## The assignment

You are requested to implement an image processing pipeline: the pipeline takes a color image as its input, convert it to grayscale, use the image's histogram to contrast enhance the grayscale image and, eventually, returns a smoothed grayscale version of the input image. For simplicity and accura-cy all operations are done in floating point. The program must be benchmarked on the NVIDIA GTX480 GPUs on the DAS-4. A brief description of the four algorithms follows.

## Converting a color image to grayscale

Our input images are RGB images; this means that every color is rendered adding together the three components representing Red, Green and Blue. The gray value of a pixel is given by weighting this three values and then summing them together. The formula is:
gray = 0.3*R + 0.59*G + 0.11*B.

## Histogram Computation

The histogram measures how often a value of gray is used in an image. To compute the histogram, simply count the value of every pixel and increment the corresponding counter. There are 256 pos-sible values of gray.

## Contrast Enhancement

The computed histogram is used in this phase to determine which are the darkest and lightest gray values actually used in an image - i.e., the lowest (min) and highest (max) gray values that have "scored" in the histogram above a certain threshold. Thus, pixels whose values are lower than min are set to black, pixels whose values are higher than max are set to white, and pixels whose values are inside the interval are scaled.

## Smoothing

Smoothing is the process of removing noise from an image. To remove the noise, each point is re-placed by a weighted average of its neighbors. This way, small-scale structures are removed from the image. We are using a triangular smoothing algorithm, i.e. the maximum weight is for the point in the middle and decreases linearly moving from the center. As an example, a 5-point triangular smooth filter in one dimension will use the following weights: 1, 2, 3, 2, 1. In this assignment you will use a two-dimensional 5-point triangular smooth filter

## The challenge: accelerate the application!

A sequential version of the application is provided, for your convenience, in the directory “sequen-tial”. Please read the code carefully, and try to understand it. A template for the parallel version is also provided, in the “cuda” directory. We recommend that you to start from the template imple-mentation that is provided. You need to parallelize and offload the previously described algorithms to the GPU. The "Kernel" comment in the code indicates the part that must be parallelized.
There are no assumptions about the size of the input images, thus the code must be capable of run-ning with color images of any size. The output must match the sequential version; a compare utility is provided to test for this. The output that you should verify is the final output image, named smooth.bmp. You are free to also save the intermediate images, e.g. for debugging, but do not in-clude the time to write this images in the performance measurements.
In the directory “images”, 16 different images are provided for testing. You can measure

the total execution time of the application, the execution time of the four kernels and the (introduced) memory transfer overheads. This way, you can compute the speedup over the sequential implemen-tation, the achieved GFLOP/s and the utilization.

## Performance

Try to optimize (at least) the histogram computation and the smoothing filter (hint: use shared memory). As an indication, the execution times (in milliseconds) measured for the computation of the largest image in the set (gpu/images/image09.bmp), can be below the following thresholds:

- Grayscale Conversion < 5.5 ms
- Histogram < 68 ms
- Contrast Enhancement < 8.7 ms
- Smoothing < 84 ms
- Total execution < 1023 ms

## Compiling and Running Your Application

Please use the provided Makefiles for compiling.
Use the same procedure as for the other assignments to run the code.