Exploiting Vitis Framework for Accelerating Sobel Algorithm

Panagiotis Mousouliotis
Dept. of Electrical and Computer Engineering
Aristotle University of Thessaloniki
Thessaloniki, Greece
p.mousouliotis@esda-lab.gr

Stavros Zogas
Dept. of Electrical and Computer Engineering
University of Peloponnese
Patra, Greece
s.zogas @esda-lab.gr

Panagiotis Christakos
Dept. of Electrical and Computer Engineer
University of Peloponnese
Patra, Greece
p.christakos @esda-lab.gr

Georgios Keramidas
School of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
gkeramidas@csd.auth.gr

Nikos Petrellis
Dept. of Electrical and Computer Engineering
University of Peloponnese
Patra, Greece
npetrellis@uop.gr

Christos Antonopoulos
Dept. of Electrical and Computer Engineering
University of Peloponnese
Patra, Greece
ch.antonop@esda-lab.gr

Nikolaos Voros
Dept. of Electrical and Computer Engineering
University of Peloponnese
Patra, Greece
voros@esda-lab.gr

***Abstract*— Edge detection is one of the most common operations needed in the image processing domain. In this work, alternative implementations of the Sobel algorithm are tested on a ZCU102 Xilinx embedded platform, demonstrating how different optimization techniques can be conveniently configured in Xilinx Vitis environment. We exploit (a) Xilinx Runtime library (XRT) that allows the reprogramming of the reconfigurable logic at real time and (b) the various high-level attributes offered by the OpenCL API for efficient resource allocation in the state-of-the-art Xilinx Ultrascale Multi-Processor System-on-Chips (MPSoC). Specifically, different implementations of the Sobel algorithm (varying the data transfer models and data packing modes) are demonstrated and analyzed. Our experimental results shows that starting from a CPU implementation with 656 ms latency, the frame processing time is reduced to a range between 17 ms and 22 ms depending on the allocated resources, leading to a solution that is up to 38 times faster.**

Keywords*— *Sobel algorithm, OpenCL, Image Processing, Pipeline, FPGA Acceleration, Artificial Intelligence, Xilinx Vitis, Xilinx HLS

I. INTRODUCTION

Image processing is one of the areas that scientists and companies continuously invest to find solutions and efficient implementations. There are many algorithms that have been implemented for feature extraction from digital images or video streams. Edge detection is a common image processing operation and quite remarkable of them are i) the Sobel algorithm due to its low complexity and ii) the Canny algorithm due to its increased sensitivity.

Sobel algorithm was presented for the first time in 1968 [1]. It is a quite common method for edge extraction from a digital image. Edges correspond to discontinuities of the image and can be used for the recognition of the shape of the displayed objects and patterns. Canny algorithm is a more complex algorithm than Sobel. Its functionality is based on frame-level statistics. Although it is more accurate than Sobel, it has a higher latency and is more computationally intensive.

Several approaches target hardware optimizations with different implementation techniques and algorithms. In [2], the authors implemented a hardware optimized architecture of Sobel Algorithm targeting the Xilinx ML510 FPGA platform by importing a pipeline logic and reducing the FPGA resources usage by 40%. In [3] and [4], a different approach is followed. In these papers, there various hardware implementations of Canny Edge Detection algorithm arE proposed targeting the Xilinx Virtex-5 FPGA platform. Focus is given on the reduction of the computation latency. On the other hand, the authors in [5] developed a Sobel Edge Detection algorithm on Xilinx Spartan-6 FPGA platform achieving better latency and higher operational frequency. In [6], a ZynqMP UltraScale MPSoC is used, focusing on the advantages of the xfOpenCV acceleration library to develop a faster Sobel Edge Detection method. In [7], the Vivado HLS v2015.3 tool is used and a comparison between software and hardware implementations of the Sobel Algorithm is presented. Furthermore, in [8] the authors implemented a frame-level Canny algorithm on Xilinx Virtex-5 FPGA platform reducing the resources and achieving better performance. Pujare et al [9] put effort into developing the Sobel algorithm based on Nexys 4 platform and they also used the Vivado v18.2 software tool. The results are displayed through VGA and MATLAB toolboxes that were employed to compare the results. The platform used in [10] is Xilinx Virtex 4 LX200. An interesting pipeline technique that has been employed reduces the hardware resources and the latency of the execution. Finally, the implementation of the Canny algorithm in processing video frames with the assistance of an Artix-7 Xilinx FPGA core is presented in [11].

In this paper, we employed the state-of-the-art Xilinx Vitis and XRT tools in order to describe various Sobel implementations in a more compact way compared to its predecessor (Xilinx SDSoC). The OpenCL API that is introduced in Xilinx Vitis allows fast experimentation with alternative implementations for efficiency and resource allocation comparison. Xilinx Vitis also supports higher portability between different hardware platforms. The developed Sobel edge detection filter was targeted for a Xilinx ZCU102 Field Programmable Gate Array (FPGA) platform. We combined three memory models (with different data copy requirements and data movement latency) with three data packing options (128, 256 and 512-bits) in order to take advantage of the FPGA port width. The overall time needed for processing a single 512×512 pixel frame was reduced from 656 ms needed by a software implementation to 17.52 ms (including the time needed for data transfer) when a 512-bit data-packing scheme is employed.

The paper is organized as follows: In Section II, there is a short introduction of the Sobel algorithm. Out implementation method is described in Section III while the experimental results are discussed in Section IV. To sum up, in Section V we present the conclusion of our experimentation and our future work.

II. Introduction to Sobel Algorithm

As noted, the Sobel filter is used in image processing and computer vision applications as an edge detection algorithm. It is based on an isotropic 3×3 Image Gradient Operator presented for the first time in 1968 and a detailed description can be found in [1]. A discrete differentiation is implemented as an approximation of the gradient of the image intensity. A convolution is applied in a window around at each pixel of the image. Thus, it is considered as a relatively low complex edge detection algorithm. The result of this operation can be the corresponding gradient vector or the norm of this vector. The Sobel algorithm is applied in a horizontal or vertical direction in the grayscale version of the initial RGB image *A*. The output G_x of the Sobel algorithm when applied horizontally can be expressed as:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +3 & 0 & -1 \end{bmatrix} * grayscale(A) \qquad (1)$$

where * is the convolution operator. The grayscale conversion followed in this paper is based on integer operations that are speed/memory efficient. The gray level p_g of a pixel p with initial color values p_R, p_G, p_B for red, green and blue, respectively is estimated as:

$$p_g = (19p_R + 37p_G + 7p_B)/64 \qquad (2)$$

The approximation of the gradient operation makes the Sobel algorithm less sensitive to high-frequency variations.

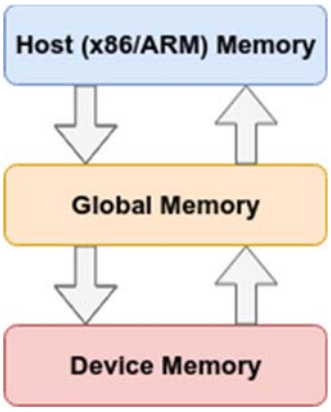


Fig. 1. The OpenCL Memory Model.

III. Sobel Algorithm Implementation in Vitis

The Xilinx Vitis tool [13] is the evolution of the Xilinx SDSoC [14] and SCAccel [15] tools. As its predecessors, the Vitis tool targets both edge and datacenter FPGA applications. Key innovations of the Vitis tools are the Xilinx Runtime library, called XRT, and the adoption of the OpenCL memory model. XRT provides the flexibility to reprogram the FPGA during the execution of an application both for the edge and the datacenter domains. This feature was not possible in the SDSoC tool, where for the edge domain the user was building an application which would use the FPGA programmed at boot time and with no flexibility to reprogram it as the application is executed. Another feature of the Vitis tool is the adoption of the OpenCL memory model hierarchy [4], where data is progressively moved from the Host (x86 or ARM) Memory to a Global Memory, accessible to both the host and the device, and finally from the Global Memory to the local Device Memory (DM). The Device Memory is typically the Block Random Access Memory (BRAM) of the FPGA. Fig. 1 provides a high level illustration of this hierarchy.

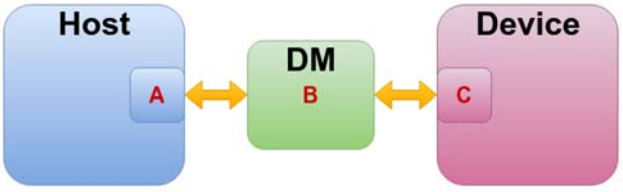


Fig. 2. The SDSoC Data Motion Network Architecture.

As opposed to the SDSoC tool where the user had the responsibility of constructing the Data Motion Network, the Vitis tool provides C++ based OpenCL functions for moving the data through the memory hierarchy as defined by the OpenCL API. This model leads to a more clear, simple, and standard way of describing FPGA applications. In the SDSoC case, the Data Motion Network, shown in Fig. 2, required low level actions as the following:

- Definition of the system port (indicated as “A” in Fig. 2); the host-side memory parts used in the data movement

- Definition of the data mover (the intermediate node between the host and the device; denoted as "B" in Fig. 2)
- Definition of the device interface for receiving/transmitting the data ("C" in Fig. 2)

TABLE I. COMPARISON OF THE DATA MOVEMENT PROCEDURE BETWEEN THE SDSOC AND THE VITIS TOOLS

	SDSoC	Vitis
Host	- Memory allocation and mapping using the sds_alloc() function. - Definition of the system port using the SDSoC pragmas.	- A constructor for memory allocation.
Intermediate	- Definition of the data mover using SDSoC pragmas. - Definition of the design data arguments using SDSoC pragmas. - Code for data packing.	- Functions for setting the design data arguments. - Functions for data migration (host to device and vice versa)
Device	- Definition of the design interface using HLS pragmas and design memories. - HLS code for data packing.	- Definition of the design interface using HLS pragmas and design memories.

Generally speaking and from the designer's perspective, the data motion network description used to model (as defined in SDSoC) the data movement between the host and the FPGA device was a quite complex and error prone procedure. It included the memory allocation in the host side, the memory mapping to physical memory addresses, the host memory port definition, the data mover definition, and finally, the FPGA design (device) memory interface.

On the contrary, in the Vitis tool, this complex procedure has been substituted by a few simple high-level function calls. Table I shows a comparison of the data movement approaches in the SDSoC and the Vitis tools. The SDSoC tool requires a mix of code and "pragma directives" in different positions of the application source code files, such as the main code files and in various header files. Moreover, special code is required for defining data packing, so as to take advantage of the full data movement port width. In the Vitis tool the whole procedure is simpler, clearer, and more standardized. Finally, some configuration issues, such as the data packing are inferred by the FPGA design and automatically realized without the need to explicitly determine all the implementation details.

Fig. 3. The three kernels used in the Sobel algorithm implementation.

Our Sobel design is implemented using three sequential kernels annotated in the Xilinx Vitis tool (Fig. 3). The first kernel in the Grayscale Converter (in the code it is named as grayconvert) converts an RGB image to a grayscale as described in Section II. This kernel supports the configuration of the data packing at the interface. In this way, the design of the grayconvert can reduce the required processing latency by consuming more image pixels and processing them concurrently.

The second kernel performs Edge Detection (imgscan), according to Sobel filtering by processing row-wise (see eq. (1)) the results of the first kernel. This kernel design is also parameterized regarding data packing, but since most of the clock cycles are used for the computation of the filtering, the data packing optimization has a negligible overhead. The third kernel used is a Boundary Detection and correction kernel, called borderscan in the design files. This kernel is not optimized at all since it consumes sparse data around the borders of the imgscan kernel's result image.

TABLE II. EXECUTION TIME OF THE GRAYCONVERT+IMGSCAN KERNELS AND TOTAL SOBEL EXECUTION TIME IN PARENTHESES

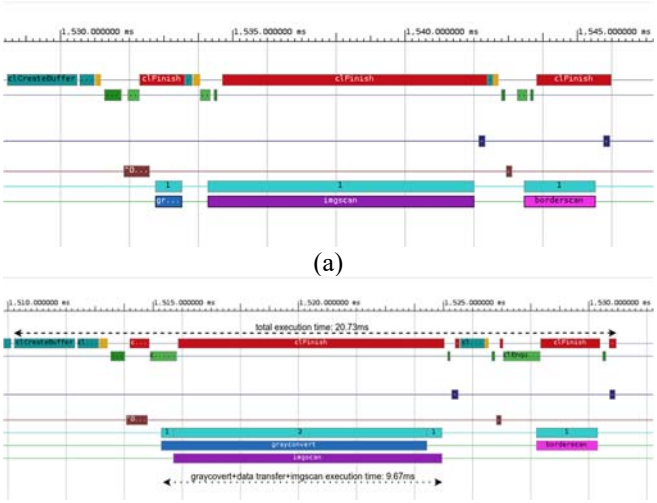
Transfer Mode	128-data-pack ms	256-data-pack ms	512-data-pack ms
All OpenCL stages	12.44 (21.71)	11.7 (20.42)	11.19 (20.34)
Global Mem. only	10.60 (18.84)	9.68 (18.07)	9.24 (17.52)
K2K stream only	10.01 (20.57)	9.78 (20.60)	9.67(20.73)

IV. EXPERIMENTAL RESULTS

A number of experiments has been conducted related to the degree of data packing as well as the method used for moving the data from the host to the device (kernel). We have defined three data moving methods:

- Use of all the OpenCL memory model stages for all data movements (host, global, device).
- Use of only global memory for sequential producer consumer kernels.
- Use of only kernel-to-kernel data streams for sequential producer consumer kernels.

The profile of the sequential producer-consumer kernels fits for the case of the grayconvert and the imgscan kernels which have been designed to produce and consume their data sequentially. Table II shows the results, in terms of latency, measured on the FPGA device for the execution of the sequence of these two kernels for the three aforementioned data movement methods and for different data packing sizes. In parenthesis, in Table II, is shown the total time required for the data transfer and execution of all three kernels required for the Sobel calculation. Although the K2K execution in Table II would be expected to achieve smaller latencies than the Global Memory only execution, this is not the case. We attribute this result to the overheads introduced by the asynchronous execution of the OpenCL command queue as it can be seen by the comparison of the Global Memory and the K2K timelines of Fig. 4.



(a)

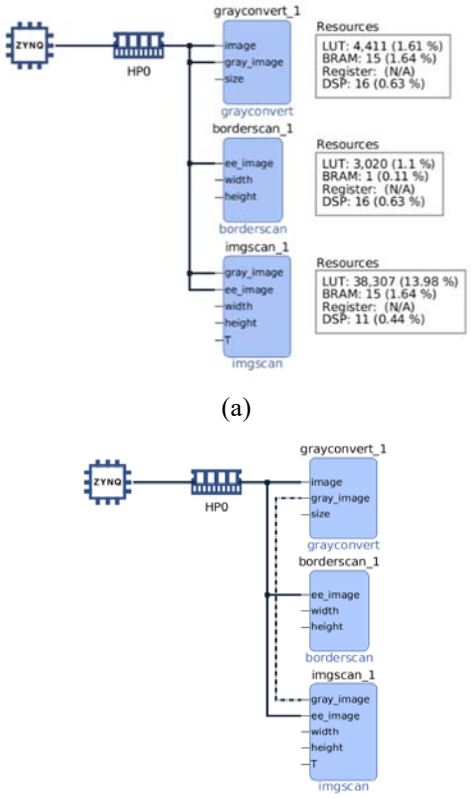
(b)

Fig. 4. Comparison of Global Memory only and K2K execution timelines. Global Memory only execution timeline - in-order OpenCL command queue execution in red color (a). K2K execution timeline - out-of-order OpenCL command queue execution in red color (b).

Regarding the synthesized system architecture and the resources used, Vitis Analyzer offers comprehensive reports. For example, in Fig. 5a the build report for all tested architectures is shown. The High Speed port is used to connect the processing system (Zynq) with the hardware kernels. The resources required by each kernel in terms of Look-Up Tables (LUT), BRAMs and Digital Signal Processing (DSP) engines is reported on the right of Fig. 5a. Run report for streaming interface appears in Fig. 5b. As can be seen from this figure, greyconvert and imgscan kernels are now directly interconnected (dashed line).

The full software implementation of the Sobel algorithm can be easily tested in the Vitis environment by simply removing grayconvert, imgscan and borderscan from the list of the hardware kernels. This reference software implementation required 656 ms to process a single 512×512 pixel frame on an Intel i7, 3.4GHz processor with 16GB RAM.

As can be seen from Table II, all the tested implementations required only 21.71 ms in the worst case (All-OpenCL-stages, 128-data-packing). The facilities offered by Xilinx Vitis allowed to compare several alternative implementations easily in order to further optimize the design. Although the results are not strictly comparable, Table III lists what has been compared in the reference approaches. A slower reference implemented either in software or in hardware is compared with an accelerated hardware implementation. In this work, we were able to rapidly test nine alternative implementations and find one that is 38 times faster than the referenced software one.



(a)

(b)

Fig. 5. Vitis Analyzer synthesis reports: (a) Kernel resources and interconnections in build reports, (b) Run-time report for streaming data transfer.

TABLE III. COMPARISON WITH REFERENCED APPROACHES

Ref.	Comparison	Proposed solution	Notes
[3]	5.97ms @ OpenCV 3.3GHz	0.721ms @ XC5VSX240	Canny algorithm
[4]	0.372ms .. 2.69ms @ various Xilinx/ Altera platforms	0.456ms @ Virtex-5	Canny algorithm
[5]	1.1ms @ 236MHz	0.52ms @ Sparta-6, 504MHz	Sobel algorithm
[6]	~8ms@PC, ~66ms@ OpenCV	~4.17ms @ xfOpenCV	Canny alg., Zynq7000
This work	656ms@Intel i7, 3.4GHz	9.24ms (kernels), 17.52ms(+transfer time)	Sobel, Global mem., 512 data pack.

V. CONCLUSIONS

In this work, the Sobel edge detection algorithm was used as a case study to demonstrate optimal hardware acceleration mechanisms that can be achieved in Xilinx Vitis/XRT environment. All the combinations between three data transfer models and three data packing options were tested reaching a hardware architecture that is 38 times faster than the referenced full software implementation.

Future work will focus on experimentation with additional hardware acceleration techniques. These techniques will be exploited in more complicated, computational intensive applications.

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 871738 - CPSoSaware: Cross-layer cognitive optimization tools & methods for the lifecycle support of dependable CPSoS.

REFERENCES

[1] I. Sobel and G. Feldman. "A 3x3 Isotropic Gradient Operator for Image Processing," Pattern Classification and Scene Analysis, 1973, pp. 271-272.

[2] S. Singh and A. K. Saini, R. Saini and A. S. Mandal, C. Shekhar and A. Vohra. " Area Optimized FPGA-Based Implementation of The Sobel Compass Edge Detector," ISRN Machine Vision, 2013, doi: 10.1155/2013/820216.

[3] Q. Xu, S. Varadarajan, C. Chakrabarti and L. J. Karam. "A Distributed Canny Edge Detector: Algorithm and FPGA Implementation," IEEE Transactions on Image Processing, July 2014, vol. 23, no. 7, pp. 2944-2960.

[4] D. Sangeetha and P. Deepa. "An Efficient Hardware Implementation of Canny Edge Detection Algorithm," in Proceedings of the 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID), Kolkata, 2016, pp. 457-462.

[5] N. Nausheen, A. Seal and P. Khanna and S. Haldar. "A FPGA based Implementation of Sobel Edge Detection," Microprocessors and Microsystems, 2017, doi: 56. 10.1016/j.micpro.2017.10.011

[6] R. Xie and X. Feng. "A Method of Quick Edge Detection Based on Zynq," in Proceedings of the IEEE 3rd International Conference on Cloud Computing and Internet of Things (CCIOT), Dalian, China, 2018, pp. 468-471.

[7] R. K. Megalingam, M. Karath, P. Prajitha and G. Pocklassery, "Computational Analysis between Software and Hardware Implementation of Sobel Edge Detection Algorithm," in Proceedings of the International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0529-0533.

[8] D. Sangeetha and P. Deepa. " FPGA implementation of cost-effective robust Canny edge detection algorithm," Journal of Real-Time Image Processing, 2019, doi: 16. 10.1007/s11554-016-0582-2

[9] A. Pujare, P. Sawant, H. Sharma, and K. Pichhode. "Hardware Implementation of Sobel Edge Detection Algorithm," ITM Web of Conferences, 2020, doi: 32. 03051. 10.1051/itmconf/20203203051.

[10] S. Abed. "Implementation of an Edge Detection Algorithm using FPGA Reconfigurable Hardware," Journal of Engg. Research, 2020, vol. 8, no. 1, pp. 179-197.

[11] L. Q. Tan and B. He. "The Research of Implementation Method of Canny Edge Detection of Video on FPGA," in Proceedings of the International Conference on Information Science, Parallel and Distributed Systems (ISPDS), Xi'an, China, 2020, pp. 1-4.

[12] M. Qasaimeh, K. Denolf, A. Khodamoradi, M. Blott, J. Lo, L. Halder, K. Vissers, J. Zambreno and P. Jones, "Benchmarking vision kernels and neural network inference accelerators on embedded platforms, " Journal of Systems Architecture, 2020, doi: 101896. 10.1016/j.sysarc.2020.10189

[13] K. Vinod. "Xilinx Vitis unified software platform," in Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020, pp. 173-174.

[14] K.Vinod, J. Hwang, W. Sun, Y. Chobe, T. Shui, and J. Carrillo. "SDSoC: A higher-level programming environment for Zynq SoC and Ultrascale+ MPSoC," in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2016, pp. 4-4.

[15] G. Giulia, E. Reggiani, L. Di Tucci, G. Durelli, M. Blott, and M. D. Santambrogio. "On how to improve fpga-based systems design productivity via sdaccel," in 2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW), 2016, pp. 247-252.

[16] M. Aaftab. "The opencl specification," in IEEE Hot Chips Symposium (HCS), 2009, pp. 1-3