AN EFFICIENT MULTIPLICATION-FREE AND LOOK-UP TABLE-FREE ADAPTIVE BINARY ARITHMETIC CODER

Evgeny Belyaev[1], *Andrey Turlikov*[2], *Karen Egiazarian*[1] *and Moncef Gabbouj*[1]

[1]Tampere University of Technology, Finland
Department of Signal Processing, {evgeny.belyaev, karen.egiazarian,moncef.gabbouj}@tut.fi
[2]Saint-Petersburg State University of Aerospace Instrumentation, Russia
Department of Safety in Information Systems, turlikov@vu.spb.ru

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 the proposed algorithm is faster and provides a better compression efficiency compared to the M-coder in the CABAC entropy coding scheme of the H.264/AVC video coding standard.

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

1. 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] and Dirac [4]. Arithmetic coders implemented in these codecs are based on the so called Q-coder [5] 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 [6], which is the core of the Context-adaptive binary arithmetic coding (CABAC) used in the H.264/AVC standard. However, since the M-coder is a key component in this compression scheme, the implementation of ABAC with a small memory footprint and a low computation complexity remains an important challenge. In addition, given that redundancy removal methods such as motion compensation and intra-frame prediction are reaching their efficiency limits (especially in High Efficiency Video Coding [7]), a further increase in video coding performance can be achieved by improving ABAC's compression efficiency.

There exist several approaches to improve ABAC; however, all of them require either a multiplication operation in the interval division part, or consume additional memory. In the H.264/AVC standard, 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 [4], which unfortunately leads to an increase in memory consumption for storing different look-up tables. Another solution can be based on look-up table-free approach based on virtual sliding window [8] (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.

Further decrease of computation complexity can be achieved by a modification of renormalization procedure. In [9], a faster renormalization method is proposed, but it is based on additional look-up tables. An adaptive binary range coder which does a byte renormalization at a time is proposed in [10]. However, range coder also uses a multiplication in the interval division part of ABAC.

In this paper, we present an efficient adaptive binary arithmetic coder which does not use multiplications and look-up tables. The proposed algorithm is a modification of ABAC based on VSW. It also allows to assign a specific window length depending on the statistical properties of the corresponding binary source. Therefore, in comparison to the state-of-the-art M-coder, it is faster and requires no look-up tables. Furthermore, it improves the compression efficiency.

The rest of the paper is organized as follows. Section 2 reviews the integer implementation of binary arithmetic coding. Section 3 is dedicated to the look-up table-free probability es-

This work was supported by the Academy of Finland (pr. no. 213462, Finnish Program for Centers of Excellence in Research 2006-2011).

timation of ones for a binary source. Section 4 introduces the proposed multiplication-free and look-up table-free adaptive binary arithmetic coder. The comparative results for the proposed adaptive binary arithmetic coding and the M-coder are presented in Section 5 and conclusions are drawn in Section 6.

2. 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 $\mathbf{x}^N = \{x_1, x_2, ..., x_N\}$, $x_t \in \{0, 1\}$ is represented as $\lceil -\log_2 P(\mathbf{x}^N) + 1 \rceil$ bits of a number

$$Q(\mathbf{x}^N) + P(\mathbf{x}^N)/2, \tag{1}$$

where $P(\mathbf{x}^N)$ and $Q(\mathbf{x}^N)$ are the probability and the cumulative probability of a sequence $\mathbf{x}^N$, respectively, which can be calculated by the recurrent relations[1]:
If $x_t = 0$, then

$$\begin{cases} Q(\mathbf{x}^t) \leftarrow Q(\mathbf{x}^{t-1}) \\ P(\mathbf{x}^t) \leftarrow P(\mathbf{x}^{t-1})(1-p), \end{cases} \tag{2}$$

if $x_t = 1$, then

$$\begin{cases} Q(\mathbf{x}^t) \leftarrow Q(\mathbf{x}^{t-1}) + P(\mathbf{x}^{t-1})(1-p) \\ P(\mathbf{x}^t) \leftarrow P(\mathbf{x}^{t-1})p. \end{cases} \tag{3}$$

An integer implementation of an arithmetic encoder is based on two registers: L and R (see Algorithm 1). Register L corresponds to $Q(\mathbf{x}^N)$ and register R corresponds to $P(\mathbf{x}^N)$. 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 [11] is used for each output symbol (see Algorithm 2).

Algorithm 1 Binary symbol x_t encoding procedure

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

3. 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 $\hat{p}_t$ 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 [12]. 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 a probability of ones is estimated by the Krichevsky-Trofimov formula: $\hat{p}_{t+1} = \frac{s_t + 0.5}{W+1}$, 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 [12]. 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 time 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

$$s_{t+1} \leftarrow s_t - y_t, \tag{4}$$

where y_t is a random value generated with probabilities

$$\begin{cases} Pr\{y_t = 1\} = \dfrac{s_t}{W}, \\ Pr\{y_t = 0\} = 1 - \dfrac{s_t}{W}. \end{cases} \tag{5}$$

Algorithm 2 Renormalization procedure

1: **while** $R < 2^{b-2}$ **do**
2: $\quad$ **if** $L \geq 2^{b-1}$ **then**
3: $\quad\quad bits\_plus\_follow(1)$
4: $\quad\quad L \leftarrow L - 2^{b-1}$
5: $\quad$ **else if** $L < 2^{b-2}$ **then**
6: $\quad\quad bits\_plus\_follow(0)$
7: $\quad$ **else**
8: $\quad\quad bits\_to\_follow \leftarrow bits\_to\_follow + 1$
9: $\quad\quad L \leftarrow L - 2^{b-2}$
10: $\quad$ **end if**
11: $\quad L \leftarrow L \ll 1$
12: $\quad R \leftarrow R \ll 1$
13: **end while**

[1] In our paper we use "$\leftarrow$" as the assignment operation, "$\ll$" and "$\gg$" as left and right arithmetic shift, and "!" as bitwise NOT operation

Step 2. Add a new symbol from the source

$$s_{t+1} \leftarrow s_{t+1} + x_t. \tag{6}$$

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 [8]. 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

$$s_{t+1} \leftarrow s_t - \frac{s_t}{W}. \tag{7}$$

Step 2. Add a new symbol from the source

$$s_{t+1} \leftarrow s_{t+1} + x_t. \tag{8}$$

By combining (7) and (8), the final rule for recalculating the number of ones can be given as follows:

$$s_{t+1} = \left(1 - \frac{1}{W}\right) \cdot s_t + x_t. \tag{9}$$

Based on (9), a probability estimation using virtual sliding window was proposed in [8]. In comparison with the M-coder, it 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. Nevertheless, it requires multiplications, which is not efficient, especially for hardware implementation.

4. PROPOSED ALGORITHM

To remove the multiplication, we use a similar reasoning as the one described in [13]. After the renormalization procedure in Algorithm 2, register R satisfies the following inequality:

$$\frac{1}{2}2^{b-1} \leq R < 2^{b-1}. \tag{10}$$

From (10) it follows that a multiplication can be approximated in the following way:

$$T = R \times \hat{p}_t \approx \alpha 2^{b-1} \times \hat{p}_t, \tag{11}$$

where $\alpha \in [\frac{1}{2}, ..., 1]$. Let us multiply both sides of (9) by $\alpha 2^{b-1}$:

$$s'_{t+1} = \left(1 - \frac{1}{W}\right) \cdot s'_t + \alpha 2^{b-1} x_t, \tag{12}$$

where $s'_t = \alpha 2^{b-1} s_t$. Let us define $W = 2^w$, where w is an integer positive value. Then, after integer rounding of equation (12), we obtain

$$s'_{t+1} = \begin{cases} s'_t + \left\lfloor \dfrac{\alpha 2^{b-1} 2^w - s'_t + 2^{w-1}}{2^w} \right\rfloor, \text{ if } x_t = 1 \\ s'_t - \left\lfloor \dfrac{s'_t + 2^{w-1}}{2^w} \right\rfloor, \text{ if } x_t = 0, \end{cases} \tag{13}$$

and

$$T = R \times \hat{p}_t \approx \alpha 2^{b-1} \times \hat{p}_t = \frac{s'_t}{2^w}. \tag{14}$$

To improve precision of the approximation (as in [6]) we quantize the interval $[\frac{1}{2}2^{b-1}; 2^{b-1})$ to four points:

$$\left\{ \frac{9}{16}2^{b-1}, \frac{11}{16}2^{b-1}, \frac{13}{16}2^{b-1}, \frac{15}{16}2^{b-1} \right\}. \tag{15}$$

To implement this, we first calculate a state s'_t using (13) for $\alpha = \frac{9}{16}$. Then we approximate the multiplication in the following way:

$$T = R \times \hat{p}_t \approx \frac{s'_t + \Delta \times \frac{1}{4}s'_t}{2^w}, \text{ where } \Delta = \frac{R - 2^{b-2}}{2^{b-4}}. \tag{16}$$

Taking into account that the approximation of the multiplication is correct for $\hat{p}_t < \frac{2}{3}$ [13], we should work with $\hat{p}_t \in [0, .., 0.5]$ and use the Most Probable Symbol (MPS) and the Least Probable Symbol. In our case, the MPS value should be changed if

$$\hat{p}_t = \frac{s'_t}{2^w} \frac{1}{\alpha 2^{b-1}} > 0.5 \;\; or \;\; s'_t > \alpha 2^{b-2} 2^w. \tag{17}$$

Thus, taking into account (13), (16) and (17), an adaptive binary arithmetic coding is proposed in the following way (see Algorithm 3):

Algorithm 3 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** $x_i \neq$ MPS **then**
6: $\quad L \leftarrow L+R$
7: $\quad R \leftarrow T$
8: $\quad s \leftarrow s + ((\alpha 2^{b-1} 2^w - s + 2^{w-1}) \gg w)$
9: $\quad$ **if** $s > \alpha 2^{b-2} 2^w$ **then**
10: $\quad\quad$ MPS $\leftarrow$!MPS;
11: $\quad\quad s \leftarrow \alpha 2^{b-2} 2^w$;
12: $\quad$ **end if**
13: **else**
14: $\quad s \leftarrow s - ((s + 2^{w-1}) \gg w)$
15: **end if**
16: *call* Renormalization procedure

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

5. SIMULATION RESULTS

For practical experiments the proposed adaptive binary arithmetic coder was embedded into the JM codec [14] which is the reference software of H.264/AVC video coding standard. Compression results were obtained for the test video sequence "Pedestrian Area" for the Main profile of the standard.

Bit-rate savings in percentage related to the M-coder for different quantization parameters are presented in Table 1. For both coders L and R are 10-bits registers ($b = 10$).

Table 1. Bit-rate savings (in %) related to the M-coder for "Pedestrian Area" (1920×1080, 60 frames)

QP	0	10	20	30	40	50
$W = 2^5$	0.80	0.46	0.40	0.40	0.39	0.28
$W = 2^6$	1.12	0.67	0.58	0.47	0.21	-1.04

Figure 1 shows the encoding time for 10^8 input binary symbols of binary memoryless sources with fixed ones probability using Processor Intel Core 2 DUO, 3GHz.

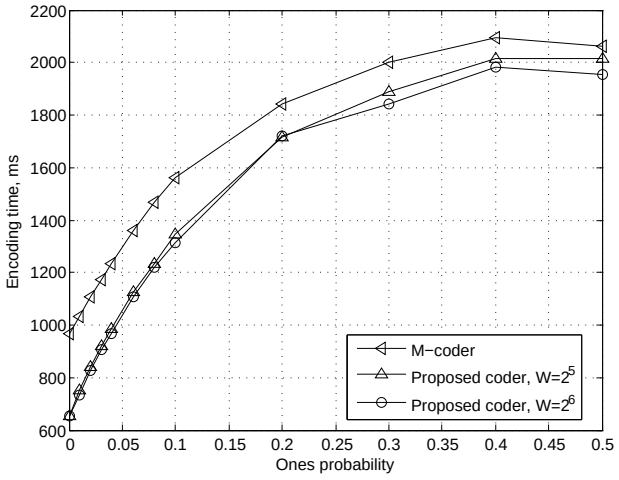

Fig. 1. Encoding time for 10^8 binary symbols

Our results show that the proposed adaptive binary arithmetic coder allows to improve compression ratio of the H.264/AVC video coding standard for fixed visual quality. At the same time it has less encoding time than the M-coder.

6. CONCLUSION

A new efficient multiplication-free adaptive binary arithmetic coder was presented. In comparison to the M-coder, it is faster, achieves better compression efficiency and does not require look-up tables. Taking into account that the proposed coder is a modification of ABAC based on VSW, for non-stationary binary sources, this coder can also achieve a trade-off between adaptation speed and precision of the probability estimation due to the utilization of different window sizes w. Therefore, it can be more preferable for image and video coding standards and non standardized codecs.

7. REFERENCES

[1] ITU-T and ISO/IEC JTC1, "Digital Compression and cod- ing of continuous-tone still images", ISO/IEC 10918-1 - ITU-T Recommendation T.81 (JPEG), 1992.

[2] ITU-T and ISO/IEC JTC 1, "JPEG 2000 Image Coding System: Core Coding System, ITU-T Recommendation T.800 and ISO/IEC 15444-1" *JPEG 2000 Part 1*, 2000.

[3] Advanced video coding for generic audiovisual services, *ITU-T Recommendation H.264 and ISO/IEC*, 2009.

[4] H. Eeckhaut, B. Schrauwen, M. Christiaens, J. Van Campenhout, "Tuning the M-coder to improve dirac's entropy coding", *WSEAS transactions on Information Science and Applications*, pp. 1563–1571, 2005.

[5] W. Pennebaker, J. Mitchel, G. Langdon, R. Arps, " An overview of the basic principles of the q-coder adaptive binary arithmetic coder", *IBM J. Research and Development*, vol.32, pp.717-726, 1988.

[6] D. Marpe, T. Wiegand, "A Highly Efficient Multiplication-Free Binary Arithmetic Coder and Its Application in Video Coding", *IEEE International Conference on Image Processing*, 2003.

[7] High Efficiency Video Coding, http://www.h265.net/

[8] E. Belyaev, M. Gilmutdinov, A. Turlikov, "Binary arithmetic coding system with adaptive probability estimation by Virtual Sliding Window", *Proceedings of the 10th IEEE International Symposium on Consumer Electronics*, pp.194–198, 2006.

[9] D. Hong, A. Eleftheriadis, "Memory-efficient semi-quasi-arithmetic coding", *IEEE International Conference on Image Processing*, 2005

[10] E. Belyaev, A. Veselov, A. Turlikov and Kai Liu, "Complexity analysis of adaptive binary arithmetic coding software implementations", *The 11th International Conference on Next Generation Wired/Wireless Advanced Networking*, 2011.

[11] A. Moffat, R. Neal, I. Witten, "Arithmetic Coding Revisited", *ACM Transactions on Information Systems*, vol. 16, pp. 256-294, 1998.

[12] B. Ryabko, "Imaginary sliding window as a tool for data compression", *Problems of Information Transmission*, pp. 156–163, 1996.

[13] D. Taubman and M. Marcellin, "JPEG2000: Image Compression, Fundamentals, Standards, and Practice", *Kluwer Academic Publishers*, 2002.

[14] H.264/AVC JM Reference Software, `http://iphome.hhi.de/suehring/tml/`