Parallel Integer Motion Estimation for High Efficiency Video Coding (HEVC) Using OpenCL

Augusto Gomez(✉), Jhon Perea, and Maria Trujillo

Multimedia and Computer Vision Group, Universidad Del Valle, Cali, Colombia
{augusto.gomez,jhon.edinson.perea,maria.trujillo}@correounivalle.edu.co

Abstract. High Efficiency Video Coding is able to reduce the bit-rate up to 50% compared to H.264/AVC, using increasingly complex computational processes for motion estimation. In this paper, some motion estimation operations are parallelised using Open Computing Language in a Graphics Processing Unit. The parallelisation strategy is three-fold: calculation of distortion measurement using 4×4 blocks, accumulation of distortion measure values for different block sizes and calculation of local minima. Moreover, we use 3D-arrays to store the distortion measure values and the motion vectors. Two 3D-arrays are used for transferring data from GPU to CPU to continue the encoding process. The proposed parallelisation is able to reduce the execution time, on average 52.5%, compared to the HEVC Test Model. Additionally, there is a negligible impact on the compression efficiency, as an increment in the BD-BR, on average 2.044%, and a reduction in the BD-PSNR, on average 0.062%.

Keywords: GPU · HEVC · Motion estimation · OpenCL · Parallel programming

1 Introduction

High Efficiency Video Coding (HEVC) achieves increase coding efficiency, while preserving video quality with lower bit-rate compared to H.264/AVC. However, it also increases dramatically the encoding computational complexity [15]. Motion estimation is the most time consuming task in a video encoder. In HEVC, the motion estimation requires about 77%–81% of the total encoding time due to distortion measure calculations and flexibility in block sizes [8]. Moreover, motion estimation configurations, such as Advanced Motion Vector Prediction (AMVP) [19], generate data dependency between neighbour blocks that make difficulties in using hardware with Single Instruction Multiple Data (SIMD) architecture [9]. The HEVC specifications include the following parallel processing options: Wavefront Parallel Processing (WPP) [5], Slices [11] and Tiles [3]. Moreover, there are approaches on parallel motion estimation presented in the literature, for instance *Wang* et al. in [17].

Graphics Processor Units (GPUs) have been used for reducing the encoding time, especially for motion estimation in HEVC – for instance in [12]. *Luo* et al. presented in [10] a quaternion of (L, T, R, B) for indexing a look-up table with

PU sizes from 4×4 to 32×32; the performance evaluation was implemented on HM10.0 [6]. *Wang* et al. proposed in [16] a two-fold approach: an initial motion estimation is obtained on GPU and then, it is refined on CPU. The authors considered 256 symmetric partitions of a Coding Tree Unit (CTU) using an initial partition of 4×4 and accumulating block sizes to reach a 64×64 block; the implementation was done on x265 video encoder [13]. *Jiang* et al. introduced in [7] an approach focused on reducing the overhead, which is based on dividing a CTU into 256 blocks of 4×4, and the implementation was done on HM12 [6]. They reported results, using the PeopleOnStreet video sequence, of a time reduction of 34.64% with QP equal to 32 and 40.83% with QP equal to 42 compared to the HM12.

In this paper, we propose a parallel strategy for integer motion estimation with negligible impact on the coding efficiency, using OpenCL as programming framework [4]. The proposed approach has been divided into three main steps: distortion measure calculations, distortion measure accumulations and rate distortion minima estimations, as the foundations of estimating motion vectors. Two 3D-arrays are built for storing distortion measure values and estimated motion vectors. In this way, data is transferred to CPU. Experimental results shown an average of 52.5% reduction of the execution time compared to the HEVC Test Model [6], with a slight increment of the BD-BR, of 2.044% on average, and a slight reduction of the BD-PSNR, of 0.062% on average, using the Bjøntegaard Delta [1] as a metric to compare compression efficiency.

2 Parallel Integer Motion Estimation

The goal of the Parallel Integer Motion Estimation (PIME) is to reduce the computational time required for calculating Motion Vectors (MV)s during the encoding process. HEVC supports CTU with maximum size of 64×64. A CTU is divided into Coding Units (CU) of different sizes. A total of 593 CUs are obtained from a CTU for inter-frame prediction. Unlike the HEVC standard, PIME does not create the tree structure top-down for calculating MVs. Instead, a CTU is divided into 4×4 size blocks and distortion measures are calculated for creating the tree structure bottom-up.

The integer motion estimation is performed in the GPU and MVs are transferred to the CPU for continuing the encoding process. For each CTU, luma data and the search area are transferred to the GPU global memory. The workflow of PIME is presented in Fig. 1. Using a block matching algorithm: Firstly, distortion measures are calculated. Secondly, distortion measures are accumulated. Finally, minima of rate distortions are estimated.

2.1 Distortion Measure Calculations

The calculation of the distortion measure is computed using OpenCL, it uses work-items for running a single instruction. A work-item executes the calculation of a distortion measure in a 4×4 block. Also, work-items are arranged in a

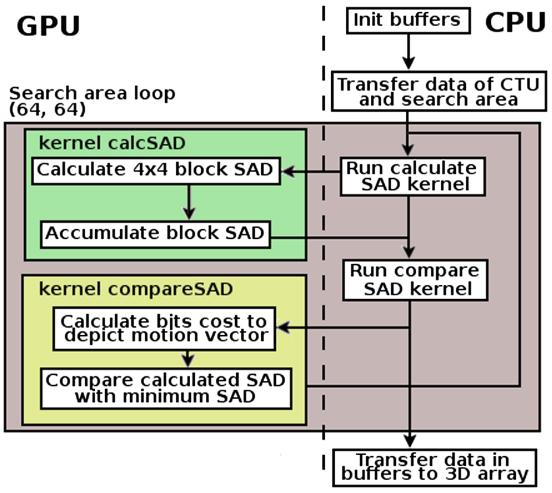

GPU
CPU
Init buffers
Transfer data of CTU and search area
Search area loop (64, 64)
kernel calcSAD
Calculate 4x4 block SAD
Accumulate block SAD
Run calculate SAD kernel
Run compare SAD kernel
kernel compareSAD
Calculate bits cost to depict motion vector
Compare calculated SAD with minimum SAD
Transfer data in buffers to 3D array

Fig. 1. Workflow of Parallel Integer Motion Estimation (PIME)

work-group for accessing a local memory and taking advantage of transferring speed. A work-group contains 16 × 16 work-items – a total of 256 work-items. Obtained values are stored into three temporal buffers: horizontal, vertical and asymmetric (AMP).

2.2 Distortion Measure Accumulations

Using the temporal buffers, distortion measure values are recursively accumulated horizontally and vertically to estimate the distortion in larger blocks, starting from 4 × 4 blocks to 64 × 64 block, including asymmetric partitions using parallel reduction [2]. A work-item adds a pair of corresponding horizontal or vertical distortion measures to obtain the distortion measure for each block. Temporal buffers, in local memory, store recursively accumulations and the buffer, in global memory, stores distortion measures of 593 blocks, defined in the HEVC specifications.

Table 1 shows positions in the global memory buffer of each stored CU.

Table 1. Index used for storing distortion measure values in the GPU global buffer (U = Up, D= Down, L = Left, R = Right)

Index	PU	Index	PU	Index	PU	Index	PU	Index	PU
0–127	8 × 4	336–351	4 × 16R	516–519	32 × 8D	544–559	16 × 16	580	16 × 64L
128–255	4 × 8	352–367	12 × 16L	520–523	32 × 24U	560–567	32 × 16	581	16 × 64R
256–271	16 × 4U	368–383	12 × 16R	524–527	32 × 24D	568–575	16 × 32	582	48 × 64L
272–287	16 × 4D	384–447	8 × 8	528–531	8 × 32L	576	64 × 16U	583	48 × 64R
288–303	16 × 12U	448–479	16 × 8	532–535	8 × 32R	577	64 × 16D	584–587	32 × 32
304–319	16 × 12D	480–511	8 × 16	536–539	24 × 32L	578	64 × 48U	588–589	64 × 32
320–335	4 × 16L	512–515	32 × 8U	540–543	24 × 32R	579	64 × 48D	590–591	32 × 64

2.3 Estimating Minima of Rate-Distortion

A global minima buffer, in the global memory, is used for storing the distortion measures corresponding to the obtained minima calculated using the rate distortion measure, over the search area. Rate distortion is a widely used measure, defined in [14]:

$$J_{\mathrm{MV}} = SAD(MV) + \lambda\, R(MV). \tag{1}$$

where J_{MV} is the cost function, $SAD(MV)$ is a distortion function, λ is the Lagrangian multiplier and $R(MV)$ symbolised the bits required to code the MV.

Two additional buffers are created to keep the $x-$ and the $y-$ coordinates of obtained minima. In this case, the work-items are independent, instead of being organised in a work-group. A work-item compares the current J_{MV} to the obtained local minimum at the moment. Once the block matching is completed, the distortion measure minima are stored in the global minima buffer. Figure 2 show the architecture of PIME.

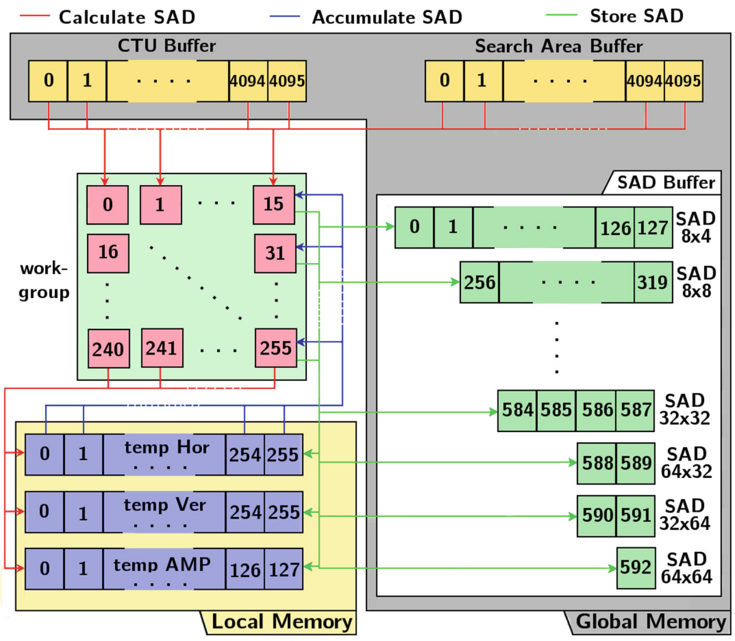


Fig. 2. Parallel integer motion estimation architecture

2.4 GPU and CPU Communication

The PIME process is performed in the GPU device while the encoding process is performed in the CPU (not concurrently). We introduce the use of 3D-arrays for data communication between GPU and CPU. In this way, the OpenCL buffers are mapped into two 3D-arrays in CPU for storing distortion measures and MVs. A 3D-array is built as follows: the first dimension corresponds to the reference frame lists (eRefPicList), the second dimension points to the index for the reference frame (iRefIdxPred) and the third dimension points to the indexes for accessing the CU (CUindex) in Table 1. A 3D-array is illustrated in Fig. 3.

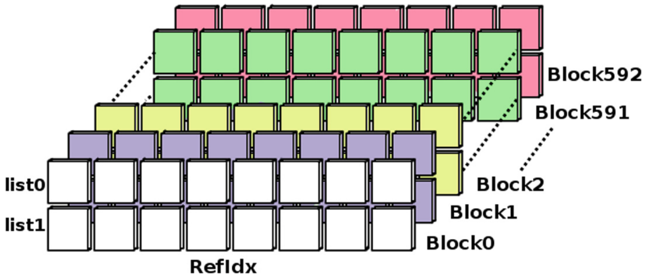


Fig. 3. The 3D-array used for storing

3 Experimental Evaluation

3.1 Experimental Setup and Encoder Configuration

Performance evaluation of PIME is conducted using the HEVC Test Model [6] version 16.4, with the default profile Random Access. The modified source code is available at https://github.com/gomezpirry/HM-OpenCL. The used parameter configurations are in Table 2.

Table 2. Parameters configuration of the HM

Encoder setting/parameter					
MaxCUSize:	64	HadamardME:	True	Search:	Full
MaxPartitionDepth:	4	Fast decision:	True	RateControl:	false
IntraPeriod:	32	AMP:	True	GOP size:	2
SearchRange:	64	Bi-SearchRange:	4	QP:	22, 27, 32, 37

The hardware platform used in these experiments is composed of a CPU Intel Core i7–4500U processor with 1.8 GHz clock base frequency and 8 GB DDR3 of system memory. The GPU used is a NVidia GeForce 840 M, with 384 cores, 2 GB DDR3 of memory and running at 1032 MHz. The encoder was compiled using GCC 4.8.4 and OpenCL 1.2, executed on Ubuntu 14.04 64 bits.

The compression efficiency is judged with the Bjøntegaard Delta Bit-Rate (BD-BR) and the Bjøntegaard Delta PSNR (BD-PSRN). Also, the speed-up using the proposed model is measured using the delta of the execution time (ΔTR %):

$$\Delta TR(\%) = \frac{T_{\mathrm{HM}} - T_{\mathrm{HM-PIME}}}{T_{\mathrm{HM}}} \times 100. \tag{2}$$

where T_{HM} is the execution time required by the HM, and $T_{\mathrm{HM-PIME}}$ is the execution time required by the proposed approach.

The selected test video sequences for presenting evaluation results include one video class A (2560×1600p) and three videos class B (1920×1080p) [18]. All video sequences have a bit depth of 8 and 4:2:0 chroma sub-sampling. For evaluation purposes, 150 frames were encoded from each video sequence.

3.2 Experimental Results

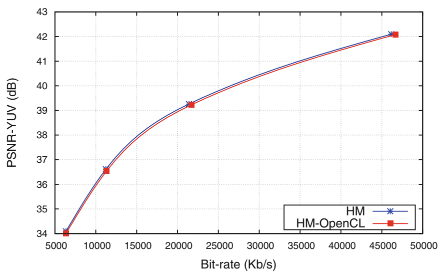
The results in Table 3 show that the proposed parallel strategy is able to reduce the execution time of 52.5% on average at the cost of less than 2.044% coding efficiency. Moreover, there is a negligible impact on the frame reconstruction quality that is reflected in a reduction on the PSNR of 0.062%.

Table 3. Average values over QPs of the BD-BR, the BD-PSRN and the ΔTR

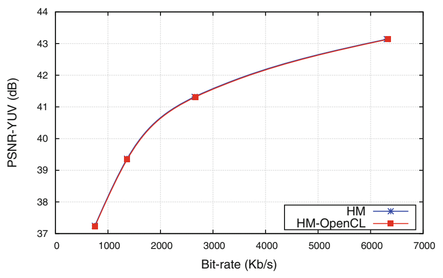
Class	Video sequence	BD-BR(%)	BD-PSNR(%)	ΔTR(%)
A	PeopleOnStreet	2.483	−0.089	51.2
B	PedestrianArea	2.805	−0.084	52.9
B	ParkScene	2.096	−0.056	53.0
B	Kimono	0.793	−0.020	53.0
Mean values		2.044	−0.062	52.5

Jiang et al. in [7] reported a time reduction on average of 37.74% using the PeopleOnStreet video sequence. Table 3 shows on average 2.48% BD-BR, −0.089% BD-PSRN and 51.2% ΔTR, using the proposed approach with the same video sequence.

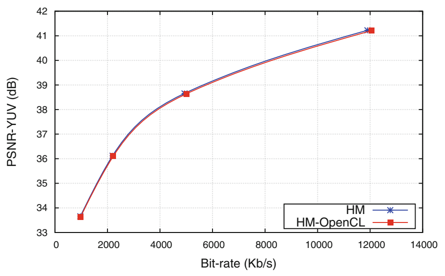
Figure 4 contains the Rate-Distortion curves calculated using the four QP values. Differences between curves are almost imperceptible.



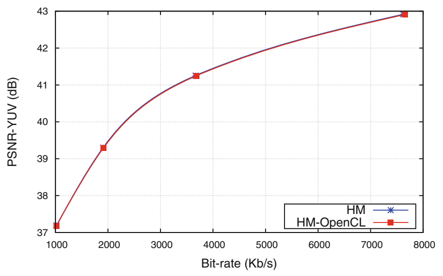
(a) PeopleOnStreet



(b) PedestrianArea



(c) ParkScene



(d) Kimono

Fig. 4. Rate-Distortion curves of the PeopleOnStreet (top-left), the PedestrianArea (top-right), the ParkScene (bottom-left) and the Kimono (bottom-right) sequences calculated using different QPs

4 Conclusions

In this paper, a parallelisation strategy is presented for reducing the computational time required for calculating motion vectors. The proposed strategy is based on OpenCL and achieves higher transfer speed by arranging work-items into a work-group during distortion measure calculations. Additionally, we introduced the use of 3D-arrays for data communication between GPU and CPU. The experimental tests have shown a significant reduction in the execution time using the proposed strategy whilst the compression efficiency suffers from a slight reduction. The parallel hardware may break data flow during the parallel processing and it may be the cause of loss of compression efficiency.

As a future work, the proposed approach will be adjusted for up-scaling in order to evaluate more CTUs at the same time. In this case, the use of a GPU with more processing units is required for mapping each 4×4 block into one processing unit (each CTU needs 256 processing units). For the communication, a dimension in the 3D-array is added in order to store a motion vector per CTU. Moreover, it will be explored the use of fast search algorithms and embedded hardware for calculating CTUs in a concurrent way.

References

1. Bjøntegaard, G.: Calculation of average PSNR differences between RD-curves. Technical report, ITU-T Video Coding Experts Group (VCEG) (VCEG-M33 2001) (2001)
2. Catanzaro, B.: OpenCL optimization case study: simple reductions (2010). http://developer.amd.com/resources/documentation-articles/articles-whitepapers/opencl-optimization-case-study-simple-reductions/. Accessed Mar 2015
3. Fuldseth, A., Horowitz, M., Xu, S., Segall, A., Zhou, M.: Tiles. Technical report JCTVC-F335, March 2011
4. Group, K: Open Computing Language (OpenCL). https://www.khronos.org/opencl/
5. Henry, F., Pateux, S.: Wavefront parallel processing. Technical report JCTVC-E196, March 2011
6. JCT-VC: HEVC Test Model (HM). https://hevc.hhi.fraunhofer.de/
7. Jiang, X., Song, T., Shimamoto, T.L.W.: High efficiency video coding (HEVC) motion estimation parallel algorithms on GPU. In: IEEE International Conference on Consumer Electronics, pp. 115–116, May 2014
8. Kim, S., Park, C., Chun, H., Kim, J.: A novel fast and low-complexity motion estimation for UHD HEVC. In: Picture Coding Symposium (PCS), pp. 105–108, December 2013
9. Lawson, H.: Parallel Processing in Industrial Real-Time Applications. Prentice Hall Series in Innovative Technology. Prentice Hall, New York (1992)
10. Luo, F., Ma, S., Ma, J., Qi, H., Su, L., Gao, W.: Multiple layer parallel motion estimation on GPU for High Efficiency Video Coding (HEVC). In: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1122–1125, May 2015
11. Misra, K., Zhao, J., Segall, A.: Entropy slices for parallel entropy coding. Technical report JCTVC-B111, July 2010
12. Monteiro, E., Maule, M., Sampaio, F., Diniz, C., Zatt, B., Bampi, S.: Real-time block matching motion estimation onto GPGPU. In: IEEE International Conference on Image Processing (ICIP), pp. 1693–1696, October 2012
13. MulticoreWare: x265 HEVC Encoder. http://x265.org/
14. Sullivan, G., Wiegand, T.: Rate-distortion optimization for video compression. IEEE Signal Process. Mag. **15**, 74–90 (1998)
15. Sullivan, G., Ohm, J., Han, W.J., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. **22**, 1649–1668 (2012)
16. Wang, F., Zhou, D., Goto, S.: OpenCL based high-quality HEVC motion estimation on GPU. In: IEEE International Conference on Image Processing (ICIP), pp. 1263–1267, October 2014
17. Wang, X., Song, L., Chen, M., Yang, J.: Paralleling variable block size motion estimation of HEVC on CPU plus GPU Platform. In: IEEE International Conference on Multimedia and Expo Workshops (ICMEW), pp. 1–5, July 2013
18. xiph.org: Xiph.org Video Test Media [derf's collection]. https://media.xiph.org/video/derf/. Accessed Mar 2015
19. Zhao, L., Guo, X., Lei, S., Ma, S., Zhao, D.: Simplified AMVP for high efficiency video coding. In: IEEE Visual Communications and Image Processing (VCIP), pp. 1–4, November 2012