Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών Lab 3 Report

Δάιος Γρηγόριος - **AEM** 10334
Παπαδάκης Κωνσταντίνος Φώτιος - **AEM** 10371

January 9, 2026

Contents

1 Introduction

Lab3 improves on Lab2 by:

- Vectorizing the input/output decreasing the memory accesses
- Storing the input/output to different banks achieving parallelized memory access

2 Application changes

To be able to vectorize the pixels correctly we utilize 512 bit unsigned integers which contains 16 integers each.

$$\frac{512}{sizeof(int)} = \frac{512}{32} = 16$$

Compared to lab2's code, here we cannot simply add 3 buffers containing a row and its neighboring element since we are using another layer of abstraction among the list of: - chunks and - buffers.

This extra layer is the unsigned 512 bit integer which another entity that can contain our pixel.

The solution we came upp with involves creating a single buffer which contains *BUFFER_SIZE **+(WIDTH/VECTOR_SIZE)*2*** 512bit elements. This added quantity allows us to alway carry the furthermost element ,towards the front or rear end of the buffer, that we might need. These two elements are:

- the up element of the buffer's first element and
- the down element of the buffer's last element.

The real matrix's width divided by how many 32 bit integers are inside a 512 bit integer essentially tells us how many 512bit elements to the right or to the left our sought after up and down elements are.Adding two times that covers both edge cases.

Now that we have all the info that we need in our *C_local* buffer we can continue by loading it to our intermediate variables which are used to apply sharpening and clipping. Ultimately they are stored in the *C_filt* output pointer argument and the cycle continues until *C_filt* is fully populated.

3 New Pragmas

There are two new pragmas used on this lab:

```
#pragma HLS DATAFLOW
#pragma HLS stream variable = A_local depth = 64
```

Dataflow enables parallelization through pipelining but on a task level which lies in contrast with *PIPELINE* which works on an instruction level. On the other side stream is applied on an array, for example variable A from the code sample, and is used when that array is consumed or produced in a sequential manner. Then a FIFO loop is used instead of RAM achieving more efficient communication.

4 Data comparison

4.1 Kernels & Compute Units

Kernel Execution	lab2 64*64	lab3 64*64	lab3 64*64 4banks	lab3 128*128
Enqueues	1	0.013	1	1
Total Time (ms)	0.513	0.013	0.014	0.042
Min Time (ms)	0.513	0.013	0.014	0.042
Avg Time (ms)	0.513	0.013	0.014	0.042
Max Time (ms)	0.513	0.013	0.014	0.042

4.2 Kernel Data Transfers

Top Kernel Data Transfer	lab2 64*64	lab3 64*64	lab3 64*64 4banks	lab3 128*128
Number of Transfers	15079	297	297	1329
Avg Bytes per Transfer	8.000	115.000	115.000	110.000
Transfer Efficiency %	0.196	2.825	2.825	2.691
Total Data Transfer (MB)	0.121	0.034	0.034	0.146
Total Write (MB)	0.033	0.016	0.016	0.066
Total Read (MB)	0.088	0.018	0.018	0.081
Total Transfer Rate (MB/s)	872.740	12002.800	12985.400	12063.900

4.3 Host Data Transfer

Host Transfer	lab2 64*64	lab3 64*64	lab3 64*64 4banks	lab3 128*128
Number of READs	1	1	1	1
Number of WRITEs	2	2	2	2
READ Transfer Rate (MB/s)	0.761	0.794	0.750	3.118
WRITE Transfer Rate (MB/s)	1.185	1.211	1.327	5.253
READ Average Size (kB)	32.768	32.768	32.768	131.072
WRITE Average Size (kB)	40.960	40.960	40.960	163.840

4.4 Comments:

Using the vectorization method we achieved an acceleration of **3946.15%**. Thanks to the implementation's performance increase we are able to use 128×128 matrices while remaining significantly faster than the non-vectorized 64×64 version (still **1221%** acceleration). Writing and reading, of the kernel/host to and from global memory, is evidently a lot faster this time thanks to the more compact way the data is transmitted.

Applying further optimizations, the 4bank setup manages to quicken writing and reading on the kernel side (kernel $\rightleftarrows$ global memory) by ~1GB/s, thanks to the parallelization of memory accesses, although on the host side we notice faster writes and slower reads from and to global memory. The overall 0.001ms rise in execution time could be attributed to the emerging topology where different memory banks might exist in different SLR regions making meeting timing harder.

5 Screenshots 64*64 - 1 bank

One Bank Structure:

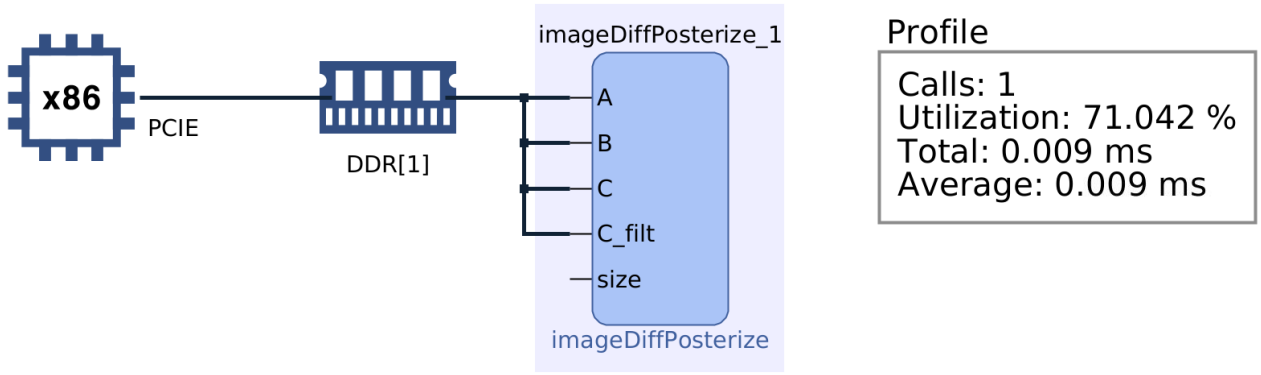

Figure 1: bank-structure-1b

Kernels & Compute Units

Kernel Execution (includes estimated device times)

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
imageDiffPosterize	1	0.013	0.013	0.013	0.013

Figure 2: kernel-compute-units-64*64-1b

Top Kernel Transfer

Compute Unit	Device	Number of Transfers	Avg Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
imageDiffPosterize_1	xilinx_u200_gen3x16_xdma_2_202110_1-0	297	115.000	2.825	0.034	0.016	0.018	12002.800

Figure 3: kernel-data-64*64-1b

Host Data Transfers

Host Transfer

Context:Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (KB)	Total Time (ms)	Avg Time (ms)
context0:1	READ	1	0.794	N/A	32.768	N/A	N/A
context0:1	WRITE	2	1.211	N/A	40.960	N/A	N/A

Figure 4: host-data-64*64-1b

6 Screenshots 64*64 - 4 banks

Four Banks Structure:

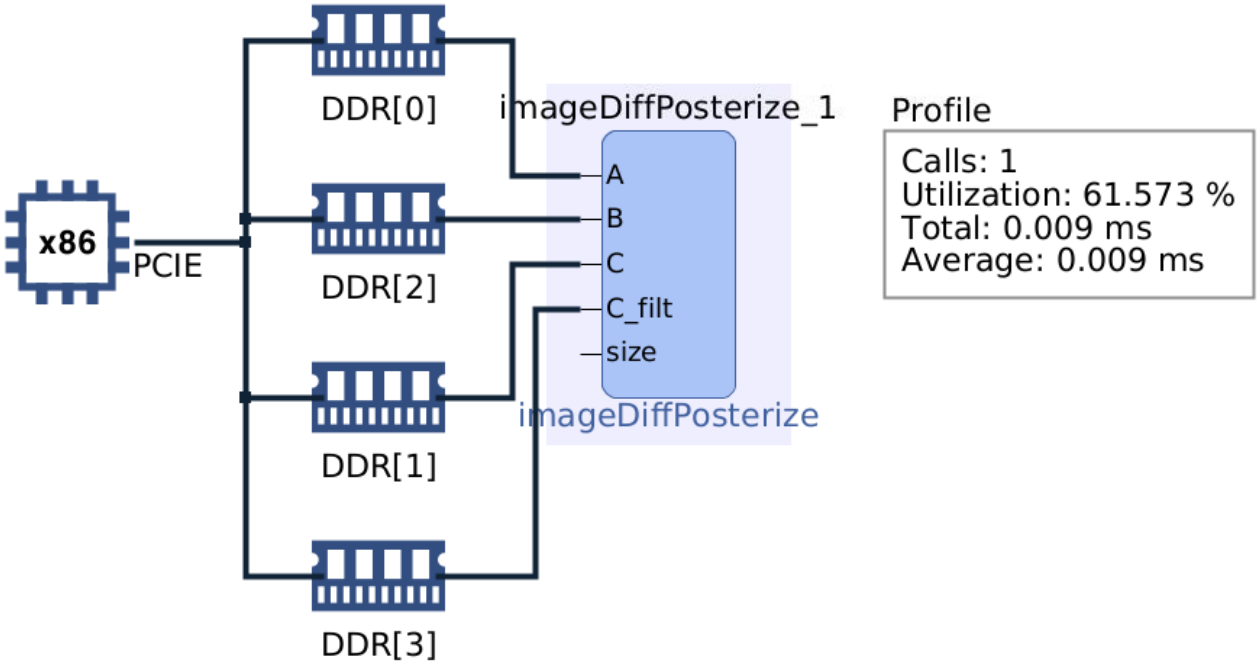

Figure 5: bank-structure-4b-64*64-4b

Kernels & Compute Units

Kernel Execution (includes estimated device times)

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
imageDiffPosterize	1	0.014	0.014	0.014	0.014

Figure 6: kernel-compute-units-64*64-4b

Top Kernel Transfer

Compute Unit	Device	Number of Transfers	Avg Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
imageDiffPosterize_1	xilinx_u200_gen3x16_xdma_2_202110_1-0	297	115.000	2.825	0.034	0.016	0.018	12985.400

Figure 7: kernel-data-64*64-4b

Host Data Transfers

Host Transfer

Context:Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (KB)	Total Time (ms)	Avg Time (ms)
context0:1	READ	1	0.750	N/A	32.768	N/A	N/A
context0:1	WRITE	2	1.327	N/A	40.960	N/A	N/A

Figure 8: host-data-64*64-4b

7 Screenshots 128*128 - 1 bank

Kernels & Compute Units

Kernel Execution (includes estimated device times)

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
imageDiffPosterize	1	0.042	0.042	0.042	0.042

Figure 9: kernel-compute-units-128*128-1b

Top Kernel Transfer

Compute Unit	Device	Number of Transfers	Avg Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
imageDiffPosterize_1	xilinx_u200_gen3x16_xdma_2_202110_1-0	1329	110.000	2.691	0.146	0.066	0.081	12063.900

Figure 10: kernel-data-128*128-1b

Host Data Transfers

Host Transfer

Context:Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (KB)	Total Time (ms)	Avg Time (ms)
context0:1	READ	1	3.118	N/A	131.072	N/A	N/A
context0:1	WRITE	2	5.253	N/A	163.840	N/A	N/A

Figure 11: host-data-128*128-1b

8 Zip Contents

- lab3.cpp
  - lab3's kernel.
- tb_lab3.cpp
  - The host which manages the lab3's kernel.
- lab3.pdf
  - This report