Lab 4 report

Marc Aguilar and Alejandro Fernández

Exercise 1

The functions which apply the filters of both exercises have been implemented similarly. First of all we compute which is the position of the thread executing the filter by its `threadIdx` and `blockIdx`. Then, if the thread is inside the domain of the image we apply the corresponding filter.

In the main part of the code we first allocate space in both the CPU and the GPU. Then, we copy the intial image synchronously from the host to the device and we call the corresponding filters. Then, we perform a synchronous copy of the filtered images from the device to the host.

The execution of the code from the GPU allocation to the last memory transfer takes about **30 ms**, which is substantially better than the OpenAcc implementation.

Exercise 2

First of all, in this case in order to allocate the pointers of the GPU we used `cudaMallocHost()` since it is compulsory for a proper asynchronous operation (as explained in class). As before, we copy the original image to the GPU pointer and then we apply the filters in different streams (one for each filter). In this way we apply the filters in an asynchronous way. Notice, though, that the original image is copied synchronously, since all filters need it.

After that, we start the asyncrhonous memory transfers from the GPU to the CPU. Since we were applying the filters in different streams, each of the memory copies are done in its respective stream. Otherwise the image would be transfered without verifying whether the image has been completely filtered.

In this case the execution is quite faster and it only takes **5.93 ms**. It has to be said that at first, before the streams were explained in theory class, we did not use `cudaMallocHost()` and the code did not work as it should.