Laboratory 5 - Report

Group Members:
Javier Almario, NIP: 962449
Alvaro Provencio, NIP: 960625

1 Kernel Implementation

This exercise have been tested in Berlin server, using 1 of the 2 available GPUs NVIDIA A10, with a maximum of 72 Compute Units, meaning it can process up to 72 work-groups, with 1024 work-items per work-group.

The program receives 2 arguments, a value for σ and an image number in order to test several images without modifying the code. With this approach we aim to test the kernel for different image sizes (more pixels mean more work-items) and with different gaussian masks, since the number of σ determines its dimension on 6 x σ + 1 (The bigger the mask the more FLOP operations per convolution).

We have decided to use the image data type for this exercise to learn a different way to interact with data in the kernel, even though it is possible to use directly a vector with all the pixel RGB values. This approach has a drawback and it is that image data type requires RGBA values which increases the memory on an additional component per pixel that in our case will not be used.

With this approach, we pass to the kernel a reference to the input and output images in global memory, the gaussian mask as a constant cache and the height and width dimensions as private memory.

Lastly, since each image has its own dimensions, which defines the global size of work-items, we have decided to set the local size as NULL in the clEnqueueNDRangeKernel call. This allows the OpenCL driver to automatically choose an appropriate work-group size for the device, which on NVIDIA GPUs could be 256, meaning work-groups of 16x16 or 8x32.

2 Impact of problem size on performance

2.1 Execution Time

Image	Size (pixels)	Program time (ms)	Kernel time (ms)
cat_275x183.jpg	50 325	445.62	0.01638
cat_250x334.jpg	83 500	482.45	0.01912
cat_600x600.jpg	360 000	470.68	0.05018
cat_760x570.jpg	433 200	474.36	0.05091
cat_1000x600.jpg	600 000	507.80	0.06929
galaxia_2976x3323.jpg	9 889 248	1509.63	0.94515
galaxia_3906x3906.jpg	15 256 836	2036.77	1.44930
galaxia_7910x6178.jpg	48 867 980	5533.60	10.04
galaxia_8736x5630.jpg	49 183 680	6065.23	9.56

Table 1: Program and kernel execution time as a function of image size for the Gaussian filter ($\sigma = 1$, matrix dimension 7×7).

Table 1 shows that, for all tested image sizes, the kernel execution time represents less than 0.2% of the total program time. For small images this fraction is around 0.004%, and for the largest images it grows only to about 0.16%. Therefore, the application is overhead-bound rather than compute-bound, the runtime is dominated by OpenCL initialization, memory management and image I/O, not by the Gaussian kernel itself.

The kernel behaves roughly linear in the number of pixels where more pixels mean more work to do, which represents monotonic increase in kernel time. This fits the Gaussian filter cost: one work-item per pixel, constant work per pixel.

2.2 Bandwidth

From the experiment we extracted the approximate bandwidth between the kernel and device global memory, which is showed on the table 2.

Size (pixels)	**H2D time (ms)**	**H2D BW (GB/s)**	**D2H time (ms)**	**D2H BW (GB/s)**[1]
50 325	0.02000	9.43	0.02186	8.59
83 500	0.03076	10.15	0.03424	9.09
360 000	0.13151	11.62	0.22963	6.65
433 200	0.13847	11.63	0.29678	5.22
600 000	0.19368	11.55	0.41402	5.52
9 889 248	3.22713	11.42	5.49885	6.71
15 256 836	5.18494	10.96	9.38215	6.06
48 867 980	16.64227	10.94	29.17787	6.24
49 183 680	16.05673	11.41	27.24213	6.73

Table 2: Host-to-device (H2D) and device-to-host (D2H) transfer times and bandwidth as a function of image size for the Gaussian filter ($\sigma = 1$, matrix dimension 7×7).

The results in Table 2 show how the host–device transfer behavior is mainly governed by image size. As the number of pixels increases, both H2D and D2H times grow approximately linearly, which is consistent with the fact that the amount of transferred data is proportional to the image resolution. The bandwidth clearly saturates around $\approx 11GB/s$ for host-to-device and $\approx 6GB/s$ for device-to-host. This highlights an asymmetry between upload and download: sending data to the GPU is significantly faster than reading it back. In practical terms, this suggests that for large problems the cost of transfers can be modeled with a simple constant time per pixel.

2.3 Throughput of the kernel

We obtained the throughput of the kernel, the results are showed in the figure 1, from the plot we observe the kernel throughput increases with image size and quickly reaches a plateau of about 3 TFLOP/s for medium and large images. This indicates that, in this range, the GPU is well utilized and the kernel achieves high computational efficiency. However, for the largest images (around 49M pixels), both pixels/s and GFLOP/s drop by roughly a factor of two, which suggests that performance is limited by memory-system effects when the working set becomes very large.

[1] H2D: host to device D2D: Device to host

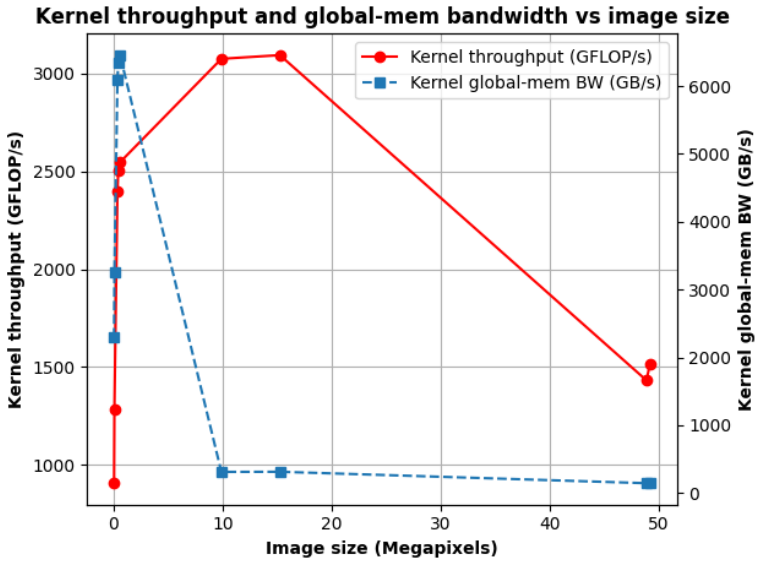

Figure 1: Kernel throughput and global-memory bandwidth as a function of image size for the Gaussian filter ($\sigma = 1$, matrix dimension 7×7).

2.4 Memory footprint of the program.

Size (pixels)	Host mem (MB)	Device mem (MB)	Kernel private mem / work-item (Bytes)
50 325	0.3841	0.3841	60
83 500	0.6372	0.6372	60
360 000	3.1252	3.1252	60
433 200	3.3052	3.3052	60
600 000	4.5778	4.5778	60
9 889 248	75.4492	75.4492	60
15 256 836	116.4010	116.4010	60
48 867 980	372.8330	372.8330	60
49 183 680	375.2420	375.2420	60

Table 3: Memory footprint on host and device, and kernel private memory per work-item, as a function of image size for the Gaussian filter ($\sigma = 1$, matrix dimension 7×7).

The table shows that both host and device memory footprints grow almost linearly with the number of pixels. For example, increasing the image size from about 5×10^4 to 4.9×10^7 pixels increases the device memory usage from roughly 0.4 MB to about 375 MB. The host and device footprints are essentially identical for each case, which is consistent with allocating one input and one output image buffer on both sides. The private memory used by the OpenCL kernel is constant (60 bytes per work-item), meaning that the kernel's per-thread state does not depend on the image resolution but on its private variables to process a pixel, that will be duplicated per each work-item; the global memory required by the image data dominates the overall memory usage of the program.

3 Experimental analysis

3.1 Local Size Variation

Local size	Work-items/group	Program time (ms)	Kernel time (ms)	Throughput (pixels/s)	Kernel throughput (GFLOP/s)	Kernel global-mem throughput (GFLOP/s)
8×8	64	1650.72	0.9370	1.0555×10^{10}	3103.09	314.56
16×8	128	1536.52	0.9404	1.0520×10^{10}	3092.98	313.53
16×16	256	1568.42	0.9192	1.0759×10^{10}	3163.10	320.64
32×8	256	1568.89	0.9400	1.0523×10^{10}	3093.80	313.61
32×32	1024	1556.16	1.0735	9.2252×10^{9}	2712.22	274.93

Table 4: Effect of local work-group size on kernel performance for the Gaussian filter (image galaxia_3906x3906, $\sigma = 1$, matrix dimension 7×7). Each row reports the mean over three executions.

Taking into consideration that the maximum work-group size is 1024, varying this size in lower values does not affect practically the GPU performance, as seen on the following table. It is worth mentioning that for the maximum work-group size, it is slightly slower computing the operations, since fewer work-groups can reside simultaneously on each compute unit, giving the scheduler less flexibility. This confirms that the maximum work-group size exposed by the device is only a hardware upper bound and not necessarily the optimal configuration. In our case, medium-sized work-groups provide better kernel performance than the largest allowed group size.

3.2 Sigma Variation

σ	Matrix dim	Kernel time (ms)	Throughput (pixels/s)	Kernel throughput (GFLOP/s)	Kernel global-mem throughput (GFLOP/s)
0.5	5	0.478208	2.06798×10^{10}	3101.97	616.31
1.0	7	0.935936	1.05662×10^{10}	3106.45	314.90
1.5	11	4.903100	2.01694×10^{9}	1464.30	60.11
2.0	13	5.598210	1.76650×10^{9}	1791.23	52.65
3.0	19	14.476300	6.83134×10^{8}	1479.67	20.36

Table 5: Median kernel performance as a function of σ for the Gaussian filter (image galaxia_2976x3323, automatic OpenCL local work-group size).

From the table 5 shows how increasing σ and as well the matrix size, it strongly impacts the kernel's cost per pixel. As σ grows from 0.5 (5×5) to 3.0 (19×19), the kernel time rises from about 0.48 ms to 14.48 ms, while the pixel throughput drops by almost two orders of magnitude, from $\sim 2.07 \times 10^{10}$ to $\sim 6.83 \times 10^{8}$ pixels/s. For the kernel throughput in GFLOP/s remains almost constant for small kernels but it becomes more irregular for larger kernels, suggesting that other factors (such as register pressure or reduced cache reuse) start to limit arithmetic efficiency. Finally we can see the estimated global-memory bandwidth decreases monotonically with σ, from about 616 GB/s to only 20 GB/s, because each pixel requires more computation and the GPU spends proportionally more time doing arithmetic than streaming data.