ACCELERATION OF KVAAZAR HEVC INTRA ENCODER WITH MACHINE LEARNING

Alexandre Mercat, Ari Lemmetti, Marko Viitanen and Jarno Vanne

Tampere University
Korkeakoulunkatu 10, Tampere, 33720, Finland
{alexandre.mercat, ari.lemmetti, marko.viitanen, jarno.vanne}@tuni.fi

ABSTRACT
The complexity of High Efficiency Video Coding (HEVC) poses a real challenge to HEVC encoder implementations. Particularly, the complexity stems from the HEVC quad-tree structure that also has an integral part in HEVC coding efficiency. This paper presents a Machine Learning (ML) based technique for pruning the HEVC quad-tree without deteriorating coding gain. We show how ML decision trees can be used to predict a depth interval for a quad-tree before the Rate-Distortion Optimization (RDO). This approach limits the number of RDO candidates and thus speeds up encoding. The proposed technique works particularly well with high-quality video coding and it is shown to accelerate the very slow preset of practical Kvazaar HEVC intra encoder by 1.35× with 0.49% bit rate increase. Compared with the corresponding preset of x265 encoder, Kvazaar is 2.12× as fast at a cost of under 1.21% bit rate overhead. These results indicate that the optimized Kvazaar is the leading open-source encoder in high-quality HEVC intra coding.

Index Terms—High Efficiency Video Coding (HEVC), Intra Encoder, Machine Learning (ML), Complexity Reduction, Quad-Tree

1. INTRODUCTION
Nowadays, numerous applications encode and stream video content. Cisco [1] reports that 75% of total IP traffic is dedicated to video in 2017 and estimates it to grow to 82% by 2022. The latest international video coding standard, High Efficiency Video Coding (HEVC) [2], is developed to address this growth. HEVC is published as twin text by ITU, ISO, and IEC as ITU-T H.265 | ISO/IEC 23008-2. When compared with the previous MPEG AVC standard, HEVC Main profile reduces the bit rate by 50% for similar objective video quality [3, 4] but at a cost of over five times as high encoding complexity [5]. This overhead stems mostly from the new quad-tree block partitioning scheme, which exponentially increases the execution time of Rate-Distortion Optimization (RDO) process in HEVC.

As shown in our previous work [6], quad-tree partitioning of Coding Tree Unit (CTU) has a potential to reduce energy up to 78% in a practical software HEVC intra encoder. Previous studies on low complexity quad-tree partitioning can be classified into two categories: 1) the early termination mechanisms which dynamically terminate processing during the RDO process when future gains are unlikely; and 2) the prediction-based complexity reduction techniques which are applied before the RDO process to predict the quad-tree partitioning with lower complexity than the full RDO process. In this paper, we focus on the latter techniques.

Authors in [7,8] proposed to reduce the complexity of the HEVC encoder by skipping some depth levels of the quad-tree partitioning. The skipped depths were selected based on the correlation between the minimum depths of the collocated CTUs in the current and previous frames. Results in [8] showed an average time savings of 45% for a Bjøntegaard Delta Bit Rate (BD-BR) increase of 1.9%.

Works in [9–14] used CTU texture complexities to predict the quad-tree partitioning. Authors in [10] classified a Coding Unit (CU) as split, non-split, or undetermined. They used global and local edge complexities in four different directions (horizontal, vertical, 45°, and 135° diagonals) of CU and sub-CUs. This method provided a complexity reduction of 52% for a BD-BR penalty of 0.8%. Feng et al. [11] used information entropy of CUs and sub-CUs saliency maps to predict the CU sizes. The method reduced the complexity by 37.9% for a BD-BR cost of 0.62%.

For the time being, several Machine Learning (ML) based solutions have been proposed to reduce the complexity of the HEVC encoder. Authors in [15, 16] presented an intra CU size classifier based on data-mining with an offline classifier training. The classifier was a three-node decision tree that used mean and variance of CUs and sub-CUs as features. This algorithm reduced coding time by 52% at the expense of BD-BR increase of 2%. Duanmu et al. [17] presented a fast CU partitioning scheme using ML for screen content coding. They used many features such as CU luma variances, color Kurtosis of CU, and gradient Kurtosis of CU. Shen and Yu [18] proposed an early termination algorithm for CU splitting. It was based on weighted Support Vector Machine (SVM). The Rate-Distortion (RD)-cost losses due to the misclassification are used as features (weights) in SVM training. In [19], authors modeled the CU depth decision process in HEVC with a three-level hierarchical decision problem using SVM classifiers. Liu et al. [20] presented Convolution Neural Network (CNN) based CU partitioning prediction scheme that infers CU and Prediction Unit (PU) split decision. The presented solution reduced the coding time by 61.1% at the expense of BD-BR increase of 2.67%.

In this paper, we propose a new method to predict HEVC quad-tree partitioning in order to reduce the complexity of the practical Kvazaar HEVC encoder. The proposed complexity reduction technique makes use of an ML algorithm to predict an adaptive quad-tree partitioning interval for a CU before starting the RDO process. The existing complexity reduction techniques in the literature worked on the HEVC Test Model (HM) [21] software encoder and their relative performance figures do not necessarily scale to practical encoders due to the inherent complexity of HM. Unlike them, our work focuses on complexity reduction under a practical framework.

The rest of this paper is organized as follows. Section 2 presents a brief overview of the Kvazaar HEVC intra encoder. Section 3 details our ML based algorithm for quad-tree partitioning prediction. Performance of the proposed complexity reduction technique is presented in Section 4. Finally, Section 5 concludes the paper.
2. KVAZAAR HEVC INTRA ENCODER

Kvaazar [22] is an academic, cross-platform software HEVC encoder. It is open-sourced under the LGPLv2.1 license. Unlike the reference encoder HM, Kvaazar takes advantage of multiple processor cores and Single Instruction Multiple Data (SIMD) instructions [23]. Kvaazar is also supported by FFmpeg and Libav projects where it can be used as an external library. The veryslow preset of Kvaazar intra encoder is described next at a high level as it is used in this work.

2.1. Coding Tree Unit Partitioning

The best CTU structure is the result of a recursive depth-first search in the quad-tree. Progressing top-down, CU split decisions are made by comparing the RD-costs of the CU at current depth versus the combined RD-cost of the 4 sub-CUs, where sub-CUs may be split even further. The RD-cost $J$ is computed as $J = D + \lambda \cdot R$, where $D$ is the distortion, $R$ is the bit rate, and $\lambda$ Lagrange multiplier [24]. Distortion $D$ is computed by Sum of Squared Differences (SSD).

Two early termination mechanisms are implemented to speed up coding tree partitioning. The first one terminates the search when all transformed coefficients of the current CU are equal to zero. The second one prevents further search of the sub-CUs if their accumulated RD-cost (1 to 3 sub-CUs) is higher than that of their parent CU at the previous depth.

2.2. Intra Mode Decision

The intra search algorithm consists of two stages. First, a logarithmic search is performed to find the minimum distortion among angular intra modes. In a rough step, luma distortion is computed by Sum of Absolute Transformed Differences (SATD) between luma samples of the source image and prediction blocks. Luma mode bits are multiplied by square root of Lagrange multiplier $\lambda$ and then added to the distortion for an estimated RD-cost. Most probable angular modes, planar mode, and DC mode are also considered.

In the RDO stage of the search, previously estimated modes are sorted according to their costs. Depending on the current block size, at most 2 or 3 best modes are forwarded to the RDO search. Most probable modes are again added to the list of modes if they are not present. Rate-distortion optimized quantization is performed for the selected modes which are then completely reconstructed. SSD between the reconstructed and source image samples is computed by adding up luma and both chroma channels. The number of used bits is calculated through CABAC, including transform tree and transformed coefficient bits. The mode with a minimum RD-cost [25] is selected as the best mode.

3. PROPOSED COMPLEXITY REDUCTION TECHNIQUE

The aim of the proposed technique is to replace the brute force scheme usually employed in HEVC encoders with a low-complexity algorithm that predicts a depth interval of CTU partitioning in which the HEVC encoder is constrained to apply the RDO process. In general, limiting the search to a certain interval reduces encoding complexity. The proposed technique is divided into two stages: 1) the ML-based one-shot prediction of quad-tree partitioning and 2) the interval prediction for quad-tree partitioning.

3.1. ML-Based One-shot Prediction of Quad-Tree Partitioning

The quad-tree prediction is called one-shot as the prediction is applied only once, before starting the RDO process of the CTU.

\[ \text{Fig. 1. Classification problem between CUs at depths } d \text{ and } d-1. \]

3.1.1. Quad-Tree Partitioning as a Classification Problem

Following a bottom-up approach (from CU size of $4 \times 4^1$ to $32 \times 32$), the main idea is to determine the best partitioning of a given CU between $2N \times 2N$ pixels and $N \times N$ pixel sub-blocks at each depth. Fig. 1 illustrates the classification problem which predicts whether the CU at depth $d$ has to be merged with CU at depth $d-1$.

At each depth $d$, the classification problem is solved by a ML approach across data-mining classifiers. The aforementioned state-of-the-art studies gather many characteristics used to predict the coding tree decomposition of a CTU. To predict the coding tree in one-shot, only characteristics independent from the encoding process with a limited overhead of computation are considered.

To avoid overfitting, i.e., overspecializing a model to a training set, the sequences are split in two data sets: the training set composed of one sequence per class and the experimental set composed of the other sequences. A training data pool is extracted from a fixed number of CTUs of each sequence of the training data set. For each depth $d$, 80,000 instances are randomly sampled from the previous defined data pool, composed of 40,000 instances of each prediction decision. The training setup of the learning algorithm and the choice of the features is detailed in [26]. The features have been deduced from an extensive study of two factors: the information gain provided by the Waikato Environment for Knowledge Analysis (WEKA) software and the overhead of computation under a practical encoder. The set of features is composed of the following 12 features:

- **CU var** [9, 13–17]: the variance of the CU luma samples at depth $d$ (1 feature).
- **Lower-CU var** [9, 13, 15–17]: the variances of the 4 sub-CU luma samples at depth $d+1$ (4 features).
- **Upper-CU var** [9, 13–16]: the variances of the upper CU luma samples at depth $d-1$ (1 features).
- **Nhbr-CU var** [13, 17]: the variances of the neighboring CU luma samples at depth $d$ in the Z-scan order (3 features).
- **Var of lower-CU mean** [15, 16]: the variance of the mean of the 4 sub-CU luma samples at depth $d+1$ (1 feature).
- **Var of lower-CU var** [15, 16]: the variance of the variance of the 4 sub-CU luma samples at depth $d+1$ (1 feature).
- **Quantization Parameter (QP)**: the Quantization Parameter (QP) of the frame (1 feature).

The training of the decision trees is performed with the C4.5 algorithm [27]. As the information gain, the C4.5 algorithm uses Kullback-Leibler Divergence (KLD) to select the best features for each decision. The C4.5 algorithm is iterating among all training instances and searches the threshold that achieves the best classification for each feature, i.e., with the highest information gain. Then, the features and their corresponding thresholds are used to divide the training instances into two subsets. To finish, the process is recurrently iterated on the two different subsets of training instances.

Table 1 summarizes the trained tree sizes, number of leaves, and the Percentage of Correctly Classify Instances (PCCI) of the 4 decision trees, where PCCI (given by the 10-fold cross-validation) de-

\[ 18 \times 8 \text{ CU split into } 4 \times 4 \text{ prediction units is considered as } 4 \times 4 \text{ CUs.} \]
The algorithm does not try to merge neighboring blocks of different depths as illustrated in Fig. 3. If the former condition is true, the algorithm browses the CTU prediction by taking the block size into account. Afterwards, the algorithm tests if the 4 neighboring blocks in the Z-scan order have the same depth.

The algorithm merges the group of four neighboring blocks (in the Z-scan order) if they are at the same depth. The number of merged blocks depends on the input P and is not predictable. The merged blocks are indicated in red in Fig. 4.

After testing each depth, the one-shot quad-tree prediction is ready and the obtained P is delivered to the next stage (Section 3.2) for interval prediction.

3.2. Interval Prediction for Quad-Tree Partitioning

The goal of this stage is to relax the predicted CTU partitioning by generating a prediction interval around the input P (Section 3.1.2). The interval is specified between the upper prediction $P_U$ and the lower prediction $P_L$. The algorithm is split into three steps:

1. The first step applies the algorithm called Upper Expansion on the input P. The algorithm merges the group of four neighboring blocks (in the Z-scan order) if they are at the same depth. The number of merged blocks depends on P and is not predictable. The merged blocks are indicated in red in Fig. 4.

2. The second step applies the “complementary” algorithm, called Lower Expansion, to generate the lower prediction $P_L$.

3. Finally, the third step applies a second pass of the Upper Expansion algorithm on the prediction yielded in the step 1 to generate the upper prediction $P_U$.

When constrained between the $P_L$ and $P_U$, the encoder tests 2 to 3 depth levels spread around the input P depending on the number of blocks merged by the third step.

3.3. Complexity Reduction Scheme

Fig. 2 presents a high-level diagram of our resulting ML complexity reduction scheme. Thanks to the offline training of the decision trees, all the frames are constrained and no learning frame is needed. First, the features (Section 3.1.1) are computed for a CTU of interest. Secondly, the ML based algorithm (Section 3.1.2) uses the computed features to generate a coarse quad-tree prediction for the CTU. Thirdly, the interval prediction algorithm (Section 3.2) uses the coarse prediction to compute a prediction interval between the upper prediction $P_U$ and the lower prediction $P_L$. Finally, the HEVC encoder is forced to limit RDO between the interval formed by $P_U$ and $P_L$.

4. PERFORMANCE ANALYSIS

The impact of the proposed complexity reduction techniques on Kvazaar performance is measured with the 18 8-bit common test sequences encoded on an Intel Core i7-5960X Extreme (8 × 3.0 GHz) processor with 32 GB of RAM memory.

Table 2 shows the BD-BR and speedup results of the proposed complexity reduction scheme over the anchor version of Kvazaar [23], version 2.8 of x265 [28], and version 16.8 of HM [21]. The benchmarking is performed with command line parameters listed in Table 3. The veyslow presets of Kvazaar and x265 are close to each other in functionality, so they are selected for the comparison. HM is set to use default configuration of All-intra coding.
Table 2. Coding speed and efficiency of the optimized Kvazaar over the original Kvazaar, x265, and HM.

<table>
<thead>
<tr>
<th>Format</th>
<th>Sequences</th>
<th>Proposed technique</th>
<th>Kvazaar vs. x265</th>
<th>Kvazaar vs. HM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Speedup 16 threads</td>
<td>BD-BR</td>
<td>Speedup 16 threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.35× 0.27%</td>
<td>1.48× 1.48%</td>
<td>8.41× 68.69%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.36× 0.39%</td>
<td>1.89× 1.98%</td>
<td>8.66× 70.19%</td>
</tr>
<tr>
<td>Class A</td>
<td>PeopleOnStreet</td>
<td>1.47× 0.84%</td>
<td>2.22× 1.62%</td>
<td>10.69× 86.86%</td>
</tr>
<tr>
<td></td>
<td>Traffic*</td>
<td>1.33× 0.63%</td>
<td>1.95× 2.17%</td>
<td>8.09× 65.75%</td>
</tr>
<tr>
<td>Class B</td>
<td>BasketballDrive</td>
<td>1.40× 0.42%</td>
<td>2.09× 1.77%</td>
<td>8.83× 71.79%</td>
</tr>
<tr>
<td></td>
<td>BQTerrace</td>
<td>1.48× 0.82%</td>
<td>2.33× 0.93%</td>
<td>11.54× 93.13%</td>
</tr>
<tr>
<td></td>
<td>Cactus*</td>
<td>1.36× 0.45%</td>
<td>2.08× 1.19%</td>
<td>8.38× 68.28%</td>
</tr>
<tr>
<td>Class C</td>
<td>BasketballDrill</td>
<td>1.40× 1.02%</td>
<td>2.03× 2.75%</td>
<td>7.98× 63.37%</td>
</tr>
<tr>
<td></td>
<td>BQMall*</td>
<td>1.33× 0.42%</td>
<td>1.97× 1.11%</td>
<td>7.78× 63.59%</td>
</tr>
<tr>
<td></td>
<td>PartyScene</td>
<td>1.27× 0.10%</td>
<td>1.89× 1.08%</td>
<td>5.70× 47.01%</td>
</tr>
<tr>
<td></td>
<td>RaceHorses</td>
<td>1.38× 0.38%</td>
<td>2.10× 1.13%</td>
<td>7.48× 60.12%</td>
</tr>
<tr>
<td>Class D</td>
<td>BasketballPass*</td>
<td>1.38× 0.28%</td>
<td>2.46× -0.39%</td>
<td>7.51× 60.44%</td>
</tr>
<tr>
<td></td>
<td>BlowingBubbles</td>
<td>1.26× 0.10%</td>
<td>2.23× 0.55%</td>
<td>5.39× 43.90%</td>
</tr>
<tr>
<td></td>
<td>BQSquare</td>
<td>1.31× 0.13%</td>
<td>2.38× 0.45%</td>
<td>5.84× 47.20%</td>
</tr>
<tr>
<td></td>
<td>RaceHorses</td>
<td>1.35× 0.23%</td>
<td>2.39× 0.05%</td>
<td>6.67× 52.70%</td>
</tr>
<tr>
<td>Class E</td>
<td>FourPeople</td>
<td>1.31× 0.83%</td>
<td>1.94× 1.62%</td>
<td>9.86× 79.67%</td>
</tr>
<tr>
<td></td>
<td>Johnny*</td>
<td>1.31× 0.95%</td>
<td>2.27× 1.20%</td>
<td>12.70× 99.55%</td>
</tr>
<tr>
<td></td>
<td>KristenAndSara</td>
<td>1.35× 0.60%</td>
<td>2.14× 1.19%</td>
<td>11.84× 92.09%</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>1.35× 0.49%</td>
<td>2.12× 1.21%</td>
<td>8.52× 68.57%</td>
</tr>
</tbody>
</table>

*test sequences used for the training phase

4.1. Speedup over the original Kvazaar

The proposed complexity reduction technique speeds up the encoding process by 1.35× on average for an BD-BR increase of 0.49%. First of all, it is noticeable in Table 2 that sequences used to constitute the training data set (marked by *) do not achieve better results compared with other sequences, which show the non-overfitted behavior of the decision trees. Therefore, training sequences are also included in Table 2.

The results also show that the class D has less degradation in terms of BD-BR (+0.18% in average) than the other classes. This stems from the selected strategy for CTU partitioning prediction (Section 3.2), where neighboring blocks of different depths cannot be merged. This approach tends to result in finer-grained CTU partitioning which favors smaller resolutions.

4.2. Speedup over x265 and HM

Kvazaar and x265 were run with 16 threads using all available optimizations. Both Kvazaar and x265 support multi-threading and SIMD optimizations for 8-bit content. Kvazaar encodes each test sequence faster for similar RD performance in a majority of cases. The average speedup of Kvazaar over x265 is 2.12× with a 1.21% increase in BD-BR. The average complexity overhead of the proposed technique is around 2% in Kvazaar.

As HM does not implement multi-threading, Table 2 includes Kvazaar results with a single thread for the sake of more straightforward algorithm level comparison. Kvazaar is shown to be 8.52× as fast as HM, when using only one thread, and more than 68.5× as fast as HM with 16 threads. In spite of the large speedup, the BD-BR is deteriorated by only 2.09% against HM.

5. CONCLUSION

This paper presented a complexity reduction technique that makes use of an ML algorithm to predict adaptive quad-tree partitioning interval for a CTU before the RDO process. The proposed technique used a one-shot quad-tree partitioning prediction based on decision trees. It accelerates the practical Kvazaar HEVC encoder by 1.35× with 0.49% BD-BR overhead. The optimized Kvazaar is 2.12× as fast as the veryslow preset of x265 encoder with a BD-BR increase of 1.21%. These rate-distortion-complexity results show that Kvazaar is currently the front-runner among the existing open-source HEVC intra encoders when complexity aspect is taken into account.

6. ACKNOWLEDGEMENT

This work was supported in part by the European Celtic-Plus Project VIRTUOSE and the Academy of Finland (decision no. 301820). The authors would also like to thank all contributors of the Kvazaar open-source project [22].

7. REFERENCES


