Run-time Machine Learning for HEVC/H.265 Fast Partitioning Decision

Svetislav Momcilovic, Nuno Roma, Leonel Sousa
INESC-ID, IST, Universidade Lisboa
Lisbon, Portugal
Email: {Svetislav.Momcilovic, Nuno.Roma, Leonel.Sousa}@inesc-id.pt

Ivan Milentijevic
Faculty of Electronic Engineering, University of Nis
Nis, Serbia
Email: ivan.milentijevic@elfak.ni.ac.rs

***Abstract*—A novel fast Coding Tree Unit partitioning for HEVC/H.265 encoder is proposed in this paper. This method relies on run-time trained neural networks for fast Coding Units splitting decisions. Contrasting to state-of-the-art solutions, this method does not require any pre-training and provides a high adaptivity to the dynamic changes in video contents. By an efficient sampling strategy and a multi-thread implementation, the presented technique successfully mitigates the computational overhead inherent to the training process on both the overall processing performance and on the initial encoding delay. The experiments show that the proposed method successfully reduces the HEVC/H.265 encoding time for up to 65%, with a negligible rate-distortion penalties.**

***Keywords*-HEVC/H.265; video coding; neural networks; multi-threading; machine learning**

I. Introduction

The newest HEVC/H.265 video coding standard has shown to achieve high coding efficiency gains (on average 40%), when compared with the previous H.264/AVC [1]. However, such gains are achieved in the cost of an increased computational complexity, specially at the encoder side, where it can be up to 5 times higher than H.264/AVC [2].

The highly flexible quad-tree partitioning scheme applied in HEVC/H.265, represents a fundamental improvement considering its impact on the Rate-Distortion (RD) efficiency. An exhaustive RD Optimization (RDO), adopted in the HEVC/H.265 reference implementation, applies a set of demanding inter/intra prediction modules on each partitioning level, to select the partitioning structure with maximum coding efficiency. However, such RDO introduces a dramatical computational burden, making the encoder impractical for usual applications in commodity computers [2].

At this respect, several works have proposed various heuristic solutions for fast partitioning decision [3], [4]. By relying on pyramid motion divergence for early termination of the RDO, the method proposed in [4] decreases a computational time for up to 60%, with an RD penalty of less than 4%. Alternative solutions [5], [6] apply machine learning models to adjust the decision parameters using large training data sets. In [6] a support vector machine is applied to support a fast partitioning decision. This method achieves an average reduction of the encoding time of 37% (although it can be as high as 71%), with a maximum RD cost of 6%. However, the efficiency of such solutions highly depends on the selected training corpus, which is processed beforehand in a time-consuming off-line training procedure.

In contrast to these solutions, the novel method, proposed herein, applies the run-time trained Neural Networkss (NNs) for early termination of the RDO. Such training is completely offloaded from the encoding thread and it does not introduce any initial encoding delay or significant processing overheads. By carefully selecting the input variables and by an efficient sampling strategy we succeed to achieve the reduction of the encoding time for up to 65%, and RD penalties of less than 2.5%, with significantly smaller data sets. Finally, by applying a strict validation policy, the proposed method dynamically adapts to the video contents and reduces the impact of the miss-prediction error propagation on the resulting RD performance. In accordance, the main contributions of this paper can be summarized as follows:

- complete solution for a fast HEVC/H.265 partitioning prediction based on run-time machine learning;
- prediction performance higher than state-of-the-art solutions, without requiring any offline pre-training;
- run-time adaptivity to video content changes;
- efficient sub-sampling strategy;

II. CTU Partitioning Prediction in HEVC/H.265

The HEVC/H.265 performs the video coding on the level of the equal square-shape Coding Tree Units (CTUs), whose predefined size may vary from 64×64 to 8×8 pixels. Each CTU is further split into one or several square-shaped Coding Units (CUs), for which the prediction mode is signaled as *inter* or *intra* (see Fig. 1). Inter prediction CUs are further split into one, two or four Prediction Units (PUs), according to 8 different partitioning modes. To attain the best performance, the exhaustive RDO applies the motion compensation on all possible partitioning structures, which dramatically increases the computation load of the encoder.

A. Proposed Fast CTU Partitioning Method

The proposed method is integrated in HEVC/H.265 encoder to efficiently predict the correct splitting decision without testing further CTU partitioning. On each partitioning level (i.e. 64×64, 32×32, and 16×16) this decision is supported by a separate run-time trained NN, controlled by corresponding *finite state machine* (see Section III-A).

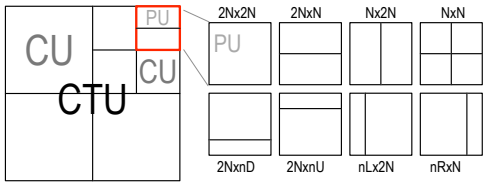

Figure 1. CTU partitioning.

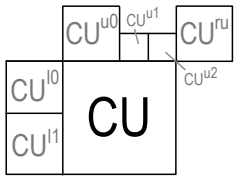

Figure 2. CU spatial neighbors predictors.

Algorithm 1 The Proposed Fast CTU Partitioning Method

EncodeSequence()

1: Encode I Frame and initial P/B Frame
2: **for** all remaining P/B frames **do**
3: Initialize/Update prediction states[1]
4: **for** all CTUs **do**
5: EncodeCU(cu, 0)
6: **end for**
7: **end for**

EncodeCU(cu,depth)

1: $minRDcost$ = min(TestMerge(), TestAllPUs())
2: $split = (depth < maxDepth)$
3: **if** $sm\_state(depth) == predict$[1] **then**
4: $split$ = predictSplit($inputs(cu)$,$depth$)
5: **end if**
6: **if** $split$ **then**
7: tempRDcost = 0
8: **for** all children **do**
9: tempRDcost += EncodeCU(cu_child, $depth$+1)
10: **end for**
11: $split$ = ($tempRDcost < minRDcost$)
12: $minRDcost$ = min($minRDcost$, $tempRDcost$)
13: **end if**
14: **if** $sm\_state(depth) == sample$[1] and $is\_samle(cu)$[2] **then**
15: AddTrainingSample($sample(cu, split)$, $depth$)
16: **end if**
17: return $minRDcost$

[1] see Section III-A
[2] see Section III-B

A general description of the proposed fast partitioning is presented in Algorithm 1. Since this method relies not only on spatial, but also on temporal predictors (see Section II-B), the *finite state machines* are initialized only after the encoding of first P/B-frame (*EncodeSequence*, line 1) and updated before each subsequent frame (line 3). The encoding of each CTU starts with the encoding of zero-depth CU (line 5).

The $EncodeCU$ procedure starts by testing all PUs and the $Merge$ mode (*EncodeCU*, line 1) to find the prediction with the minimum RD cost ($minRDcost$). In the case of splitting, this procedure is recursively called for all the *children* partitions (lines 6–13). However, while the exhaustive RDO assumes the positive splitting decision as long as the maximum partitioning depth ($maxDepth$) is not reached (line 2), the proposed solution relies on NNs to predict this decision (line 4). The $predict$ state (line 3) of *finite state machines* signals that the assigned NN is ready to be applied. Contrastingly, in the $sample$ state, the splitting is initially assumed as $true$ (line 2) and the partitioning is proceed (line 9). However, according to the obtained RD-cost, it is verified if the splitting was really advantageous or not (lines 11–12). This decision is further used to form the training sample (line 15) together with the selected input variables.

Table I
INPUT VARIABLES CONSIDERED IN THE PROPOSED METHOD.

Parameter	Explanation
$BMaxDepthU$	Maximum CU^u depth
$BMaxDepthL$	Maximum CU^l depth
$BDepthRU$	CU^{ru} depth
$MaxDepthP$	Maximum CU^p depth
$BRDCostU^{(1)}$	$\Sigma^i(RDcost(CU^{u_i})/h(2)(CU^{u_i}))$
$BRDCostL^{(1)}$	$\Sigma^i(RDcost(CU^{l_i})/h^{(2)}(CU^{l_i}))$
$BRDCostRU^{(1)}$	$RDcost(CU^{ru})$
$RDCostP^{(1)}$	$RDcost(CU^p)$
$RDCost2N\times2N^{(1)}$	$RDcost(CU^{ru})$
$RDCostMerge^{(1)}$	$RDcost(CU^{merge})$
$RDCost2N\times N^{(1)}$	$RDcost(CU^{2N\times N})$
$RDCostN\times2N^{(1)}$	$RDcost(CU^{N\times 2N})$
$MergeFlag$	Merge flag
$RDCostParrent^{(1),(3)}$	$RDcost$ of upper-level CU

(1) All RDcosts are scaled with *minRDcost*
(2) $h(CU^i)$=height(CU^i)/8, i.e. 1 for 8×8, 2 for 16×16; 4 for 32×32; and 8 for 64×64 CUs
(3) *RDCostParrent* is ignored for *depth=0*

B. Selection of the Input Variables

The proposed prediction relies on the set of input variables, gathered in the motion compensation procedure for already processed CUs. The left (CU^l), up (CU^u) and right-up (CU^{ru}) spatial neighbors are considered, as well as the temporal neighbor from the previous frame (CU^p) (see Table I). In particular, the maximum partitioning depths among all the neighbors on both up ($BMaxDepthU$) and the left ($BMaxDepthL$) CU borders were selected (see Fig. 2). Moreover, we selected the depth of the right-up neighbor ($BDepthRU$), and the maximum depth of the CU^p temporal neighbor ($MaxDepthP$). The CU^p is specially relevant for high frame rates, where the adjacent frames slightly differ to each other. For all these CUs we also considered respective RD-costs, i.e. the sums of the RD-costs on the up ($BRDCostU$) and left ($BRDCostL$) borders and the RD-costs of the CU^{ru} ($BRDCostRU$), and CU^p ($RDCostP$), all scaled with the respective CU height.

Furthermore, the RD costs of the processed PUs (i.e. $RDCost2N\times2N$, $RDCostN\times2N$, and $RDCost2N\times N$), and the $Merge$ mode ($RDCostMerge$), scaled with the $minRDcost$ (see Algorithm 1), are considered. In fact, if the $minRDcost$ corresponds to the finer-grained PUs, a further splitting is expected. Contrastingly, if the $minRDcost$ is

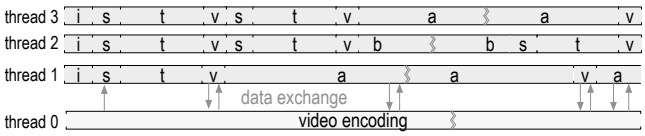

Figure 3. Video encoding and partitioning decision threads. Prediction state machine: i - init; s-sample; v - validate; a - apply; b - block (suspend)

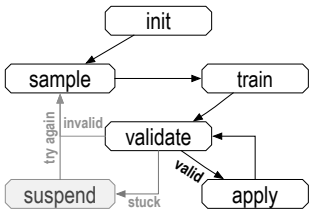

Figure 4. Prediction state machine.

found for 2N×2N or $Merge$ mode, it is likely that the splitting is not needed. The $MergeFlag$ [1] is also considered as an input, since this mode usually corresponds to highly homogeneous areas, where no splitting is expected. Finally, the RD cost of the coarser grained CU ($RDCostParrent$) was also considered. In fact, if the achieved RD cost is not smaller than the parent CU cost, a splitting is not expected.

III. Application of neural networks for Fast Partitioning Decision

To predict the best CTU partitioning, 3 small NNs were applied, i.e. one per partitioning level, controlled by respective *finite state machines*. The NNs are trained asynchronously from the encoding thread (see Fig. 3), and applied as soon as the training is completed. Due to its reduced computational complexity, the quasi-Newton training method is adopted [8]. To additionally reduce the training time, the sub-sampling was applied for non-zero CU depths.

A. Neural networks control state machine

The proposed method is controlled by three independent finite state machines, updated after the encoding of each frame. They enclose 6 states, namely: *init*, *sample*, *training*, *validation*, *suspend* and *apply* (see Fig. 4).

The initial *init* state does not consider any action. As soon as the first I-frame and the first P/B-frame are encoded, the *sample* state is set. In this state, the training samples are extracted from the encoded CUs, until the specified number of samples is reached, and the *train* state is activated.

In the *train* state, the quasi-Newton training is performed in 3 asynchronous threads, using the extracted samples. However, such run-time training allows only the application of reduced NNs, with no more than a single intermediate layer. When the training is finished, the *validate* state is set.

In the *validate* state, the prediction is validated according to the resulting prediction error, i.e. the ratio between the wrong early terminations, and the total number of the applied predictions. To prevent the propagation of prediction errors, this threshold is set to less than 5%. If a validation succeeds, the *apply* state is activated (see Fig. 3, *thread 1*). To reduce the overhead, the sampling is partially performed even in the *validate* state. However, these samples are only used if a validation fails (*thread 2/3*). If the validation fails two successive times, the prediction is *suspended* (*thread 2*).

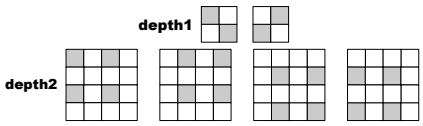

Figure 5. Sampling strategy in successive video frames (CUs selected for sampling are in gray).

In practice, the state machine is set to *suspend* when it is estimated that the desired prediction efficiency is not achievable using the samples from the current video content. To relax the subsequent computations, the training is suspended for a predefined number of frames. As soon as this number is reached, the *sample* state is activated again.

In the *apply* state the NNs are applied to make the CU splitting decision. Finally, to adapt to changes in video content, the validation is repeated with a predefined frequency.

B. Sampling strategy

In the proposed method, the number of training samples quadruples with each partitioning level. Consequently, a too large training corpus can stuck the state machine at the *train* state for a long period. In such a case, the trained NN might be no longer valid for already changed video content. To prevent this, an efficient sampling strategy is deployed.

In particular, a sampling rate of 2 is applied for the $depth1$ CUs (32×32). The position of *sampled* CUs is alternatively shifted left and right for subsequent frames (see Fig. 5). For the $depth2$ (16×16) a sampling rate of 4 is applied, while the position of the sampled CU turns clockwise, within a 2×2 square. However, such sub-sampling is only applied when the upper partitioning depth is not in the *apply* state, since the early termination on the coarser-grained level can significantly reduce the number of samples.

IV. Experimental Evaluation

The experimental evaluation of the proposed fast partitioning was performed by relying on HM 16.0 HEVC reference software [9], and the open source NN library OpenNN [10]. The offloading of the training threads is performed by the C++ 11 multi-threading instructions. The evaluation was performed on platforms equipped with an Intel Core i7 4770K CPU at 3.5GHz, and DDR3 4x8GB, with Linux OS.

The test video sequences are selected in a way to represent different resolutions, frame rates, and spatial/movement characteristics (see Table II). The RD-performance is evaluated using the Bjøntegaard's method [7], with quantizer

Table III
COMPARISON OF THE PROPOSED SOLUTION WITH THE STATE-OF-THE-ART METHODS

Sequence	Proposed			[4]			[6]		
	$\Delta T(\%)$	BD$_{br}$ (%)	BD$_{psnr}$ (dB)	$\Delta T(\%)$	BD$_{br}$ (%)	BD$_{psnr}$ (dB)	$\Delta T(\%)$	BD$_{br}$ (%)	BD$_{psnr}$ (dB)
BasketballDrive	30.33	1.24	-0.03	51.89	3.88	-0.09	39.73	1.75	-0.04
ParkScene	65.24	-0.01	0.03	42.65	2.01	-0.07	41.83	1.08	-0.03
Kimono	45.14	1.35	-0.03	46.95	1.6	-0.06	30.31	0.36	-0.01
Cactus	43.74	1.76	-0.03	42.19	2.14	-0.05	45.67	1.92	-0.04
BQTerrace	65.24	1.04	-0.02	51.51	0.6	-0.02	47.36	0.79	-0.02
Traffic	35.29	1.6	-0.05	46.32	2.5	-0.10	48.53	1.84	-0.01
average	47.5	1.17	-0.02	46.92	2.12	-0.07	42.24	1.29	-0.03

Table II
THE CHARACTERISTICS OF THE TEST VIDEO SEQUENCES.

Sequence	**Format**	**Frame rate**	#Frames
$Johnny$	1280×720	60	600
$Kristen\&Sara$	1280×720	60	600
$ParkScene$	1920×1080	24	240
$Cactus$	1920×1080	50	500
$BQTerrace$	1920×1080	60	600
$BasketballDrive$	1920×1080	50	500
$Kimono$	1920×1080	24	240
$Traffic$	2560×1600	30	150
$SteamLocom.$	2560×1600	60	150

Table IV
PERFORMANCE OF THE PROPOSED METHOD FOR ADDITIONAL TEST SEQUENCES.

Sequence	$\Delta T(\%)$	**BD$_{br}$ (%)**	**BD$_{psnr}$ (dB)**
$Johnny$	65.53	2.47	-0.06
$Kristen\&Sara$	48.41	2.13	-0.07
$SteamLocom.$	62.67	1.31	-0.03

values of QP={22,27,32,37}, and HM software as a reference. The average differences are expressed in both bitrate (BD$_{br}$ in %) and PSNR (BD$_{PNSR}$ in dB). The $encoder\_lowdelay\_P\_main$ configuration is adopted.

Table III presents the comparison of the proposed method with two different state-of-the-art approaches, based on a heuristic [4], and a machine learning solutions [6]. As it can be seen, the proposed solution achieves a higher reduction of the encoding time, for smaller RD penalties. However, in contrast to the proposed run-time training method, the solution proposed in [6] considers an offline training procedure.

The performance obtained for additional test sequences that do not appear in [4] and [6] is presented in Table IV. As it can be observed, the proposed method reduces the computational time for up to 65%, while the RD penalties do not surpass 2.5% for BD$_{br}$, and 0.08 dB for BD$_{psnr}$.

V. CONCLUSION

A fast CTU partitioning for HEVC/H.265 was proposed, based on run-time trained NNs. The method does not require any offline pre-training, and dynamically adapts to the changes in the video content. The training time is additionally reduced by an efficient sampling strategy. The experiments show that the method reduces the encoding time for up to 65%, with less than a 2.5% of RD penalty, which is performance superior to the state-of-the-art solutions.

ACKNOWLEDGMENT

This work was supported by national funds through FCT (Fundao para a Cincia e a Tecnologia), under projects PTDC/EEI-ELC/3152/2012 and UID/CEC/50021/2013, and by the COST under NESUS ICT COST Action IC1305. Svetislav Momcilovic also acknowledges FCT for the Post-Doc scholarship SFRH/BPD/75261/2010.

REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard." *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[2] G. Correa, P. Assuncao, L. Agostini, and L. da Silva Cruz, "Performance and computational complexity assessment of high-efficiency video encoders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1899–1909, Dec 2012.

[3] H.-M. Yoo and J.-W. Suh, "Fast coding unit decision algorithm based on inter and intra prediction unit termination for HEVC," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, Jan 2013, pp. 300–301.

[4] J. Xiong, H. Li, Q. Wu, and F. Meng, "A fast HEVC inter CU selection method based on pyramid motion divergence," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 559–564, 2014.

[5] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Complexity scalability for real-time HEVC encoders," *Journal of Real-Time Image Processing*, pp. 1–16, 2014.

[6] X. Shen and L. Yu, "CU splitting early termination based on weighted SVM," *EURASIP J Image Video Process*, vol. 2013, no. 1, 2013.

[7] G. Bjøntegaard, "Calculation of Average PSNR Differences between RD curves," *ITU, Doc. VCEG-M33*, April 2001.

[8] R. Setiono and L. C. K. Hui, "Use of a quasi-newton method in a feedforward neural network construction algorithm," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, pp. 273–277, 1995.

[9] F. Bossen, D. Flynn, and K. Sühring, "HEVC reference software manual," *JCTVC-D404, Daegu, Korea*, 2011.

[10] R. Lopez, "Open NN: An Open Source Neural Networks C++ Library [software]," 2014.