J Supercomput (2015) 71:613–628
DOI 10.1007/s11227-014-1313-8

Accelerating HEVC using heterogeneous platforms

Gabriel Cebrián-Márquez · José Luis Hernández-Losada · José Luis Martínez · Pedro Cuenca · Minhao Tang · Jiangtao Wen

Published online: 18 October 2014

Abstract The high efficiency video coding (HEVC) standard achieves double compression efficiency compared with H.264/advanced video coding at the cost of huge computational complexity. Parallelizing HEVC encoding is an efficient way of fulfilling this computational requirement. The parallelization algorithms considered in HEVC, such as Tiles or wavefront parallel processing (WPP), rely on creating picture partitions that can be processed concurrently in a multi-core architecture. However, this paper focuses on the design of a heterogeneous parallel architecture composed of a graphic processing unit (GPU) plus a multi-core central processing unit (CPU) to take advantage of these techniques. Experimental results indicate that our approach outperforms WPP in terms of speed-up and reduces the delay introduced by alterna-

This work has been jointly supported by the Spanish Ministry of Economy and Competitiveness (MINECO) and the European Commission (FEDER funds) under the project TIN2012-38341-C04-04.

G. Cebrián-Márquez (✉) · J. L. Hernández-Losada · J. L. Martínez · P. Cuenca
Albacete Research Institute of Informatics (I3A),
University of Castilla-La Mancha, Albacete, Spain
e-mail: Gabriel.Cebrian@uclm.es

J. L. Hernández-Losada
e-mail: JoseLuis.Hernandez2@alu.uclm.es

J. L. Martínez
e-mail: JoseLuis.Martinez@uclm.es

P. Cuenca
e-mail: Pedro.Cuenca@uclm.es

M. Tang · J. Wen
Department of Electrical Engineering, Tsinghua University, Beijing, China
e-mail: tmh920811@163.com

J. Wen
e-mail: jtwen@tsinghua.edu.cn

tive techniques such as the group of pictures-based processing pattern. Moreover, the proposed algorithms obtain speed-up values of over 4× on an Intel quad-core CPU and an NVIDIA GPU with negligible quality losses.

Keywords HEVC · Parallelization · GPU · Multicore · Heterogeneous computing

1 Introduction

The new high efficiency video coding (HEVC) standard [3] has recently been 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). HEVC was initially conceived with the purpose of achieving adequate efficiency and performance to deliver high-quality multimedia services over bandwidth-constrained networks, and also to support formats beyond high definition (HD) resolution, such as the new 4 and 8K formats. This standard is based on a well-known block-based hybrid video coding architecture, as was its predecessor, namely H.264/MPEG4 part 10—advanced video coding (AVC) [9], which it outperforms in terms of bitrate reduction at the same quality [12]. Among other features, HEVC incorporates multiple new coding tools, such as highly flexible quad-tree coding block partitioning, which includes new concepts such as coding unit (CU), prediction unit (PU) and transform unit (TU) [2,12].

All these improvements imply a considerable increase in the encoding time. Fortunately, this computational cost can be efficiently reduced by adapting the sequential algorithm to parallel architectures. Over the past 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, there are devices called graphic processing units (GPUs) which are normally used as co-processors to assist the central processing unit (CPU). CPUs and GPUs have different instruction set architectures, forming what it is known as a heterogeneous computing platform [7].

As an aid to this parallelism, HEVC places special emphasis on a hardware-friendly design and parallel-processing architectures. These parallelization approaches are *Tiles* [11] and wavefront parallel processing (WPP) [8], and these will be described in Sect. 2. Basically, these parallelization algorithms rely on creating picture partitions that break some dependencies for prediction, context-adaptive binary arithmetic coding (CABAC) modelling, and/or slice header overhead. As a result, coding losses may appear.

At this point, this paper proposes a GPU-based algorithm that makes use of this device in order to efficiently parallelize the motion estimation (ME) carried out in the HEVC inter-prediction algorithm. This algorithm is completely asynchronous from the CPU, in such a way that it can execute operations from other modules, resulting in lower execution times. Additionally, in terms of coding efficiency, this algorithm makes use of a full search pattern, providing even better Bjøntegaard Delta rate (BD-rate) results than the reference algorithm in some cases.

Furthermore, this algorithm can be combined with multiple coarse-grained parallelization techniques such as the aforementioned ones in a heterogeneous architec-

ture. In fact, this paper shows the results of combining this GPU-based proposal with two different parallelization techniques: WPP and a group of pictures (GOP) based coarse-grained algorithm used in previous standards. These algorithms allow a multicore system to process multiple coding tree units (CTUs) by splitting the frame in rows or the sequence in GOPs, respectively. In either case, the motion estimation of these regions is issued to the device, reducing the idle time of the whole architecture.

The proposed architecture is tested by comparing their results with the ones provided by the HEVC Test Model (HM) [10], outperforming them in terms of speed-up and coding efficiency. Results show speed-ups of up to 4.78× on a quad-core CPU using four threads plus simultaneous multithreading (SMT) with negligible rate-distortion (RD) penalties. Additionally, Radicke et al. [13] propose another GPU-based algorithm that makes use of a diamond search. Compared with this another proposal, our algorithm achieves better BD-rate values, ranging from 0.8 to 1.8 %.

The remainder of this paper is organized as follows: Sect. 2 includes the technical background of the new HEVC standard, while Sect. 3 identifies the related work which is being carried out on the topic. Section 4 introduces our proposed architecture. Experimental results are shown in Sect. 5. Section 6 concludes the paper and includes some possible lines of action for future work.

2 Technical background

As mentioned in the previous section, the main target of HEVC is to achieve lower bitrates for video streams while maintaining the same observable quality. In order to make this possible, HEVC introduces new coding tools with respect to its predecessor, H.264/AVC, and these enable a notable increase in coding efficiency. One of the most important changes affects picture partitioning, which is now performed following a quadtree structure. HEVC dispenses with the terms macro-block (MB) and block for the ME and the transform, respectively, and introduces three new concepts: CU, PU and TU. This structure leads to a flexible coding to suit the particularities of the frames. Each picture is partitioned into 64 × 64 pixel. pixel square regions called CTUs, which replace the MB structure of previous standards. Each CTU can be recursively divided into four smaller sub-areas called CUs, whose size ranges from 8 × 8 to 64 × 64 pixel. These regions may contain one or several PUs and TUs. This quadtree structure is shown in Fig. 1, in which each node represents one of the four CUs in which a CTU might be divided. For each level (current size of the CU), the split flag (SF) indicates whether it has been decided to split the corresponding CU or not.

For intra-picture prediction, a PU uses the same 2N × 2N size as that of the CU to which it belongs, allowing it to be split into quad N × N PUs only for CUs at the maximum depth level. Therefore, the PU size ranges from 64 × 64 to 4 × 4 pixels. For inter-picture prediction, several rectangular block shapes (square and non-square) are available, defining eight different PU sizes (2N × 2N, 2N × N, n × 2N, N × N, 2N × U, 2N × D, nL × 2N, nR × 2N). The prediction residual obtained in each PU is transformed using the residual quad tree (RQT) structure, which supports various TU sizes from 32 × 32 to 4 × 4. For the transform of intra 4× 4 PU residuals, an integer approximation of the discrete sine transform (DST) is used instead.

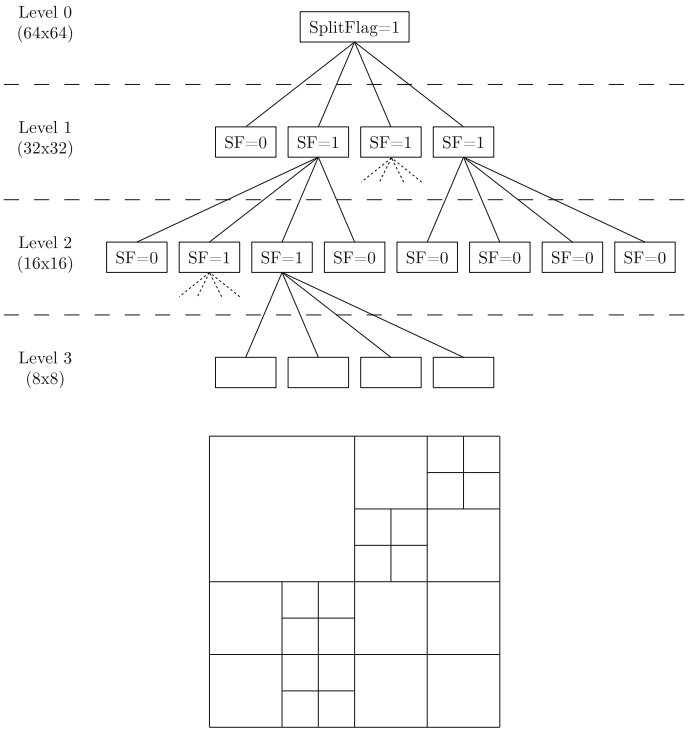
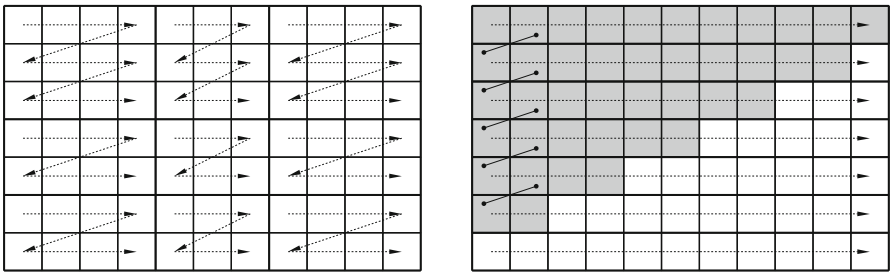


Fig. 1 CTU quadtree structure partitioning

High efficiency video coding (HEVC) checks most of the PUs (inter and intra modes) to decide whether it should split a CU or not, depending on the best RD case. Furthermore, in the case of inter prediction, for each of these PU partitions an ME algorithm is called. The wide range of possibilities makes this module much more computationally expensive than its predecessor, the one in the H.264/AVC standard. HEVC introduces changes in other modules too, such as intra prediction (where a total of 35 different coding modes can be selected), the new PU modes (asymmetric modes), additional image filters or new transform sizes, among others. As expected, the selection of the optimal partitioning for each CU/PU/TU is an intensive time-consuming process due to the huge number of combinations that have to be evaluated.

With the aim of reducing this huge complexity, the new HEVC codec also includes new parallelization techniques such as Tiles [11] and WPP [8]. On the one hand, Tiles are square or rectangular shape partitions where dependencies are broken across tile boundaries [11], making it possible to process them independently, taking into account that coding losses may appear. The in-loop filters (deblocking and sample adaptive offset, SAO), however, can still cross these boundaries. The number of tiles and their location can be defined for the entire sequence or changed from picture to picture. On the other hand, WPP allows the creation of picture partitions (rows) that can be processed in parallel. Unlike Tiles, entropy encoding and prediction are allowed to cross partitions in order to minimize coding losses. Nevertheless, coding dependencies



(a) Tiles partitioning **(b)** WPP partitioning

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

make it necessary to have a delay of at least two CUs between consecutive rows, in a similar way to segmentation in a computer architecture [5,8]. For this reason, not all the processes can start encoding these rows at the same time, which means low CPU utilization at the beginning and at the end of a frame, thus incurring the so-called "ramping inefficiencies". Both techniques are depicted in Fig. 2, in which an example of partitioning is shown for Tiles and WPP, respectively, as well as the corresponding processing order of the inner CUs.

Tiles and WPP have different merits and disadvantages. WPP is generally well suited for the parallelization of 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 vary. Additionally, WPP does not introduce artifacts at partition boundaries, as is the case for Tiles. In order to simplify the implementation, it is not possible to use Tiles and WPP simultaneously in the same compressed video sequence.

In either case, these approaches need parallel architectures to exploit their potential and, hence, to reduce the computational complexity of HEVC. In this respect, new architectures composed of multi-core CPUs and GPUs are being introduced in high-performance computing. A multi-core processor is composed of several processors sharing the same chip, while GPUs are composed of hundreds of similar simple processing cores, designed and organized with the goal of achieving high performance. These cores are grouped into stream processors that perform single instruction multiple data (SIMD) operations, which are suitable for arithmetic intensive applications. The main feature of these devices is a large number of processing elements integrated on a single chip at the expense of a significant reduction in cache memory.

3 Related work

As far as the related work in the literature is concerned, there have been many approaches focusing on accelerating different modules of the H.264/AVC encoding algorithm by means of parallel computing [4,14,16]. However, in the framework of HEVC, the first parallel approaches were focused on reducing the complexity of the decoding algorithm; in [5], the authors improve the WPP approach. The idea consists

of once there are no available rows in the current picture, the next one starts being processed. In this way, the ramping inefficiencies of WPP can be mitigated by overlapping the execution of consecutive pictures. This proposal was called overlapped wavefront (OWF). As a limitation, search areas need to be constrained to the region of the reference frame that has been already reconstructed.

In the context of this paper, which is more focused on the encoder side, there are not many approaches. OWF might work for the encoder, but no results were given in [5]. Yu et al. [18] proposed in a parallel candidate list to parallelize the motion vector prediction, but the proposal is not standard compliant. Later, in [17], the authors reduced the encoding time by up to 13 times using a 64-core architecture, which is far less accessible than the one used in this paper. Finally, Wang et al. proposed in [15] 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 HM reference software [10] and, thus, the RD results provided are worse due to the fact that not all coding tools were implemented [15].

In [13], the authors propose a GPU-based motion estimation 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 motion estimation is also performed on 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 algorithms

As seen before, parallelization is possible in both the encoder and the decoder using the algorithms defined in the standard, among others. Nonetheless, these are designed to be executed on a multi-core CPU, taking advantage of the capabilities that multiple threads may offer, but not taking into account other devices. Heterogeneous architectures such as the ones formed by the association of a multi-core CPU and a GPU are utilized in this paper, making use of the immeasurable power they can provide. The combination of a GPU-based motion estimation algorithm and two coarse-grained parallelization techniques is described in the following subsections.

4.1 GPU-based inter prediction algorithm

As motion estimation is the most resource intensive operation on the encoder side [12], this algorithm aims to reduce the time spent on the CPU by performing these searches on a GPU device. Nevertheless, taking into account that data transfers between host and GPU are highly time-consuming, these operations are performed asynchronously. In this way, time spent on the integer motion estimation (IME) is negligible compared with the default search algorithm.

As soon as a GOP starts being processed, it is possible to transfer the original frames that will be encoded to the device, making them available for subsequent uses. Later

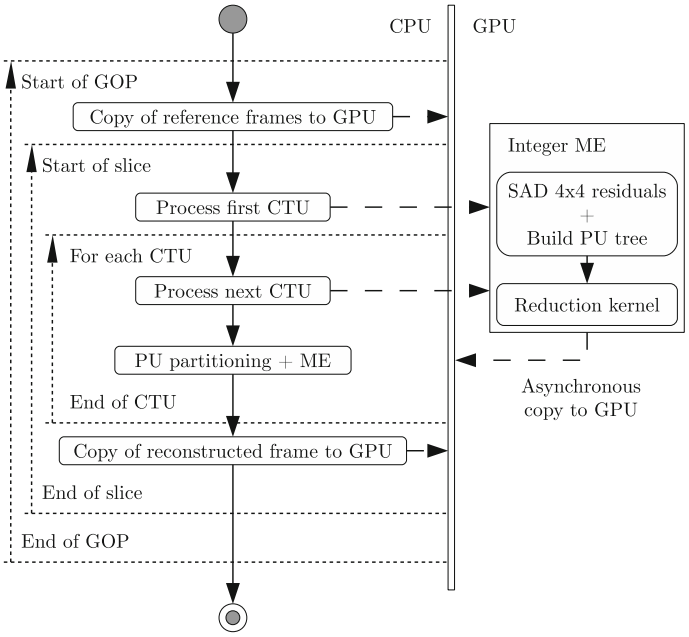


Fig. 3 GPU-based interprediction algorithm activity diagram

on, these frames are updated with their reconstructed version when they are encoded (and decoded in-loop) to correctly carry out motion estimation on the device. These operations are shown in Fig. 3.

When the encoder starts processing a slice, the host queues the execution of two consecutive kernels that perform the integer motion estimation of every PU partition in the first CTU. 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 motion vector (MV) may offer the best possible result.

This algorithm 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 per sample in the reference search area. Every thread is responsible for calculating all the 4×4 SAD blocks in a CTU, taking as MV its position in the search area. 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 second kernel runs a reduction algorithm over the residual data obtained from the first one, so that the result of the GPU algorithm is a single table containing the best MV for every PU partition, which is copied asynchronously to the

host. After the transfer is finished, motion search operations related to the next CTU are then issued to the device.

By the time the host needs to perform the motion estimation of the CTU, integer MVs should be ready to be queried, it only being necessary to perform fractional motion estimation (FME) of the PU partitions which have not been skipped by the encoder.

4.2 Joint algorithms

As a consequence of the computational limit of a single processor, the idea of having multiple cores on the same chip was successfully introduced some time ago. One of its most relevant benefits is that a parallel application can achieve speed-up values in direct proportion to the number of cores. This, along with the ever-expanding existence of this kind of architectures, motivated the JCT-VC to include parallelism in HEVC, which was implemented by breaking some dependencies while trying to provide as much coding efficiency as possible.

These new techniques, however, have some issues that make them inadequate in some particular situations. Nevertheless, many algorithms used in previous standards are still applicable to HEVC. In this way, this subsection will present two different joint algorithms, one based on WPP and another one on a traditional GOP-based division pattern. This will show that our GPU-based proposal can be combined with many coarse-grained parallelization techniques.

4.2.1 WPP + GPU-based inter prediction

Unlike Tiles, WPP is able to keep some dependencies across partition boundaries, thus obtaining very good coding efficiency results. Even though this might have an effect on the maximum speed-up that it can achieve [5], WPP has been the algorithm chosen as the basis for the first of our joint proposals, aiming to achieve a trade-off between both parameters.

In our heterogeneous architecture, both the multi-core CPU and the GPU algorithms are independent. While WPP performs a coarse-grained parallelization of the whole encoding process, our GPU-based algorithm is able to carry out the IME operation. This independence makes it possible to combine these algorithms in a single proposal, thus obtaining higher speed-up values at the expense of a negligible increment in coding efficiency losses.

As depicted in Fig. 4, WPP allows multiple threads to process several CTUs concurrently (shaded regions). A single GPU device can carry out the integer motion estimation of these CTUs, queueing several kernels into the GPU. In particular, all these CTUs are located in the current frame and share the same reference list, i.e. the search areas can be found in the same frame, as shown in the figure. As a result of processing multiple CTUs, the device is fully utilized, lowering idle times. In addition, the GPU can process different kernels independently of the CPU in such a way that the host can continue processing other modules concurrently.

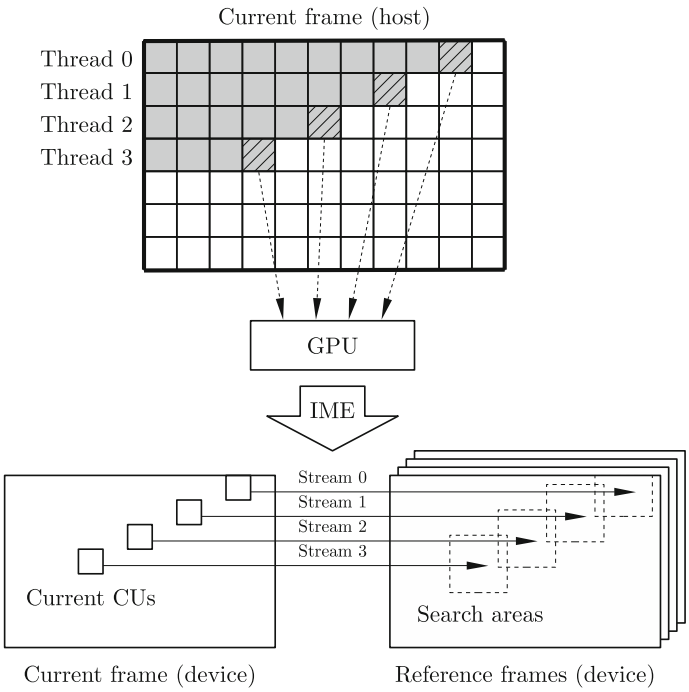


Fig. 4 Combination of WPP and the GPU-based proposal (4 threads)

4.2.2 GOP-based pattern + GPU-based inter prediction

Just as in previous video coding standards, HEVC offers an independent GOP-based encoding pattern into which a sequence can be divided. In this way, this second joint algorithm consists in assigning each GOP to a different thread, taking into account that no dependencies have to be broken between them. Due to this independence, the RD results obtained by this algorithm are very similar to the ones from the sequential algorithm. Moreover, this architecture is scalable to any number of cores.

The proposed architecture has a module splitter and joiner to allocate tasks to available cores and receive their results. Moreover, a dynamic module has been included to set up the execution schedule of the threads, taking into account that GOPs can have different delays. This is mainly because (1) the encoding algorithm can be carried out without following a sequential order and (2) different threads may have different time requirements because of variable residual data and, thus, time spent to encode it may diverge for each one.

Figure 5 shows how the GOPs into which a sequence is divided are issued to every thread. In this way, each thread processes a different frame and, hence, the motion estimation is performed on different CTUs. Therefore, the corresponding search areas are not located in the same frame. However, in a similar way to the previous joint algorithm, a single GPU device carries out the integer motion estimation of multiple threads, filling up its occupancy and taking full advantage of the available resources.

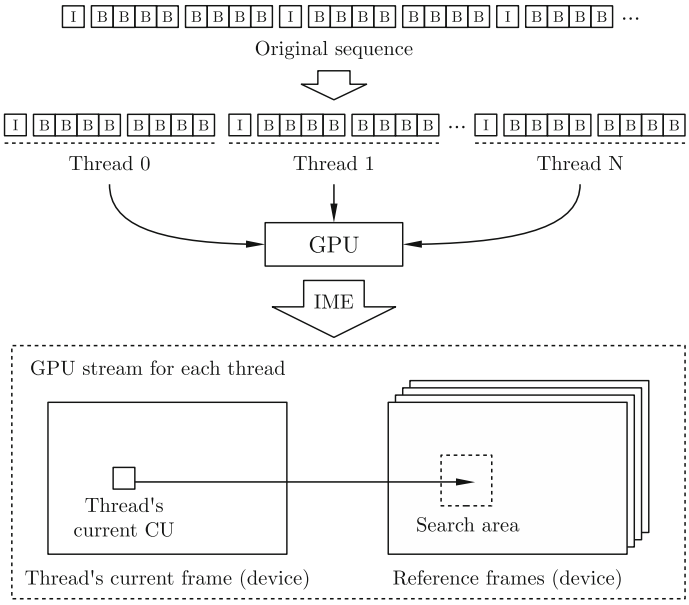


Fig. 5 Combination of the GOP-based algorithm and the GPU-based proposal

Once again, these operations are performed asynchronously, so that the CPU can keep carrying out other operations of the encoding process.

5 Performance evaluation

In order to ensure a common framework, the JCT-VC group defined a document [1] in which some test conditions are set out to homogenize comparisons between experiments. Therefore, this performance evaluation has been carried out in accordance with these guidelines, including the defined sequences and configuration parameters. This document also defines the Bjøntegaard Delta metric (BD-rate) used to measure the coding efficiency obtained by a proposal.

Random access (RA) with a color sub-sampling of 4:2:0 and 8-bit encoding was the configuration chosen to carry out the evaluation, as it is the most widely used configuration in real scenarios, but any other alternative might also work with our proposed algorithm. The selected quantization parameter (QP) values were {22, 27, 32, 37}, but only their average results will be presented. No other changes were made to the default parameters.

In this scenario, the aforementioned document [1] defines the set of video sequences to be encoded, which are grouped into classes according to their resolution, with the exception of Class F. This parameter will be relevant when presenting the results achieved by WPP.

- Class A (2,560 × 1,600): *Traffic*, *PeopleOnStreet*.
- Class B (1,920×1,080): *Kimono*, *ParkScene*, *Cactus*, *BasketballDrive*, *BQTerrace*.

- Class C (832×480): *BasketballDrill*, *BQMall*, *PartyScene*, *RaceHorses*.
- Class D (416×240): *BasketballPass*, *BQSquare*, *BlowingBubbles*, *RaceHorses*.
- Class F: *BasketballDrillText* (832 × 480), *ChinaSpeed* (1,024 × 768), *SlideEditing* (1,280×720), *SlideShow* (1,280×720).

The sequential algorithm of HM version 10.0 [10] has been used as the reference algorithm to calculate the corresponding speed-up and coding efficiency values. The proposed GPU-based algorithm has been tested separately to calculate its influence on the overall processing time. Later, the results achieved by the combination of this algorithm along with WPP and the GOP-based algorithm are also presented, aiming to show that the GPU-based proposal can be combined with several types of coarse-grained algorithms.

All the measurements have been performed on a quad-core Intel Core i7-2600 CPU running at 3.40 GHz and an NVIDIA GTX 560 Ti GPU running 384 CUDA cores at the frequency of 1.6 GHz. Consequently, tests have been carried out with 2 and 4 threads, as well as 4 plus SMT, enabling the processor to execute eight threads.

To start with, Table 1 shows the results of the proposed GPU-based algorithm. As can be seen, performing the IME operation on the GPU involves accelerating the encoding process by 1.12× while incurring very low coding efficiency losses (due to MV prediction), or even improving it in some cases. This is because the proposed algorithm performs a more exhaustive search. It is necessary to emphasize that these results are the theoretical limit of the integer ME, as the GPU has already calculated every MV when the host needs to perform this operation. In other words, the IME is performed in virtually perfect time.

On the other hand, Table 2 shows a comparison between the results provided by the combination of WPP and the GPU-based algorithm, and the ones provided by WPP itself. As can be seen, the proposal can reach speed-up values close to the ones from a parallel efficient algorithm (i.e. threads are almost fully utilized), providing that the frame size is large enough to exploit the available parallelism (see Class A). Accordingly, WPP obtains lower speed-up results with lower resolutions (see class D).

These results also show that combining both WPP and the GPU-based algorithm surpasses the results of WPP in terms of speed-up, reaching values of up to 4.33× on average (for Class A) compared with 3.92×, respectively. This behaviour is also present in other classes besides Class A, although the achieved speed-up values may be lower due to the aforementioned limitations of WPP. With regard to coding efficiency, this time reduction has a negligible impact of 1.3 % average BD-rate.

Table 3, in turn, shows the same comparison on the basis of the GOP-based parallelization technique. Unlike WPP, it can be seen that this other algorithm does not depend on the dimensions of the sequence, i.e. every class is able to achieve the same amount of parallelism. Moreover, as there are no dependencies between GOPs, and hence between threads, this algorithm is almost perfectly scalable.

Once again, it is shown that the combination of a coarse-grained algorithm, such as the GOP-based pattern, and the GPU-based motion estimation module outperforms the sole utilization of the former. In this way, this joint version can reach speed-ups of 4.46× on average (for Class A), compared with 4.13×, respectively. Moreover, it

Table 1 Speed-up and coding efficiency results of the GPU-based proposal

Class	Video sequence	Speed-up	BD-rate (%)
A	Traffic	1.07	0.2
	PeopleOnStreet	1.17	−0.5
B	Kimono	1.14	0.5
	ParkScene	1.07	0.0
	Cactus	1.11	0.2
	BasketballDrive	1.19	3.5
	BQTerrace	1.08	−1.0
C	BasketballDrill	1.14	−0.5
	BQMall	1.11	0.8
	PartyScene	1.09	−0.3
	RaceHorses	1.20	0.0
D	BasketballPass	1.15	0.0
	BQSquare	1.06	−0.1
	BlowingBubbles	1.07	−0.4
	RaceHorses	1.16	−0.5
F	BasketballDrillText	1.13	−0.7
	ChinaSpeed	1.15	−3.1
	SlideEditing	1.05	3.0
	SlideShow	1.08	4.3
	Class A	1.12	−0.2
	Class B	1.12	0.7
	Class C	1.14	0.0
	Class D	1.11	−0.2
	Class F	1.10	0.9
	Total average	1.12	0.2

achieves speed-up values of up to 4.78×. The average BD-rate of this joint algorithm stands at 0.6%.

As can be seen, the results of both joint algorithms can be connected with the ones from Table 1, as the difference in speed-up and BD-rate compared with the GPU-based algorithm by itself stands at around 1.10× and 0.2%, respectively. This means that the device is almost fully utilized, taking advantage of its resources and potential.

Choosing one or another coarse-grained parallelization technique will depend on the scenario in which the HEVC encoder is involved, e.g. WPP is adequate for large resolutions, while the GOP-based algorithm is ideal for transcoding operations. In either case, our GPU-based proposal is able to improve upon the achieved speed-up results with negligible coding efficiency penalties.

5.1 Comparison with related works

As stated in Sect. 3, Radicke et al. [13] propose a GPU-based motion estimation algorithm that is substantially different from the one presented in this paper. This subsection aims to evaluate these differences in terms of speed-up and coding efficiency.

Table 2 Speed-up and BD-rate results comparison between WPP and its corresponding joint algorithm

Class	Video sequence	Speed-up						BD-rate (%)	
		2 threads		4 threads		4 th. + SMT			
		WPP	w/GPU	WPP	w/GPU	WPP	w/GPU	WPP	w/GPU
A	Traffic	1.88	1.99	3.37	3.56	3.90	4.12	0.7	0.9
	PeopleOnStreet	1.89	2.17	3.35	3.85	3.95	4.53	0.7	0.1
B	Kimono	1.89	2.13	3.36	3.80	3.80	4.31	1.2	1.7
	ParkScene	1.88	2.01	3.33	3.56	3.70	3.97	0.7	0.7
	Cactus	1.89	2.07	3.31	3.65	3.72	4.13	1.1	1.4
	BasketballDrive	1.90	2.22	3.40	3.98	3.76	4.43	1.5	5.0
	BQTerrace	1.88	2.00	3.31	3.53	3.79	4.06	1.2	0.1
C	BasketballDrill	1.80	2.00	2.73	3.09	2.72	3.09	1.4	1.1
	BQMall	1.81	1.96	2.83	3.11	2.83	3.09	1.5	2.3
	PartyScene	1.78	1.91	2.70	2.94	2.71	2.96	0.6	0.2
	RaceHorses	1.78	2.09	2.79	3.30	2.84	3.35	0.8	0.8
D	BasketballPass	1.67	1.88	1.75	2.01	1.75	2.01	0.9	0.9
	BQSquare	1.60	1.66	1.84	1.92	1.84	1.92	1.3	1.3
	BlowingBubbles	1.59	1.66	1.80	1.91	1.80	1.90	0.9	0.6
	RaceHorses	1.63	1.85	1.81	2.10	1.81	2.10	0.9	0.4
F	BasketballDrillText	1.79	1.97	2.71	3.05	2.70	3.05	1.4	0.7
	ChinaSpeed	1.86	2.12	3.15	3.54	3.28	3.74	0.8	-2.3
	SlideEditing	1.80	1.86	3.13	3.24	3.33	3.46	1.0	3.9
	SlideShow	1.80	1.92	3.00	3.21	3.24	3.45	2.2	6.7
	Class A	1.88	2.08	3.36	3.71	3.92	4.33	0.7	0.5
	Class B	1.89	2.08	3.34	3.70	3.75	4.18	1.1	1.8
	Class C	1.79	1.99	2.77	3.11	2.78	3.13	1.1	1.1
	Class D	1.62	1.76	1.80	1.99	1.80	1.98	1.0	0.8
	Class F	1.81	1.97	3.00	3.26	3.14	3.43	1.3	2.3
	Total average	1.80	1.98	2.85	3.15	3.08	3.41	1.0	1.3

In order to achieve comparable results, the same test conditions as in [13] have been used. In this way, the same four sequences have been encoded (*Traffic* and *PeopleOnStreet* from Class A, and *Kimono* and *ParkScene* from Class B) using the *Low Delay P* encoding configuration. This mode involves using P slices and, thus, biprediction is not taken into consideration. As a result, both GPU-based algorithms achieve higher speed-ups. The rest of the configuration parameters remain equal to the ones provided in [1].

Our proposed heterogeneous architecture performs the fractional motion estimation module on the host, as opposed to [13], which carries it out on the device. Therefore, the speed-up results provided in Table 4 are not completely comparable. Nevertheless, the fractional motion search consists on executing the same algorithm once again on a

Table 3 Speed-up and BD-rate results comparison between the GOP-based parallelization technique and its corresponding joint algorithm

Class	Video sequence	Speed-up						BD-rate (%)	
		2 threads		4 threads		4 th. + SMT			
		GOP	w/GPU	GOP	w/GPU	GOP	w/GPU	GOP	w/GPU
A	Traffic	1.95	2.05	3.60	3.80	4.10	4.19	0.0	0.3
	PeopleOnStreet	1.96	2.24	3.66	4.16	4.16	4.73	0.1	−0.6
B	Kimono	1.94	2.18	3.44	3.95	3.95	4.48	2.3	2.8
	ParkScene	1.93	2.06	3.61	3.86	4.09	4.38	1.5	1.6
	Cactus	1.96	2.14	3.68	4.03	4.21	4.61	−0.5	−0.4
	BasketballDrive	1.96	2.28	3.60	4.16	4.08	4.78	0.4	4.0
	BQTerrace	1.96	2.07	3.64	3.88	4.07	4.33	−0.9	−1.9
C	BasketballDrill	1.96	2.19	3.70	4.13	4.26	4.75	0.3	−0.1
	BQMall	1.94	2.11	3.43	3.81	3.95	4.38	0.4	1.2
	PartyScene	1.95	2.09	3.53	3.83	4.01	4.41	0.9	0.7
	RaceHorses	1.93	2.27	3.54	4.20	4.06	4.71	−0.5	−0.4
D	BasketballPass	1.85	2.12	3.42	3.95	3.98	4.55	0.4	0.5
	BQSquare	1.95	2.02	3.67	3.81	4.16	4.30	0.8	0.6
	BlowingBubbles	1.87	1.98	3.40	3.64	3.73	4.04	1.8	1.4
	RaceHorses	1.92	2.19	3.52	4.03	4.11	4.60	−0.1	−0.6
F	BasketballDrillText	1.97	2.17	3.68	4.08	4.24	4.70	0.4	−0.4
	ChinaSpeed	1.95	2.21	3.64	4.15	4.09	4.72	−0.2	−3.2
	SlideEditing	1.96	2.00	3.60	3.67	4.02	4.16	−1.0	2.1
	SlideShow	1.86	1.96	3.23	3.44	3.45	3.75	2.0	6.8
	Class A	1.95	2.14	3.63	3.98	4.13	4.46	0.0	−0.2
	Class B	1.95	2.15	3.59	3.98	4.08	4.52	0.6	1.2
	Class C	1.95	2.16	3.55	3.99	4.07	4.56	0.3	0.3
	Class D	1.90	2.08	3.50	3.86	4.00	4.37	0.7	0.5
	Class F	1.93	2.09	3.54	3.84	3.95	4.33	0.3	1.3
	Total average	1.94	2.12	3.56	3.93	4.05	4.45	0.4	0.6

Table 4 Comparison between this proposal and the related work

Class	Video sequence	Speed-up		BD-rate (%)	
		Proposal	Radicke et al. [13]	Proposal	Radicke et al. [13]
A	Traffic	1.12	1.92	0.0	1.8
	PeopleOnStreet	1.32	2.07	−1.0	0.6
B	Kimono	1.30	2.15	−0.1	0.7
	ParkScene	1.17	1.94	0.1	1.4

subsampled version of the region referenced by the best motion vector found. In this way, it would be possible to adapt our proposed algorithm to perform this additional operation. As a result, it would be expected to obtain equivalent speed-up results.

The main difference, however, lies in the fact that our proposed algorithm performs a full search, while the algorithm in [13] carries out a diamond search, a similar pattern to the one used as the default algorithm in HM. Taking into account that this pattern focuses the search on the centre positions of the reference area, it might be unable to find the best MV in the absence of motion predictors. On the contrary, a full search is able to obtain the best MV. Since both proposals perform the same fractional motion estimation operation afterwards, it is possible to compare the coding efficiency results shown in Table 4. As can be seen, our algorithm achieves BD-rate savings ranging from 0.8 to 1.8 % compared with the one proposed by Radicke et al. [13].

As a conclusion, the effects of choosing the right search pattern may have a noticeable impact on the coding efficiency results. In this way, a full search is able to obtain the best results with no performance impact on a SIMD architecture such as a GPU.

6 Conclusion and future work

In this paper, we have proposed a GPU-based motion estimation algorithm that, in combination with some coarse-grained parallelization techniques, forms an efficient parallel framework for the HEVC encoder. In this framework, these parallelization techniques are performed on a multi-core CPU, while the GPU carries out the ME operation concurrently. Comparing our approach to some of these techniques (WPP and a GOP-based algorithm), our experiments show that the corresponding joint algorithms achieve better performance in terms of speed-up with negligible coding efficiency penalties. Ongoing work will focus on using multiple GPUs and parallelizing other modules, as well as considering other architectures such as Intel Integrated Graphics or Intel Xeon Phi [6].

References

1. Bossen F (2013) Common test conditions and software reference configurations (Doc. JCTVC-L1100). http://phenix.int-evry.fr/jct/doc_end_user/current_document.php?id=7281. Accessed 14 May 2013
2. Bossen F, Bross B, Suhring K, Flynn D (2012) HEVC complexity and implementation analysis. IEEE Trans Circuits Syst Video Technol 22(12):1685–1696. doi:10.1109/TCSVT.2012.2221255
3. Bross B, Han W, Ohm J, Sullivan G, Wang YK, Wiegand T (2013) High efficiency video coding (HEVC) text specification draft 10 (Doc. JCTVC-L1003). http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=7243. Accessed 21 March 2013
4. Cheung NM, Fan X, Au O, Kung MC (2010) Video coding on multicore graphics processors. IEEE Signal Process Mag 27(2):79–89. doi:10.1109/MSP.2009.935416
5. Chi CC, Alvarez-Mesa M, Juurlink B, Clare G, Henry F, Pateux S, Schierl T (2012) Parallel scalability and efficiency of HEVC parallelization approaches. IEEE Trans Circuits Syst Video Technol 22(12):1827–1838. doi:10.1109/TCSVT.2012.2223056
6. Fang J, Varbanescu AL, Sips H (2013) Identifying the key features of Intel Xeon Phi: a comparative approach. Parallel and Distributed Systems Report Series PDS-2013-006, Delft University of Technology
7. Wc Feng, Manocha D (2007) High-performance computing using accelerators. Parallel Comput 33(10–11):645–647

8. Henry F, Pateux S (2011) Wavefront Parallel Processing (Doc. JCTVC-E196). http://phenix.int-evry.fr/jct/doc_end_user/current_document.php?id=2122. Accessed 25 March 2013
9. ITU-T, ISO/IEC JTC (2012) Information technology—coding of audio-visual objects—part 10: advanced video coding. ITU-T Recommendation H.264 and ISO/IEC 14496–10
10. JCT-VC (2013) HM reference Software. https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/. Accessed 23 April 2013
11. Misra K, Segall A, Horowitz M, Xu S, Fuldseth A, Zhou M (2013) An overview of Tiles in HEVC. IEEE J Sel Top Signal Process 7(6):969–977. doi:10.1109/JSTSP.2013.2271451
12. Ohm J, Sullivan G, Schwarz H, Tan TK, Wiegand T (2012) 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. doi:10.1109/TCSVT.2012.2221192
13. Radicke S, Hahn J, Grecos C, Wang Q (2014) A highly-parallel approach on motion estimation for high efficiency video coding (HEVC). In: IEEE international conference on consumer electronics (ICCE), 2014, pp 187–188. doi:10.1109/ICCE.2014.6775965
14. Su H, Wu N, Zhang C, Wen M, Ren J (2011) A multilevel parallel intra coding for H.264/AVC based on CUDA. In: Sixth international conference on image and graphics (ICIG), 2011, pp 76–81. doi:10.1109/ICIG.2011.99
15. Wang X, Song L, Chen M, Yang J (2013) Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform. In: IEEE international conference on multimedia and expo workshops (ICMEW), 2013, pp 1–5. doi:10.1109/ICMEW.2013.6618412
16. Yan C, Dai F, Zhang Y (2011) Parallel deblocking filter for H.264/AVC on the TILERA many-core systems. In: Advances in multimedia modeling, lecture notes in computer science, vol 6523, Springer, Berlin, pp 51–61. doi:10.1007/978-3-642-17832-0_6
17. Yan C, Zhang Y, Dai F, Li L (2013) Highly Parallel Framework for HEVC motion estimation on many-core platform. In: Data compression conference (DCC), 2013, pp 63–72. doi:10.1109/DCC.2013.14
18. Yu Q, Zhao L, Ma S (2012) Parallel AMVP candidate list construction for HEVC. In: Visual communications and image processing (VCIP), 2012 IEEE, pp 1–6. doi:10.1109/VCIP.2012.6410775