A novel efficient encoding engine for CABAC

Evgeny Belyaev, Karen Egiazarian and Moncef Gabbouj
Department of Signal Processing, Tampere University of Technology, Finland

1. Idea description

All details related to the proposed approach can be found in this article (see it below).


In short, in our proposal we have replaced M-coder, which is an encoding engine of CABAC, by our VSW-coder. In our case the reconstructed video for the corresponding quantization parameter (QP) is identical for the original software and the proposed software, so PSNR as well as SSIM values are also identical.

For the given video sequences and QP’s set, the proposed VSW-coder provides bitrate savings from 0.22 to 1.03% on average (see Table I) at the same quality and has a comparable computational complexity (see detailed computational complexity analysis in the article below).

<table>
<thead>
<tr>
<th>Video</th>
<th>22</th>
<th>27</th>
<th>32</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>0.50</td>
<td>0.85</td>
<td>1.50</td>
<td>2.28</td>
</tr>
<tr>
<td>BasketballPass</td>
<td>-0.10</td>
<td>0.07</td>
<td>0.38</td>
<td>0.78</td>
</tr>
<tr>
<td>BQMall</td>
<td>0.09</td>
<td>0.12</td>
<td>0.33</td>
<td>0.67</td>
</tr>
<tr>
<td>City</td>
<td>0.44</td>
<td>0.46</td>
<td>0.69</td>
<td>1.25</td>
</tr>
<tr>
<td>ElFuente</td>
<td>0.10</td>
<td>0.37</td>
<td>0.83</td>
<td>1.32</td>
</tr>
<tr>
<td>hall</td>
<td>0.45</td>
<td>0.57</td>
<td>1.08</td>
<td>1.68</td>
</tr>
<tr>
<td>Kimono1</td>
<td>0.16</td>
<td>0.20</td>
<td>0.43</td>
<td>0.90</td>
</tr>
<tr>
<td>MobileCalendar</td>
<td>0.68</td>
<td>0.70</td>
<td>0.93</td>
<td>1.51</td>
</tr>
<tr>
<td>ParkScene</td>
<td>0.21</td>
<td>0.16</td>
<td>0.43</td>
<td>1.03</td>
</tr>
<tr>
<td>PartyScene</td>
<td>-0.01</td>
<td>-0.06</td>
<td>-0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>PeopleOnStreet</td>
<td>0.21</td>
<td>0.19</td>
<td>0.22</td>
<td>0.33</td>
</tr>
<tr>
<td>RaceHorses</td>
<td>0.23</td>
<td>0.28</td>
<td>0.35</td>
<td>0.45</td>
</tr>
<tr>
<td>silent</td>
<td>-0.11</td>
<td>-0.01</td>
<td>0.50</td>
<td>1.09</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.22</strong></td>
<td><strong>0.30</strong></td>
<td><strong>0.59</strong></td>
<td><strong>1.03</strong></td>
</tr>
</tbody>
</table>

2. Archive file description

The archive file includes original (folder HM-12.0-org) and proposed software (folder HM-12.0-vsw) which we have used in our experiments as well as frame-by-frame results for each video and QP (folder results). Taking into account that in latest versions of HM software the encoding decisions is depend on encoding engine, we have modified the original and the proposed software to have the same input for both encoders by setting of m_entropyBits array to zero values in file ContextModel.cpp
An efficient adaptive binary arithmetic coder with low memory requirement

Evgeny Belyaev, IEEE Member, Andrey Turlikov, Karen Egiazarian, IEEE Senior Member and Moncef Gabbouj, IEEE Fellow

Abstract—In this paper we propose a novel efficient adaptive binary arithmetic coder which is multiplication-free and requires no look-up tables. To achieve this, we combine the probability estimation based on a virtual sliding window with the approximation of multiplication and the use of simple operations to calculate the next approximation after the encoding of each binary symbol. We show that in comparison with the M-coder the proposed algorithm provides comparable computational complexity, less memory footprint and bitrate savings from 0.5 to 2.3% on average for H.264/AVC standard and from 0.6 to 3.6% on average for HEVC standard.

Index Terms—arithmetic coding, H.264/AVC, HEVC, M-coder.

I. INTRODUCTION

Adaptive binary arithmetic coding (ABAC) is an essential component in most common image and video compression standards and several non-standardized codecs such as JPEG [1], JPEG2000 [2], H.264/AVC [3], HEVC [4] and Dirac [5]. Arithmetic coders implemented in these codecs are based on the so-called Q-coder [6] which is a multiplication-free adaptive binary arithmetic coder with a bit renormalization and look-up tables used for multiplication approximation and probability estimation.

The most efficient ABAC implementation is the M-coder [7], which is the core of the Context-adaptive binary arithmetic coding (CABAC) used in the H.264/AVC standard. The emerging HEVC standard [4], [9] also uses the M-coder as an encoding engine in CABAC, but in contrast with H.264/AVC the context-modeling in HEVC is significantly simplified. Therefore, the computation complexity and memory consumption portions of the M-coder in CABAC of HEVC are higher than those in H.264/AVC.

Since the M-coder is a key component of the entropy encoding in both compression schemes, the implementation of ABAC with a smaller memory footprint, a lower computation complexity and a higher compression efficiency remains an important challenge.

There exist several approaches to improve the compression efficiency of ABAC; however, all of them require either a multiplication operation in the interval division part, or consume additional memory. In H.264/AVC and HEVC standards, CABAC first divides the input data into several non-stationary binary sources using context modeling. Next, each binary source is compressed by an M-coder which estimates the probabilities using one state machine for binary sources with different statistical properties. However, from a compression efficiency point of view, it is better to find the trade-off between adaptation speed and precision of the probability estimation for each binary source. This task can be solved by utilizing several state machines with different adaptation speeds and precision of probability estimation [5], which unfortunately leads to an increase in memory consumption for storing different look-up tables. Another solution uses look-up table-free approach based on virtual sliding window [10] (VSW). In this approach, the probability estimation is calculated using a simple rule with one parameter – window length, and a trade-off is reached by assigning a specific window length selected according to the statistical properties of the corresponding binary source. The main disadvantage of this approach is that a multiplication operation is required. Improving of compression efficiency can be also achieved by modification of the context modeling in CABAC [11], [12], [13] but it also accompanied by significant increase of computation complexity.

Decreasing the computational complexity can be achieved by modifying the renormalization procedure. In [14], a faster renormalization method is proposed, but it is based on additional look-up tables. As an alternative to arithmetic coders, range coders use bytes as output bit stream element and do byte renormalization at a time [15], [16], [17], [18]. In [19] it was shown that adaptive binary range coder allows to decrease computation complexity. However, range coder is not efficient for compression of small length binary sequences and it also uses a multiplication in the interval division part of ABAC, so it is not suitable for hardware applications. Additional interesting direction of the computation complexity decrease can be based on development of a CABAC with a parallelism in data processing [20], [21].

Decreasing memory consumption of CABAC is also a very important issue, especially for hardware implementation. That is why a number of proposals for HEVC standard aim at minimizing memory consumption. In [22] the new CABAC architecture was proposed. This architecture reduces the memory requirement by 67%, but at the expense of 0.25–6.84% coding loss. In [23] a reduction of initialization tables for CABAC contexts lead to coding losses up to 0.4%. In [24] the LPS range look-up table size was reduced by 34% without any degradation of the compression efficiency. To reduce the
chip area for hardware implementations in [25], the authors proposed to remove look-up tables, but had to introduce multiplications.

In this paper, we present an efficient adaptive binary arithmetic coder. The main contributions of the paper are the following:

1) A new adaptive binary arithmetic coder is proposed which, in contrast with the previous coders, does not use multiplications nor any look-up tables.

2) We evaluated the proposed coder’s performance and show that it allows to easily control the trade-off between the speed of probability adaptation and the precision of probability estimation through a single parameter, the window length. For software applications we show that the proposed coder has comparable computation complexity with M-coder.

3) Two application scenarios for the proposed coder are investigated. In the first scenario, in comparison to the M-coder the proposed coder provides bitrate savings from 0.5 to 2.3% for H.264/AVC standard and 0.6–3.6% for HEVC standard. In the second scenario, a compression improvement of 0.3% in comparison to the first scenario is achieved.

The rest of the paper is organized as follows. Section II reviews the integer implementation of binary arithmetic coding. Section III is dedicated to the look-up table-free probability estimation of ones for a binary source. Section IV describes the state-of-the-art adaptive binary arithmetic encoder implementations. Section V introduces the proposed multiplication-free and look-up table-free adaptive binary arithmetic coder. Section VI evaluates the performance of the proposed coder and M-coder using three main features: coding redundancy, speed of probability adaptation and computation complexity. Two scenarios of usage of the proposed coder and comparative results with the M-coder are presented in Section VII. Conclusions are drawn in Section VIII.

II. INTEGER IMPLEMENTATION OF BINARY ARITHMETIC CODING

Let us consider a stationary discrete memoryless binary source with $p$ denoting the probability of ones. In a binary arithmetic encoding codeword for a binary sequence $x^N = \{x_1, x_2, ..., x_N\}$, $x_t \in \{0, 1\}$ is represented as $\lceil - \log_2 P(x^N) + 1 \rceil$ bits of a number

$$Q(x^N) + P(x^N)/2,$$  

where $P(x^N)$ and $Q(x^N)$ are the probability and the cumulative probability of a sequence $x^N$, respectively, which can be calculated by the recurrent relations:

If $x_t = 0$, then

$$\begin{cases} Q(x^t) \leftarrow Q(x^{t-1}) \\ P(x^t) \leftarrow P(x^{t-1})(1 - p), \end{cases}$$  

and

if $x_t = 1$, then

$$\begin{cases} Q(x^t) \leftarrow Q(x^{t-1}) + P(x^{t-1})(1 - p) \\ P(x^t) \leftarrow P(x^{t-1}). \end{cases}$$  

In this paper we use “$\leftarrow$” as the assignment operation, “$\ll$” and “$\gg$” as left and right arithmetic shift, and “$\lnot$” as bitwise NOT operation.

An integer implementation of an arithmetic encoder is based on two registers: $L$ and $R$ size of $b$ bits (see Algorithm 1). Register $L$ corresponds to $Q(x^N)$ and register $R$ corresponds to $P(x^N)$. The precision required to represent registers $L$ and $R$ grows with the increase of $N$. In order to decrease the coding latency and avoid registers underflow, the renormalization procedure [27] is used for each output symbol (see Algorithm 2).

Algorithm 1: Binary symbol $x_t$ encoding procedure

1: $T \gets R \times p$
2: $T \gets \max(1, T)$
3: $R \gets R - T$
4: if $x_t = 1$ then
5: $L \gets L + R$
6: $R \gets T$
7: end if
8: call Renormalization procedure

Algorithm 2: Renormalization procedure

1: while $R < 2^{b-2}$ do
2: if $L \geq 2^{b-1}$ then
3: $\text{bits}_\text{plus}_\text{follow}(1)$
4: $L \gets L - 2^{b-1}$
5: else if $L < 2^{b-2}$ then
6: $\text{bits}_\text{plus}_\text{follow}(0)$
7: else
8: $\text{bits}_\text{to}_\text{follow} \leftarrow \text{bits}_\text{to}_\text{follow} + 1$
9: $L \gets L - 2^{b-2}$
10: end if
11: $L \gets L < 1$
12: $R \gets R < 1$
13: end while

III. PROBABILITY ESTIMATION

In real applications the probability of ones is unknown. In this case for an input binary symbol $x_t$ the probability estimation of ones $p_t$ is calculated and used in line 1 of Algorithm 1 instead of $p$. One of the well-known probability estimation algorithms is based on a sliding window concept. The probability of a source symbol is estimated by analyzing the content of a special buffer [28]. This buffer keeps $W$ previously encoded symbols, where $W$ is the length of the buffer. After the encoding of each symbol the buffer’s content is shifted by one position, a new symbol is written to the free cell and the earliest symbol in the buffer is erased.
For the binary sources, the probability of ones is estimated by the Krichevsky-Trofimov formula [29]:

$$\hat{p}_{t+1} = s_t + 0.5 \frac{s_t}{W+1},$$

where $s_t$ is the number of ones in the window before encoding the symbol with the index $t$.

The advantage of using a sliding window is the possibility of a more accurate evaluation of the source statistics and a fast adaptation to changing statistics. However, the window has to be stored in the encoder and the decoder memory, which is a serious drawback of this algorithm. To avoid this, the Imaginary Sliding Window technique (ISW) was proposed [28]. The ISW technique does not require storing the content of the window, and instead, it estimates symbols count from the source alphabet stored in the window.

Let us consider the ISW method for a binary source. Define $x_t \in \{0, 1\}$ as a source input symbol with index $t$, $y_t \in \{0, 1\}$ as a symbol deleted from the window after adding $x_t$. Suppose at every instant a symbol in a random position is erased from the window instead of the last one. Then the number of ones in the window is recalculated by the following recurrent randomized procedure.

**Step 1.** Delete a random symbol from the window

$$s_{t+1} = s_t - y_t,$$

where $y_t$ is a random value generated with probabilities

$$P_r(y_t = 1) = \frac{s_t}{W},$$

$$P_r(y_t = 0) = 1 - \frac{s_t}{W}.$$

**Step 2.** Add a new symbol from the source

$$s_{t+1} = s_{t+1} + x_t.$$

For the implementation of the ISW algorithm, a random variable must be generated. This random variable should take the same values at the corresponding steps of the encoder and the decoder. However, there is a way to avoid generating a random variable [10]. At step 1 of the algorithm let us replace a random value $y_t$ with its probabilistic average. Then the rule for recalculating the number of ones after encoding of each symbol $x_t$ can be presented in two steps.

**Step 1.** Delete an average number of ones from the window

$$s_{t+1} = s_t - \frac{s_t}{W}.$$

**Step 2.** Add a new symbol from the source

$$s_{t+1} = s_{t+1} + x_t.$$

By combining (8) and (9), the final rule for recalculating the number of ones can be given as follows:

$$s_{t+1} = \left(1 - \frac{1}{W}\right)s_t + x_t.$$

Equation (10) corresponds to the following probability estimation rule:

$$p_{t+1} = \left(1 - \frac{1}{W}\right)p_t + \frac{1}{W}x_t.$$
Algorithm 3 : Binary symbol \( x_t \) encoding procedure

1: \( \Delta \leftarrow (R - 2^{b+2}) \gg (b - 4) \)
2: \( T \leftarrow \text{TabRangeLPS}[s][\Delta] \)
3: \( R \leftarrow R - T \)
4: if \( x_t \neq \text{MPS} \) then
5: \( L \leftarrow L + R \)
6: \( R \leftarrow T \)
7: if \( s = 0 \) then
8: \( \text{MPS} \leftarrow \text{MPS} \)
9: end if
10: \( s \leftarrow \text{TransStateLPS}[s] \)
11: else
12: \( s \leftarrow \text{TransStateMPS}[s] \)
13: end if
14: call Renormalization procedure

of (10) by \( W \):

\[
\hat{p}_1 = \frac{s'_t + Wx_t}{2^w}.
\]

where \( s'_t = Wx_t \). Let us define \( W = 2^w \), where \( w \) is an integer positive value. After integer rounding of equation (16), we obtain

\[
s_{t+1}^t = \begin{cases} s'_t + \left[ \frac{2^w - s'_t + 2^{w-1}}{2^w} \right], \text{if } x_t = 1 \\ s'_t - \left[ \frac{s'_t + 2^{w-1}}{2^w} \right], \text{if } x_t = 0, \end{cases}
\]

and the probability of ones is calculated as

\[
\hat{p}_t = \frac{s'_t}{2^w}.
\]

Thus, the adaptive binary arithmetic coding algorithm based on virtual sliding window is presented in Algorithm 4.

Algorithm 4 : Binary symbol \( x_t \) encoding procedure

1: \( T \leftarrow \left( R \times s \right) \gg \left( 2^w \right) \)
2: \( T \leftarrow \max(1, T) \)
3: \( R \leftarrow R - T \)
4: if \( x_t = 1 \) then
5: \( L \leftarrow L + R \)
6: \( R \leftarrow T \)
7: \( s \leftarrow s + \left( (2^w - s + 2^{w-1}) \gg w \right) \)
8: else
9: \( s \leftarrow s - \left( (s + 2^{w-1}) \gg w \right) \)
10: end if
11: call Renormalization procedure

In comparison with the M-coder, Algorithm 4 provides a better compression efficiency due to an assignment of a specific virtual sliding window length selected according to the statistical properties of the corresponding binary source [10] and using multiplication in interval division part of the arithmetic coder instead of its approximation. Nevertheless, using these multiplications is not efficient, especially for a hardware implementation.

V. THE PROPOSED ADAPTIVE BINARY ARITHMETIC CODER

To eliminate the multiplication operation from Algorithm 4, we use a similar reasoning as the one described in Section IV. Let us multiply both sides of (10) by \( \alpha 2^{b-1} \):

\[
s'_{t+1} = \left( 1 - \frac{1}{W} \right) \cdot s'_t + \alpha 2^{b-1} x_t,
\]

where \( s'_t = \alpha 2^{b-1} s_t \). After integer rounding of equation (19), we obtain

\[
s'_{t+1} = \begin{cases} s'_t + \left[ \frac{\alpha 2^{b-1} 2^w - s'_t + 2^{w-1}}{2^w} \right], \text{if } x_t = 1 \\ s'_t - \left[ \frac{s'_t + 2^{w-1}}{2^w} \right], \text{if } x_t = 0, \end{cases}
\]

Taking into account (14) and (15)

\[
T = R \times \hat{p}_t \approx \alpha 2^{b-1} \times \hat{p}_t = \frac{s'_t}{2^w},
\]

To improve precision of the approximation (as in M-coder) we quantize the interval \( \left[ \frac{1}{2^{b-1}}, 2^{b-1} \right] \) to four points:

\[
\left\{ \frac{9}{16}, \frac{11}{16}, \frac{13}{16}, \frac{15}{16}, \frac{17}{16} \right\}.
\]

To implement this, we first calculate a state \( s'_t \) using (20) for \( \alpha = \frac{9}{16} \). Then we approximate the multiplication in the following way:

\[
T = R \times \hat{p}_t \approx s'_t + \Delta \times \frac{1}{2^w} s'_t, \text{where } \Delta = \frac{R - 2^{b-2}}{2^{b-4}}.
\]

Taking into account that the approximation of the multiplication is correct for \( \hat{p}_t < \frac{1}{2} \) [30], we should work with \( \hat{p}_t \in [0, 0.5] \) and use the Most Probable Symbol and the Least Probable Symbol. In our case, the MPS value should be changed if

\[
\hat{p}_t = \frac{s'_t}{2^w \alpha 2^{b-1}} > 0.5,
\]

or

\[
s'_t > \alpha 2^{b-2} 2^w.
\]

Thus, taking into account (20), (23) and (25), an adaptive binary arithmetic coding is proposed in the following way (see Algorithm 5):

Since \( \Delta \in \{0, 1, 2, 3\} \), a multiplication in line 2 of Algorithm 5 can be implemented based on conditional and addition operations.

From Algorithm 5, it follows that if initial state \( s_0 \geq 2^{w-1} - 1 \), then the minimum value of the probability estimation is

\[
\hat{p}_{min} = \frac{2^{w-1} - 1}{\alpha \cdot 2^{b-1} \cdot 2^w},
\]
Algorithm 5: Binary symbol $x_i$ encoding procedure

1: $\Delta \leftarrow (R - 2^{b-2}) \gg (b-4)$
2: $T \leftarrow (s + \Delta \times (s \gg 2)) \gg w$
3: $T \leftarrow \max(1, T)$
4: $R \leftarrow R - T$
5: if $x_i \neq \text{MPS}$ then
6: $L \leftarrow L + R$
7: $R \leftarrow T$
8: $s \leftarrow s + ((\alpha 2^{b-1}w - s + 2^{w1-1}) \gg w)$
9: if $s > \alpha 2^{b-2}w$ then
10: $\text{MPS} \leftarrow \text{MPS}$;
11: $s \leftarrow \alpha 2^{b-2}w$;
12: end if
13: else
14: $s \leftarrow s - ((s + 2^{w1-1}) \gg w)$
15: end if
16: call Renormalization procedure

VI. PERFORMANCE EVALUATION OF THE PROPOSED CODER

A. Coding redundancy analysis

Let us assume that a stationary binary memoryless source with probability of ones $p$ is to be compressed. Then the probability estimation process can be represented by the Markov chain with a finite number of states. Therefore, a coding redundancy $r(p)$ caused by the probability estimation precision can be calculated as:

$$r(p) = \sum_i \pi(s_i) \cdot \left(- (1-p) \cdot \log_2(1-\hat{p}_i) - p \cdot \log_2 \hat{p}_i\right) - h(p),$$

where $\pi(s_i)$ is a stationary probability of state $s_i$, $\hat{p}_i$ is a probability estimation value for this state, and $h(p)$ is the entropy of the source.

In practice, a coding redundancy also depends on the number of bits $b$ for representation of registers $L$ and $R$ [31]. Taking this into account, we estimate the coding redundancy as

$$r(p) = \lim_{N \rightarrow \infty} \left( \frac{R}{N} \right) - h(p),$$

where $N$ is the number of input binary symbols and $R$ is the size of the compressed bit stream.

Table I shows the coding redundancy (28) related to the binary entropy for the M-coder and the proposed coder with window lengths $W = 2^4$, $2^5$ and $2^6$, respectively. For both coders we set the values $b = 10$ and $N = 10^5$.

First, one can see that the coding redundancy for the proposed algorithm depends on the window length $W$: longer window length provides less redundancy. Second, in most of the cases, in comparison with the M-coder the proposed coder exhibits a higher redundancy if $W = 2^4$ and a lower redundancy if $W = 2^6$. For $W = 2^5$ it has a higher redundancy for low probabilities and a lower redundancy for high probabilities of ones. Finally, Table II shows the minimum probability estimation value $\hat{p}_{\min}$ for both coders. One can see that the proposed coder provides significantly less $\hat{p}_{\min}$ value and; therefore, it is more efficient for compressing low entropy sources.

Table II

<table>
<thead>
<tr>
<th>$p$</th>
<th>M-coder</th>
<th>Proposed coder $W = 4$</th>
<th>Proposed coder $W = 5$</th>
<th>Proposed coder $W = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>0.029</td>
<td>0.0039</td>
<td>0.0037</td>
<td>0.0037</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>0.0287</td>
<td>0.0037</td>
<td>0.0033</td>
<td>0.0033</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>0.0281</td>
<td>0.0034</td>
<td>0.0033</td>
<td>0.0033</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>0.0239</td>
<td>0.0021</td>
<td>0.0019</td>
<td>0.0016</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>0.0102</td>
<td>0.0111</td>
<td>0.0078</td>
<td>0.0052</td>
</tr>
<tr>
<td>$0.02$</td>
<td>0.008</td>
<td>0.0156</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>$0.03$</td>
<td>0.009</td>
<td>0.016</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>$0.04$</td>
<td>0.012</td>
<td>0.017</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>$0.06$</td>
<td>0.017</td>
<td>0.014</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>$0.08$</td>
<td>0.02</td>
<td>0.014</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>$0.1$</td>
<td>0.021</td>
<td>0.014</td>
<td>0.006</td>
<td>0.006</td>
</tr>
<tr>
<td>$0.2$</td>
<td>0.021</td>
<td>0.013</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>$0.3$</td>
<td>0.022</td>
<td>0.014</td>
<td>0.007</td>
<td>0.007</td>
</tr>
<tr>
<td>$0.4$</td>
<td>0.02</td>
<td>0.013</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>$0.5$</td>
<td>0.02</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

B. Speed of probability adaptation analysis

In the case of non-stationary source compression, the coding efficiency is highly depend on the speed of the probability adaptation. For quantitative comparison for different probability estimation algorithms, we propose the following approach. First, the initial value of the probability estimation is set to $\hat{p}_0 = 0.5$. Then, the binary source with probability of ones $p < 0.5$ is generated and compressed. Finally, the speed of the probability adaptation is defined as the inverse of the average number of binary symbols $t$ which are needed for the first attainment of probability estimation value $\hat{p}_t \leq p$.

Table III shows the average number of binary symbols for the M-coder and the proposed coder with window lengths $W = 2^4$, $2^5$ and $2^6$. First, one can see that the adaptation speed for the proposed algorithm depends on the window length $W$: a shorter window length provides a higher adaptation speeds. Second, the adaptation speed of the M-coder is less than the one for the proposed coder with $W = 2^4$ and higher for $W = 2^5$ and $W = 2^6$.

C. Computation complexity analysis

From Algorithms 1, 3, 4 and 5 it follows, that the number of operations in the interval division part of an arithmetic coder is directly proportional to the number of input binary symbols.
TABLE III

<table>
<thead>
<tr>
<th>$p$</th>
<th>M-coder</th>
<th>Proposed coder $W = 4$</th>
<th>Proposed coder $W = 5$</th>
<th>Proposed coder $W = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.45</td>
<td>19</td>
<td>11</td>
<td>33</td>
<td>71</td>
</tr>
<tr>
<td>0.4</td>
<td>25</td>
<td>18</td>
<td>45</td>
<td>104</td>
</tr>
<tr>
<td>0.3</td>
<td>34</td>
<td>28</td>
<td>65</td>
<td>145</td>
</tr>
<tr>
<td>0.2</td>
<td>44</td>
<td>35</td>
<td>80</td>
<td>181</td>
</tr>
<tr>
<td>0.1</td>
<td>54</td>
<td>44</td>
<td>99</td>
<td>219</td>
</tr>
<tr>
<td>0.05</td>
<td>66</td>
<td>52</td>
<td>115</td>
<td>249</td>
</tr>
<tr>
<td>0.02</td>
<td>76</td>
<td>62</td>
<td>133</td>
<td>283</td>
</tr>
</tbody>
</table>

On the other hand, the number of times steps in lines 2–12 are used is proportional to the number of bits in the output bit stream (see Algorithm 2). Let us define $N$ as a number of the input binary symbols, $R$ as a size of the output bit stream. Then, the complexity per input symbol for the binary arithmetic coder $C$ can be written as follows [19]:

$$C = \frac{\alpha \cdot N + \beta \cdot R}{N}, \quad (29)$$

where $\alpha$ is the computation complexity of the interval division part per input binary symbol, $\beta$ is the computation complexity of the renormalization part per output binary symbol. For arithmetic coder the output bit stream size

$$R = N \cdot (h(p) + r(p)), \quad (30)$$

where $h(p)$ is an entropy of binary memoryless source with probability of ones $p$, $r(p)$ is an coding redundancy. Therefore, the complexity per input symbol is a linear function of the source entropy and coding redundancy:

$$C(p) = \alpha + \beta \cdot (h(p) + r(p)). \quad (31)$$

From (31) it follows, that for zero-entropy source ($h(p) = 0$) the complexity $C(p)$ is the lowest possible and is mainly determined by the complexity of the division part. With an increase of $h(p)$ the renormalization part is used more and more often; therefore, the complexity also increases and reaches the maximum value when $h(p) = 1$. Moreover, from the model (31) it follows, that if everything else being equal, the compression scheme which has a higher coding redundancy has a higher complexity.

Table IV shows the complexity $C(p)$ for the M-coder and the proposed coder. In this work, as in many papers related to complexity comparisons (see, for example [32], [33]), we measure the complexity in CPU cycles. For the measurements both coders were implemented as separate software programs which include the encoding of a binary string only. For each probability $p$ we have generated $N = 10^8$ input binary symbols and, then, the CPU cycles of Processor Intel Core i3M 2.1 GHz was measured by the Average CPU Cycles software\(^2\).

First, one can observe that the complexity for the proposed algorithm depends on the window length $W$, which determines different coding redundancies: a longer window length provides less redundancy and, as it is shown in the model (31), less complexity. Second, the proposed coder has up to 18% less computation complexity for very low entropy sources, because its minimum possible probability estimation $\hat{p}_{\text{min}}$ is more than 10 times smaller than in the M-coder (see Table II). Therefore it has less redundancy $r(p)$ and, following model (31), the renormalization step is not so often used as in the case of the M-coder, which explains the lower complexity. Finally, for the remaining probabilities, the complexity difference is in the range of $\pm 5\%$, which means that in these cases the complexities of both coders are comparable.

TABLE IV

<table>
<thead>
<tr>
<th>$p$</th>
<th>$0$</th>
<th>$0.05$</th>
<th>$0.1$</th>
<th>$0.2$</th>
<th>$0.3$</th>
<th>$0.4$</th>
<th>$0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed coder, $W = 4$</td>
<td>15.5</td>
<td>28.3</td>
<td>35.4</td>
<td>45.1</td>
<td>51.7</td>
<td>56.5</td>
<td>56.6</td>
</tr>
<tr>
<td>Proposed coder, $W = 5$</td>
<td>15.5</td>
<td>27.6</td>
<td>34.7</td>
<td>44.5</td>
<td>50.2</td>
<td>55.2</td>
<td>55.3</td>
</tr>
<tr>
<td>Proposed coder, $W = 6$</td>
<td>15.5</td>
<td>27.4</td>
<td>34.3</td>
<td>44.4</td>
<td>49.1</td>
<td>54.3</td>
<td>54.0</td>
</tr>
<tr>
<td>Proposed coder, average</td>
<td>15.5</td>
<td>27.8</td>
<td>34.8</td>
<td>44.3</td>
<td>50.3</td>
<td>55.3</td>
<td>55.3</td>
</tr>
<tr>
<td>M-coder</td>
<td>19.0</td>
<td>28.1</td>
<td>35.5</td>
<td>45.9</td>
<td>52.3</td>
<td>55.3</td>
<td>53.7</td>
</tr>
<tr>
<td>Average gain, $%$</td>
<td>18.8</td>
<td>11.1</td>
<td>2.1</td>
<td>3.5</td>
<td>3.8</td>
<td>6.0</td>
<td>-3.0</td>
</tr>
</tbody>
</table>

VII. APPLICATION OF THE PROPOSED CODER IN H.264/AVC AND HEVC STANDARDS

A. Scenario 1: Using fixed window length for all contexts

For practical experiments the proposed adaptive binary arithmetic coder was embedded into the JM codec v.12.1 [34] which is the reference software of the H.264/AVC video coding standard and into the HM 3.0 reference software [8] of the HEVC video coding standard. JM codec was used the Main profile, while HM codec was running with the low-delay configuration.

The initial state for each binary context was calculated as:

$$s_0 = \max \left\{2w - 1, \left[\sqrt[2w-1]{2w \hat{p}_0 + 0.5}\right]\right\} \quad (32)$$

where $\hat{p}_0$ is the initial probability predefined in the standards.

In fact, the predefined initial probability can differ significantly from the best initial probability for the current binary source from the compression efficiency point of view. Therefore, it is important to use a high speed of probability adaptation at the initial stage of the coding, especially in the case of compression of short binary sequences. Then it is better to switch to a probability estimation with a lower adaptation speed, but with a better precision of the probability estimation. This approach is adopted in JPEG2000 [2] utilizing three adaptation mechanisms, which are embedded into one state machine. It has also been used in [10] by applying Krichevsky-Trofimov formula at the initial stage of the coding. However, in both cases, an additional table or a division operation is needed.

In this paper we implement a similar idea in the following way. First, for a binary sequence compression a small window length $W = 2^w$ is used. After the compression of $n_1$ binary symbols the window length and the state $s$ are increasing two times to $W = 2^{w+1}$ and $2s$. Then, after the compression of $n_2$ binary symbols the window length and the state are increased two times again, and so on, until the maximum window length

\(^2\)\url{http://user.tninet.se/~jad615g/averagecpu/}
is achieved. In this case, as it is shown in Section VI, a high speed of probability adaptation is achieved by setting small window lengths at the initial stage of coding and, then, a high probability estimation precision is achieved by setting longer window lengths.

Bitrate savings in percentage related to the M-coder for different quantization parameters are presented in Tables V–VI. In all cases the reconstructed video for the corresponding quantization parameter (QP) is identical for the M-coder and for the proposed coder. Practical results were obtained for the first 60 frames of the test video sequences [35], [36] with different frame resolutions: 352×288 (“Foreman”, “Mobile”, “Akyio”, “Mother Daughter”), 640×480 (“Vas- sar”, “Ballroom”), 704×576 (“City”, “Crew”) and 1920×1080 (“Pedestrian Area”, “Rush Hour”, “Station”, “Tractor”). In our experiments, in case of H.264/AVC, for all contexts an initial window length is $W = 2^4$, the maximum window length is $W = 2^6$ and the values $n_1 = 24$ and $n_2 = 48$. In case of HEVC, an initial window length is $W = 2^4$, the maximum window length is $W = 2^6$ and the values $n_1 = 12$, $n_2 = 24$ and $n_3 = 48$.

The results show that for fixed visual quality the proposed adaptive binary arithmetic coder allows to decrease the bitrate by 0.5–2.3% for H.264/AVC standard and 0.6–3.6% for HEVC standard on average. Herewith, for low QP’s (0–30) the bitrate savings is caused mostly by less coding redundancy of window length $W = 2^w$, while for high QP’s (40–50) the savings results mostly from the higher speed of the probability adaptation, which is very crucial for encoding of short binary sequences. It is important to notice, that for an implementation of the proposed approach additional tables or multiplication/division operations are not needed. Therefore, in this scenario the computation complexity is comparable with the M-coder.

**TABLE V**

<table>
<thead>
<tr>
<th>QP</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>0.93</td>
<td>0.46</td>
<td>0.43</td>
<td>0.10</td>
<td>0.02</td>
<td>0.15</td>
</tr>
<tr>
<td>Mobile</td>
<td>0.04</td>
<td>0.17</td>
<td>0.30</td>
<td>0.46</td>
<td>0.20</td>
<td>0.49</td>
</tr>
<tr>
<td>Akyio</td>
<td>0.85</td>
<td>0.43</td>
<td>0.12</td>
<td>-0.17</td>
<td>0.68</td>
<td>5.04</td>
</tr>
<tr>
<td>Mother Daughter</td>
<td>1.14</td>
<td>0.81</td>
<td>0.56</td>
<td>0.06</td>
<td>0.44</td>
<td>4.81</td>
</tr>
<tr>
<td>Vassar</td>
<td>1.35</td>
<td>0.77</td>
<td>0.81</td>
<td>-0.42</td>
<td>1.72</td>
<td>8.25</td>
</tr>
<tr>
<td>Ballroom</td>
<td>1.23</td>
<td>0.66</td>
<td>0.62</td>
<td>0.50</td>
<td>0.51</td>
<td>0.84</td>
</tr>
<tr>
<td>City</td>
<td>0.79</td>
<td>0.53</td>
<td>0.58</td>
<td>0.73</td>
<td>0.80</td>
<td>1.24</td>
</tr>
<tr>
<td>Crew</td>
<td>1.01</td>
<td>0.43</td>
<td>0.73</td>
<td>0.59</td>
<td>0.79</td>
<td>1.38</td>
</tr>
<tr>
<td>Pedestrian Area</td>
<td>1.14</td>
<td>0.69</td>
<td>0.72</td>
<td>0.79</td>
<td>1.07</td>
<td>1.47</td>
</tr>
<tr>
<td>Rush Hour</td>
<td>1.19</td>
<td>0.87</td>
<td>0.88</td>
<td>0.98</td>
<td>0.97</td>
<td>1.46</td>
</tr>
<tr>
<td>Station</td>
<td>1.39</td>
<td>1.06</td>
<td>1.07</td>
<td>0.59</td>
<td>0.70</td>
<td>1.20</td>
</tr>
<tr>
<td>Tractor</td>
<td>1.02</td>
<td>0.53</td>
<td>0.61</td>
<td>0.79</td>
<td>0.84</td>
<td>0.96</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1.01</td>
<td>0.62</td>
<td>0.62</td>
<td>0.49</td>
<td>0.71</td>
<td>2.29</td>
</tr>
</tbody>
</table>

**TABLE VI**

<table>
<thead>
<tr>
<th>QP</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>0.64</td>
<td>0.35</td>
<td>0.32</td>
<td>0.18</td>
<td>0.08</td>
<td>2.76</td>
</tr>
<tr>
<td>Mobile</td>
<td>0.45</td>
<td>0.35</td>
<td>0.52</td>
<td>0.72</td>
<td>1.23</td>
<td>2.80</td>
</tr>
<tr>
<td>Akyio</td>
<td>0.75</td>
<td>0.73</td>
<td>0.63</td>
<td>0.74</td>
<td>2.07</td>
<td>6.18</td>
</tr>
<tr>
<td>Mother Daughter</td>
<td>0.74</td>
<td>1.06</td>
<td>0.89</td>
<td>0.89</td>
<td>2.21</td>
<td>5.54</td>
</tr>
<tr>
<td>Vassar</td>
<td>0.49</td>
<td>0.29</td>
<td>0.96</td>
<td>0.73</td>
<td>1.57</td>
<td>6.27</td>
</tr>
<tr>
<td>Ballroom</td>
<td>0.80</td>
<td>0.27</td>
<td>0.44</td>
<td>0.46</td>
<td>1.01</td>
<td>1.93</td>
</tr>
<tr>
<td>City</td>
<td>0.52</td>
<td>0.56</td>
<td>0.91</td>
<td>1.04</td>
<td>1.94</td>
<td>3.37</td>
</tr>
<tr>
<td>Crew</td>
<td>0.38</td>
<td>0.39</td>
<td>0.78</td>
<td>0.52</td>
<td>1.18</td>
<td>3.98</td>
</tr>
<tr>
<td>Pedestrian Area</td>
<td>0.72</td>
<td>0.75</td>
<td>0.61</td>
<td>0.66</td>
<td>0.87</td>
<td>1.54</td>
</tr>
<tr>
<td>Rush Hour</td>
<td>0.79</td>
<td>0.91</td>
<td>0.73</td>
<td>0.82</td>
<td>0.91</td>
<td>1.86</td>
</tr>
<tr>
<td>Station</td>
<td>0.78</td>
<td>1.01</td>
<td>0.87</td>
<td>0.82</td>
<td>0.95</td>
<td>1.74</td>
</tr>
<tr>
<td>Tractor</td>
<td>0.48</td>
<td>0.52</td>
<td>0.62</td>
<td>1.38</td>
<td>2.73</td>
<td>4.31</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.61</td>
<td>0.63</td>
<td>0.69</td>
<td>0.73</td>
<td>1.46</td>
<td>2.59</td>
</tr>
</tbody>
</table>

B. Scenario 2: Adaptive window length selection for each context

As it was mentioned in the introduction, from a compression efficiency point of view, it is better to find the trade-off between the adaptation speed and the precision of the probability estimation for each binary source. In the proposed adaptive binary arithmetic coder, this trade-off can be realized due to an assignment of a specific window length selected according to the statistical properties of the corresponding binary source.

In this paper we proposed to select the window length adaptively during the encoding and decoding process. Let us demonstrate this approach for the H.264/AVC standard. First $n_1 + n_2$ binary symbols are coded and decoded by the basic scheme described in Section VII-A. Then after the processing each binary symbol, the encoder and the decoder synchronously estimate the bit stream size $\hat{r}(w)$ for each window length from the set $W = 2^w \in \{2^4, 2^5, 2^6, 2^7\}$. To accomplish this, Algorithm 5 with the virtual renormalization procedure (see Algorithm 6) is applied for each window length. Then after processing of $n_3$ symbols the encoder and the decoder uses the window length

$$w^* = \arg \min_{i \in \{w\}} \hat{r}(i)$$

until the end of the binary sequence.

**Algorithm 6**: Virtual renormalization procedure

1: while $R(w) < 2^{b-2}$ do
2: if $L(w) \geq 2^{b-1}$ then
3: $\hat{r}(w) \leftarrow \hat{r}(w) + 1$
4: $L(w) \leftarrow L - 2^{b-1}$
5: else if $L(w) < 2^{b-2}$ then
6: $\hat{r}(w) \leftarrow \hat{r}(w) + 1$
7: end if
8: $\hat{r}(w) \leftarrow \hat{r}(w) + 1$
9: $L(w) \leftarrow L - 2^{b-2}$
10: end while

Bitrate savings related to the M-coder in the case of adaptive window length selection with $n_3 = 512$ are presented in Table VII. One can see that using different window lengths decreases up to 0.3% the bitrate additionally in comparison to the first scenario. On the other hand, because use of virtual renormalization, the complexity of this approach is much higher than the one in Scenario 1.
TABLE VII
BITRATE SAVINGS (IN %) COMPARED TO THE M-CODER IN CASE OF
ADAPTIVE WINDOW LENGTH SELECTION FOR EACH CONTEXT

<table>
<thead>
<tr>
<th>QP</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>1.01</td>
<td>0.56</td>
<td>0.44</td>
<td>0.10</td>
<td>0.02</td>
<td>0.15</td>
</tr>
<tr>
<td>Mobile</td>
<td>0.05</td>
<td>0.25</td>
<td>0.37</td>
<td>0.50</td>
<td>0.21</td>
<td>0.48</td>
</tr>
<tr>
<td>Akoyo</td>
<td>0.89</td>
<td>0.45</td>
<td>0.12</td>
<td>−0.17</td>
<td>0.68</td>
<td>5.04</td>
</tr>
<tr>
<td>Mother Daughter</td>
<td>1.26</td>
<td>0.92</td>
<td>0.59</td>
<td>0.06</td>
<td>0.39</td>
<td>4.81</td>
</tr>
<tr>
<td>Vassar</td>
<td>1.56</td>
<td>1.94</td>
<td>1.00</td>
<td>0.43</td>
<td>1.63</td>
<td>8.33</td>
</tr>
<tr>
<td>Ballroom</td>
<td>1.42</td>
<td>0.83</td>
<td>0.71</td>
<td>0.51</td>
<td>0.32</td>
<td>0.83</td>
</tr>
<tr>
<td>City</td>
<td>0.88</td>
<td>0.63</td>
<td>0.73</td>
<td>0.82</td>
<td>0.72</td>
<td>1.32</td>
</tr>
<tr>
<td>Crew</td>
<td>1.14</td>
<td>0.60</td>
<td>0.85</td>
<td>0.63</td>
<td>0.82</td>
<td>1.61</td>
</tr>
<tr>
<td>Pedestrian Area</td>
<td>1.34</td>
<td>0.90</td>
<td>0.88</td>
<td>0.91</td>
<td>1.22</td>
<td>1.68</td>
</tr>
<tr>
<td>Rush Hour</td>
<td>1.41</td>
<td>1.09</td>
<td>1.08</td>
<td>1.13</td>
<td>1.11</td>
<td>1.75</td>
</tr>
<tr>
<td>Station</td>
<td>1.09</td>
<td>1.30</td>
<td>1.55</td>
<td>0.70</td>
<td>0.87</td>
<td>1.92</td>
</tr>
<tr>
<td>Tractor</td>
<td>1.18</td>
<td>0.70</td>
<td>0.76</td>
<td>0.91</td>
<td>0.93</td>
<td>1.11</td>
</tr>
<tr>
<td>Average</td>
<td>1.15</td>
<td>0.76</td>
<td>0.74</td>
<td>0.75</td>
<td>0.76</td>
<td>2.38</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION
A new efficient multiplication-free and look-up table-free adaptive binary arithmetic coder was presented. In comparison to the M-coder it has the following advantages:
1) It does not use any look-up tables allowing a reduction in memory consumption or chip area in a hardware implementation.
2) It provides a simply mechanism to control the trade-off between the speed of probability adaption and the precision of probability estimation through a single parameter, the window length.
3) It provides bitrate savings from 0.5 to 2.3% for H.264/AVC standard and from 0.6 to 3.6% for HEVC standard on average and has a comparable computational complexity.
4) It does not require any changes in the context-modeling and can be easily embedded for performance improvement to any compression scheme which is based on binary arithmetic coding.

Taking into account these advantages, the proposed coder can be more preferable for a future image and video coding standards or non standardized codecs.

REFERENCES
Evgeny Belyaev (M’12) received the Engineer degree in automated systems of information processing and control and the Ph.D. (candidate of science) degree in technical sciences from State University of Aerospace Instrumentation (SUAI), Saint-Petersburg, Russia, in 2005 and 2009, respectively. He is currently a Researcher with the Institute of Signal Processing, Tampere University of Technology, Finland. His research interests include real-time video compression and transmission, video source rate control, scalable video coding, motion estimation and arithmetic encoding.

Andrey Turlikov is a professor at the Department of Information Systems and Data Protection of St. Petersburg State University of Aerospace Instrumentation, St. Petersburg, Russia. Since 1987 he has been involved in teaching activity. He is the author of more than 100 research papers and has been the invited speaker at the number of international symposiums and seminars. His research interests cover multi-user telecommunication systems, real-time data transmission protocols, reliability theory, and video compression algorithms.

Karen Egiazarian (SM’96) was born in Yerevan, Armenia, in 1959. He received the M.Sc. degree in mathematics from Yerevan State University in 1981, the Ph.D. degree in physics and mathematics from Moscow State University, Moscow, Russia, in 1986, and the D.Tech. degree from the Tampere University of Technology (TUT), Tampere, Finland, in 1994. He has been Senior Researcher with the Department of Digital Signal Processing, Institute of Information Problems and Automation, National Academy of Sciences of Armenia. Since 1996, he has been an Assistant Professor with the Institute of Signal Processing, TUT, where he is currently a Professor, leading the Computational Imaging Group. His research interests are in the areas of applied mathematics, signal processing, and digital logic.

Moncef Gabbouj (F’11) received the B.S. degree in electrical engineering from Oklahoma State University, Stillwater, in 1985, and the M.S. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1986 and 1989, respectively.

He is currently an Academy of Finland Professor with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. He was the Head of the department from 2002 to 2007. He was on sabbatical leave at the American University of Sharjah, Sharjah, United Arab Emirates, from 2007 to 2008, and a Senior Research Fellow with the Academy of Finland, Helsinki, Finland, from 1997 to 1998 and in 2008. He is the Co-Founder and Past CEO with SuviSoft Oy, Ltd., Tampere. From 1995 to 1998, he was a Professor with the Department of Information Technology, Pori School of Technology and Economics, Pori, Finland, and from 1997 to 1998 he was a Senior Research Scientist at the Academy of Finland. From 1994 to 1996, he was an Associate Professor with the Signal Processing Laboratory, Tampere University of Technology. From 1990 to 1993, he was a Senior Research Scientist with the Research Institute for Information Technology, Tampere. His current research interests include multimedia content-based analysis, indexing and retrieval, nonlinear signal and image processing and analysis, and video processing and coding.