A Heterogeneous Implementation of the Sobel Edge Detection Filter Using OpenCL

Theodora Sanida
Department of Electrical & Computer Engineering
University of Western Macedonia
Kozani, 50131, Greece
thsanida@uowm.gr

Argyrios Sideris
Department of Electrical & Computer Engineering
University of Western Macedonia
Kozani, 50131, Greece
asideris@uowm.gr

Minas Dasygenis
Department of Electrical & Computer Engineering
University of Western Macedonia
Kozani, 50131, Greece
mdasyg@ieee.org

***Abstract*—Today, edge detection is a cornerstone technique as edges are essential in many applications, such as image processing and biometric imaging. One popular algorithm for edge detection is the Sobel. Many researchers have focused on accelerating the Sobel filtering, but to the best of our knowledge we are the first to propose a 5×5 convolution kernel implementation using OpenCL. In this work, we implement the Sobel filter, one of the most effective and popular edge detection algorithms in image processing, in the OpenCL programming language. From the implementation of the Sobel algorithm we compare the performance of the CPU and GPU through OpenCL, in typical images ranging from 64×64 to 4096×4096 pixels. The Sobel operator uses a pair of 3×3 horizontal and vertical convolution kernels for edge detection functions. We apply 3×3 and 5×5 convolution kernels using OpenCL and compare them. The results have shown that for all image sizes, the GPU speed up ranges from 11,18 to 15,46 times with 3×3 convolution kernels, while speed up is from 10,05 to 13,46 times for the 5×5 convolution kernels. Finally, the results of our implementation are compared to other existing implementations and found to achieve better performance.**

***Index Terms*—Sobel Edge Detection, Sobel Filter, Sobel Operator, OpenCL, Image Processing, Convolution Kernel.**

I. INTRODUCTION

All the important information and all the key features contained in an image can be extracted from the edges. The edges are characterized by strong local fluctuations and define boundaries between regions. An edge is formed between the borders of two different regions in an image [1], [2].

Edge detection is the process of locating an edge in an image. This process results in a significant reduction in image size, while filtering out information that may be considered less relevant and retaining the important structural properties of an image. Thus, in the process of detecting the edges, any type of redundancy contained in the image is removed. In addition, this process helps in segmentation, data compression and object recognition [3], [4]. The main purpose of edge detection is to simplify the pixels of the boundaries of an image aiming minimization the amount of data to be processed. The Sobel Edge Detection Filter is one of the most effective edge detection algorithms shaving a relatively low arithmetic complexity and is used as a pre-processing step in several computer and machine vision algorithms [5].

In our research, we use the OpenCL 1.2 programming language for edge detection with the Sobel algorithm. OpenCL has the advantage of portability because it can be executed on many different architectures. We performed our experiments on a set of images with resolution ranging from 64×64 to 4096×4096 pixels. The Sobel filter was applied to gray scale images and we used 3×3 and 5×5 convolution kernels to evaluate the CPU performance of GPU devices.

The remainder of the paper is organized as follows: in Section II, we give an overview of works similar to ours. In Section III we discuss the Sobel edge detection operator. Section IV gives an outline of the procedure followed for accelerating the Sobel algorithm using OpenCL. In Section V, we present the results of our research. Finally, the conclusion of our research is outlined in Section VI.

II. RELATED WORK

In this section, we present research that is similar to ours. In [6], the authors propose a function in MatLab software to find Sobel edges using kernels whose dimensions are 5×5. The new function in MatLab uses a two-dimensional gray-scale image. They use a set of 8 images (256×256 pixels) with 3×3 and 5×5 convolution kernels. Experiments have shown that the larger the convolution kernel, the lower the image sensitivity to noise.

The authors in paper [7], develop a function with MatLab software for a 5×5 convolution kernel in the Sobel algorithm. They compare 5 images with 3×3 and 5×5 convolution kernels. Experiments have shown that with the 5×5 convolution kernels the Sobel operator is slower to compute but exhibits less noise sensitivity. They concluded, the larger the convolution kernel is, the lower the image sensitivity to noise is as well, while the Sobel operator exhibits higher output values for similar edges.

On [8] the authors compare the edge detection method with Canny and Sobel algorithms in MRI (Magnetic Resonance Imaging) images. In order to compare each slice of MRI images they tested both methods. The Sobel operator uses a 3×3 mask while the Canny operator uses an adjustable mask. Their results showed that the Canny edge detection was better than the Sobel detector result but with a muchhigher complexity.

In [9], the authors extended Sobel, Prewitt, and Kirsch edge operators from 3×3 kernels to 5×5 kernels in mammographic images. The results showed that 5×5 kernels in Sobel, Prewitt and Kirsch algorithms are better than 3×3 kernels. Amongst Sobel, Prewitt and Kirsch edge operators with 5×5 kernels, Sobel gives comparatively better results.

In contrast to these authors, we present the first heterogeneous implementation of Sobel Edge Detection algorithm with 3×3 and 5×5 convolution kernels using OpenCL. The proposed design is optimized for performance compared to existing parallel implementations.

III. SOBEL EDGE DETECTION OPERATOR

The Sobel Edge Detection algorithm separately detects horizontal and vertical boundaries in an image. The Sobel operator performs a 2-D spatial gradient measurement on images and usually uses a pair of 3×3 horizontal and vertical convolution kernels [5].

A pair with 3×3 convolution kernels is shown in Figure 1. The first kernel (Gx) calculates the slope in the x (columns) and the other (Gy) calculates the tilt in the y (rows) direction. Figure 2 shows the convolution using a 3×3 kernel. The Sobel Edge Detection convolution kernels are expanded to 5×5 dimensions [6], [7], [9]. A pair of 5×5 convolution kernels is shown in Figure 2.

-1	0	+1
-2	0	-2
-1	0	-1

Gx

+1	+2	+1
-2	0	+2
-1	0	+1

Gy

Fig. 1. Sobel masks with 3×3 dimensions.

Sobel operator based edge detection plays a significant role in a wide variety of image processing applications because it neutralizes effectively the noise sensitivity and marks also effectively the edges in an image. A more detailed description of the Sobel algorithm is described in [10], [11].

-5	-4	0	4	5
-8	-10	0	10	8
-10	-20	0	20	10
-8	-10	0	10	8
-5	-4	0	4	5

Gx

5	8	10	8	5
4	10	20	10	4
0	0	0	0	0
-4	-10	-20	-10	-4
-5	-8	-10	-8	-5

Gy

Fig. 2. Sobel masks with 5×5 dimensions.

IV. IMPLEMENTATION

In this section, we present our implementation of the Sobel algorithm using the OpenCL 1.2 programming language, in different image sizes.

We decided to use the OpenCL language, the de-facto progamming standard for heterogeneous implementations, which mean that the same code can be compiled and executed in a variety of architectures, from CPU up to custom FPGA designs. In our work, we applied the OpenCL 1.2 with the Sobel operator to 3×3 convolution kernels (as shown in Figure 1) and 5×5 (as shown in Figure 2) with resolution ranging from 64×64 to 4096×4096 pixels.

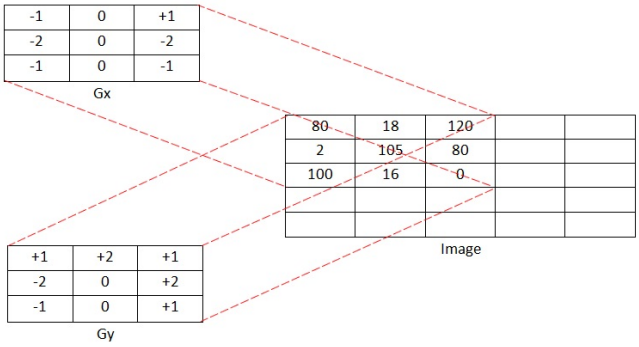

Fig. 3. Convolution using a 3×3 kernel.

First, CPU reads the original BMP color image. The original color BMP image is converted into a grayscale image. A pixel color in an image is a combination of three colors Red, Green, and Blue (RGB). The Equation (1), is used to convert a pixel to gray scale. The R, G, and B symbols correspond to the channel name and $fvalue$ contains the final pixel value. Converting the original BMP color image to a gray scale image is not included in our measurements.

$$fvalue = (0.21R + 0.72G + 0.07B) \quad (1)$$

The parameters of the Sobel algorithm are the BMP gray scale image and the convolution kernel to be executed (3×3 or 5×5). Then the platform and the device (CPU or GPU) are selected, the context and command queue for the device are created. Here is the definition of the objects that are transferred from host CPU to OpenCL device global memory.

The objects that are transferred to the GPU are the input grayscale image, image rows, image columns, output grayscale image and others memory buffers. A convolution kernel describes how each pixel in an image is influenced by its neighbors. Figure 3 shows the convolution using a 3×3 kernel. In the kernel the input image is read-only and the output image is write-only.

We define two dimensions: get_global_id, the first id corresponds to the current focused pixel and the second to the output pixel position. We read the values of 8 pixels around the current pixel in the neighborhood of 3×3 or of 24 pixels around the current pixel in the neighborhood of 5×5 and compute the slope x (columns) and the tilt in the y (rows). The output value is written in exactly the same position in the output image. The kernel that uses constant memory is executed so that all the kernel interfaces are stored in the cache during execution.

Constant memory of OpenCL contains data that is immutable during the kernel.

A gray scale image has only one channel and this channel represents the intensity of whites. Therefore, for a gray scale image, the Sobel algorithm will be executed once. Finally, when the kernel completes its computation, the output data is stored in global memory and transferred back to host CPU. Figure 4 shows the implementation architecture of the Sobel filter with OpenCL.

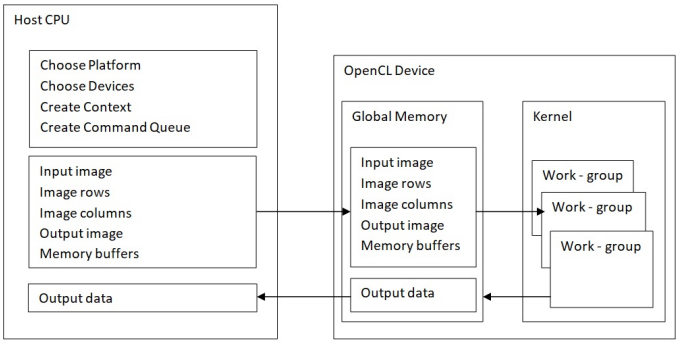

Fig. 4. Implementation architecture of the Sobel filter.

V. EXPERIMENTAL RESULTS

In this section, we present and compare the Sobel algorithm on GPUs and CPUs through OpenCL 1.2, in different image sizes.

Experiments were performed on gray scale BMP images with resolution: 64×64, 128×128, 256×256, 512×512, 1024×1024, 2048×2048 and 4096×4096 pixels. We used the OpenCL1.2 programming language. OpenCL has the advantage of portability. The implementation code was written in C programming language.

A. Evaluation Environment

We used Visual Studio 2019 for the software implementation. In these experiments we utilized the Intel Core i7 8750H processor and the Nvidia GeForce GTX 1060 graphics card. The detailed technical specifications of the hardware and the software used in this document are reported in Table I.

TABLE I
HARDWARE AND SOFTWARE TECHNICAL SPECIFICATIONS

Component	Description
Hardware	Processor: Intel Core i7 8750H (2.20GHz), RAM Memory: 16GB DDR4 - 2666MHZ, Graphics Board: Nvidia GeForce GTX 1060, 6144MB
Software	Windows 10 Professional 64-bit, Visual Studio 2019
Drivers	Nvidia CUDA toolkit version: 10.1

B. Evaluation result

Time measurement was performed using the time.h library and clock_gettime() function was used. Time measurement includes the image loading to GPU memory, kernel creation time and kernel execution time. Time measurement does not include program initializations, image input and output image. The transformation of the color space was not included in our measurements similar with other researchers. Each time measurement was performed with the execution of 20 images per size and their average execution time was recorded. Experiments show that the performance of the GPU, for all image sizes, is improved in both 3×3 and 5×5 convolution kernels.

In figure 5 the first image is the original, the second image shows the resulting image with a 3×3 convolution kernels and the third image shows the resulting image with a 5×5 convolution kernels. From the figure we observe that the 5×5 convolution kernels provide much greater accuracy in edge detection and less noise sensitivity.

Fig. 5. Resulting images with 3×3 and 5×5 convolution kernels.

Table II and Table III show the results of our measurements in milliseconds (ms), with 3×3 and 5×5 convolution kernels at different image sizes. On the device OpenCL–GPU speed up with 3×3 convolution kernels ranges from 11,18 to 15,46 times while 5×5 convolution kernels range is from 10,05 to 13,46 times. Figure 6 shows that increasing the image resolution, decreases the speedup which is justifiable by the fact that increasing the image size, increases the ammount of data to be transfered from the CPU to the GPU, and thus the problem transforms from a computation intensive algorithm to a memory access intenstive algorithm. The memory bus incurs a bottleneck penalty. We have noticed it and are working on techniques to alleviate this.

TABLE II
RESULTS WITH 3×3 CONVOLUTION KERNELS AT DIFFERENT IMAGE SIZES

Image resolution (px)	OpenCL-CPU (ms)	OpenCL-GPU (ms)	Speed Up OpenCL-GPU (times)
64×64	1,469	0,095	15,46
128×128	1,754	0,114	15,39
256×256	2,470	0,162	15,25
512×512	2,611	0,172	15,18
1024×1024	7,988	0,601	13,29
2048×2048	11,373	0,926	12,28
4096×4096	12,472	1,116	11,18

Finally, we compare the Sobel algorithm with 3×3 convolution kernels on GPUs through OpenCL 1.2 with similar implementations of other researchers we found in the literature. Table IV shows the execution time in milliseconds (ms). In the conducted experiments, the image 2048×2048 has

TABLE III
RESULTS WITH 5×5 CONVOLUTION KERNELS AT DIFFERENT IMAGE SIZES

Image resolution (px)	OpenCL-CPU (ms)	OpenCL-GPU (ms)	Speed Up OpenCL-GPU (times)
64×64	1,633	0,121	13,46
128×128	1,892	0,145	13,03
256×256	2,682	0,209	12,81
512×512	3,016	0,239	12,62
1024×1024	9,987	0,837	11,93
2048×2048	12,673	1,174	10,79
4096×4096	13,572	1,351	10,05

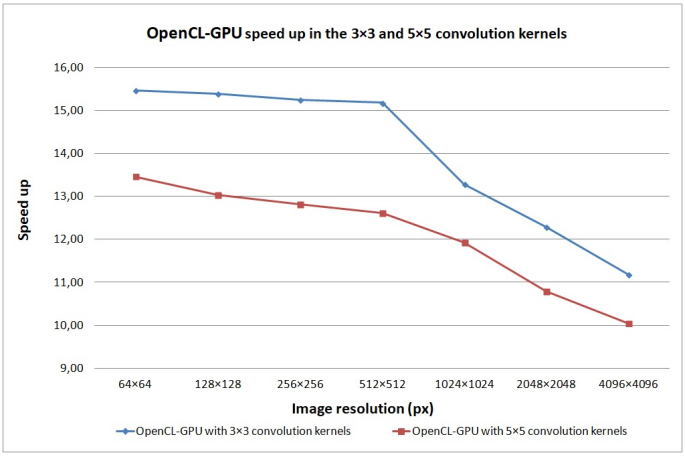

Fig. 6. OpenCL–GPU speed up in the 3×3 and 5×5 convolution kernels.

the best performance. The comparison of results shows that the proposed implementation of the Sobel operator with 3×3 convolution kernels at different image sizes on GPU using the OpenCL 1.2, improves the speed compared to the Cuda on GPU. In terms of similar parallel implementations, we are showing better acceleration from 11,64% to 27,00% and have the advantage of independent architecture using the OpenCL. This was achieved because we significantly improved the performance of the kernel using constant memory and 16×1 pixel/work-item.

VI. CONCLUSIONS

In this work, we accelerated the Sobel filter with the OpenCL 1.2 programming language. We used a set of images with different resolution: 64×64, 128×128, 256×256, 512×512, 1024×1024, 2048×2048 and 4096×4096 pixels. The Sobel filter was applied to gray scale images and we used 3×3 and 5×5 convolution kernels.

The results have shown that for all image sizes, the GPU speed up is greater for both 3×3 and 5×5 convolution kernels. The Sobel operator with a 5×5 convolution kernels offers much better edge detection accuracy in the image than the 3×3 convolution kernels. In addition, it is observed that with 5×5 convolution kernels there is less sensitivity to the noise of the image. The GPU shows speed up ranges from 11,18 to 15,46 times with 3×3 convolution kernels while range is from 10,05 to 13,46 times with the 5×5 convolution kernels.

TABLE IV
THE COMPARISON OF OUR IMPLEMENTATION WITH OTHER STATE OF THE ART, REVEALS THAT OUR'S NOT ONLY CAN BE EXECUTED ON A VARIETY OF ARCHITECTURES DUE TO THE OPENCL LANGAGE, BUT IT HAS ALSO A LOWER COMPUTATIONAL TIME.

Image resolution (px)	Our work OpenCL GPU (ms)	Cuda [12]	Cuda [13]	Cuda [14]
128×128	0,114			0,150
256×256	0,162			0,190
512×512	0,172			0,200
1024×1024	0,601			0,700
2048×2048	0,926	1,210	1,472	2,500

We are the first to propose a heterogeneous implementation of Sobel Edge Detection filter with 5×5 convolution kernels, using OpenCL. In our future research we will optimize further this algorithm, using pipeline and better data reuse, and we will evaluate the algorithm in FPGA architectures. Finally, the proposed implementation of the Sobel filter using OpenCL, improves the speed compared with other similar parallel implementations.

REFERENCES

[1] Juneja, M., & Sandhu, P. S. (2009). Performance evaluation of edge detection techniques for images in spatial domain. International journal of computer theory and Engineering, 1(5), 614.

[2] Perona, P., & Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on pattern analysis and machine intelligence, 12(7), 629-639.

[3] Fisher, Y. (2012). Fractal image compression: theory and application. Springer Science & Business Media.

[4] Acharjya, P. P., Das, R., & Ghoshal, D. (2012). Study and comparison of different edge detectors for image segmentation. Global Journal of Computer Science and Technology.

[5] Sobel, I. (1990). An Isotropic 3× 3 Gradient Operator, Machine Vision for Three–Dimensional Scenes. Freeman, H., Academic Pres, NY, 376379.

[6] Aybar, E. (2006). Sobel edge detection method for matlab. Anadolu University, Porsuk Vocational School, 26410.

[7] Gupta, S., & Mazumdar, S. G. (2013). Sobel edge detection algorithm. International journal of computer science and management Research, 2(2), 1578-1583.

[8] Othman, Z., Haron, H., Kadir, M. R. A., & Rafiq, M. (2009). Comparison of canny and Sobel edge detection in mri images. Computer Science, Biomechanics & Tissue Engineering Group, and Information System, 133-136.

[9] Kekre, H. B., & Gharge, S. M. (2010). Image segmentation using extended edge operator for mammographic images. International journal on computer science and Engineering, 2(4), 1086-1091.

[10] Vincent, O. R., & Folorunso, O. (2009, June). A descriptive algorithm for sobel image edge detection. In Proceedings of Informing Science & IT Education Conference (InSITE) (Vol. 40, pp. 97-107). California:Informing Science Institute.

[11] Jin-Yu, Z., Yan, C., & Xian-Xiang, H. (2009, April). Edge detection of images based on improved Sobel operator and genetic algorithms. In 2009 International Conference on Image Analysis and Signal Processing (pp. 31-35). IEEE.

[12] Nugteren, C., Corporaal, H., & Mesman, B. (2011, July). Skeleton-based automatic parallelization of image processing algorithms for GPUs. In 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (pp. 25-32). IEEE.

[13] Wu, J., Song, Z., & Jeon, G. (2014). GPU-parallel implementation of the edge-directed adaptive intra-field deinterlacing method. Journal of Display Technology, 10(9), 746-753.

[14] Zhang, N., Chen, Y. S.,& Wang, J. L. (2010, March). Image parallel processing based on GPU. In 2010 2nd International Conference on Advanced Computer Control (Vol. 3, pp. 367-370). IEEE.