms | 0.264 ms |
| Filter | 2952 ms | 394.087 ms | 1.114 ms |
| Encode + save | 3.037 s | 2.968 s | 2.965 s |
| Total | 7.482 s | 4.513 s | 4.030 |

As the benchmark shows, the load and save times are pretty much the same, since it is sequential operation in all cases. However, in filtering and grayscale conversion, there is a significant performance difference when using OpenCL, especially GPU.

## Discussion

While the implementation is quite flexible and well-performing, we could improve performance in the image filtering slightly; since we are filtering grayscale images, all channels except alpha (RGB) have the same value. This would allow filtering a single channel as a simple vector, which would speed up the computing a bit. However, this implementation allows for more flexible operations, as it works well for non-grayscale images as well.

The OpenCL implementations use edge clamping (via `sampler_t`), but the sequential implementation does not, which results in a slight darkening on the very edge of the image.

## References

[1] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[2] https://computergraphics.stackexchange.com/a/41

[3] *The OpenCl Programming Guide* and *The OpenCL Specification 1.2*

## Reporting

| **Task** | **Hours** |
|---|---|
| C implementation | 15 h |
| OpenCL implementation | 20 h |
| **Total** | **35 h** |