Complexity Control of HEVC through Quadtree Depth Estimation

Guilherme Correa #1, Pedro Assuncao †2, Luciano Agostini *3, Luis A. da Silva Cruz #4

# Instituto de Telecomunicações – University of Coimbra – Coimbra, Portugal
† Instituto de Telecomunicações – Polytechnic Institute of Leiria – Leiria, Portugal
* Group of Architectures and Integrated Circuits – Federal University of Pelotas – Pelotas, Brazil

Abstract—The emerging HEVC standard introduces a number of tools which increase compression efficiency in comparison to its predecessors at the cost of greater computational complexity. This paper proposes a complexity control method for HEVC encoders based on dynamic adjustment of the newly proposed coding tree structures. The method improves a previous solution by adopting a strategy that takes into consideration both spatial and temporal correlation in order to decide the maximum coding tree depth allowed for each coding tree block. Complexity control capability is increased in comparison to a previous work, while compression losses are decreased by 70%. Experimental results show that the encoder computational complexity can be downscaled to 60% with an average bit rate increase around 1.3% and a PSNR decrease under 0.07 dB.

Keywords: HEVC, video coding, computational complexity, complexity control, complexity reduction

I. INTRODUCTION

In the last few years, the High Efficiency Video Coding (HEVC) standard has been developed by the Joint Collaborative Team on Video Coding (JCT-VC). The standard is intended to be launched by early 2013 and to gradually substitute the current state-of-the-art H.264/AVC video coding standard in both academy and industry. The goal of HEVC is to double the compression rates achieved by its predecessor while still maintaining the same level of subjective image quality. To reach this goal, several new compression techniques and encoding modes have been added to the standard draft [1] and implemented in the HEVC Test Model (HM) [2], effectively increasing the computational complexity of HEVC in comparison with H.264/AVC in ranges from 9% up to 502%, depending on the encoder configuration [3, 4].

Computational complexity is a very important factor in the development of video codecs, despite all recent gains in processing speed brought by advances in computer-aided design and semiconductor device fabrication. Moreover, the evolution of networking technologies has also made possible the transmission and reception of high resolution video formats, which require a much larger amount of processing operations and thus increase power consumption. Such problems will become even more important in HEVC due to its high computational complexity, especially when the encoding has to be performed in reduced energy-budget platforms such as mobile devices.

Several works have been published in the last years aiming at controlling or reducing the computational complexity of video encoding and decoding algorithms, generally focusing on the motion estimation (ME) and mode decision (MD) processes, which represent a large share of the total processing cost of video encoding [5-8]. In [5, 6, 8] a complexity control algorithm is proposed to dynamically adjust the ME search method, either adopting a fast full search approach or a combination of different fast search techniques. In [7], complexity is managed through adjustments to parameters that jointly control both ME and MD, such as the search area in ME and the number of candidate coding modes in MD. However, these methods are not directly applicable to the HEVC standard, where high computational complexity is intrinsically related to the use of new quadtree-based coding structures.

We have proposed in [9] the first method for complexity control in HEVC which is based on the adjustment of coding tree structures according to a target computational complexity. Despite providing a fair complexity control for most video sequences without significantly decreasing compression efficiency, the method may not work as well when encoding video with fast motion scenes under small target complexities, as explained in section III. To solve this problem, we present in this paper a new algorithm based on the technique proposed in [9], which allows estimating the best maximum coding tree depth based on both spatial and temporal correlations observed in the coding tree depths.

This paper is organized as follows: section II presents a short overview on HEVC encoding structure and its intrinsic computational complexity. Section III reviews our previous work and introduces the new algorithm for complexity control. Section IV shows experimental results and section V concludes the paper.
II. QUADTREE-BASED ENCODING STRUCTURE

Even though HEVC maintains the hybrid coding architecture used in previous standards, its frame-level coding structures are significantly different. To better cope with image regions with different characteristics adaptive quadtrees have been employed instead of the macroblocks used in H.264/AVC, adding more flexibility to the encoding processes.

Each picture is divided into a number of square blocks of equal size called coding tree blocks (CTBs), which are used as the roots for each coding quadtree (or coding tree). Each leaf of the coding tree is called a coding unit (CU) and its dimensions can vary from 8x8 up to the CTB dimensions, depending on the tree depth at which it is located. The smaller the coding tree depth, the larger is the coding unit. The encoder chooses the best coding tree configuration through the use of a Rate-Distortion Optimization (RDO) process which evaluates every possible quadtree configuration and comparing all of them in terms of bit rate and image quality. Fig. 1 (a) shows an example of a 64x64 CTB divided into 13 CUs, whose dimensions vary from 32x32 to 8x8 pixels.

CUs are further subdivided into other entities used in prediction, defining the so-called prediction units (PUs), and in prediction residue transformation, the transform units (TUs). Each CU can be divided into two or four prediction units (PU), which are predicted separately. Even though PUs are not organized in a quadtree structure, the best PU division is also determined through exhaustive iterations of the RDO process taking into consideration the resulting bit rate and image quality for each PU division possibility. Fig. 1 (b) shows an example of a 32x32 CU divided into two 16x32 PUs.

During the transform coding of the prediction residual, each CU is assumed to be root of another quadtree-based structure called residual quadtree (RQT). Each leaf of the RQT is called a transform unit (TU), which sizes can vary from 4x4 up to the CU size, depending on the depth at which the TU is located. Similarly to the coding tree, the RQT structure is also defined by exhaustive RDO iterations. Fig. 1 (c) shows an example of a 32x32 CU divided into 16 TUs, whose dimensions vary from 16x16 to 4x4 pixels.

The number of coding tree possibilities grows exponentially with the maximum depth allowed. Moreover, for each possible CU in each possible coding tree configuration, all possible PUs and RQT configurations are tested in the Rate-Distortion Optimization (RDO) process, which involves the computation of bit rate and reconstructed image fidelity after performing intra-interprediction, direct and inverse transform and quantization, entropy coding and deblocking operations for each configuration. Limiting the computational burden related to the definition of the encoding structure is thus essential if one aims at optimizing HEVC encoder for operation at or below a given complexity level for use in systems with reduced energy allowances or limited computational power.

III. COMPUTATIONAL COMPLEXITY CONTROL STRATEGY

Since the encoding computational complexity grows with the number of possible CUs evaluated, which in turn increases exponentially with the maximum tree depth allowed [9], the method proposed in this paper controls complexity by limiting the maximum tree depth for each coding tree. By restricting the maximum tree depth, the video sequence is encoded ensuring the computational complexity is kept under a target value defined by the user or by the device itself based, for instance, on the remaining battery life.

In order to allow adjusting computational complexity during the encoding process according to a target complexity, two types of frames are defined in the strategy used in this work and [9]: unconstrained frames (Fu) and constrained frames (Fc), the former being encoded using the full RDO process to define all tree structures (i.e., the maximum possible coding complexity is allowed) and the latter is encoded with tree depths limited to a value lower than the maximum possible, as defined in the next sub-sections. The complexity control is performed as shown in Fig. 2 by adjusting \( N_c \), the number of Fc frames that occur between two consecutive Fu frames.

Even though the algorithm introduced in [9] is able to limit the encoder computational complexity to a given target value, \( N_c \) is updated in unitary steps, which in some cases causes slow convergence of \( N_c \) to the best value for the video segment being processed and the desired target complexity. Besides that, the algorithm in [9] is not completely adequate for fast motion sequences. In such cases, a determined image area may move away from the position where it was located in the last Fu frame before another Fu frame is encoded to update the maximum tree depth values. As a result, the Fc frames occurring between these two consecutive Fu frames...
may be encoded using maximum tree depths that do not match the current frame content, with the mismatch becoming worse towards the end of the group of $F_c$ frames. This problem is even more severe in cases where the target complexity is small and thus $N_c$ is large, causing decreases in the encoding performance.

The algorithm presented in this work copes with these issues by using a scheme which converges faster to the best $N_c$ value for determined sequence and target complexity by exploiting existing spatio-temporal correlations to determine the maximum coding tree depth allowed for each CTB.

### A. Maximum Coding Tree Depth Decision

In [9] we have proposed a method which relies on the fact that the maximum coding tree depth tends to be constant during long periods of the sequence in co-located frame areas, as verified through extensive experiments. In [10], we propose a new method which also considers the CTB motion activity by taking into account the coding tree depth of the motion compensated CTB in the previous frame.

In the present work, we also explore the tendency of neighboring coding tree blocks to be encoded with the same or similar maximum coding tree depths. Several experiments were conducted in order to analyze how frequently a certain CTB $y$ is encoded with the same or smaller maximum coding tree depth than its neighboring CTBs. The results of these experiments are presented in Table I and show that in most cases the CTBs surrounding $y$ are encoded with maximum depths that are equal to or exceed the maximum depth of $y$. The experiments were performed considering the top, left and top-left CTBs, since they are the neighboring CTBs available when encoding $y$.

As all coding tree depths from 0 to $n$ are tested through RDO when a depth $n$ is selected as maximum, no rate-distortion efficiency losses would be observed if a depth greater than the one that should be used for encoding $y$ was selected as maximum.

**TABLE I**

<table>
<thead>
<tr>
<th>Neighboring CTB</th>
<th>Greater or equal depth (%)</th>
<th>Smaller depth (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>83.47</td>
<td>16.53</td>
</tr>
<tr>
<td>Left</td>
<td>82.24</td>
<td>17.76</td>
</tr>
<tr>
<td>Top-left</td>
<td>91.84</td>
<td>8.16</td>
</tr>
</tbody>
</table>

Based on these observations and on the algorithm proposed in [9], in this new proposal the maximum coding tree depths allowed for each CTB are decided taking into consideration the type of frame ($F_c$ or $F_u$) as well as the maximum depths used in the temporal and spatial neighboring CTBs and the current $N_c$ value.

Let $CTB_{i,j}^{k}$ be a coding tree block located at position $i, j$ of frame $k$. If $k$ is an $F_u$ frame, the CTB is encoded with no complexity limitation, which means that the maximum coding tree depth possible is allowed. If $k$ is an $F_c$ frame, the maximum coding tree depth allowed is defined by taking the highest value among the maximum coding tree depths used at the:

i. Left side neighboring tree block – $CTB_{i-1,j}^{k}$
ii. Top neighboring tree block – $CTB_{i,j-1}^{k}$
iii. Top-left neighboring tree block – $CTB_{i-1,j-1}^{k}$
iv. Co-located tree block in the previous frame – $CTB_{i,j}^{k-1}$
v. Motion compensated tree block in the previous frame – $CTB_{i,j}^{k-1}$

Except for the last value among the five listed above, all are straightforward to obtain by simply storing coding tree depths used in each CTB. Two maximum tree depth matrices (MTDM) are used to store values corresponding to CTBs in the current and previous frames ($k$ and $k-1$ in display order): MTDM$^k$ and MTDM$^{k-1}$. Since $CTB_{i,j}^{k}$ is encoded and its structure is defined, its maximum depth is stored into MTDM$^{k}_{i,j}$. The process of using values stored into the matrices and updating them while encoding a frame is summarized in the algorithm presented in Fig. 3.

![Fig. 3. Pseudo-code for the maximum coding tree depth decision.](image)

In cases with very small target complexities, the maximum coding tree depth allowed is decreased by one additional unit if the maximum value for $N_c$ is already achieved but complexity is still above the target (lines 09 and 10). The $N_c$ limit is defined in this work as half the frame rate. This will be addressed in more details in sub-section III.C.

The maximum coding tree depth of item (v) listed above, the motion compensated tree block ($CTB_{(o,p)}^{k}$), is represented as CMTDM$^k_{(o,p)}$ in the algorithm (line 08) and is derived as explained in the next sub-section.

### B. Motion Compensated MTDM

As video sequences are frequently composed of regions with medium to high motion activity levels, the search for the maximum coding tree depth for a determined CTB should also consider the displacement of that image region from one frame to another. This way, as explained in sub-section III.A, the maximum depth for a determined $CTB_{(o,p)}^{k}$ shown in white in Fig. 4, also depends on the maximum depth used to encode $CTB_{(o,p)}^{k-1}$ in the previous frame, shown in black in Fig. 4,
where $o$, $p$ corresponds to the position in which the image region represented by $CTB_{i,j}$ was located in frame $k-1$.

```
\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig4.png}
\caption{Predicting the displacement of a CTB through motion compensation.}
\end{figure}
```

This compensation of the motion effect is performed using motion information from the previous frame in order to predict the most probable displacement from frame $k-1$ to frame $k$ for the image region corresponding to each CTB. The motion vector (MV) from the largest possible PU in each coding tree, which has the same dimensions as the CTB itself, is used to determine the CTB motion direction. Let us take a certain $CTB^{k-1}_{\theta,p}$ as an example and consider that the motion vector of its largest PU is $MV^{k-1}_{\theta,p} = (m, n)$, shown as a dashed line arrow in Fig. 4, and the reference frame is $r$. Let us assume that the motion speed is roughly constant in a group of frames (GOP). When this assumption is valid the CTB displacement from frame $k-2$ to frame $k-1$ can be computed by dividing $MV^{k-1}_{\theta,p}$ components $m$ and $n$ by the number of frames between frame $k-1$ and $r$. Still assuming constant motion speed, we can predict where $CTB_{\theta,p}$ may be located in frame $k$. The solid line arrow in Fig. 4 shows the estimated motion displacement from frame $k-1$ to frame $k-2$ and the pointed line arrow shows the predicted motion displacement from frame $k-1$ to frame $k$. The maximum coding tree depth for each $CTB_{i,j}$ in frame $k$ is copied from the corresponding position in MTDM$^{k-1}$ and stored into a motion compensated MTDM for frame $k$, named CMTDM$^k$, as shown in the algorithm presented in Fig. 3 (line 08).

### C. Complexity Adjustment

As previously mentioned, the computational complexity of HEVC is adjusted by increasing or decreasing $N_c$, the number of constrained frames ($F_c$). This value is updated proportionally to the difference between the target encoding complexity and the predicted complexity to encode the whole sequence.

In real time transmission systems, the video sequence size is unknown at the beginning of the encoding process. This way, a video segment of a limited size is used by the complexity control algorithm and the algorithm is restarted when a new segment starts. In the experiments presented in this work, 150 frames were used in the video segment.

As the actual maximum encoding complexity is still unknown at the beginning of the encoding process, its value is estimated according to equation (1) after encoding the first five frames as $Fu$. In (1) $EMC$ is the estimated maximum complexity, $CF_i$ is the computational complexity used to encode the $i^{th}$ frame and $N$ is the number of frames in the video segment. $EMC$ is then used in the calculation of $ETC$ as defined in (2), where $ETC$ represents the estimated target complexity to encode the whole video segment and $CT$ is the input target complexity (expressed as a percentage of the maximum complexity).

\[
EMC = \frac{1}{5} \sum_{i=1}^{5} CF_i \cdot N \tag{1}
\]

\[
ETC = CT \cdot EMC \tag{2}
\]

After computing $ETC$, the algorithm starts monitoring the computational complexity while encoding the video segment. The monitoring is performed by computing a predicted overall computational complexity for the whole segment ($PC$) according to equation (4), as explained later. While $PC$ is within the limits imposed by $ETC$ all frames are encoded as $Fu$. Whenever $PC$ increases beyond $ETC$, the complexity control is activated and frames start being encoded as $Fou$ or $Fc$ according to the algorithm decisions.

The value of $N_c$ is adjusted as defined in (3), where $prevNc$ is the $N_c$ value used in the previous group of $Fc$ frames, $\alpha$ is the adjusting parameter, and $FR$ is the video frame rate. The value of $\alpha$ is a function of $\beta = (ETC - PC) / ETC$ as shown in Fig. 5, in which the difference between $PC$ and $ETC$ is shown in the horizontal axis and the decrease or increase of $N_c$ is shown in the vertical axis.

\[
N_c = prevNc + \alpha \cdot (FR / 2) \tag{3}
\]

```
\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig5.png}
\caption{Adjusting $N_c$ according to the difference between $PC$ and $ETC$.}
\end{figure}
```

Once the maximum or minimum $N_c$ is achieved, the algorithm keeps it unchanged. As previously mentioned, we have used half the frame rate as the $N_c$ limit in this work. Higher limits can be set in order to decrease computational complexity even more, but rate-distortion performance losses will become larger in such cases.

After the $N_c$ value is calculated, a $Fou$ frame is encoded and the maximum coding tree depth used for each treeblock of that
frame is stored in the MTDM for future use, as described in sub-section III.A. The next $N_c$ constrained frames $F_c$ are then encoded with the RDO process limited to the maximum coding tree depths allowed, as explained in sub-sections III.A and III.B. $PC$ is then once again calculated and $N_c$ is adjusted, if necessary.

The $PC$ value is computed as shown in (4), where $CF_i$ is the computational complexity incurred while encoding the $i^{th}$ frame and $CFu$ and $CFc_i$ represent the encoding complexities of the last $Fu$ frame and of the $j^{th}$ frame of the last group of constrained frames. $NE$ is the number of frames already encoded, including the most recent $Fu$ and $F_c$ frames.

$$PC = \sum_{i=1}^{NE} CF_i + \frac{\sum_{j=1}^{N_c} CFc_j}{N_c + 1} \cdot (N - NE)$$

The complexity control algorithm is presented in Fig. 6. Whenever encode frame appears, the algorithm presented in Fig. 3 is used.

```plaintext
01 start a new video segment
02 for each i from 0 to 5
03 frame_type = Fu
04 encode frame i
05 if last frame go to line 01
06 calculate EMC (equation 1)
07 calculate ETC (equation 2)
08 calculate PC (equation 4)
09 if PC < ETC go to line 02
10 else
11 adjust Nc (equation 3)
12 frame_type = Fu
13 encode frame i
14 for each i from 0 to Nc
15 frame_type = Fc
16 encode frame i
17 update CMTDM
18 if last frame go to line 01
19 calculate PC (equation 4)
20 go to line 09
```

Fig. 6. Pseudo-code for the computational complexity control.

### IV. EXPERIMENTAL RESULTS

The method proposed in this paper was evaluated by measuring its complexity control accuracy and video encoding performance under specific target complexities. The algorithm was implemented in the HM encoder (version 8.2) and its computational complexity was estimated as proportional to the execution time measured with the VTune Amplifier XE 2011 software profiler from Intel. The experiments were performed using six video sequences of different resolutions composed of 150 frames, as shown in Table II. The encoder performance was evaluated under five target complexities: 60%, 70%, 80%, 90%, and 100% of the uncontrolled complexity case. Four different quantization parameters (QP) were used in the tests: 27, 32, 37, and 42. As this work focuses on computational complexity control of video coding operations carried out in devices with power constraints, the Low Delay P, Main configuration [11] was used in all tests.

#### TABLE II

<table>
<thead>
<tr>
<th>Name</th>
<th>Frame Count</th>
<th>Frame Rate (Hz)</th>
<th>Bit Depth</th>
<th>Spatial Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQMall</td>
<td>150</td>
<td>60</td>
<td>8</td>
<td>832x480</td>
</tr>
<tr>
<td>SlideShow</td>
<td>200</td>
<td>20</td>
<td>8</td>
<td>1280x720</td>
</tr>
<tr>
<td>BasketballDrive</td>
<td>150</td>
<td>50</td>
<td>8</td>
<td>1920x1080</td>
</tr>
<tr>
<td>BQTerrace</td>
<td>150</td>
<td>60</td>
<td>8</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Cactus</td>
<td>150</td>
<td>50</td>
<td>8</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Traffic</td>
<td>150</td>
<td>30</td>
<td>8</td>
<td>2560x1600</td>
</tr>
</tbody>
</table>

Fig. 7 shows a graph of the actual complexities obtained after encoding the video sequences as a function of the target complexities. The quantization parameter (QP) used for these simulations was 32, but other QP values were also tested, leading to similar results. The dashed line in the graph represents the target complexity, while the other lines represent the actual running complexity for various test video sequences. As it can be seen, the actual running complexity for each tested sequence is close to the ideal case, thus showing that the proposed method is quite accurate and is capable of controlling computational complexity to within a tight interval around the desired value.

Concerning the video encoding performance, Table III presents average results when the complexity control algorithm proposed in this paper is used and compares it with the method proposed in [9]. Besides the coding performance indicators variations, bit rate increases and average luminance PSNR decreases, the table also shows average running complexities, for all target complexities in comparison to case where no complexity control is applied, i.e., when target complexity is 100%. All tests were performed under four different QPs (27, 32, 37 and 42). An increase in the bit rate was observed in all sequences coded at low complexity points,
especially when the target complexity (CT) is set to 60%. This happens due to the fact that limiting the coding tree depth for complexity control leads to a smaller number of small size CUs than in the case of no complexity control, which results in prediction residues with larger magnitudes and less noise-like, which require more bits to be encoded. In comparison to [9], the algorithm proposed in this paper produces better RD results for all target complexities and with encoding efficiency closer to the values obtained when no complexity control is applied.

### TABLE III

AVERAGE PERFORMANCE RESULTS AND COMPARISON WITH [9]

<table>
<thead>
<tr>
<th>Target Complexity</th>
<th>Algorithm [9]</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Δ Bit rate (%)</td>
<td>ΔY-PSNR (dB)</td>
</tr>
<tr>
<td>90%</td>
<td>-0.32</td>
<td>-0.01</td>
</tr>
<tr>
<td>80%</td>
<td>+1.09</td>
<td>-0.03</td>
</tr>
<tr>
<td>70%</td>
<td>+1.89</td>
<td>-0.05</td>
</tr>
<tr>
<td>60%</td>
<td>+4.48</td>
<td>-0.08</td>
</tr>
</tbody>
</table>

### V. CONCLUSION

A complexity control method for HEVC encoders was proposed in this paper. The method relies on dynamically adjusting quadtree-based data structures depth according to computational or power resources availability. The maximum coding tree depth allowed for each coding tree block is determined during encoding time based on both spatial and temporal correlation. Computational complexity can thus be reduced to a predefined target at the cost of small losses in RD efficiency.

Experimental results show that for a complexity reduction of up to 40% the average PSNR drop observed is 0.07 dB and the average bit rate increase is around 1.4%. When compared to our previous work, the new algorithm reduces compression losses in 70%, while still reducing computational complexity at the same rates.

The method is especially useful in power-constrained portable multimedia devices to reduce energy consumption and to extend the battery life. Besides, it can also be applied to non-portable multimedia devices operating in real-time with limited computational resources.

### REFERENCES


