Chapter

Performance Analysis of OpenCL and CUDA Programming Models for the High Efficiency Video Coding

Randa Khemiri, Soulef Bouaafia, Asma Bahba, Maha Nasr and Fatma Ezahra Sayadi

Abstract

In Motion estimation (ME), the block matching algorithms have a great potential of parallelism. This process of the best match is performed by computing the similarity for each block position inside the search area, using a similarity metric, such as Sum of Absolute Differences (SAD). It is used in the various steps of motion estimation algorithms. Moreover, it can be parallelized using Graphics Processing Unit (GPU) since the computation algorithm of each block pixels is similar, thus offering better results. In this work a fixed OpenCL code was performed firstly on several architectures as CPU and GPU, secondly a parallel GPU-implementation was proposed with CUDA and OpenCL for the SAD process using block of sizes from 4x4 to 64x64. A comparative study established between execution time on GPU on the same video sequence. The experimental results indicated that GPU OpenCL execution time was better than that of CUDA times with performance ratio that reached the double.

Keywords: HEVC, ME, SAD, GPU, CUDA, OpenCL

1. Introduction

The Graphics Processing Unit (GPU) [1] is a microprocessor present on graphic cards or game consoles. It has a strong parallel framework initially dedicated to accelerating graphics tasks. Having this innovation and programming language General Purpose computation on GPUs (GPGPU) languages such as Compute Unified Device Architecture (CUDA) [2] and Open Computing Language (OpenCL) [3] enabled applications development in many domains.

CUDA is an NVIDIA Corporation programming model that runs only on NVIDIA GPUs. The OpenCL method, an effort of the Khronos Community, is very close to the CUDA method. However, this is a requirement open for parallel programming on various platforms: CPUs, GPUs, Digital Signal Processors (DSPs) and other types of processors. Taking into account that, OpenCL is able to manage several devices. The concept of context makes it possible to deal with this problem. A context designates a set of devices.

However, there are two major differences. The first difference is that OpenCL codes are much larger than CUDA C codes. The multiplatform side of OpenCL explains this. The second difference is that the kernel is built from the host code during runtime using the OpenCL runtime library [4].The OpenCL kernel can be used in two ways, expressly defining the working group's local and global size and the local size or indirectly leaving OpenCL to select its global size of working group. The size of a working group equals a CUDA thread size block, the size of a working group is also known as ND Range configuration, as seen in **Figure 1**.

The two languages provide similar hierarchical decomposition of the computation index space explained on **Table 1**. The synchronization is available on thread block/ work-group level only.

This paper proposed an implementation of the Sum of Absolute Differences (SAD) of the High Efficiency Video Coding (HEVC) Motion Estimation (ME) algorithm on an NVIDIA GPU using CUDA and OpenCL languages to compare their performances.

This manuscript is structured as follows: Section 2 introduces the HEVC SAD algorithm. In Section 3, an overview of ME is given. Section 4 gives and describes the SAD kernel proposed. In Section 4 the experimental results and the discussion are given. Finally, Section 5 concludes this paper.

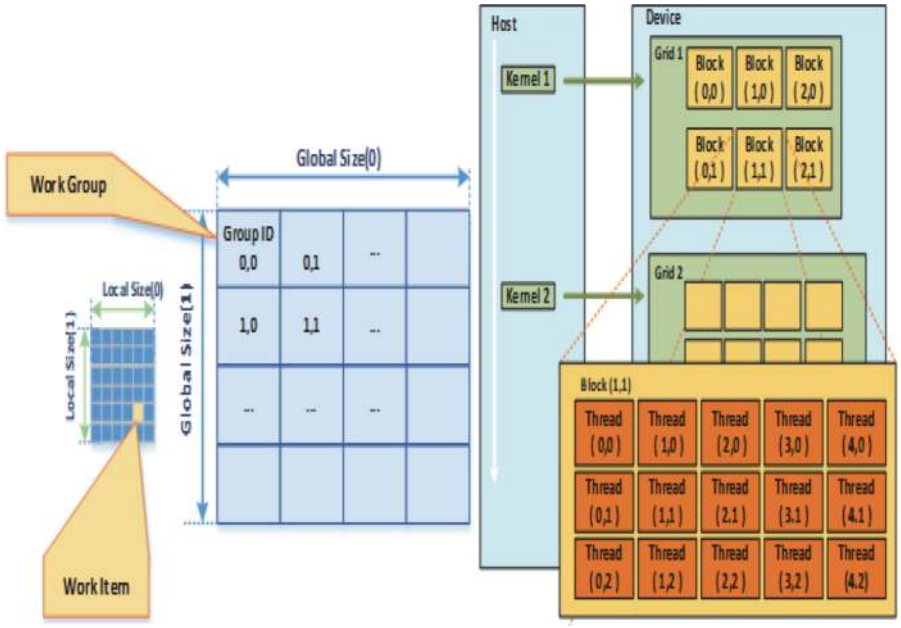

Figure 1.
Model of software programming.

CUDA	OpenCL
Grid	NDRange
Thread Block	Work group
Thread	Work item
Thread ID	Global ID
Block index	Block ID

Table 1.
Execution model terminology mapping.

2. HEVC ME feature

The key element of HEVC is the ME, which represent the most time-consuming task in video coding. Actually, the complexity of ME increases significantly due to the increase in the coding block size [5]. Inter-prediction requires a great complexity burden of up to 80% [6] in the total encoding process, due to the ME, which consumes around 70% of the inter-prediction time, as mentioned **Figure 2** [6].

ME is performed on a block-by-block basis and supports variable block sizes in HEVC. This coding tree unit (CTU) structure, which offers a compromise between a good quality and a less bit-rate, is based on three new concepts: coding unit (CU), prediction unit (PU), and transform unit (TU) [7, 8].

Each picture is divided into CTU of size 64 × 64 pixels, which can be partitioned after that into 4 CUs [9] sized from 8 × 8 to 64 × 64 pixels. These regions of CU contain one or several PUs and TUs.

In the HEVC ME algorithm, SAD and SSD are the most requested functions. These several cost functions are used to decide the best coding mode and its associated parameters. An idea of the SAD is given in the next subsection.

3. HEVC SAD algorithm

The calculation of the Sum of Absolute Difference (SAD) is commonly used for motion estimation in video coding. This is usually the computational intensive part of video processing [10, 11]. It computes the difference between the pixel intensity of the current and reference frame macro block. The motion compensation block size is N × N, where, $Current_{i,j}$, and $Reference_{i,j}$are current and reference frame block [12].

$$SAD = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|Current_{i,j} - Reference_{i,j}\right| \tag{1}$$

SAD is also used as an error calculation in order to define the similar block and to evaluate the motion vector in the motion estimation phase [13]. SAD is a simple and fast evaluation metric. This calculation takes every pixel in a block into an account. For many motion estimation algorithms, it is therefore very efficient (**Figure 3**).

4. Proposed SAD kernel

The calculation of the SAD can be parallelized using GUP since it treats each pixel separately, which corresponds to the architecture of the graphics processors

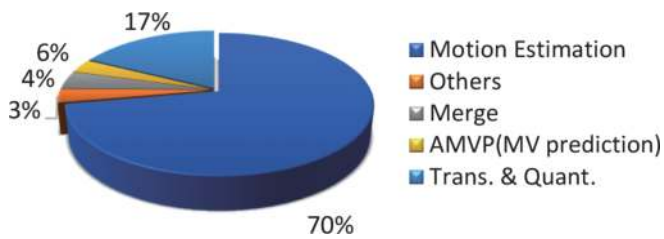

Figure 2.
HEVC inter-prediction time distribution [6].

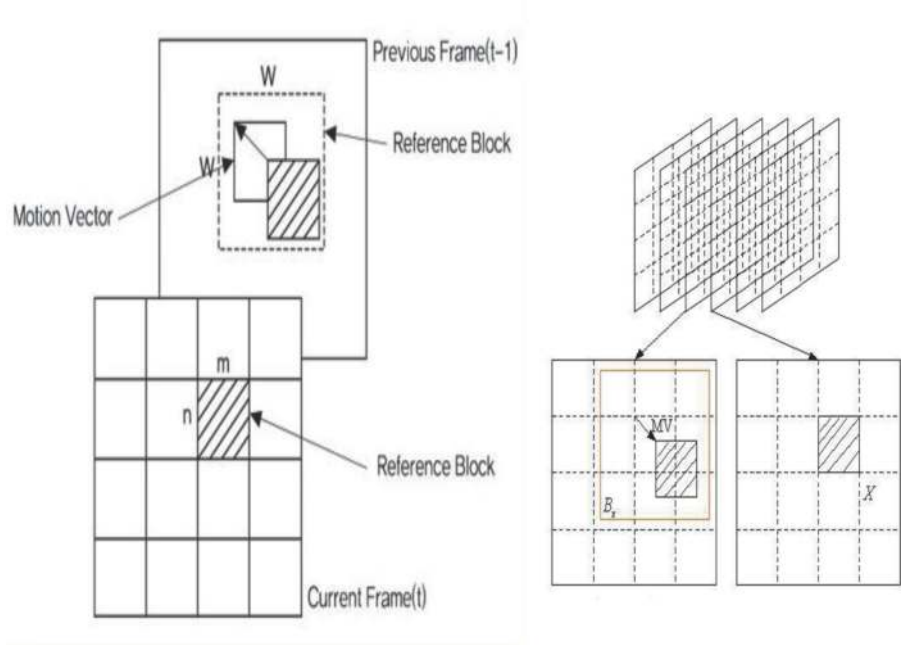

Figure 3.
Block matching algorithm based on SAD.

2D-grid of threads blocks which computes all disparities for 2D blocks of the image. Each thread computes the SAD value for a block in the search range, and a thread block calculates the entire SAD value for an image block. The benefit is that all SADs are calculated in the same thread block for an image block.

In [14] the authors implemented the SAD on the general purpose GPU architecture. A significant acceleration of 204x for an image size of 1024 × 768 was obtained for SAD on the GeeForce GTX 280 compared to the serial implementation as shown in **Figure 4**.

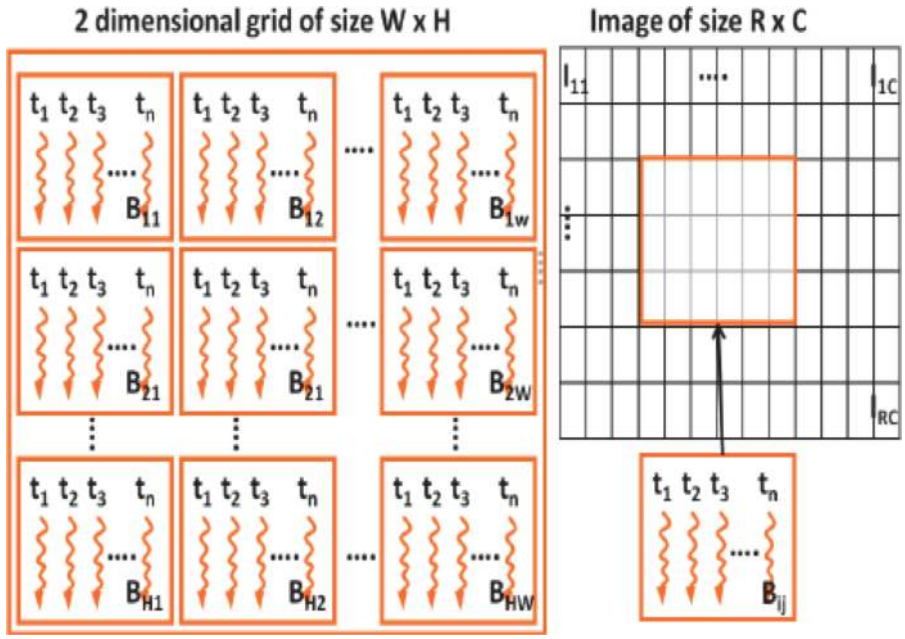

Figure 4.
Typical mapping of a block-matching algorithm to a GPU.

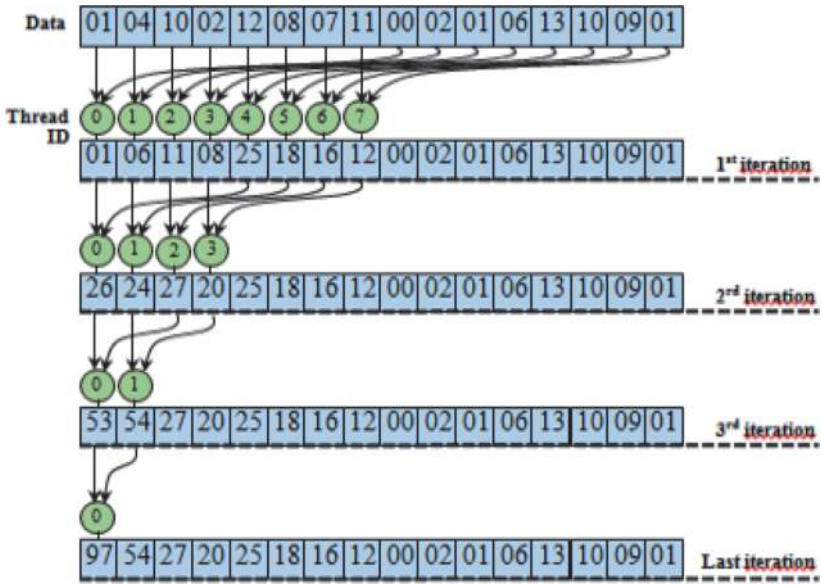

Figure 5.
Reduction technique.

The SAD kernel is composed of two main steps. The subtraction of the PU pixels then the summation. The addition was achieved on the GPU with the parallel reduction. In step1, the first N/2 elements are added to the other N/2. In the result, in the step 2, we have N/2 elements to add up. The first half was added to the next half. The same steps are repeated until there is only one number remaining as shown in **Figure 5** [15].

5. Experimental results

5.1 OpenCL performance on GPU compared the CPU one

OpenCL offers a convenient way to construct heterogeneous computing systems and opportunities to improve parallel application performance. As first step, the OpenCL SAD kernel was implemented in two platforms: CPU with 4 cores at frequency 2.5 GHz and an NVDIA GPU 920 m of 954 MHz as frequency. The SAD block dimensions are from 4×8 to 64×64 pixels. A comparative analysis was made on the same video between the CPU and GPU is seen in **Figure 6**. It is clear from the next figure that the GPU execution time is greater than CPU execution (**Figure 7**) [16].

When using the Eq. (2), **Figure 6** indicates the speed up [17] of the both implementations.

$$Speed - up = CPU\ execution\ time / GPU\ execution\ time \tag{2}$$

The speed up shows that the GPU platform is more efficient than the CPU platform, and this is due to the efficient parallel architecture of GPU compared to CPU. To validate the OpenCL code compared to the CUDA code the next study is proposed.

5.2 Execution performance OpenCL GPU compared to CUDA GPU

Running the application through GPU requires these steps as it is shown in **Figure 8**. For OpenCL, approach contains GPU detection and kernel compilation.

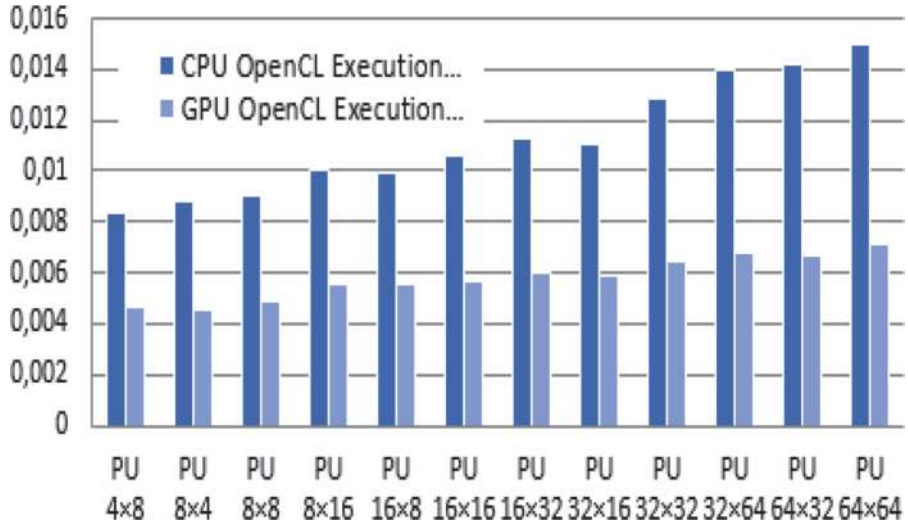

Figure 6.
Performance OpenCL comparison with GPU and CPU platforms.

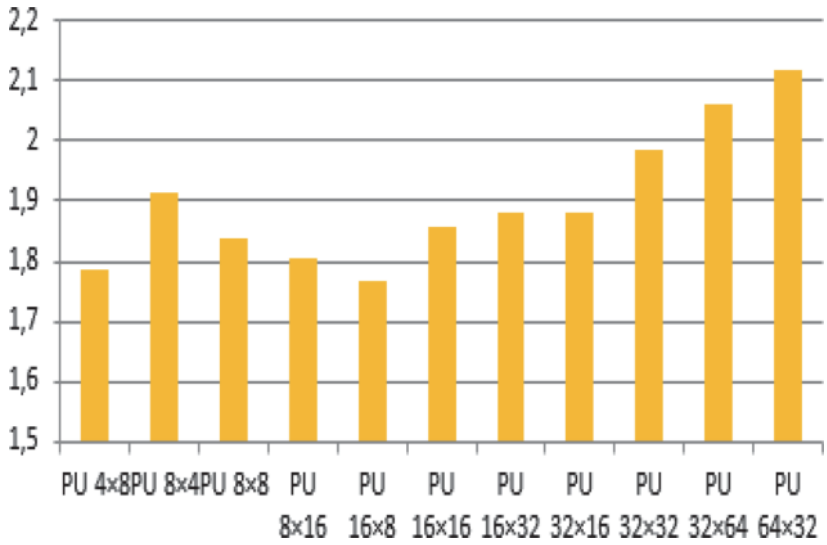

Figure 7.
Speed-up using OpenCL language.

The CPU input is read from the host to the device by all frameworks; the kernel is executed on the GPU; the device is returned to the host by copy data. Finally, the results are displayed on CPU.

Table 2 reports the kernel running time for different size of Prediction Unit (PU) (designed the block size used). In order to get repeated average times, we fixed each problem 10 times for both CUDA and OpenCL.

We use a normalized performance metric, called Performance (PR), to compare the performance of CUDA and OpenCL (**Figure 9**).

$$PR = CUDA \text{ execution time}/OpenCL \text{ execution time} \quad (3)$$

If performance ratio is greater than 1, OpenCL will give a better results compared to CUDA language. As shown in **Figure 8**, the performance ratio indicates that the OpenCL kernel running time is better than CUDA kernel running for each

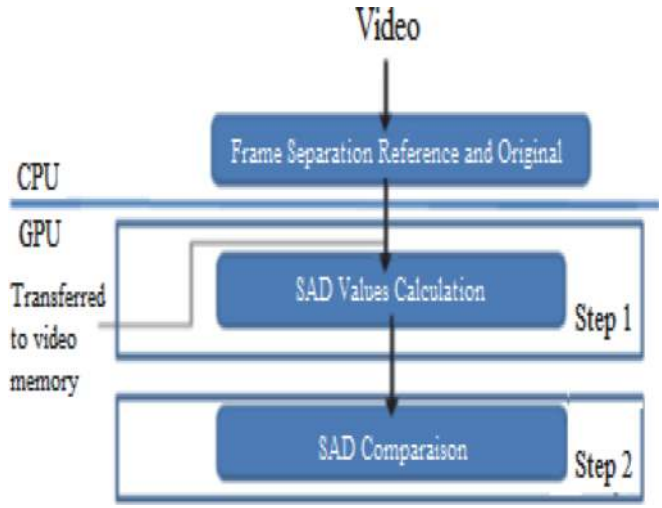

Figure 8.
Algorithm flow.

Block sizes		GPU execution time (µs)	
		CUDA language	OpenCL language
CU 8 × 8	PU 4 × 8	6.63	5.592
	PU 8 × 4	6.885	5.482
	PU 8 × 8	7.573	5.831
CU 16 × 16	PU 8 × 16	8.224	6.013
	PU 16 × 8	8.402	5.909
	PU 16 × 16	8.992	6.356
CU 32 × 32	PU 16 × 32	10.17	6.451
	PU 32 × 16	9.037	6.503
	PU 32 × 32	9.729	6.877
CU 64 × 64	PU 32 × 64	10.265	7.964
	PU 64 × 32	13.1	7.297
	PU 64 × 64	13.687	8.614

Table 2.
GPU and CPU application running times in seconds.

size block. Similar results are obtained by Frang et al. [18] and Exterman [19], respectively.

5.3 Comparative study

In this section, we compared the time performance of our proposed implementation to State-of-the-Art process [20, 21].

In the work presented by Xiao et al. [20], when comparing the result of the proposed with the HEVC reference software, experimental results show that the proposed GPU implementation achieves 34.4% encoding time reduction on average while the BD-rate increase is only about 2% for a typical low delay setting. Another interesting work is proposed by Karimi et al. [21] used a specific real-world application to compare the performance of CUDA with NVIDIA's implementation of

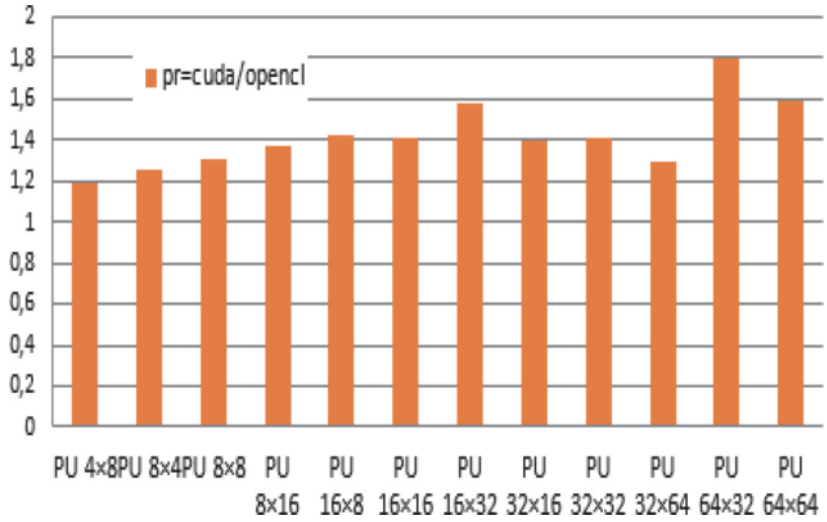

Figure 9.
Performance ratio.

OpenCL. Contrary to our results, CUDA's kernel execution was here consistently faster than OpenCl's, despite the two implementations running nearly identical code. CUDA seems to be a better choice for applications where achieving as high a performance as possible is important. Otherwise the choice between OpenCL and CUDA can be made by considering factors such as prior familiarity with either system, or available development tools for the target GPU hardware. The performance will be dependent on some variables, including code quality, algorithm type and hardware type.

6. Conclusion

OpenCL is quite competitive with CUDA on the NVIDIA graphics processor in terms of performance. In this work, the use of OpenCL as a portable language for the development of GPGPU applications was studied. SAD is the largest part of runtime and calculation in motion estimation the reduction technique was used to implement the SAD, which significantly allows reducing the run time. The performance ratio was equals to 2 when comparing the OpenCL implementation to the CUDA one.

Paralleling multiple GPU algorithms could improve performance. In addition to the ME algorithm of the Joint Collaborative Video Coding Team (JCT-VC) [22], we assume that the suggested concept can also be applied.

Conflict of interest

The authors declare no competing interests.

Author details

Randa Khemiri[1,2*], Soulef Bouaafia[1], Asma Bahba[3], Maha Nasr[3] and Fatma Ezahra Sayadi[3]

1 Faculty of Sciences, Electronics and Microelectronics Laboratory, University of Monastir, Monastir, Tunisia

2 Higher Institute of Computer Science and Multimedia Gabes, University of Gabes, Tunisia

3 Networked Objects, Control, and Communication Systems Laboratory (NOCCS), National Engineering School of Sousse, University of Sousse, Sousse, Tunisia

*Address all correspondence to: randa.khemiri@gmail.com

IntechOpen

References

[1] Osama, M., Wijs, A.: Parallel SAT Simplification on GPU Architectures. In: Vojnar T., Zhang L. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS. Lecture Notes in Computer Science, 114(27) (2019).

[2] Yang, X., Jian, L., Wu, W. et al. J Real-Time Image Proc, https://doi.org/10.1007/s11554-018-0803-y, 2019.

[3] Karimi, K., Dickson, N. G., Hamze, F.: A Performance Comparison of CUDA and OpenCL (2010).

[4] Tsuchiyama, R., Nakamura, T., Iizuka, T., Asahara, A.: The OpenCL Programming Book. Fixstars Corporation (2010).

[5] Richardson, I.: 'HEVC an introduction to high efficiency video coding', VCodexVideo Compression, h ttp://vcodex.com/, accessed 15 January 2016.

[6] Kim, J., Jun, D.S., Jeong, S., et al.: 'An SAD-based selective bi-prediction method for fast motion estimation in high efficiency video coding', ETRI J., 2012, 34, (5), pp. 753–758.

[7] Tai, S., Chang, C., Chen, B., et al.: 'Speeding up the decisions of quad-tree structures and coding modes for HEVC coding units', Adv. Intell. Syst. Appl. (SIST 21), 2013, 2, pp. 393–401.

[8] Hyang-Mi, Y., Jae-Won, S.: 'Fast coding unit decision algorithm based on inter and intra prediction unit termination for HEVC'. IEEE Int. Conf. Consumer Electronics (ICCE), Las Vegas, 2013, pp. 300–301.

[9] Khemiri, R., Bahri, N., Belghith, F., Bouaafia, S., Sayadi, F.E., Atri, M., Masmoudi, N. : 'Fast Motion Estimation's Configuration Using Diamond Pattern and ECU, CFM, and ESD Modes for Reducing HEVC Computational Complexity'. In book: Digital Imaging. Publisher: IntechOpen, pp. 1-18.

[10] Praveen, K., Kannappan, P., Ankush, M., Guna, S.: Parallel Blob Extraction using Multicore Cell Processor. Advanced Concepts for Intelligent Vision Systems (ACIVS), 320–332 (2009).

[11] Vanne, J., Aho, E., Hamalainen, T. D., Kuusilinna, K.: A High-Performance Sum of Absolute Difference Implementation for Motion Estimation," Circuits and Systems for Video Technology, IEEE Transactions on, 16 (7), 876-883 (2006).

[12] The individual procedures of OpenCL: processing program coding, separating individual tasks and transferring them to the respective processors. © Khronos-Group.

[13] Junaid, T., Sam, K., Hui, Y.: HEVC intra mode selection based on Rate Distortion (RD) cost and Sum of Absolute Difference (SAD), Journal of Visual Communication and Image Representation, 35, 112-119 (2016).

[14] Jinglin, Z., Jean François, N., Jean-Gabriel, C.: Implementation of Motion Estimation Based On Heterogeneous Parallel Computing System withOpenc. 14th Ieee International Conference On High Performance Computing and Communications (HPCC), Liverpool, United Kingdom (2012).

[15] Khemiri, R., Kibeya, H., Sayadi, F. E., Bahri, N., Atri, M., Masmoudi, N.: Optimization of HEVC motion estimation exploiting SAD and SSD GPU-based implementation. IET Image Processing, 12(2), 243-253 (2018).

[16] Khemiri, R., Chouchene, M., Barhi, H., et al.: Fast SAD algorithm of HEVC

video encoder on two successive GPU generations', Int. J. Imaging Robot, 17, (2), 1–11 (2017).

[17] Chouchene, M., Sayadi, F., Bahri, H., Dubois, J., Miteran, J., Atri. M.: Optimized Parallel Implementation of Face Detection based on GPU component. Microprocessors and Microsystems, 393–404 (2015).

[18] Jianbin, F., Ana Lucia, V., Henk, S.: A Comprehensive Performance Comparison of CUDA and OpenCL. International Conference on Parallel Processing, ICPP Taipei, Taiwan (2011).

[19] Exterman, D.: CUDA vs OpenCL: Which to Use for GPU Programming, (2021).

[20] Xiao, W., Wu, F., Xu, J., Shi, G.: Fast HEVC Encoding with GPU Assisted Reference Picture Selection, In: Huet B., Ngo CW., Tang J., Zhou ZH., Hauptmann A.G., Yan S. (eds) Advances in Multimedia Information Processing – PCM 2013. PCM 2013. Lecture Notes in Computer Science, Springer, 8294, https://doi.org/10.1007/978-3-319-03731-8_22 (2013).

[21] Karimi, K., Dickson, N. G., Hamze F., A Performance Comparison of CUDA and OpenCL, Cornell University, https://arxiv.org/abs/1005.2581 (2011).

[22] The Joint Collaborative Team on Video Coding (JCT-VC) https://hevc.hhi.fraunhofer.de/.