IEIE Transactions on Smart Processing and Computing

Integer-Pel Motion Estimation for HEVC on Compute Unified Device Architecture (CUDA)

Dongkyu Lee[1], Donggyu Sim[2], and Seoung-Jun Oh[1]

[1] Department of Electronic Engineering, Kwangwoon University / Seoul, Korea {dongkyu, sjoh}@media.kw.ac.kr
[2] Department of Computer Engineering, Kwangwoon University / Seoul, Korea dgsim@kw.ac.kr

* Corresponding Author: Dongkyu Lee

Received February 20, 2014; Revised April 20, 2014; Accepted August 28, 2014; Published December 31, 2014

** Short Paper*

** Extended from a Conference: Preliminary results of this paper were presented at the IEEE ICCE Jan. 2013.*

***Abstract*:** A new video compression standard called High Efficiency Video Coding (HEVC) has recently been released onto the market. HEVC provides higher coding performance compared to previous standards, but at the cost of a significant increase in encoding complexity, particularly in motion estimation (ME). At the same time, the computing capabilities of Graphics Processing Units (GPUs) have become more powerful. This paper proposes a parallel integer-pel ME (IME) algorithm for HEVC on GPU using the Compute Unified Device Architecture (CUDA). In the proposed IME, concurrent parallel reduction (CPR) is introduced. CPR performs several parallel reduction (PR) operations concurrently to solve two problems in conventional PR; low thread utilization and high thread synchronization latency. The proposed encoder reduces the portion of IME in the encoder to almost zero with a 2.3% increase in bitrate. In terms of IME, the proposed IME is up to 172.6 times faster than the IME in the HEVC reference model.

***Keywords*:** HEVC, Motion estimation, CUDA, Parallel reduction

1. Introduction

The popularity of high definition (HD) videos, the diversity of service and the emergence of beyond HD formats create demands for a new video coding standard that can achieve higher coding efficiency and better visual quality than the H.264/AVC (Advanced Video Coding) standard. The new standard, High Efficiency Video Coding (HEVC), was recently established by the Joint Collaborative Team on Video Coding (JCT-VC), an expert group proposed by the ISO/IEC Moving Expert Group (MPEG) and the ITU-T Video Coding Expert Group (VCEG) [1]. Compared to H.264/AVC, HEVC provides 50 % better compression efficiency with equal visual quality. The significant bit-rate savings are achieved through the use of a set of new encoding tools and algorithms, along with some well-known features adopted from existing standards. On the other hand, such compression performance requires a dramatic increase in the computational complexity, making real-time video coding difficult to achieve. Motion estimation (ME) is the most complex and time-consuming process. In HEVC encoders, ME typically requires approximately 70% of the total computational load. Therefore, some researches have focused on ways to reduce the computational load of ME. Diamond search [2] and three step search [3] are two well-known fast-search algorithms.

Recently, the academic and industrial parallel processing communities have turned their attention to Graphic Processing Units (GPUs). Modern GPUs have evolved into massively-parallel processors with tremendous power so that the computational capability of GPUs far surpasses that of Central Processing Units (CPUs). Some stages of the fixed-function pipeline in modern GPUs have been replaced by programmable modules. Therefore, GPUs are currently utilized not only for accelerating graphics processing and 3D rendering but also for speeding up non-graphics applications, such as linear algebra, physical simulation, and image processing. This type of utilization is called General-Purpose computing on GPUs (GPGPU). With this trend, Compute Unified Device Architecture (CUDA) was introduced by

NVIDIA in 2006 to provide a programming API to exploit the high degree of inherent parallelism on GPUs [4]. It is scalable and allows the programmer to implement task- and data-parallel parts of the software in an abstract yet transparent manner by providing a small set of extensions to standard programming languages.

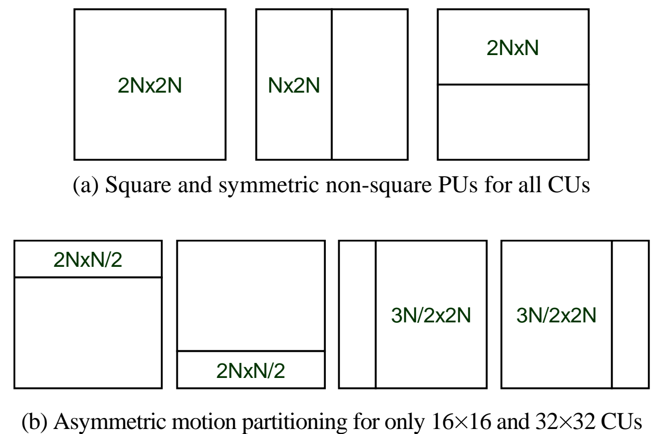
This paper extends our earlier work on a GPU-based full-search integer-pel ME (IME) algorithm for H.264/AVC [5] to HEVC by taking the characteristics of HEVC into account and evaluates its performance in terms of coding efficiency and computational complexity after integrating it into an HEVC encoder. In the proposed IME, concurrent parallel reduction (CPR) is presented. CPR increases the thread utilization and reduces the thread synchronization by performing several PR operations simultaneously to solve two problems in conventional parallel reduction (PR). Because HEVC supports larger block sizes, more flexible block partitioning, and asymmetric motion partitioning (AMP), all these characteristics were considered in the proposed algorithm. In the proposed encoder, GPU performs the IME algorithm in a pre-pass and transfers the integer-pel motion vector (IMV) information to the CPU. The CPU then performs all other processing, such as prediction and transform. In the inter predictions, all IMVs and costs obtained by the GPU are used directly for motion vector refinement.

The remainder of this paper is organized as follows. The next section describes the related works. Section 3 introduces the conventional IME in HEVC and the CUDA architecture. Section 4 provides details of the proposed approach. Section 5 includes performance evaluations of the proposed encoder, followed by the final conclusions in the last section.

2. Related Work

Some studies aiming to accelerate ME using a GPU for H.264/AVC can be found [6-8]. Chen et al. presented an efficient 4×4 block-level parallelized full search algorithm [6]. Their work divided the ME into different steps to achieve high parallelism through low data transfer between the CPU and GPU memories and it considered a block-level parallelism. On the other hand, this approach has high memory access latency that results from the use of the off-chip memory of the GPU. Zhou et al. [7] presented another approach that takes full advantage of the on-chip memory of the GPU to reduce memory access latency and achieved up to 50 fold performance improvement compared to CPU implementation. Monteiro et al. [8] implemented an ME algorithm with three distinct hardware architectures: a parallel solution for multi-core processor using OpenMP, a distributed solution for cluster/grid machines, and a CUDA-based solution. Their results showed that the CUDA-based solution provides significant speed-up compared to the others. Some GPU-based ME algorithms have been proposed for HEVC [9-11]. Radicke et al. applied AMP and diamond search pattern, and evaluated their algorithm on various GPUs [9]. Wang et al. presented an HEVC encoder with a CPU plus GPU platform [10]. The ME process was performed on GPU simultaneously while other processes were performed on the CPU. They employed wavefront parallel processing for synchronization between the CPU and GPU. Another fast encoding system performed the ME process line-by-line on GPU to overlap the CPU and GPU computations [11].

These studies, however, did not consider the PR operation. Conventional PR inherently has very low thread utilization, resulting from halving the number of active threads for consecutive iterations and required thread synchronization between the iterations due to the data hazards, which can be solved by the proposed CPR.



(a) Square and symmetric non-square PUs for all CUs

(b) Asymmetric motion partitioning for only 16×16 and 32×32 CUs

Fig. 1. PU partitions for a 2*N*x2*N* CU.

3. Background

3.1 Integer-Pel Motion Estimation in HEVC

IME is performed on a block-by-block basis and supports variable block sizes in HEVC. Every frame is subdivided into coding tree units (CTUs). This is comparable to the concept of macroblocks in H.264/AVC. Each CTU can be divided recursively into multiple square coding units (CUs), ranging from 8×8 up to the CTU size. Each CU includes several prediction units (PUs) and transform units (TUs), which allows the prediction and transform to be handled in an independent manner. There are various PU partitions in a $2N\times2N$ CU, as shown in Fig. 1. AMP is a major innovation of HEVC compared to H264/AVC; it improves the coding efficiency because it can represent an irregular image pattern quite efficiently.

The IME process identifies the best matched block in a search area (SA) of a reference frame with respect to the target PU in a current frame and provides a map of the displacement using a motion vector (MV). After a reference frame is selected, an SA is determined in the reference frame, typically centered at a motion vector predictor (MVP). A search for the most similar block is performed in the SA. Matching is normally performed by minimizing the similarity criterion defined by

$$C = D + \lambda_{pred} \cdot B_{pred}, \tag{1}$$

where C is the total cost. D normally represents the sum of absolute differences (SAD) between the current PU and the matching block in the reference frame. λ_{pred} is the Lagrangian multiplier and B_{pred} is the number of bits required to code a motion vector difference (MVD) between the MVP and the MV candidate. Once the best matched block is selected, the MV candidate pointing to its position becomes the final MV of the PU.

3.2 Compute Unified Device Architecture (CUDA)

CUDA is a parallel programming framework for utilizing GPUs for general purpose computing. A CUDA program consists of a host program executed by sequential threads on CPU and a device program executed by thousands of threads on GPU. There is a function called a kernel that is launched by CPU and executed on GPU. CUDA employs a thread hierarchy to deal with the large number of threads. A thread block consists of concurrently executing threads and a grid includes independent thread blocks. All threads in a thread block can cooperate among themselves through barrier synchronization and communicate data via the on-chip memory of GPU. Fine-grain data parallelism can be captured with threads, and more coarse-grain parallelism can be described with thread blocks. CUDA GPUs create, manage, schedule, and execute threads in groups of 32 parallel threads called warps. i.e., a thread block can also be split into warps. All threads in a warp execute one common instruction at a time, so full efficiency is realized when all 32 threads of the warp agree on their execution path. Before launching a kernel, a programmer needs to determine how many threads are needed for kernel execution and specify a thread block and a grid dimension.

This study has been designed on a Kepler architecture GPU. A Kepler GPU consists of streaming multiprocessors (SMXs). An SMX includes several CUDA cores, registers, configurable shared memory/L1 cache, warp schedulers, etc.. Once a kernel is launched, a specified grid is configured and the thread blocks are assigned to SMXs depending on the required resources, such as the number of threads per thread block and the amount of shared memory. The CUDA cores and other execution units in an SMX execute the threads. In the GPU, there are several memory types: register, local memory, shared memory, global memory, constant memory, and texture memory [12]. The register is the fastest memory and only visible to a corresponding thread. That is, each thread has its own registers and cannot access those of the other threads. Local memory is used to hold automatic variables when the device compiler determines if there is insufficient register space to hold them. In contrast to registers, access to local memory is highly time-consuming because it is an off-chip memory. Shared memory is an on-chip memory and as fast as registers when proper access patterns are used. This memory is only accessible to a thread block. All threads within a thread block use this memory to share data and communicate with other threads. All threads can access global memory and conduct read-only access to constant memory and texture memory. These memories have cache memories to speed up memory access and are optimized for different memory usages. Texture memory offers fast 2D memory access using a texture cache and different addressing modes as well as data filtering for some specific data formats.

4. Proposed Integer-Pel Motion Estimation Algorithm

In this study, the full-search is used because it is more suitable for GPU computation than other fast-search methods. An SA size should be chosen carefully because the computational complexity depends directly on it. From the evaluations for coding efficiency with various SA sizes under common test conditions (CTCs) [13], a SA size of 16 was chosen. In MVP decisions, the spatial predictors are unavailable because all blocks are processed at the same time. Therefore, zero MVP is assumed for all PUs.

The proposed algorithm consists of three stages: SAD calculation, hierarchical-SAD computing (H-SAD) [6], and CPR. Before the IME process, a current frame and its reference frame are transferred to the texture memory of GPU. Every CTU is processed by one thread block. Therefore, the thread blocks can process the corresponding CTUs simultaneously. As every thread block performs the same process, the algorithm is described in the view of a thread block in the next subsections.

4.1 SAD Calculation

The SAD value of every search position in a search window needs to be calculated. The search window size is set to 32×32 according to the SA size of 16. The thread block size is set to 32×32 to map each thread to each search position. The default CTU size is 64×64 and the smallest PU size is 4×8 or 8×4 in HEVC. First, both the CTU in the current frame and its reference block are cached into shared memory. Thread synchronization is called to guarantee that all the data is loaded completely. All SAD values of a 4×4-block in a CTU are then calculated, i.e., each SAD value is calculated by each thread and stored into global memory. These SAD values are denoted as a 4×4-SAD group. This parallel SAD calculation is repeated until all 4×4-SAD groups for the CTU are computed. Note that there are 256 4×4-SAD groups for the CTU.

4.2 Hierarchical SAD Computing

H-SAD computing is the process of deriving the SAD values in a larger block from the previously calculated SAD values in smaller blocks. The basic idea is that an SAD value of a block at a given search position is equal to the sum of the SAD values of all sub-blocks it contains. Therefore, an SAD value with few addition operations can be obtained instead of calculating the actual SAD value. In the H-SAD stage, a thread block size is set to 32×32 to map each thread to each search position. Some 4×4-SAD groups are cached into shared memory, which are then combined to generate the SAD values of both PUs in an

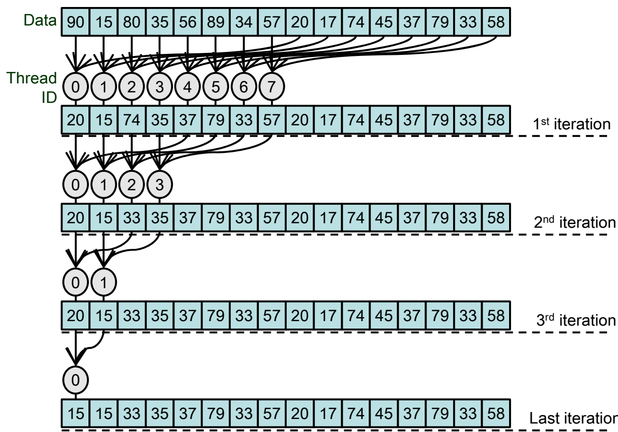


Fig. 2. Example of the PR process for the minimum.

8×8-CU and AMPs of a 16×16-CU. The generated SAD groups are stored into global memory. In the same manner, all SAD groups in the CTU are constructed. Note that there are 585 SAD groups in the CTU.

4.3 Concurrent Parallel Reduction (CPR)

The position of a block with the minimum cost defined by Eq. (1) needs to be found to obtain an MV. Each SAD group has its own MV. Therefore, a PR operation can be applied to find the minimum cost. Fig. 2 gives an example of the PR process with 8 threads for the minimum among 16 data. The number of processed data is halved after each iteration, resulting in the minimum after the last iteration. In this work, a PR operation with 512 threads can be applied after calculating 1024 costs using Eq. (1). For a number of SAD groups in a CTU, the PR operations can be applied sequentially. On the other hand, this sequential approach has some critical disadvantages in terms of thread utilization as well as thread synchronization latency. In PR, the number of active threads is halved between iterations. This yields very low thread utilization, resulting in long latency. Low thread utilization means that less active threads are being scheduled in an SMX; so that a warp is less likely to hide a latency caused by another warp executing a long-latency operation.

The proposed CPR deals with these problems. To increase the thread utilization and decrease the thread synchronization, CPR processes several SAD groups in parallel. The thread block is set to contain as many threads needed to obtain high occupancy. In current GPUs, 1024 threads can be allocated in a thread block. As many SAD groups as possible are then cached into shared memory. For Kepler GPUs, only 16 SAD groups can be loaded due to the shared memory size limit. Each SAD value is used to calculate the corresponding cost using Eq. (1). In this study, 1024 costs corresponding to an SAD group are denoted as a cost group. There are 16 cost groups and 1024 threads so that 1024 threads are grouped into 16 thread groups, each of which consists of 64 threads. 16 PR operations are executed concurrently by mapping each thread group to each cost group. Each thread in a cost group finds a smaller cost among two costs in each iteration, resulting in 64 costs at the last, as shown in Fig. 3. Because every thread group works independently on its corresponding cost group, no synchronization is needed between iterations. On the other hand, after the last iteration, just one thread synchronization is needed to ensure that 64 costs are obtained. It is needed to find the minimum cost among 64 costs with 64 threads. Therefore, the conventional PR operation with warp-unrolling is applied to 64 costs by a warp, 32 threads [14]. This warp-based PR operation requires no thread synchronization. In short, CPR increases the thread utilization by treating several cost groups simultaneously and requires only one thread synchronization. In one CPR operation, 16 IMVs can be obtained so that the CPR operation is performed repeatedly until all IMVs are obtained. The GPU transfers

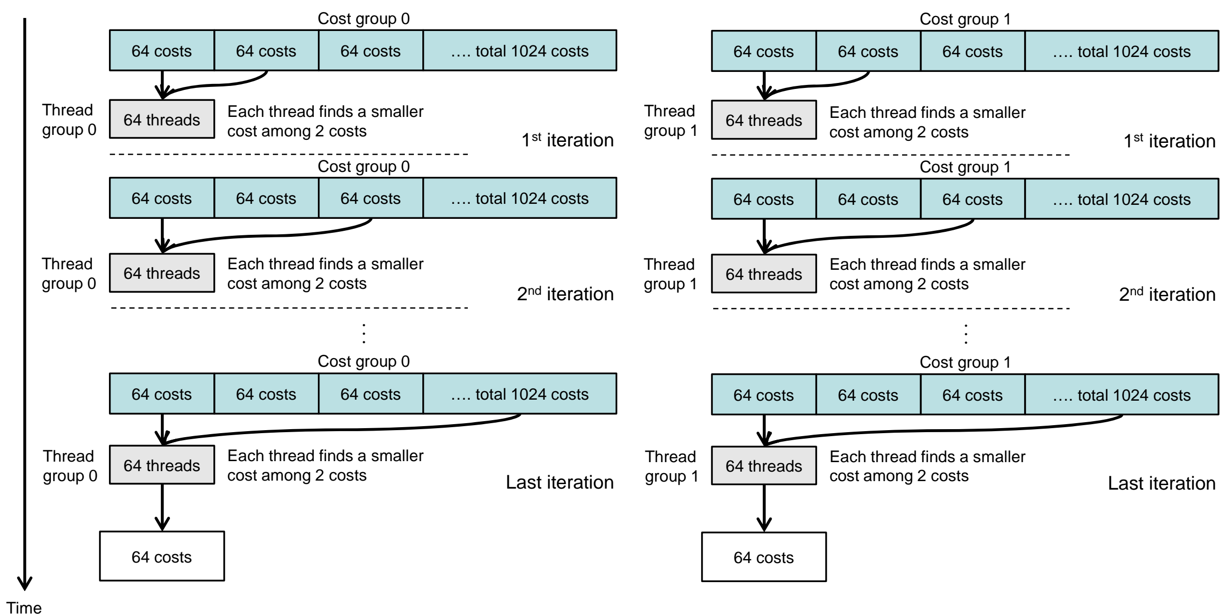


Fig. 3. Process for finding 64 small costs in a cost group in CPR.

all IMVs and corresponding costs to the CPU.

5. Performance Evaluation

For a performance evaluation of the proposed algorithm, Intel Core i7-2600 3.4 GHz CPU, 8 GB memory, Geforce GTX 780 with 3 GB DRAM, and CUDA toolkit 6.5 were used. HM 10.0 encoder under the low-delay P configuration in the CTC was used as an anchor. The proposed GPU-based IME was integrated into the anchor, which is denoted as the proposed encoder. 100 frames of class B (1920×1080) and class C (832×480) sequences with QPs of 22, 27, 32, and 37 were tested. For a performance evaluation of the proposed encoder in coding efficiency and computational complexity with respect to the anchor, the Bjontegaard-delta (BD)-bitrate of the Y component [15] and the encoding time saving (ETS) were used. The ETS is defined as

$$ETS = \frac{T_{anc} - T_{pro}}{T_{anc}} \times 100\ (\%), \quad (2)$$

where T_{anc} and T_{pro} are the encoding times of the anchor and the proposed encoder, respectively. A relative computational complexity of IME (R_{ime}) in an encoder is defined as

$$R_{ime} = \frac{T_{ime}}{T_{enc}} \times 100\ (\%), \quad (3)$$

where T_{ime} and T_{enc} are the IME execution time and the total encoding time, respectively.

Table 1 lists the coding efficiency and the ETS of the proposed encoder with respect to the anchor as well as the R_{ime} of the two encoders. The data transfer time between CPU and GPU is included in the GPU-based IME execution time measurement. The proposed encoder yielded a 2.3% increase in BD-bitrate against the anchor. The bitrate losses result from the small SA size of 16 as well as the zero center position in the SA. The performance degradation is significant for the sequence, "BasketballDrive", which has high motion activities with global motion. The use of the zero center position is the dominant factor for the loss. In terms of computational complexity, the R_{ime} of the proposed encoder is 0.3% on average whereas it is 19.4% in the anchor. The proposed encoder reduces the complexity of IME to almost zero, providing 18.1% ETS.

Table 2 lists the speed-up (SU) of the proposed IME, which is the ratio of two IME execution times defined as

$$SU = \frac{IMET_{anc}}{IMET_{pro}}, \quad (4)$$

where $IMET_{anc}$ and $IMET_{pro}$ are the total IME execution times of the anchor and the proposed encoder, respectively. Table 2 shows that the proposed IME provides SU of up to 172.6 against the IME in the anchor. The complexity of the proposed IME depends on the sequence size because it employs the full-search method. In contrast, the IME complexity in the anchor depends on both the sequence size and the characteristics of the sequence, resulting from the fast-search method.

Table 2. Speed-up of the proposed IME against the IME in the anchor for 100 frames.

Sequence	IME execution time (s)		Speed-up
	IME in the anchor	Proposed IME	
Kimono	1892.8	12.1	156.4
ParkScene	765.4	12.1	63.3
Cactus	877.7	12.1	72.5
BasketballDrive	1820.5	12.1	150.5
BQTerrace	666.8	12.0	55.6
Average	**1204.6**	**12.1**	**99.7**
BasketballDrill	224.1	2.7	83.0
BQMall	221.4	2.7	82.0
PartyScene	191.8	2.7	71.0
RaceHorses	466.0	2.7	172.6
Average	**275.8**	**2.7**	**102.2**

Table 1. Coding efficiency and ETS of the proposed encoder against the anchor for 100 frames.

Class	Sequence	BD-bitrate (%)	ETS (%)	R_{ime} (%)	
				Anchor	Proposed encoder
B	Kimono	1.3	26.7	27.3	**0.2**
	ParkScene	2.1	12.6	14.2	0.3
	Cactus	0.7	15.3	16.7	0.3
	BasketballDrive	9.7	26.8	26.6	0.3
	BQTerrace	1.1	9.4	11.8	0.3
C	BasketballDrill	2.1	17.8	18.6	0.3
	BQMall	0.6	17.3	18.4	0.3
	PartyScene	0.2	13.2	13.7	0.2
	RaceHorses	3.2	23.5	27.2	0.2
	Average	**2.3**	**18.1**	**19.4**	**0.3**

6. Conclusion

This paper proposed a GPU-based IME algorithm for HEVC. H-SAD computing was applied to reduce the computational complexity by data reuse. The proposed CPR solves two problems in the conventional PR. CPR increases the thread utilization and reduces the thread synchronization by conducting several PR operations concurrently. The proposed GPU-based encoder makes the portion of the IME negligible with a 2.3% increase in BD-bitrate. Owing to the dependency on the spatial MVs, zero center position is used for the SA, which leads the significant coding loss in a sequence with high motion activities. Designing a proper center position can improve the coding performance. In terms of the IME, the proposed IME is up to 172.6 times faster than the IME in the HM encoder.

In this study, promising performance was obtained in the IME with the huge computing power of the GPU. This GPU-based parallelization approach can be applied to other algorithms used in video coding, such as interpolation, fractional ME, de-blocking filter, and Sample Adaptive Offset (SAO) in HEVC. With these algorithms parallelized on GPU, video encoding can be accelerated much further.

Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP. [14-000-11-002, Development of Object-based Knowledge Convergence Service Platform using Image Recognition in Broadcasting Contents]

References

[1] B. Bross, W-J. Han, J-R. Ohm, G. J. Sullivan, Y-K. Wang, and T. Wiegand, "High Efficiency Video Coding (HEVC) Text Specification Draft 10," *Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T VCEG and ISO/IEC MPEG,* JCTVC-L1003, Geneva, CH, Jan. 2013. Article (CrossRef Link)

[2] S. Zhu and K-K. Ma, "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation," *IEEE Trans. Image Process.*, vol. 9, no. 2, pp. 287–290, Feb. 2000. Article (CrossRef Link)

[3] X. Jing and L-P. Chau, "An Efficient Three-Step Search Algorithm for Block Motion Estimation," *IEEE Trans. Multimedia*, vol. 6, no. 3, pp. 435–438, June 2004. Article (CrossRef Link)

[4] NVIDIA, CUDA C Programming Guide Ver 6.5, *NVIDIA Corp.*, Santa Clara, CA, Aug. 2014. Article (CrossRef Link)

[5] D-K. Lee, and S-J. Oh, "Variable Block Size Motion Estimation Implementation on Compute Unified Device Architecture (CUDA)," in *Proc. of IEEE Int. Conf. Consum. Electron.*, pp. 635–636, Jan. 2013. Article (CrossRef Link)

[6] W-N. Chen and H-M. Hang, "H.264/AVC Motion Estimation Implementation on Compute Unified Device Architecture (CUDA)," *in Proc. of IEEE Int. Conf. Multimedia and Expo (ICME)*, pp. 697–700, Apr. 2008. Article (CrossRef Link)

[7] Z. Jing, J. Liangbao, and C. Xuehong, "Implementation of Parallel Full Search Algorithm for Motion Estimation on Multi-Core Processors," in *Proc. Int. Conf. Next Generation Information Technology*, pp. 31–35, June 2011. Article (CrossRef Link)

[8] E. Monteiro, B. Vizzotto, c. Diniz, M. Maule, B. Zatt, and S. Bampi, "Parallelization of Full Search Motion Estimation Algorithm for Parallel and Distributed Platforms," *Int. J. Parallel Prog.*, vol. 42, no. 2, pp. 239–264, Aug. 2012. Article (CrossRef Link)

[9] S. Radicke, J. Hahn, C. Grecos, and Q. Wang, "Highly-Parallel HEVC Motion Estimation with CUDA," *in Proc. of IEEE European Workshop on Visual Information Processing (EUVIP)*, pp. 148–153, June 2013. Article (CrossRef Link)

[10] X-W. Wang, M. Chen, and J-J. Yang, "Paralleling Variable Block Size Motion Estimation of HEVC on Multicore CPU plus GPU Platform," *in Proc. of IEEE Int. Conf. Image Process. (ICIP)*, pp. 1836–1839, Sept. 2013. Article (CrossRef Link)

[11] S. Radicke, J. Hahn, C. Grecos, and Q. Wang, "A Highly-Parallel Approach on Motion Estimation for High Efficiency Video Coding (HEVC)," *in Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, pp. 187–188, Jan. 2014. Article (CrossRef Link)

[12] R. Farber, "CUDA Application Design and Development," *Morgan Kaufmann*, Waltham, MA, pp. 109–131, 2011. Article (CrossRef Link)

[13] F. Bossen, "Common Test Conditions and Software Reference Configurations," *Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T VCEG and ISO/IEC MPEG*, JCTVC-L1100, Geneva, CH, Jan. 2013. Article (CrossRef Link)

[14] M. Harris, "Optimizing Parallel Reduction in CUDA," *NVIDIA Developer Technology*, 2007. Article (CrossRef Link)

[15] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," *ITU-T VCEG,* VCEG-M33, Austin, TX, April, 2001. Article (CrossRef Link)

Dongkyu Lee was born in Seoul, Korea. He received his B.S. and M.S. degrees in Electronic Engineering from Kwangwoon University, Seoul, Korea, in 2012 and 2014, respectively. He is a PhD student at the Kwangwoon University. His research interests are image and video processing, video compression, and video coding.

Donggyu Sim was born in Chungchung Province, Korea, in 1970. He received his B.S. and M.S. degrees in Electronic Engineering from Sogang University, Seoul, Korea, in 1993 and 1995, respectively. He also received his Ph.D. degree there in 1999. He was with the Hyundai Electronics Co., Ltd. from 1999 to 2000, where he was involved in MPEG-7 standardization. He was a Senior Research Engineer at Varo Vision Co., Ltd., working on MPEG-4 wireless applications from 2000 to 2002. He worked for the Image Computing Systems Lab. (ICSL) at the University of Washington as a Senior Research Engineer from 2002 to 2005. His research involved ultrasound image analysis and parametric video coding. He joined the Department of Computer Engineering at Kwangwoon University, Seoul, Korea in 2005, and now is working as a Professor. He was elevated to an IEEE Senior Member in 2004. His current research interests are image processing, computer vision, and video communication.

Seoung-Jun Oh was born in Seoul, Korea, in 1957. He received both the B.S. and the M.S. degrees in Electronic Engineering from Seoul National University, Seoul, in 1980 and 1982, respectively, and the Ph.D. degree in Electrical and Computer Engineering from Syracuse University, New York, in 1988. In 1988, he joined ETRI, Daejeon, Korea, as a Senior Research Member. Later he was promoted to a Program of Multimedia Research Session in ETRI. Since 1992, he has been a Professor of Department of Electronic Engineering at Kwangwoon University, Seoul, Korea. He has been a Chairman of SC29-Korea since 2001. His research interests include image and video processing, video compression, and video coding.

Copyrights © 2014 The Institute of Electronics and Information Engineers