GPU-Based Heterogeneous Coding Architecture for HEVC

Gabriel Cebrián-Márquez, José Luis Martínez, Pedro Cuenca, and Elche donde corresponda

Albacete Research Institute of Informatics (I3A). University of Castilla-La Mancha (UCLM). Plz. de la Universidad 2, 02071 Albacete, Spain
{Gabriel.Cebrian,JoseLuis.Martinez,Pedro.Cuenca}@uclm.es
Department of Physics and Computer Architecture.
Miguel Hernández University, 03202 Elche, Spain
{los.de.elche}@umh.es

Abstract. The High Efficiency Video Coding (HEVC) standard has nearly doubled the compression efficiency of prior standards. Nonetheless, this increase in coding efficiency involves a notably higher computing complexity that should be overcome in order to achieve real-time encoding. For this reason, this paper focuses on applying parallel processing techniques to the HEVC encoder with the aim of reducing significantly its computational cost without affecting the compression performance. Firstly, we propose a coarse-grained slice-based parallelization technique that is executed in a multi-core CPU, and then, with finer level of parallelism, a GPU-based motion estimation algorithm. Both techniques define a heterogeneous parallel coding architecture for HEVC. Results show that speed-ups of up to 4.06× can be obtained on a quad-core platform with low impact in coding performance.

Keywords: H.265, HEVC, heterogeneous, parallel encoding, GPU

1 Introduction

The new High Efficiency Video Coding (HEVC) [9] standard was established in early 2013 by the Joint Collaborative Team on Video Coding (JCT-VC), an expert group proposed by the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). With the aim of improving the coding efficiency of state-of-art coding standards, HEVC introduced several new coding tools, outperforming its predecessor H.264/Advanced Video Coding (AVC) [10] by up to 50% in terms of compression performance [13]. Among others, these tools include a highly flexible quadtree coding block partitioning structure, which divides the picture into square blocks named coding tree units (CTU). CTUs, whose size can range from 64×64 to 8×8, can be further partitioned into coding units (CU), and these, in turn, into prediction units (PU) and transform units (TU) [18, 5].

While these new coding tools and the improvements in the existing ones allow to achieve good coding efficiency results, these imply a considerable increase

in encoding time. Fortunately, this computational cost can be reduced by adapting the sequential algorithm to parallel architectures. Over the last few years, the computing industry has tended towards including several processing units on a single shared chip. In fact, in terms of massive data computing, graphics processing units (GPU) are normally used as co-processors to assist the central processing unit (CPU). Given the architectural differences between CPUs and GPUs, they form what it is known as a heterogeneous computing platform [19].

As an aid to this parallelism, HEVC places special emphasis on a hardware friendly design and parallel-processing architectures. These parallelization approaches are Tiles [12] and Wavefront Parallel Processing (WPP) [8]. On the other hand, this standard has also maintained the concept of slices from previous standards. Whereas slices were first intended for error resilience by allowing resynchronization after data losses, they can also be used for parallelization purposes. Basically, all of these three parallelization techniques rely on creating picture partitions that break some dependencies for prediction, context-adaptive binary arithmetic coding (CABAC) modeling, and/or slice header overhead. As a result, coding losses may appear.

At this point, this paper proposes a heterogeneous coding architecture composed by a multi-core CPU, and a GPU as a co-processor. Firstly, the proposed algorithm makes use of slices to allow the distribution of the encoding process over different processing units. Secondly, a GPU-based motion estimation (ME) algorithm uses this device in order to efficiently perform the motion search carried out in the ME module of the encoder. The different slices are processed in parallel making use of a single GPU device, which performs its operations in a completely asynchronous way with respect to the CPU. As a result, the execution time can be notably reduced, achieving an average speed-up of 3.68× on a quad-core CPU using 4 threads with low impact on coding efficiency.

The remainder of this paper is organized as follows. Section 2 includes a technical background of the new HEVC standard, while Sect. 3 identifies the related work that is being developed about the topic. Section 4 introduces our proposed heterogeneous coding architecture. Experimental results are shown in Section 5. Finally, Sect. 6 concludes the paper.

2 Technical Background

As mentioned in the introduction of this paper, one of the main objectives of the HEVC standard is to achieve considerably higher coding efficiency compared to previous standards. In order to make this possible, HEVC describes new coding tools, but at the same time it introduces many improvements to those that were already present in its predecessors, in particular H.264/AVC. One of the most important changes affects the picture partitioning, which is now performed following a quadtree structure as shown in Fig. 1. Each picture is partitioned into 64×64 pixel square regions called CTUs, which can be recursively divided into 4 smaller sub-regions simply called CUs, whose size range from 64×64 to 8×8. These structures can be partitioned, in turn, into the so-called PUs, whose size

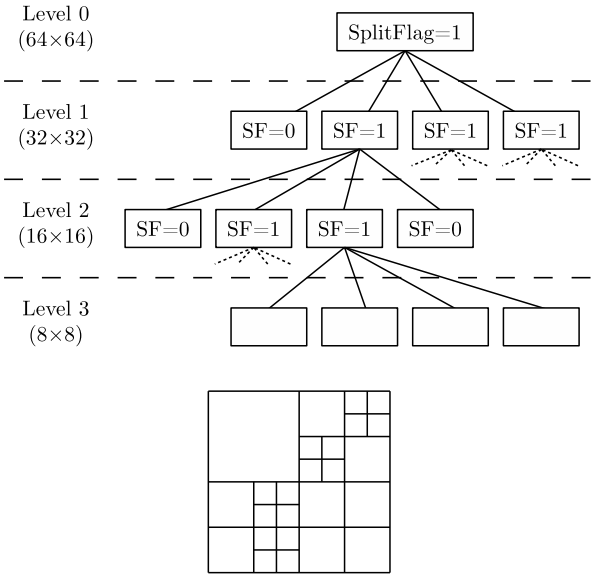
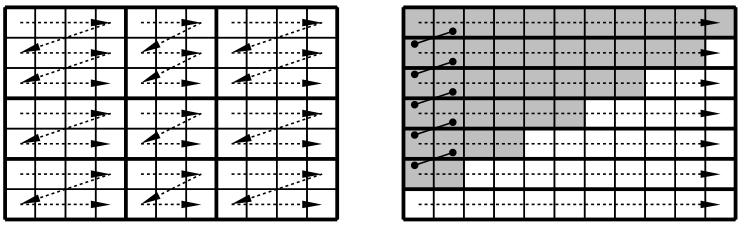


Fig. 1. Example of CTU quadtree structure defined in HEVC



(a) Tiles partitioning (b) WPP partitioning

Fig. 2. Partitioning and processing order of Tiles (a) and WPP (b)

and shape have been defined in the standard, and TUs, which are also organised in a tree structure named residual quadtree (RQT) and have a size that ranges from 32×32 to 4×4.

For intra-picture prediction, only square PUs of the same size as the parent CU can be used (2N×2N), with the exception of CUs at the maximum depth level, which can be split into four square PUs (N×N). Therefore, the PU size can range from 64×64 to 4×4. For inter-picture prediction, however, the standard defines the typical four symmetric partitions (2N×2N, 2N×N, N×2N and N×N) as well as four new asymmetric partitions (2N×nU, 2N×nD, nL×2N and nR×2N).

As can be seen, the huge amount of possibilities introduced by the standard allows the encoder to perform a very flexible coding. However, these many possible combinations involve a notable increase in the computational complexity

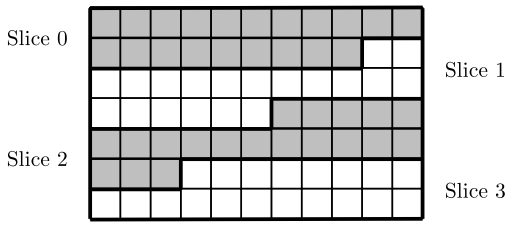


Fig. 3. Example of frame partitioning into four slices

of both the encoder and the decoder. With the aim of reducing this complexity, HEVC defines some parallelization techniques such as Tiles [12] and WPP [8]. On the one hand, the standard defines Tiles as square or rectangular shape partitions where dependencies are broken across tile boundaries, making it possible to process them independently. However, the in-loop filers can still cross these boundaries in order to improve the overall coding efficiency. An example of Tiles can be seen in Fig. 2a, in which the input picture has been divided into 9 different tiles. It should be also noted that the number of tiles can be specified for the entire sequence or it can be also changed from picture to picture. On the other hand, WPP allows the creation of rows as picture partitions that can be processed in parallel. In this case, unlike Tiles, entropy encoding and prediction are allowed to cross partitions in order to minimize coding losses. However, this is achieved by introducing a delay of two CTUs between consecutive rows in order to preserve some dependencies. For this reason, not all the processes can start the encoding at the same time, which involves lower CPU utilization at the beginning and at the end of a frame (so-called "ramping inefficiencies"). An example of this technique is shown in Fig. 2b.

Tiles and WPP have different merits and disadvantages. WPP is generally well suited for parallelizing the encoder and the decoder due to its high number of picture partitions with low compression losses. However, the amount of parallelism with Tiles can be even higher, as the number of regions into which a frame is divided may also be higher. Additionally, WPP does not introduce artifacts at partition boundaries, as is the case for Tiles. In order to simplify the implementation, Tiles and WPP cannot coexist in the same compressed video sequence, but they can be combined in some cases with the so-called slices.

A slice is a partition of an encoded frame which can be independently decoded with regard to other slices in the same frame, and they already existed in previous standards such as H.264/AVC. The size and shape of a slice can be arbitrary, although it is required that the CTUs contained in it are consecutive in coding order as shown in Fig. 3. While their main aim is to improve the error resilience of the standard by avoiding the propagation of errors across slices, they have also been used as a way of parallelism. In particular, it is possible to create completely independent slices in a frame, and thus achieve a perfect way to speed up the encoder by using multiple processing units. As a counterpart, however, each slice

introduces a header in the resulting stream that might cause some overhead in terms of coding efficiency.

3 Related Work

Several state-of-art works address the complexity analysis and parallelization strategies of the HEVC standard, such as [5, 2, 17], detailing their strengths and weaknesses. As for other parallelization techniques, most publications focus on the decoder side aiming to provide real-time decoding of high-definition (HD) and ultra high-definition (UHD) video contents [2] For example, authors in [6] and [7] present a variation of WPP called Overlapped Wavefront (OWF) in which the decoding of consecutive pictures is overlapped. In this way, when a thread finishes the processing of a CTU row, it can begin processing the next picture instead of waiting until the end of the current picture. With regard to the encoder side, it is less frequent to find works in the literature. In [14], authors propose a fine-grained parallel optimization of the ME module of the encoder, allowing to perform the motion vector prediction of all PUs available in a CU at the same time. In [16], authors show a highly parallel intra prediction algorithm for heterogeneous CPU+GPU platforms. Alternatively, other recent works focus on altering the scanning order. As an example, authors in [11] propose a diamond-based CU scanning order with the aim of obtaining a good scheme for massively parallel platforms.

Other related works make use of GPU devices in the encoding process. Authors in [20] propose a scheme similar to the one in this paper based on a GPU plus multi-core CPU, but the major shortcoming of this paper lies in the fact that they did not use the reference software, and thus the RD results provided are worse due to the fact that not all coding tools were implemented. In [15], authors propose a GPU-based ME algorithm that differs in some aspects from the one proposed in this paper. On the one hand, instead of performing a full search, the pattern used in this paper corresponds to a diamond search. This pattern may not obtain the best possible results, as not every position in the search area is checked. On the other hand, fractional ME is also performed in the device, which leads to higher speed-up values. However, the absence of motion vector predictors (MVP) may have some negative effects on the coding efficiency of the algorithm.

4 Proposed Heterogeneous Coding Architecture

As seen before, parallelism is possible in both the encoder and the decoder making use of the algorithms defined in the standard, i.e. WPP and Tiles, or other coarse-grained parallel techniques. However, these algorithms are not adequate for some parallel architectures such as those based on GPUs. This implies that it is necessary to develop efficient algorithms for these heterogeneous platforms. Fortunately, all of them can coexist and cooperate with the aim of reducing the computational complexity of the encoder.

This section will show the details of the proposed coding algorithm designed for GPU-based heterogeneous architectures. First, a coarse-grained slice-based parallelization technique will be shown, followed by a GPU-based motion estimation algorithm. After that, it will be described how both techniques can cooperate at two different levels.

4.1 Slice-Based Parallel Coding

As mentioned in Sect. 2, a slice is a partition of an encoded frame which can be independently decoded with regard to the other slices in the same frame. Similarly, slices can also be encoded in an isolated way, as there are no dependencies between them. For this reason, it is possible to exploit the inherent parallelism of these structures in such a way that every slice is encoded by a different processing unit.

Our proposed approach is to divide each frame into as many slices as available processing units. For example, if a hardware platform is composed of a quad-core CPU, each frame could be divided into four different slices as shown in Fig. 3. It has to be noted, though, that in order to homogenise the time devoted by the processing units to the encoding of the slices, each partition is assigned the same number of CTUs. In this way, the parallel efficiency of the algorithm is maximised.

In those coding configurations in which inter prediction takes place, it is required to establish a synchronization mechanism at the end of each frame. This is caused by the fact that the motion estimation is performed using the reconstructed version of the frames, and therefore, they have to be completely encoded prior to being referenced by other frames. Consequently, the decoded picture buffer (DPB) is updated with the reconstructed frame after their encoding. In order to avoid costly memory transfers, a shared memory scheme has been taken into account, in which the DPB is shared across processing units.

4.2 GPU-Based Inter Prediction Algorithm

Whereas the slice-based algorithm is able to achieve coarse-grained parallelism of the encoding process, it is still possible to further exploit the proposed heterogeneous architecture to reduce the computational complexity of the encoder. In particular, one of the most computationally expensive operations of the encoder is the ME algorithm. This operation consists in estimating the motion vectors (MV) that describe the transformation from the current frame to the reference frames with the lowest rate-distortion cost, and therefore, it is performed by subtracting the corresponding block partitions from the reference pictures, which involves a large amount of repetitive operations. Considering the nature of this algorithm (simple operations performed over multiple data), it should be noted the convenience of the GPU architecture to carry out the ME.

In order to avoid delays caused by data transfers between host and device, and the execution of the algorithm itself, this operation is performed asynchronously, i.e. the device carries out the ME while the host is devoted to other encoding

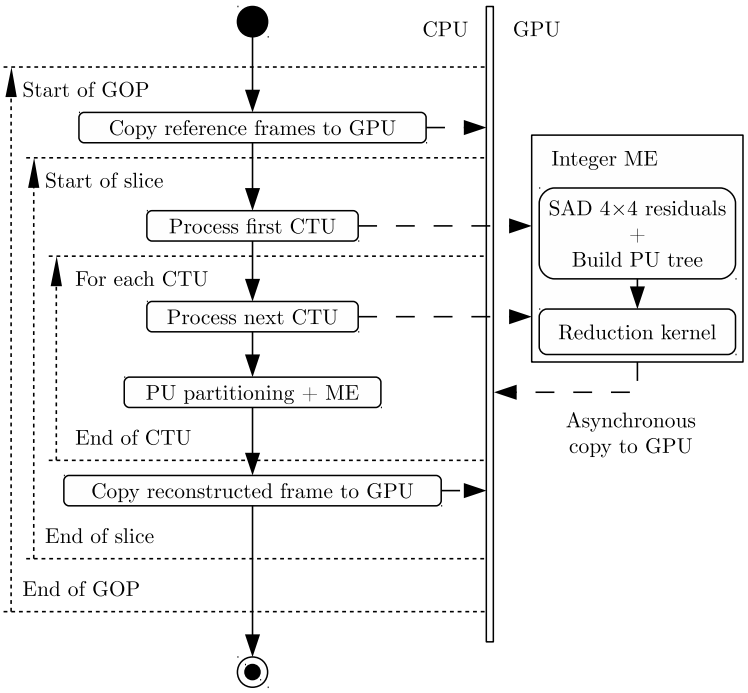


Fig. 4. GPU-based inter prediction algorithm diagram

modules. This is shown in Fig. 4. As can be seen, the original frames are copied to the device as soon as a group of pictures (GOP) starts being processed. However, whereas the blocks whose MVs are being predicted comprise a region of these pictures, the prediction is made from the reconstructed samples of the reference frames instead. For this reason, the original frames are replaced with their reconstructed versions once they are encoded (and decoded in-loop). With regard to the assignation of workload to the device, it takes place at the beginning of each CTU in which a frame is divided. At that moment, the host queues the motion estimation of the following CTU to the one that is currently being processed. In this way, host and device can cooperate asynchronously thanks to the existence of a double buffer that holds the motion information of the current CTU and, at the same time, allocates the space in which the device will copy the corresponding data. Exceptionally, for the first CTU in a frame, the host might need to wait until the device calculates the required motion information and queues the next CTU.

Similarly to the regular ME algorithm, the device calculates the MVs for all the possible PUs in which a CTU is divided and their associated costs. This is performed in two steps, and thus it is divided in two different GPU kernels. The first kernel executes the operations required to calculate the sum of absolute differences (SAD) residuals across a search area in the reference frame, while

the second one determines which MV may offer the best possible result from the aforementioned costs.

The first kernel relies on the fact that every PU size established by the standard is divisible by four, and taking into account the nature of the SAD operation, it is possible to calculate the residual information of a PU partition from the composition of its 4×4 SAD partitions. Following this approach, the previously mentioned kernel distributes a device thread to each sample in the reference search area, or in other words, each device thread is assigned a MV within the search range. Every thread is responsible for calculating all the 4×4 SAD blocks in a CTU. Once these blocks are calculated, all the running threads put them together to obtain the PU partitions into which a CTU might be divided. From another point of view, the results of this step would be equivalent to a full-search algorithm performed for every PU partition.

At this point, the device has at its disposal the prediction costs for each position in the search area and every PU. Therefore, the second kernel consists of a reduction algorithm that selects the MV that provides the lowest cost. As a result of this step, each PU is assigned only the best MV and its corresponding costs, which is the information copied to the host.

With this information, the encoder is able to skip the first step of the ME module, which corresponds to the integer ME. Instead of performing the search with integer-pel sample precision, the host only needs to obtain the MV that the device asynchronously calculated and copied beforehand, thus reducing the computational complexity of the ME. The remaining steps of the ME module are not altered, and therefore are executed after the MV is obtained.

4.3 Joint Heterogeneous Architecture

While the proposed slice-based algorithm performs a coarse-grained parallelization of the encoding, the GPU-based ME algorithm is able to speed up one of the most computationally expensive modules of the encoder. It can be noticed that both techniques work at different levels, and thus they can coexist and cooperate simultaneously. The way in which this cooperation takes place is shown in Fig. 5. As can be seen, the input frames are divided into slices as described in Sect. 4.1. In this case, however, each processing unit of the host queues a CTU in a single device, that is, multiple CTUs are processed by the device at the same time. These queries are organised in streams, so that the device can decide the order in which these CTUs are processed, or even perform the operations concurrently. As a result, the device utilization increases, and thus the relative power consumption decreases.

With regard to the scalability of the algorithm, it is only restricted by the limitations of the hardware platform in use. On the one hand, the slice-based algorithm is as scalable as the number of available processing units, and so is the joint architecture. On the other hand, while the GPU-based algorithm has been designed in a fast and efficient way, the device could limit the amount of parallelism if it is not able to process as many CTUs as required by the slice-

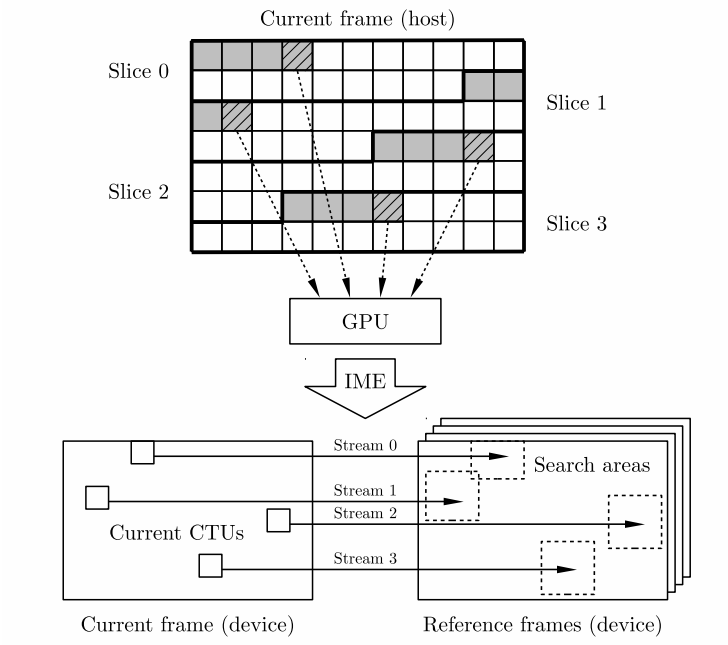


Fig. 5. Combination of the slice-based algorithm and the GPU-based proposal (4 threads)

based algorithm. In any case, it all depends on the capabilities of the hardware used.

5 Performance Evaluation

The experiments have been executed on the HEVC Test Model (HM-16.3) [1] reference software, following the guidelines and coding conditions provided by the document elaborated by the JCT-VC [4], which ensures a common framework among proposals. In this regard, Low Delay B has been the selected configuration, and the QP values being tested are 22, 27, 32 and 37. The encodings have been carried out using the Main profile and 8-bit depth. Sequences of classes C and D according to the JCT-VC classification have been used, which include the 832×480 (C) and 416×240 (D) resolutions. It should be noted, though, that the proposed algorithm can also be used with the rest of classes and coding configurations, such as Random Access.

The hardware platform used in the experiments is composed of an Intel® Core™ i7-2600 CPU running at 3.40 GHz and an NVIDIA® GeForce® GTX 560 Ti GPU running 384 CUDA cores at the frequency of 1.6 GHz. The encoder has been compiled with GCC 5.3.1 and executed on Ubuntu 16.04 (Linux 4.4.0-

Table 1. Results of the proposed algorithm using four threads

Class	Video sequence	BD-rate (%)	Encoding speed-up
C	BasketballDrill	2.12	3.91
	BQMall	2.94	3.60
	PartyScene	1.85	3.79
	RaceHorses	2.51	4.06
D	BasketballPass	4.80	3.54
	BQSquare	5.51	3.41
	BlowingBubbles	4.32	3.43
	RaceHorses	4.39	3.72
	Mean values	3.55	3.68

22). Turbo Boost and Hyper-Threading have been disabled to achieve the reproducibility of the results.

With regard to the parametrization of the encoder, the sample adaptive offset (SAO) filter has been disabled, as well as the *LFCrossSliceBoundary* flag, which allows loop filters to cross slice boundaries. This can be justified by the fact that slices should be completely independent for parallel processing, but these filters introduce some dependencies in the encoding process, so they need to be disabled for the proposed algorithm. In the case of the proposed algorithm, the encoder is executed using 4 threads and enabling the GPU device.

The results will be provided in terms of speed-up and Bjøntegaard delta rate (BD-rate). The BD-rate is a measure of coding efficiency that represents the percentage of bit rate variation between two sequences with the same objective quality [3]. Therefore, a negative value implies that the proposal is able to improve the coding efficiency of the baseline encoder.

Table 1 shows the results of the proposed GPU-based heterogeneous coding architecture compared to the baseline encoder with just one slice per frame. As can be seen, the obtained BD-rate is directly related to the resolution of the input sequence. This increase in bit rate is caused by the overhead introduced by the slice headers, as well as some other indirect effects in terms of prediction and entropy coding. For this reason, smaller resolutions also involve smaller slices, and thus more probabilities of broken dependencies. This behaviour, however, is inherent of the slice partitioning, and is not related to the proposed parallelization scheme. In terms of speed-up, the results show that the achieved parallel efficiency is considerably high, obtaining values that are even greater than the maximum theoretical of 4× for the slice-based algorithm thanks to the use of the GPU-based platform. It has to be noted, though, that it is not possible to reach that theoretical speed-up in some cases. This is caused by the fact that not all the slices take the same amount of time to be encoded, not even if all of them contain the same number of CTUs.

6 Conclusions

The HEVC standard has opened the door to a great number of new applications and video formats such as UHD with the aim of fulfilling the demands of the market. However, the improved coding efficiency of HEVC has also led to a notable increment in the computational complexity of the encoder. In order to achieve real-time encoding, it has become a requirement to develop fast and efficient coding algorithms and techniques. Accordingly, this paper proposes a GPU-based heterogeneous architecture for HEVC which combines a coarse-grained parallelization technique and a GPU-based ME algorithm. While the former is able to achieve coarse-grained parallelism by removing existing dependencies, the latter speeds up the most computationally expensive module of the encoder. As a result, the computational complexity of the encoder is reduced.

An experimental evaluation of the algorithm has shown that the encoding can be sped up by 3.68× on average with four threads, showing efficient utilization of resources of the GPU-based heterogeneous platform. As a consequence, there is a low impact in coding efficiency of 3.55% in BD-rate.

Acknowledgments. This work was jointly supported by the Spanish Ministry of Economy and Competitiveness (MINECO) and the European Commission (FEDER funds) under the project TIN2015-66972-C5-2-R, and by the Spanish Ministry of Education, Culture and Sports under the grant FPU13/04601.

References

1. HEVC Test Model (HM) Reference Software. `https://hevc.hhi.fraunhofer.de/`
2. Álvarez-Mesa, M., Chi Ching Chi, Juurlink, B., George, V., Schierl, T.: Parallel Video Decoding in the Emerging HEVC Standard. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1545–1548 (Mar 2012)
3. Bjøntegaard, G.: Calculation of Average PSNR Differences between RD-Curves. Tech. Rep. VCEG-M33, ITU-T Video Coding Experts Group (VCEG) (2001)
4. Bossen, F.: Common Test Conditions and Software Reference Configurations. Tech. Rep. JCTVC-L1100 (Jan 2013)
5. Bossen, F., Bross, B., Shring, K., Flynn, D.: HEVC Complexity and Implementation Analysis. IEEE Trans. Circuits Syst. Video Technol. 22(12), 1685–1696 (Dec 2012)
6. Chi Ching Chi, Álvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., Schierl, T.: Parallel Scalability and Efficiency of HEVC Parallelization Approaches. IEEE Trans. Circuits Syst. Video Technol. 22(12), 1827–1838 (Dec 2012)
7. Chi Ching Chi, Álvarez-Mesa, M., Lucas, J., Juurlink, B., Schierl, T.: Parallel HEVC Decoding on Multi- and Many-core Architectures. J. Sign. Process. Syst. 71(3), 247–260 (Jun 2013)
8. Henry, F., Pateux, S.: Wavefront Parallel Processing. Tech. Rep. JCTVC-E196 (Mar 2011)
9. ISO/IEC, ITU-T: High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2 (version 3) (Apr 2015)

10. ISO/IEC, ITU-T: Advanced Video Coding for Generic Audiovisual Services. ITU-T Recommendation H.264 and ISO/IEC 14496-10 (version 10) (Feb 2016)
11. Łuczak, A., Karwowski, D., Maćkowiak, S., Grajek, T.: Comput. Vis. and Graphics, chap. Diamond Scanning Order of Image Blocks for Massively Parallel HEVC Compression, pp. 172–179. Springer Berlin Heidelberg, Berlin, Heidelberg (Sep 2012)
12. Misra, K., Segall, A., Horowitz, M., Shilin Xu, Fuldseth, A., Minhua Zhou: An Overview of Tiles in HEVC. IEEE J. Sel. Topics Signal Process. 7(6), 969–977 (Dec 2013)
13. Ohm, J.R., Sullivan, G.J., Schwarz, H., Thiow Keng Tan, Wiegand, T.: Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC). IEEE Trans. Circuits Syst. Video Technol. 22(12), 1669–1684 (Dec 2012)
14. Qin Yu, Liang Zhao, Siwei Ma: Parallel AMVP Candidate List Construction for HEVC. In: IEEE Visual Communications and Image Processing (VCIP). pp. 1–6 (Nov 2012)
15. Radicke, S., Hahn, J.U., Grecos, C., Qi Wang: A Highly-Parallel Approach on Motion Estimation for High Efficiency Video Coding (HEVC). In: IEEE International Conference on Consumer Electronics (ICCE). pp. 187–188 (Jan 2014)
16. Radicke, S., Hahn, J.U., Qi Wang, Grecos, C.: A Parallel HEVC Intra Prediction Algorithm for Heterogeneous CPU+GPU Platforms. IEEE Trans. Broadcast. 62(1), 103–119 (Mar 2016)
17. Saleh, M.A., Hashim, H., Tahir, N.M., Hisham, E.: Review for High Efficiency Video Coding (HEVC). In: IEEE Conference on Systems, Process and Control (ICSPC). pp. 141–146 (Dec 2014)
18. Sullivan, G.J., Ohm, J.R., Woo-Jin Han, Wiegand, T.: Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Trans. Circuits Syst. Video Technol. 22(12), 1649–1668 (Dec 2012)
19. Wu-Feng, Manocha, D.: High-Performance Computing Using Accelerators. Parallel Comput. 33(10-11), 645–647 (Oct 2007)
20. Xiangwen Wang, Li Song, Min Chen, Junjie Yang: 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 (Jul 2013)