The Journal of Supercomputing (2023) 79:11641–11659
https://doi.org/10.1007/s11227-022-04764-1

On the use of deep learning and parallelism techniques to significantly reduce the HEVC intra-coding time

Vicente Galiano[1] · **Héctor Migallón**[1] · **Miguel Martínez-Rach**[1] · **Otoniel López-Granado**[1] · **Manuel P. Malumbres**[1]

Accepted: 10 August 2022 / Published online: 3 September 2022

Abstract

It is well-known that each new video coding standard significantly increases in computational complexity with respect to previous standards, and this is particularly true for the HEVC and VVC video coding standards. The development of techniques for reducing the required complexity without affecting the rate/distortion (R/D) performance is therefore always a topic of intense research interest. In this paper, we propose a combination of two powerful techniques, deep learning and parallel computing, to significantly reduce the complexity of the HEVC encoding engine. Our experimental results show that a combination of deep learning to reduce the CTU partitioning complexity with parallel strategies based on frame partitioning is able to achieve speedups of up to 26× when 16 threads are used. The R/D penalty in terms of the BD-BR metric depends on the video content, the compression rate and the number of OpenMP threads, and was consistently between 0.35 and 10% for the video sequence test set used in our experiments

Keywords CNN · Deep learning · HEVC · Deep learning · Parallel processing · Slices · Video coding

✉ Vicente Galiano
vgaliano@umh.es

Héctor Migallón
hmigallon@umh.es

Miguel Martínez-Rach
mmrach@umh.es

Otoniel López-Granado
otoniel@umh.es

Manuel P. Malumbres
mels@umh.es

[1] Department of Computer Engineering, Miguel Hernández University, Avda. de la Universidad s/n, Elche 03202, Alicante, Spain

1 Introduction

The high-efficiency video coding (HEVC) standard was launched in 2013 [9] by the Joint Collaborative Team on Video Coding (JCT-VC). Although HEVC can compress a video sequence using half the bitrate of its predecessor, this performance improvement comes at the expense of an increment in the computational cost [1].

Great efforts have been made to speed up the encoding process. Several works in the literature have tried to reduce the coding time using modern hardware accelerators [2–8]. In [6, 8], computation of the motion estimation (ME) was moved to the GPU, since in the same way as for previous video standards, ME is the most complex task undertaken by the encoder, requiring more than 90% of the encoding time [9]. In [2, 4, 7], the ME process was accelerated using a similar approach based on FPGAs. In other approaches, various coding processes have been moved to the FPGA, such as the 2D-DCT with variable size [3], the intra-frame prediction process [5], and the CABAC entropy encoder [10].

Other works in the literature have used parallel computing strategies to reduce the overall complexity of HEVC encoding, and to take advantage of the multi-core processors available in modern HPC servers in order to speed up the overall encoding time for a video sequence [11–16]. There are also several other approaches, which typically depend on the selected parallelisation strategy (temporal or spatial) and the level at which parallelism is applied (fine, medium, or coarse). For example, in [15], the authors applied a fine parallelism scheme to reduce the complexity of the HEVC Sample Adaptive Offset (SAO) in-loop filter, and obtained an speedup of 1.9×, while in [14], the authors employed a temporal parallelism approach based on wavefront parallel processing which consisted of a special type of pipeline processing for the Coding Tree Units (CTUs) of a given frame when several computing OpenMP computing threads were available. The latter approach obtained an speedup of 5.5× using 20 cores, with a BD-rate [17] increment of 1.2%. In [12], a higher-level parallelisation scheme (at the frame level) was proposed based on the partition of each frame using tiles (a new feature available in HEVC). In this approach, a maximum speedup of up to 9× was obtained for the all intra (AI)-coding mode using 10 cores. The study in [16] presented a thorough analysis of the need to adaptively evaluate the workload of the different tiles in order to determine the best CTU partitioning is presented. In [13], the authors developed a parallel HEVC encoder using frame-level parallelism by means of slices rather than tiles, obtaining speedups of up to 9.3× and 8.7× for the AI and Random Access (RA) coding modes, respectively. In [11], a coarse-grained parallelisation scheme was presented (at the sequence level), in which different groups of pictures could be independently encoded by several processing nodes. This parallel approach was well-suited to the distributed memory architectures of modern federated clusters, and obtained speedups of up to 11.84× using 12 cores for the RA coding mode, with a BD-rate increment of 1.3%.

Finally, there are other works that have focused on optimisation of the source code of specific parts of the HEVC encoder [18–24]. In [18, 19], a pre-analysis technique was proposed to reduce (a) the size of the search area; (b) the number of reference frames in the inter-frame prediction; (c) the number of intra-prediction modes; and (d) the number of best candidates for the intra-frame prediction process. This approach achieved a 49% reduction in coding time on average for the RA coding mode with an average BD-rate increment of 1.08%. In [21], the authors developed a fast decision method to perform efficient asymmetric mode partition, thus reducing the computational complexity. They also proposed an adaptive motion search area estimator to reduce the overall inter-coding complexity even further. Their results demonstrated that their algorithm could reduce the encoding time by 31.37% in the RA coding mode with a negligible BD-rate increment. In [20], the authors reported on a fast decision mode based on CABAC rate estimation with a coding time reduction of 15%, while in [22], a fast CTU partitioning algorithm was developed in which the CTU texture was used to prune the CTU quad-tree structure. The results proved that the proposed fast coding unit (CU) partitioning algorithm yielded savings of 41% in the encoding time on average, with a BD-rate increment of 0.69%. In [23], a decision tree-based algorithm for CTU partition was presented. The authors implemented three decision trees classifiers for all the three depths of the CU partition. However, the thresholds required by this algorithm needed to be selected manually. This technique was able to reduce the encoding time by 42.1% on average, with a BD-rate increment of 0.7%. The authors of [24] proposed a Bayesian decision rule for an early termination CU algorithm. This Bayesian decision rule was used to estimate a likelihood function and the prior probability of a new scene. The model was then updated for the following frames, to predict the CU size. Although the proposed model had a negligible training time compared with other machine learning models, its accuracy depended on the particular scene, making it inaccurate. The results showed that an average reduction in coding time of 36% could be achieved with a BD-rate increment of 1.08% for the AI coding mode.

With regard to source code optimisation techniques, several authors have developed deep learning approaches to reduce the complexity of the HEVC encoder [25–33]. For example, to reduce the complexity of inter-mode prediction in the Low Delay B coding mode (LB), Zhang et al. [29] proposed a coding unit (CU) depth decision algorithm with a three-level joint classifier based on a support vector machine (SVM), which predicted the splitting of CTUs based on as a three-level of hierarchical binary decision problem. The proposed algorithm was able to reduce the encoding time by 51.45% on average, with a BD-rate increment of 1.98%. For the intra-coding mode, Liu et al. [26] developed a convolutional neural network (CNN) approach that predicted the CTU partitioning, thus reducing the coding time by 72% on average, with a BD-rate increment of 4.79%. The authors of [28] proposed a CNN-based algorithm for predicting the CU size for both inter- and intra-prediction coding using CNN models, where the quantisation parameter (QP) was used as one of the inputs to the classifier. In this scheme, reductions in coding time of 66.47% and 62.94% were achieved for the intra- and inter-coding modes, respectively. In [31], the authors developed a CNN-based algorithm to extract texture and objects location features, which were used with a Softmax classifier to predict the CU size. The results showed a reduction in the coding time of 66.89%, with a BD-rate increment of 1.31% for the AI coding mode. In [32], the researches proposed a fast CU size decision algorithm based on a CNN architecture, where four CNNs were used as classifiers at each of the four depths to make a decision (splitting or non-splitting) for the given QP. The pruning algorithm achieved a coding time reduction of 77% with a BD-rate increment of 3.1% on average for the AI coding mode. The authors of [33] presented CtuNet, a CNN approach that predicted CTU partitioning. The CtuNet framework consisted of three CNN networks for the CU sizes of 64×64, 32×32, and 16×16, with a residual network (ResNet18) [34] as the backbone model. This model obtained reductions in the coding time of 63.68% with a BD-rate increment of 1.77% on average, for the AI coding mode.

Recently, Çetinkaya et al. [35] have published a survey of CTU depth decision algorithms that covered classical statistics-based algorithms to modern advanced deep learning algorithms such as deep neural networks. In another recent paper, Wang and Li [36] designed a one-stage decision network(OSDN) structure to determine the CU/PU partition and prediction mode for intra-coding. Their experimental results showed that the proposed method could reduce the intra-encoding time by 73.69%, with a BD-PSNR loss of 0.1673 dB on average.

The most important contributions of the present work are as follows:

1 A hybrid HEVC encoder that combines two different acceleration strategies based on parallel computing and source code optimisation techniques is designed and developed. The first acceleration technique is a parallel scheme that uses a domain decomposition model based on HEVC slice partitioning, which is particularly suitable for exploiting the shared memory parallelism of multicore processors. The second technique uses optimisation methods at the CTU level to reduce the complexity of the quad-tree splitting process by means of a CNN.
2 The benefits of our hybrid solution are demonstrated, and it is shown to be fully compliant with the HEVC standard, to give good encoding performance for the HEVC, and to achieve outstanding speedups.
3 The hybrid proposal also includes extra parallelisation of the additional processing steps required by the machine learning-based acceleration approach.

The remainder of this paper is organised as follows. In Sect. 2, we explain the deep learning approach used to predict the CU partition and the slice-based parallelism strategy. Sect. 3 describes the proposed hybrid approach for improving the speed of the HEVC coding stage, and in Sect. 4, experimental results from the proposed hybrid algorithm are presented. Finally, in Sect. 5, some conclusions are drawn.

2 Related work

In this section, we explain the main features of the techniques used in this work to create the hybrid acceleration scheme in order to significantly improve the speedup of the HEVC encoding process.

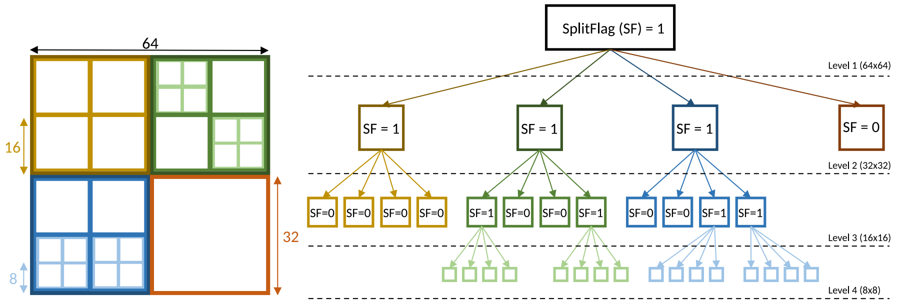
2.1 Neural network algorithm

The HEVC algorithm reduces the bit rate of the encoded video at the cost of a considerable increase in the encoding complexity. One of the most time-consuming process is the decision on the optimal quad-tree partitioning of each CTU. To find an optimal CTU partitioning from the 83522 possible partitions (see [35]), HEVC searches 85 CUs with different sizes ranging from 64×64 to 8×8 pixels. In addition to finding the correct CU depth structure, the prediction unit (PU) modes and the transform unit (TU) partitioning must be properly determined for each CU. Thus, the search for the optimal CTU structure requires the largest amount of time in the encoding process [37], since it uses a brute force approach to find the one with the minimum rate-distortion (RD) cost.

Several schemes for reducing the computational cost of the CU partition have been reviewed in Sect. 1, some of which reduce the complexity of the algorithm at the cost of an increase in bit rate to maintain the reconstructed video quality; others replace the brute force search for R/D optimisation (RDO) with a deep neural network that is trained to estimate the CTU partitioning. Of the numerous complexity reduction schemes based on deep learning that have been proposed, we highlight the one presented by Xu et al. [28]. The main factors that differentiate this proposal from the alternatives involve the definition of a hierarchical CU partition map (HCPM) to represent the CU partition. Given sufficient training data and an efficient HCPM representation, the authors propose a deep CNN structure called an early-terminated hierarchical CNN (ETH-CNN) that can be trained to explore various patterns for the CTU partition and thus reduce the complexity of the HEVC coding process.

A CTU has a size of 64×64 pixels by default, and can either contain a single CU or be recursively split into multiple smaller CUs, based on the quad-tree structure shown in Fig. 1.

In the CU partition structure in HEVC, four different CU sizes are supported by default; these are 64×64, 32×32, 16×16 and 8×8, corresponding to four CU depths of 0, 1, 2 and 3. For a coding unit U, the first-level binary label $y_1(U)$ indicates whether U is split (= 1) or not (= 0). If U is split, its sub-CUs of depth one are



(a) CUs blocks **(b)** CTU quad-tree

Fig. 1 Example of CTU quad-tree structure defined in HEVC

denoted as $\{U_i\}_{i=1}^4$. As stated above, in HEVC, the binary labels for splitting each CU are obtained using a time-consuming RDO process, but these can be predicted faster via a deep learning algorithm using a simple multi-class classification in one step call (ETH-CNN). Note that the input CTU is extracted from raw images, and only the Y channel is used in ETH-CNN. The structure of ETH-CNN consists of two pre-processing layers, three convolutional layers, and one concatenating layer [28]. Using this ETH-CNN structure, the model is trained to minimise the R/D loss function (see Equation (2)), and can finally be used to predict the CTU partitioning in the form of HCPM. For each training sample r the loss function LF_r sums the cross-entropy over all valid elements of HPCM (see Equation (1)).

$$LF_r = H\left(y_1^r(U), \hat{y}_1^r(U)\right) + \sum_{\substack{i \in \{1,..,4\} \\ y_2^r(U_i) \neq null}} H\left(\left(y_2^r(U_i)\right), \hat{y}_2^r\left(U_i\right)\right) + \sum_{\substack{i,j \in \{1,..,4\} \\ y_3^r(U_{i,j}) \neq null}} H\left(\left(y_3^r(U_{i,j})\right), \hat{y}_3^r\left(U_{i,j}\right)\right) \tag{1}$$

where $\left\{ \hat{y}_1^r(U), \hat{y}_2^r(U_i)_{i=1}^4, \hat{y}_3^r(U_{i,j})_{i,j=1}^4 \right\}_{k=1}^{NoTS}$ are the labels of the hierarchical CU partition map predicted by ETH-CNN and r represents the number of training samples (NoTS). Moreover, $H(y, \hat{y})$ is the cross-entropy between the ground-truth (y) and the predicted labels ($\hat{y}$). The proposed ETH-CNN model is trained by optimising the global loss function (LF) shown in Equation (2).

$$LF = \frac{1}{NoTS} \sum_{r=1}^{NoTS} LF_r \tag{2}$$

Given an input CTU, ETH-CNN provides the splitting probabilities at each level $P_1(U)$, $P_2(U_i)$ and $P_3(U_{i,j})$ for the binary labels $y_1(U)$, $y_2(U_i)$ and $y_3(U_{i,j})$, to predict the CU partitioning. In general, a decision threshold $\alpha_l = 0.5$ is set for levels 1, 2 and 3. Hence, a CU with $P_l(U) > \alpha_l$ is split into four sub-CUs. The author of [28] also provides a convolutional network for inter-coding called ETH-LSTM. However, as our proposal is focused on the intra-coding we will use the ETH-CNN network specially developed for intra-coding.

2.2 Slice-based parallel algorithm

The HEVC standard allows each frame of a video source to be segmented into a set of CTUs, each of which can be configured as an independent block that can be encoded in parallel. The HEVC standard offers two options for dividing the video source to be encoded into independent sets of CTUs: slice and tile partitioning. Slices are sets of correlative CTUs where the number of CTUs in each set are the same for all slices (except where necessary for the last slice containing the CTUs in the lower right-hand corner of the frame). In the HEVC standard, the number of

CTUs per slice needs to be established. The sizes of the slices (in terms of the numbers of CTUs) will determine the number of slices in each frame, depending on both the resolution of the video sequence to be encoded and the size of the CTUs. Note that each CTU is a square set of pixels for which the size is set to 64×64 pixels, as specified in the HEVC common test conditions [38].

As each slice contains a data header, it can be decoded independently of the others, even if the data from the others are not available when decoding. Since the size of the header can affect the compression ratio (i.e. the number of bits per pixel in the compressed bit stream), the number of slices in the proposed parallel algorithm should be established with care, in order to avoid an excessive bitstream overhead (see [39]). Each encoding process calculates the slice size, expressed in number of CTUs, depending on (a) the number of CTUs in a frame; (b) the identification of the encoding process $\left(I_{EP}\right)$; and (c) the total number of available encoding processes $\left(N_{EP}\right)$, as indicated in Algorithm 1. The size of the last slice (in the lower right-hand corner) is either equal to or smaller than the rest of the slices, and its size $\left(S_{Slice}\right)$ is determined based on the number of processes according to Algorithm 1.

Algorithm 1 Slice size algorithm for encoding process $I_{EP} \in \{1, \ldots, N_{EP}\}$

```
1: if $I_{EP} \neq N_{EP}$ then
2:     $S_{Slice} = N_{CTUs}$ div $N_{EP}$
3: else
4:     $S_{Slice} = N_{CTUs}$ mod $N_{EP}$
5: end if
```

The slice partitioning process in Algorithm 1 aims to achieve a balanced computational load, in which domain decomposition is performed to assign each process the same (or a similar) amount of data. Note that if the computational load assigned to each process is evaluated based on the number of CTUs in a frame $\left(N_{CTUs}\right)$ it is only possible for the encoding process of the last slice to have an imbalanced computational load. Depending on the video sequence resolution to be encoded, there may also be CTUs at the right-hand or bottom edges of a frame with fewer than 4096 (64×64) pixels. Figure 2a and b show two different partition schemes for encoding a video sequence

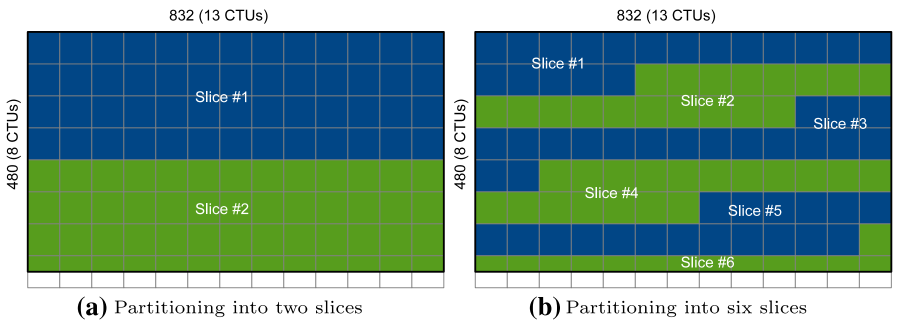


Fig. 2 Slice partitioning of a 832×480 frame

of size (832 × 480) pixels, where the total number of CTUs is 104 (13 × 8). Figure 2 shows partitioning into two slices of 52 CTUs each, while Fig. 2 shows partitioning into six slices, where the first five slices contain 18 CTUs each and the last slice contains 14. In the last slice, only the first CTU has 4096 (64 × 64) pixels, and the remaining 13 CTUs have only 2048 (64 × 32) pixels.

Once the slices have been assigned to the processes, each process must encode the CTUs contained in the assigned slice, and for each CTU, the quad-tree structure must be computed using the brute force R/D algorithm as described in Sect. 2.1.

In order to significantly reduce the computing time of the HEVC encoding process, we propose a hybridised scheme that includes both a deep learning approach to predict the CU partition and a parallel processing scheme based on slice partitioning, and this is described in the next section.

3 Hybrid acceleration proposal

The deep learning algorithm described in Sect. 2.1 and the slice-based parallel algorithm in Sect. 2.2 can be complemented by allowing for parallelisation and pre-calculation of CTU partitioning through deep learning. A general flowchart for the proposed hybrid algorithm is shown in Fig. 3. The sliced parallel algorithm is represented using red boxes, while the blue ones represent the contribution from deep learning. In the first step, all of the OpenMP threads read the configuration parameters and encode a set of frames depending on the total numbers of frames and threads. Each thread computes the HCPM for all the CTUs in the assigned frame set, and the partition map is stored in memory so that it can be accessed by all threads when the CTU partitioning tree is computed for a given slice. Once all the HPCMs have been generated and saved in a concurrent manner (which yields an improvement in computation time compared to other approaches), all threads are synchronised to encode each frame. In this sense, the slice-based parallel algorithm is applied at a higher level. As shown in Fig. 3, only the master thread reads the new frame to be encoded, in order to reduce both the number of disk accesses and the memory requirements. The frame to be encoded will therefore be stored in the shared memory, and will be accessed only for reading. In fact, each thread will only access those CTUs that are part of the slice to be encoded by it. The prediction for the CTU partition obtained from the deep learning approach is used when coding the set of CTUs for the slice assigned to each thread. When each thread has encoded the slice assigned to it, it writes its bit stream into the final bit stream, and this process must be done in the right order, as shown in Fig. 3. Hence, thread 0 is the first to become idle after storing its computed part of the bitstream. This thread can then start reading or receiving the new data, while the rest of the OpenMP threads finish writing to the bitstream.

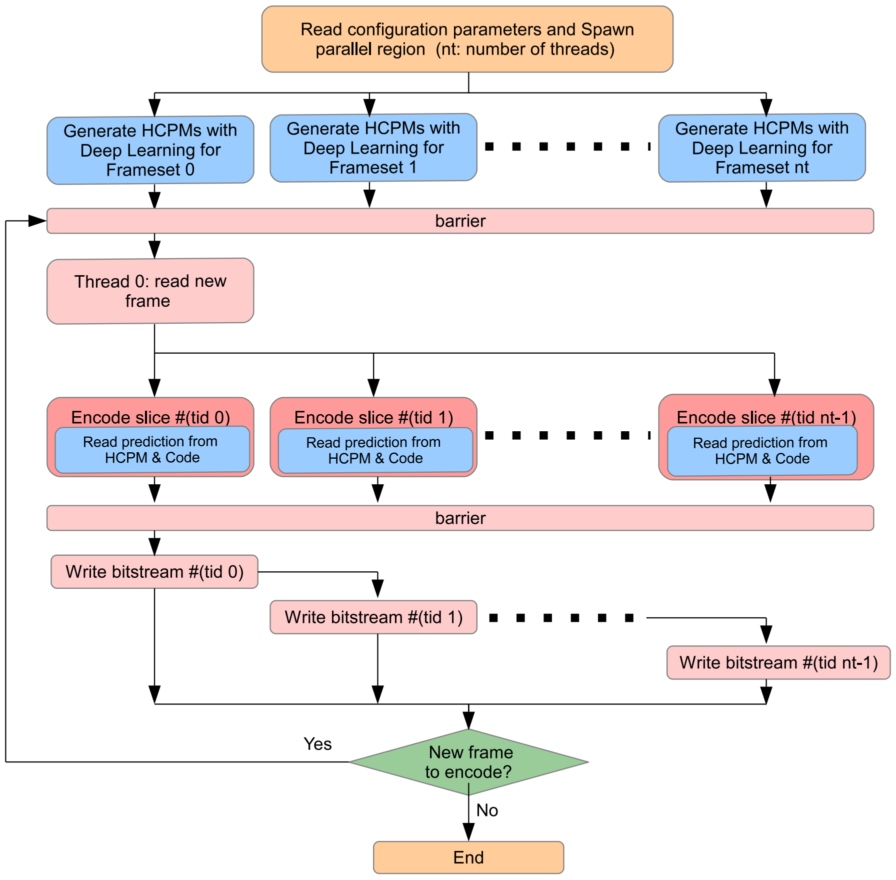

Read configuration parameters and Spawn parallel region (nt: number of threads)
Generate HCPMs with Deep Learning for Frameset 0
Generate HCPMs with Deep Learning for Frameset 1
Generate HCPMs with Deep Learning for Frameset nt
barrier
Thread 0: read new frame
Encode slice #(tid 0)
Read prediction from HCPM & Code
Encode slice #(tid 1)
Read prediction from HCPM & Code
Encode slice #(tid nt-1)
Read prediction from HCPM & Code
barrier
Write bitstream #(tid 0)
Write bitstream #(tid 1)
Write bitstream #(tid nt-1)
Yes
New frame to encode?
No
End

Fig. 3 Hybrid parallel algorithm

4 Experimental results

In this section, we present the results of a set of experiments carried out to validate the effectiveness of our proposal are presented. To evaluate the intra-frame coding performance of our hybrid scheme, we compare the slice-based parallel approach proposed in [13], the deep learning approach proposed in [28] and the proposed hybrid approach. All three methods are based on the HEVC reference software HM version 16.3 [40] (which was used as a benchmark), and the AI configuration was applied using the default configuration file `encoder_intra_main.cfg`. Four QP values (22, 27, 32, 37) were chosen for compression of the selected video sequences as recommended by the HEVC common test conditions [38]. All experiments were conducted on a server with two processors (Intel(R) Xeon(R) Gold 6140 @ 2.30 GHz) with 18 cores per processor, 400 GB RAM, four Tesla P100-PCIE GPUs and CentOS Linux release 7.6.1810 as the operating system. For the deep learning approaches, we used TensorFlow 1.8 with GPU support for CUDA 9.1 and cuDNN 7.1 is used. The trained neural networks considered in the experiments were provided by the authors of [28]. Eleven video sequences from the JCT-VC standard test set [38] were used to evaluate and compare our method, as summarised in Table 1.

Table 2 shows the speedup and Bjontegaard delta bit rate (BD-BR) [41] obtained for the Class A video sequences using the schemes in [13, 28] and our proposed approach (Prop.). The time reduction is expressed based on the speedup as an acceleration measurement in order to directly relate the coding latency to the number of OpenMP threads (Th.) used. All the speedups and the values for the BD-rate were obtained with respect to the reference software, HM version 16.3 [40].

The experimental results from the deep learning approach were similar to those obtained by the authors of [28]; for example, for the Traffic sequence, a reduction of a 73.7% in the execution time was achieved for QP = 37, corresponding to an average speedup of 3.7×. The OpenMP approach described in [13] gave speedups of up to 14.65× for 16 threads for same video sequences, with an efficiency of 75% (where efficiency is defined as the ratio of useful work to the resources expended by each thread in each core). This was as expected, since a slice-based distribution is more efficient for higher-resolution video sequences where the computational load can be equally distributed, as described by the authors of [13]. The proposed approach which combines both strategies is able to considerably reduce the coding times. For example, for the BQMall Class C video sequence encoded with QP = 37, a speedup of 37.9× was achieved for 16 threads. These results clearly show that a combination of slice-based parallelisation with a reduction in complexity from deep learning can provide significant levels of acceleration for HEVC intra-frame coding, which are greater than the accelerations obtained by the schemes in [28] and [13] (2.96× and 14.12×, respectively). In a practical scenario where the speed of intra-coding is decisive, the proposed solution offers a much higher performance than all the proposals described in Sect. 1.

The reduction in the complexity of the HEVC intra-frame coding mode is achieved at the expense of a loss of R/D performance. Tables 2 , 3, 4 and 5 show the values of BD-BR used to evaluate the R/D performance of the proposed scheme and the other two alternatives [13, 28]. As expected, the BD-BR for our hybrid proposal is approximately the sum of the penalties obtained by the approaches in [28] and [13]. For example, it can be seen from Table 5 that for QP = 37, the algorithm proposed in [28] shows an increase in the BD-rate of 1.43% for RaceHorses, whereas the penalty obtained by the algorithm proposed in [13] is 1.76% for 16 threads.

Table 1 Test video sequences

Class	Size	Sequence	Frame rate
A	2560 × 1600	PeopleOnStreet, Traffic	30fps, 30fps
B	1920 × 1024	BasketballDrive, BQTerrace, Cactus	50fps, 60fps, 50fps
C	833 × 488	BasketballDrill, BQMall, PartyScene	50fps, 60fps, 50fps
D	384 × 192	BlowingBubbles, BQSquare,RaceHorses	50fps, 50fps, 50fps

Table 2 Speedup and BD-BR for Class A video sequences

Sequence	Speedup													BD-BR		
	[28]				Th.	[13]				Prop.				[28]	[13]	Prop.
	QP					QP				QP						
	22	27	32	37		22	27	32	37	22	27	32	37			
PeopleOnStreet	2.44	2.50	2.56	2.70		1.00	1.00	1.00	0.99	2.37	2.42	2.71	2.69	2.41%	0.00%	2.24%
					2	1.91	1.90	1.92	1.94	4.37	4.51	4.98	4.98		0.17%	2.43%
					4	3.87	3.72	3.85	3.57	7.96	8.26	8.81	8.56		0.46%	2.85%
					8	7.17	6.50	6.98	6.95	14.73	14.76	15.10	15.64		1.04%	3.41%
					16	15.36	15.12	15.25	15.41	25.68	26.25	27.48	26.68		2.15%	4.52%
Traffic	2.94	3.33	3.45	3.70	1	1.00	1.00	1.00	0.99	2.28	2.38	2.69	2.62	2.35%	0.00%	2.45%
					2	1.85	1.81	1.93	1.85	4.34	4.39	4.88	4.92		0.14%	2.55%
					4	3.49	3.50	3.68	3.54	7.75	8.19	8.72	8.23		0.41%	2.75%
					8	7.10	7.02	6.33	6.56	14.25	14.60	14.88	15.19		1.14%	3.49%
					16	15.31	14.40	14.88	14.65	24.44	25.27	26.47	25.59		1.97%	4.42%

Table 3 Speedup and BD-BR for Class B video sequences

Sequence	Speedup													BD-BR		
	[28]				Th.	[13]				Prop.				[28]	[13]	Prop.
	QP					QP				QP						
	22	27	32	37		22	27	32	37	22	27	32	37			
Basketballdrive	2.54	3.51	4.42	4.76	1	1.00	1.00	1.00	0.99	2.52	3.53	4.42	4.92	4.29%	0.00%	4.29%
					2	1.89	1.98	1.89	1.93	4.82	6.58	8.34	9.09		0.49%	4.77%
					4	3.44	3.62	3.43	3.68	7.92	10.32	13.16	14.65		0.94%	5.22%
					8	6.18	6.63	6.59	6.87	13.37	17.98	23.56	25.85		2.35%	6.55%
					16	14.86	14.92	15.08	15.21	22.58	25.45	30.62	34.46		4.04%	8.33%
BQTerrace	2.45	3.47	4.82	5.04	1	1.00	1.00	1.00	0.99	2.44	3.51	4.37	4.80	4.35%	0.00%	4.39%
					2	1.82	1.82	1.76	1.89	4.61	6.51	7.94	8.82		0.48%	4.88%
					4	3.52	3.57	3.53	3.79	7.64	10.26	12.82	14.02		0.91%	5.32%
					8	6.07	6.09	6.30	6.73	13.31	17.52	22.72	25.58		2.30%	6.72%
					16	14.35	14.43	14.62	14.62	22.09	24.30	29.88	33.98		3.97%	8.40%
Cactus	2.41	3.49	5.04	5.33	1	1.00	1.00	1.00	0.99	2.36	3.49	4.23	4.73	3.81%	0.00%	3.85%
					2	1.94	1.98	1.91	1.99	4.49	6.19	7.64	8.80		0.48%	4.33%
					4	3.63	3.89	3.72	3.91	7.53	10.19	12.25	13.81		0.90%	4.76%
					8	6.66	7.12	6.91	6.98	12.84	17.44	22.31	24.32		2.24%	6.11%
					16	14.14	14.23	14.27	14.48	21.61	23.37	29.42	32.48		3.94%	7.82%

Table 4 Speedup and BD-BR for Class C video sequences

Sequence	Speedup													BD-BR		
	[28]				Th.	[13]				Prop.				[28]	[13]	Prop.
	QP					QP				QP						
	22	27	32	37		22	27	32	37	22	27	32	37			
Basketballdrill	1.97	2.05	2.26	2.75	1	1.00	1.00	1.00	0.99	1.77	1.85	2.04	2.46	1.76%	0.00%	1.69%
					2	1.95	1.79	1.83	1.91	3.47	3.32	3.72	4.73		0.12%	1.81%
					4	3.51	3.56	3.39	3.46	6.26	6.62	6.91	8.61		0.45%	2.14%
					8	6.49	6.30	6.00	6.65	11.54	11.64	12.31	16.58		1.24%	2.94%
					16	14.01	14.13	14.19	14.35	24.97	26.30	28.91	35.54		2.37%	4.08%
BQmall	1.92	1.98	2.38	2.96	1	1.00	1.00	1.00	0.99	1.74	1.79	2.14	2.66	1.67%	0.00%	1.69%
					2	1.84	1.74	1.78	1.85	3.18	3.13	3.83	4.94		0.11%	1.80%
					4	3.17	3.32	3.35	3.28	5.52	5.94	7.25	8.78		0.45%	2.14%
					8	5.77	5.83	6.19	5.77	10.04	10.42	13.27	15.47		1.19%	2.88%
					16	12.12	13.45	14.09	14.12	21.05	24.06	30.19	37.90		2.25%	3.96%
Partyscene	1.90	2.10	2.45	2.92	1	1.00	1.00	1.00	0.99	1.71	1.89	2.23	2.61	1.68%	0.00%	1.70%
					2	1.91	2.01	2.01	1.84	3.28	3.82	4.44	4.88		0.11%	1.81%
					4	3.84	3.93	3.64	3.60	6.60	7.44	8.08	9.56		0.43%	2.13%
					8	6.80	6.67	6.63	6.63	11.64	12.60	14.67	17.53		1.16%	2.87%
					16	11.54	12.97	13.40	13.63	19.85	24.65	29.63	35.91		2.17%	3.89%

Table 5 Speedup and BD-BR for Class D video sequences

Sequence	Speedup													BD-BR		
	[28]				Th.	[13]				Prop.				[28]	[13]	Prop.
	QP					QP				QP						
	22	27	32	37		22	27	32	37	22	27	32	37			
Blowingbubbles	1.65	1.67	2.1	2.54	1	1.00	1.00	1.00	0.99	1.49	1.51	1.90	2.28	1.45%	0.00%	1.37%
					2	1.97	1.92	1.94	1.88	2.93	2.90	3.70	4.31		0.09%	1.46%
					4	3.46	3.36	3.36	3.28	5.18	5.08	6.39	7.56		0.41%	1.79%
					8	5.48	4.86	4.97	5.00	8.17	7.32	9.45	11.52		1.14%	2.53%
					16	12.13	10.23	11.20	11.87	18.05	15.46	21.26	27.39		1.82%	3.21%
BQsquare	1.58	1.78	2.11	2.62	1	1.00	1.00	1.00	0.99	1.44	1.61	1.91	2.35	1.36%	0.00%	1.37%
					2	2.02	1.92	1.87	1.86	2.90	3.10	3.58	4.42		0.09%	1.46%
					4	3.52	3.14	3.33	3.28	5.03	5.06	6.33	7.79		0.40%	1.78%
					8	5.45	4.94	4.76	5.07	7.79	7.98	9.09	11.97		1.12%	2.51%
					16	11.60	10.24	11.44	11.85	16.51	16.52	21.73	28.05		1.78%	3.17%
Racehorses	1.55	1.71	2.01	2.70	1	1.00	1.00	1.00	0.99	1.41	1.55	1.82	2.42	1.43%	0.00%	1.44%
					2	1.91	1.80	1.87	1.93	2.69	2.79	3.39	4.71		0.09%	1.53%
					4	3.27	3.19	2.98	3.21	4.57	4.96	5.40	7.80		0.39%	1.84%
					8	5.09	5.00	5.17	5.27	7.16	7.76	9.35	12.81		1.08%	2.53%
					16	11.48	10.28	11.08	12.08	17.21	15.95	20.11	29.49		1.76%	3.22%

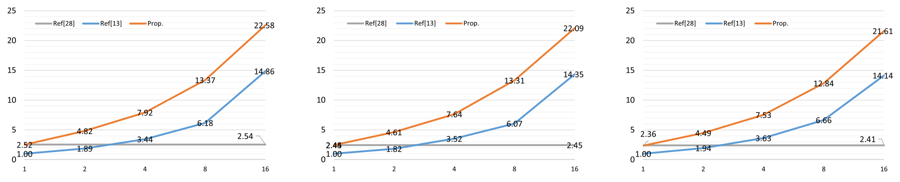
Finally, our hybrid model has a penalty of 3.22% for the BD-rate. From an analysis of these results, it can be concluded that deep learning and parallelism do not interfere with or cancel each other out in terms of the video quality.

In Fig 4, we show the speedup behaviour of the three schemes under evaluation as the number of the working threads increases, for three different Class B video sequences encoded with a QP value of 22. For the deep learning approach, the speedup is constant, as it does not use threads, whereas for the slice-based approach, we find an speedup progression that indicates good scalability behaviour, which is maintained for our hybrid proposal.

Finally, Table 6 shows the R/D performance results and the time reductions achieved by several schemes in the literature and the approach presented in this work. These results show that our scheme is able to achieve the greatest time reductions, with values that are consistently above 90%, and R/D performance losses of under 5% on average. However, if the increase in bitrate is unacceptable, a slower configurations may be chosen (with a lower number of threads), but with a minor R/D loss.

5 Conclusions

In this paper, we present a powerful technique to accelerate an HEVC encoder in the intra-frame coding mode. Our scheme combines two different approaches and exploits their characteristics to reap the benefits of both, and can considerably increase the speedup. Our proposed algorithm combines a slice-based parallel proposal for shared memory systems, with a deep learning approach. Although each scheme obtains a significant speedup when applied separately, a combination of both approaches considerably accelerates the HEVC encoder and achieves time savings of more than 90%. Our experimental results show a coding acceleration of up to 35×. There have been many attempts in the literature to speed up intra-encoding in HEVC, but they have not been jointly exploited. Our scheme achieved an acceleration of 35× with regard to the reference software, without the need for additional hardware. However, this acceleration was obtained at the expense of a loss of R/D performance. In our experiments, the maximum BD-rate penalty was 10.14% and the minimum was -0.9%. It was found that the two base algorithms did not interfere



(a) BasketBallDrive **(b)** BQTerrace **(c)** Cactus

Fig. 4 Speedup behaviour versus number of threads for the approaches in [13, 28] and the proposed scheme

Table 6 BD-BR and time reduction ΔT (%) for test video sequences for average values of QP = 22, 27, 32, and 37.

Class	Sequence	[28]		[32]		[33]		[31]		Prop.	
		ΔT(%)	BD-BR	ΔT(%)	BD-BR	ΔT(%)	BD-BR	ΔT(%)	BD-BR	ΔT(%)	BD-BR
A	PeopleOnStreet	−61.01	2.41	−77.80	3.60	−70.76	1.90	−66.86	1.17	−96.23	4.48
	Traffic	−70.79	2.35	−79.90	3.31	−69.43	2.11	−72.91	1.14	−93.20	4.18
B	BasketBallDrive	−76.32	4.29	−81.60	4.26	−76.55	3.20	−73.32	1.52	−91.21	8.33
	BQTerrace	−64.72	4.35	−75.90	3.36	−60.67	1.20	−65.02	1.68	−90.20	8.40
	Cactus	−63.27	3.81	−78.80	3.33	−68.07	1.90	−60.41	1.79	−93.40	7.82
C	BasketBallDrill	−52.98	1.76	−73.50	4.82	−58.25	2.80	−76.22	1.52	−96.21	4.08
	BQMall	−58.42	1.67	−76.10	2.96	−56.70	1.70	−57.86	1.57	−97.14	4.00
	PartyScene	−44.50	1.68	−68.90	1.52	−45.22	0.40	−54.02	0.32	−95.70	3.86
D	BlowingBubbles	−40.54	1.45	−67.80	1.73	−49.22	0.80	−52.05	0.34	−95.11	3.21
	BQSquare	−45.82	1.36	−69.10	1.56	−44.98	0.30	−65.33	1.45	−94.24	3.18
	RaceHorses	−55.76	1.43	−73.80	2.53	−53.47	1.10	−75.13	1.06	−96.10	3.22

with each other, as the results for the BD-rate obtained by the hybrid algorithm were approximately the sum of the penalties of both algorithms.

Due to the high level of computational complexity of the newest video coding standards, hybrid approaches that combines different acceleration techniques will be necessary in order to reduce the computational requirements. As a future line of research, we plan to use two levels of parallelisation based on heterogeneous platforms (shared and distributed memory) in order to get closer to real-time encoding with no change in the coding performance.

Acknowledgements This research was supported by the Spanish Ministry of Science, Innovation and Universities and the Research State Agency under Grant RTI2018-098156-B-C54 co-financed by FEDER funds (MCIU/AEI/FEDER, UE) and by the Valencian Ministry of Innovation, Universities, Science and Digital Society (Generalitat Valenciana) under Grants CIAICO/2021/278 and GV/2021/152.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability The set of test video sequences used in the experiments are available at ftp://hevc@ftp.tnt.uni-hannover.de/testsequences/ upon request to JCT-VC chairs. The datasets generated and analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Bossen F, Bross B, Suhring K, Flynn D (2012) HEVC complexity and implementation analysis. Circuits Syst Video Technol, IEEE Trans 22(12):1685–1696. https://doi.org/10.1109/TCSVT.2012.2221255
2. Alcocer E, Gutierrez R, López-Granado O, Malumbres MP (2019) Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder. J Real-Time Image Proc 16:547–557. https://doi.org/10.1007/s11554-016-0572-4
3. Chen M, Zhang Y, Lu C (2017) Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms. AEU-Int J Electron C 73:1–8. https://doi.org/10.1016/j.aeue.2016.12.024
4. Haddar R, Chaari A, Kibeya H, Ben Ayed MA, Masmoudi N (2017) FPGA-based implementation of TZsearch algorithm for H.265/HEVC standard. In: 2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA) pp. 605–610. 10.1109/STA.2017.8314939
5. Kalali E, Hamzaoglu I (2016) FPGA implementation of HEVC intra prediction using high-level synthesis. In: 2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin), pp. 163–166. 10.1109/ICCE-Berlin.2016.7684745
6. Lee D, Sim D, Cho K (2016) Fast motion estimation for HEVC on graphics processing unit (GPU). J Real-Time Image Proc 12:549–562. https://doi.org/10.1007/s11554-015-0522-6

7. Vidyalekshmi VG, Yagain D, Ganesh Rao K (2014) Motion estimation block for HEVC encoder on FPGA. In: International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), pp. 1–5. 10.1109/ICRAIE.2014.6909136
8. Xue Y-G, Su H-Y, Ren J, Wen M, Zhang C-Y, Xiao L-Q (2017) A highly parallel and scalable motion estimation algorithm with GPU for HEVC. Sci Program 2017:1–15. https://doi.org/10.1155/2017/1431574
9. Medhat A, Shalaby A, Sayed MS, Elsabrouty M (2014) A highly parallel SAD architecture for motion estimation in HEVC encoder. In: IEEE Asia Pacific Conference on Circuits and Systems (APCCAS'14), Ishigaki, pp. 280–283
10. Hahlbeck J, Stabernack, B (2014) A 4k capable FPGA based high throughput binary arithmetic decoder for H.265/MPEG-HEVC. In: 2014 IEEE Fourth International Conference on Consumer Electronics- Berlin (ICCE-Berlin)
11. Migallón H, Galiano V, Piñol P, López-Granado O, Malumbres MP (2016) Distributed memory parallel approaches for HEVC encoder. J Supercomput. https://doi.org/10.1007/s11227-016-1666-2
12. Migallón H, López-Granado O, Galiano V, Piñol P, Malumbres MP (2016) Shared memory tile-based vs hybrid memory gop-based parallel algorithms for HEVC encoder. Springer, Cham, pp 521–528. https://doi.org/10.1007/978-3-319-49583-5_40
13. Piñol P, Migallón H, López-Granado O, Malumbres MP (2015) Slice-based parallel approach for HEVC encoder. J Supercomput 71:1882–1892. https://doi.org/10.1007/s11227-014-1371-y
14. Radicke S, Hahn J, Grecos C, Qi Wang (2014) A multi-threaded full-feature HEVC encoder based on wavefront parallel processing. In: 2014 International Conference on Signal Processing and Multimedia Applications (SIGMAP), pp. 90–98
15. Ryu E, Nam J, Lee S, Jo H, Sim D (2013) Sample adaptive offset parallelism in HEVC. Multimedia and ubiquitous engineering. Lecture Notes in Electrical Engineering 240. https://doi.org/10.1007/978-94-007-6738-6_137
16. Storch I, Palomino D, Zatt B (2019) Speedup evaluation of HEVC parallel video coding using tiles. J Real-Time Image Processing. https://doi.org/10.1007/s11554-019-00900-y
17. Bjontegaard G (2001) Calculation of average PSNR differences between RD-curves. Technical Report VCEG-M33, Video Coding Experts Group VCEG, Austin
18. Cebrián-Márquez G, Martínez JL, Cuenca P (2019) Adaptive inter CU partitioning based on a look-ahead stage for HEVC. Signal Process: Image Commun 76:97–108. https://doi.org/10.1016/j.image.2019.04.019
19. Cebrián-Márquez G, Martínez JL, Cuenca P (2019) Inter and intra pre-analysis algorithm for HEVC. J Supercomput 73:414–432. https://doi.org/10.1007/s11227-016-1882-9
20. Chen W, Wang X (2016) Fast entropy-based cabac rate estimation for mode decision in HEVC. SpringerPlus 756:1–10. https://doi.org/10.1186/s40064-016-2377-0
21. Huang X, An P, Zhang Q (2017) Efficient AMP decision and search range adjustment algorithm for HEVC. J Image Video Process 75:1–15. https://doi.org/10.1186/s13640-017-0226-x
22. Maazouz M, Batel N, Bahri N, Masmoudi N (2019) Homogeneity-based fast CU partitioning algorithm for HEVC intra coding. Eng Sci Technol, Int J 22(3):706–714. https://doi.org/10.1016/j.jestch.2018.12.016
23. Westland N, Dias AS, Mrak M (2019) Decision trees for complexity reduction in video compression. In: 2019 IEEE International Conference on Image Processing (ICIP). 10.1109/icip.2019.8803302
24. Kuang W, Chan Y-L, Tsang S-H, Siu W-C (2020) Online-learning-based bayesian decision rule for fast intra mode and cu partitioning algorithm in HEVC screen content coding. IEEE Trans Image Process 29:170–185. https://doi.org/10.1109/TIP.2019.2924810
25. Correa G, Assuncao PA, Agostini LV, da Silva Cruz LA (2015) Fast HEVC encoding decisions using data mining. IEEE Trans Circuits Syst Video Technol 25(4):660–673. https://doi.org/10.1109/TCSVT.2014.2363753
26. Liu Z, Yu X, Gao Y, Chen S, Ji X, Wang D (2016) CU partition mode decision for HEVC hardwired intra encoder using convolution neural network. IEEE Trans Image Process 25(11):5088–5103. https://doi.org/10.1109/TIP.2016.2601264
27. Mallikarachchi T, Talagala DS, Arachchi HK, Fernando A (2018) Content-adaptive feature-based CU size prediction for fast low-delay video encoding in HEVC. IEEE Trans Circuits Syst Video Technol 28(3):693–705. https://doi.org/10.1109/TCSVT.2016.2619499

28. Xu M, Li T, Wang Z, Deng X, Yang R, Guan Z (2018) Reducing complexity of HEVC: a deep learning approach. IEEE Trans Image Process 27(10):5044–5059. https://doi.org/10.1109/TIP.2018.2847035
29. Zhang Y, Kwong S, Wang X, Yuan H, Pan Z, Xu L (2015) Machine learning-based coding unit depth decisions for flexible complexity allocation in high efficiency video coding. IEEE Trans Image Process 24(7):2225–2238. https://doi.org/10.1109/TIP.2015.2417498
30. Zhu L, Zhang Y, Pan Z, Wang R, Kwong S, Peng Z (2017) Binary and multi-class learning based low complexity optimization for HEVC encoding. IEEE Trans Broadcast 63(3):547–561. https://doi.org/10.1109/TBC.2017.2711142
31. Kuanar S, Rao KR, Bilas M, Bredow J (2019) Adaptive CU mode selection in HEVC intra prediction: a deep learning approach. Circuits Syst Signal Process 38:5081–5102. https://doi.org/10.1007/s00034-019-01110-4
32. Chen Z, Shi J, Li W (2020) Learned fast HEVC intra coding. IEEE Trans Image Process 29:5431–5446. https://doi.org/10.1109/TIP.2020.2982832
33. Zaki F, Mohamed AE, Sayed SG (2021) CtuNet: a deep learning-based framework for fast CTU partitioning of H265/HEVC intra- coding. Ain Shams Eng J 12(2):1859–1866. https://doi.org/10.1016/j.asej.2021.01.001
34. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–17
35. Cetinkaya E, Amirpour H, Ghanbari M, Timmerer C (2021) CTU depth decision algorithms for HEVC: a survey. Signal Process: Image Commun 99:116442. https://doi.org/10.1016/j.image.2021.116442
36. Wang Z, Li F (2021) Convolutional neural network based low complexity hevc intra encoder. Multimed Tools Appl 80(2):2441–2460. https://doi.org/10.1007/s11042-020-09231-8
37. Feng Z, Liu P, Jia K, Duan K (2018) Fast intra CTU depth decision for HEVC. IEEE Access 6:45262–45269. https://doi.org/10.1109/ACCESS.2018.2864881
38. Bossen F (2013) Common test conditions and software reference configurations. JCTVC-L1100 Joint Collaborative Team on Video Coding Technical Report, Geneva
39. Piñol P, Migallón H, López-Granado O, Malumbres MP (2015) Slice-based parallel approach for HEVC encoder. J Supercomput 71(5):1882–1892. https://doi.org/10.1007/s11227-014-1371-y
40. Fraunhofer-HHI (2015) HEVC Reference Software (HM-16.3). available at: http://hevc.hhi.fraunhofer.de/svn/
41. Bjontegaard G (2008) Improvements of the BD-PSNR model. VCEG-M33, Video Coding Experts Group (VCEG) Technical Report, Berlin