Complexity Analysis of Adaptive Binary Arithmetic Coding Software Implementations

Evgeny Belyaev[1], Anton Veselov[2], Andrey Turlikov[2], and Liu Kai[3]

[1] Tampere University of Technology, Korkeakoulunkatu 10, 33720 Tampere, Finland
evgeny.belyaev@tut.fi
[2] Saint-Petersburg State University of Aerospace Instrumentation, Bolshaya Morskaya 67, 190000 St. Petersburg, Russia
{felix,turlikov}@vu.spb.ru
[3] School of Computer Science and Technology, Xidian University, NO.2 Taibai South Road, MailBox 161, 710071 Xi'an, China
kailiu@mail.xidian.edu.cn

Abstract. This paper is dedicated to the complexity comparison of adaptive binary arithmetic coding integer software implementations. Firstly, for binary memoryless sources with known probability distribution, we prove that encoding time for arithmetic encoder is a linear function of a number of input binary symbols and source entropy. Secondly, we show that the byte-oriented renormalization allows to decrease encoding time up to 40% in comparison with bit-oriented renormalization. Finally, we study influence of probability estimation algorithm for encoding time and show that probability estimation algorithm using "Virtual Sliding Window" has less computation complexity than state machine based probability estimation algorithm from H.264/AVC standard.

Keywords: binary arithmetic coder, range coder, probability estimation, complexity analysis.

Introduction

Adaptive binary arithmetic coding is included in well known image and video compression standards and state of the art codecs like JPEG [1], JPEG2000 [2], H.264/AVC [3], Dirac [4] etc. Arithmetic coders implemented in these codecs are based on Q-coder [5] which is multiplication free adaptive binary arithmetic coder with bit renormalization and look-up tables used for probability estimation. Q-coder was introduced in 1988 and since that time the relative computational complexity of different arithmetical operations changed significantly. For example, table look-up operation takes more CPU clock cycles than a multiplication [7]. Thus, these changes should be considered for designing of a new video compression standards (especially for High Efficiency Video Coding (HEVC) [8]) or state of the art codecs.

In this paper we compare adaptive binary range coder introduced in [6] with arithmetic coder from H.264/AVC standard. At first, we show that the

S. Balandin et al. (Eds.): NEW2AN/ruSMART 2011, LNCS 6869, pp. 598–607, 2011.

byte-oriented renormalization allows to decrease encoding time up to 40% in comparison with bit-oriented renormalization. Then we investigate the influence of probability estimation algorithm for encoding time and show that using look-up table free "Virtual Sliding Window" (VSW) algorithm [14] allows to decrease the encoding time up to 10% in comparison with probability estimation algorithm from H.264/AVC standard.

Other actual topic for arithmetic encoding is complexity or power consumption modeling which is needed for power-rate-distortion analysis. Previous works [9,10] use assumption that entropy encoder complexity is approximately proportional to the output bit rate. In this paper we prove that in case of binary memoryless sources with known probability distribution encoding time for binary arithmetic encoder is a linear function of a number of input binary symbols and source entropy. If probability is not known, then encoding time is a linear function of a number of input binary symbols and output bit rate.

The rest of this paper is organized as follows. Section 1 describes the main idea of arithmetic encoding and its two integer implementations with different renormalizations. Section 2 is dedicated to integer implementations of probability estimation based on sliding window approximations. Section 3 introduces the linear model for complexity of binary arithmetic encoder and show the comparative results for described adaptive binary arithmetic coding software implementations.

1 Integer Implementations of Binary Arithmetic Encoder

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

$$\sigma(\mathbf{x}^N) = q(\mathbf{x}^N) + p(\mathbf{x}^N)/2, \tag{1}$$

where $p(\mathbf{x}^N)$ and $q(\mathbf{x}^N)$ are probability and cumulative probability of sequence $\mathbf{x}^N$ accordingly which can be calculated by using following recurrence formulas. If $x_i = 0$, then

$$\begin{cases} q(\mathbf{x}^i) \leftarrow q(\mathbf{x}^{i-1}) \\ p(\mathbf{x}^i) \leftarrow p(\mathbf{x}^{i-1}) \cdot (1-p). \end{cases} \tag{2}$$

If $x_i = 1$, then

$$\begin{cases} q(\mathbf{x}^i) \leftarrow q(\mathbf{x}^{i-1}) + p(\mathbf{x}^{i-1}) \cdot (1-p) \\ p(\mathbf{x}^i) \leftarrow p(\mathbf{x}^{i-1}) \cdot p. \end{cases} \tag{3}$$

In practice, integer implementation of arithmetic encoder is based on two v-size registers: `low` and `range` (see Algorithm 1). Register `low` corresponds to $q(\mathbf{x}^N)$, register `range` corresponds to $p(\mathbf{x}^N)$. The precision required to represent registers `low` and `range` grows with the increase of N. For decreasing coding latency and avoiding registers underflowing the *renormalization* procedure is used for each output symbol (see lines 8–26 in Algorithm 1).

As an alternative to arithmetic coders, *range coders* use bytes as output bit stream element and do byte renormalization at a time [16,17,18] (see lines 8–15 in Algorithm 2). In this paper the binary range coder with Subbotin's [19] renormalization is analyzed.

Algorithm 1. Binary symbol x_i encoding procedure in binary arithmetic coder

Input: x_i, low, range, counter

```
 1: len := range·p
 2: len := max{1,len}
 3: range := range − len
 4: if x_i =1 then
 5:    low := low+range
 6:    range := len
 7: end if
 8: while range < QUARTER do
 9:    if low ≥ HALF then
10:       PUTBIT(1)
11:       for i=1,...,counter do
12:          PUTBIT(0)
13:       end for
14:       low := low − HALF
15:    else if low < QUARTER then
16:       PUTBIT(0)
17:       for i=1,...,counter do
18:          PUTBIT(1)
19:       end for
20:    else
21:       counter := counter + 1
22:       low := low − QUARTER
23:    end if
24:    low := low/2
25:    range := range/2
26: end while
```

2 Integer Implementations of Probability Estimation

2.1 Sliding Window and Its Approximations

Algorithms of adaptive data encoding based on *sliding window* are widely known. The probability of source symbol is estimated by analysis of special buffer contents [11]. It keeps W previous encoded symbols, where W is the length of the buffer. After encoding of each symbol the buffer's content is shifted by one position, new symbol is written to the free cell and the earliest symbol in buffer is erased. This buffer is called sliding window after the method of buffer content manipulation.

Algorithm 2. Binary symbol x_i encoding procedure in binary range coder

Input: x_i, low, range

```
 1: len := range·p
 2: len := max{1,len}
 3: range := range − len
 4: if x_i =1 then
 5:    low := low+range
 6:    range := len
 7: end if
 8: while (low ⊕ (low+range))<TOP ∨ (range<BOTTOM) do
 9:    if range<BOTTOM ∧ ((low ⊕ (low+range)))≥TOP then
10:       range:= −low ∧ BOTTOM−1
11:    end if
12:    PUTBYTE(low·2^−24)
13:    range:=range·2^−8
14:    low:=low·2^−8
15: end while
```

For binary sources probability of ones is estimated by Krichevsky-Trofimov [12] formula

$$\hat{p}_{t+1} = \frac{S_t + 0.5}{W + 1}, \tag{4}$$

where S_t is the number of ones in the window before encoding symbol with the number t.

The advantage of using the sliding window is the opportunity of precise evaluation of source statistics and fast adaptation to changing statistics. However, the window has to be stored in the encoder and decoder memory, which is a serious disadvantage of this algorithm. To avoid it the *Imaginary Sliding Window* technique (ISW) proposed for a binary source [13] and for non-binary source [11]. The ISW technique does not require window content storage and estimates count of symbols from source alphabet stored in the window.

Let us consider the ISW method for a binary source. Define $x_t \in \{0, 1\}$ as source input symbol with number t, $y_t \in \{0, 1\}$ as symbol deleted from the window after addition of 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

$$S_{t+1} = S_t - y_t, \tag{5}$$

where y_t is a random value generated with probabilities

$$\begin{cases} Pr\{y_t = 1\} = \dfrac{S_t}{W}, \\ Pr\{y_t = 0\} = 1 - \dfrac{S_t}{W}. \end{cases} \tag{6}$$

Step 2. Add a new symbol from the source

$$S_{t+1} = S_{t+1} + x_t. \tag{7}$$

For implementation of ISW algorithm a random variable must be generated. This random variable should take the same values at the corresponding steps of encoder and decoder. However, there is a way to avoid generating a random variable [14]. At step 1 of the algorithm let us replace random value y_t with its probabilistic average. Then the rule for recalculating 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

$$S_{t+1} = S_t - \frac{S_t}{W}. \tag{8}$$

Step 2. Add a new symbol from the source

$$S_{t+1} = S_{t+1} + x_t. \tag{9}$$

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

$$S_{t+1} = \left(1 - \frac{1}{W}\right) \cdot S_t + x_t. \tag{10}$$

2.2 Probability Estimation Based on State Machine

One way for implementation of probability estimation can be based on the *state machine* approach. Each state of this machine corresponds to some probability value. Transition from state to state is defined by the value of the input symbol. This approach does not require multiplications or divisions for probability calculation. In addition, the fixed set of states allows to implement the multiplication-free arithmetic encoding [5].

For example, let us consider state machine based probability estimation in H.264/AVC standard [15]. Input symbols are divided into two types: Most Probable Symbols (MPS) and Least Probable Symbols (LPS). State machine contains 64 states and is based on equation (10). Each state defines probability estimation for Least Probable Symbol. Set of probability values $\{\hat{p}_0, \hat{p}_1, ..., \hat{p}_{63}\}$ is defined as:

$$\begin{cases} \hat{p}_i = (1-\gamma)\hat{p}_{i-1}, \text{ where } i = 1, ..., 63, \hat{p}_0 = 0.5, \\ \gamma = 1 - \left(\dfrac{\hat{p}_{min}}{0.5}\right)^{\frac{1}{63}}, \hat{p}_{min} = 0.01875. \end{cases} \tag{11}$$

Probability estimation for symbol x_{t+1} is calculated as

$$\hat{p}_{t+1} = \begin{cases} (1-\gamma)\hat{p}_t + \gamma, \text{ if } x_t = \text{LPS}, \\ \max\{(1-\gamma)\hat{p}_t, \hat{p}_{62}\}, \text{ if } x_t = \text{MPS}. \end{cases} \tag{12}$$

2.3 Probability Estimation Based on Virtual Sliding Window

Probability estimation using "Virtual Sliding Window" [14] is also based on equation (10), but it does not use state machine for probability calculation. For this algorithm probability estimation that x_{i+1} is equal to one is defined as

$$\hat{p}_{i+1} = \frac{s_i}{2^{2w}}, \tag{13}$$

where 2^{2w} is a window length and s_i is a virtual sliding window state which is recalculated by the following rule:

$$s_{i+1} = \begin{cases} s_i + \left\lfloor \dfrac{2^{2w} - s_t + 2^{w-1}}{2^w} \right\rfloor, \text{ if } x_i = 1 \\ s_i - \left\lfloor \dfrac{s_i + 2^{w-1}}{2^w} \right\rfloor, \text{ if } x_i = 0. \end{cases} \tag{14}$$

For stationary memoryless sources window length expansion increases the probability estimation precision and improves compression rate. For arbitrary source window length expansion may reduce estimation precision. Therefore, optimal window length selection is a complex problem because statistical properties of a binary source are unknown. In [14] the following simple heuristic algorithm of window length selection is proposed. Let us define $L = \{2^{2w_1}, 2^{2w_2}, ..., 2^{2w_l}\}$ as a set of window lengths. The output of the binary source is encoded and then window length is selected from the set L. During encoding probability estimations $\hat{p}_i(w_1), \hat{p}_i(w_2), ..., \hat{p}_i(w_l)$ are calculated. After encoding, bit stream length estimation is calculated by equation: $\hat{R}(w_k) = \sum_i \hat{r}_i(w_k)$, where

$$\hat{r}_i(w_k) = \begin{cases} -\log_2 \hat{p}_i(w_k), \text{ if } x_i = 1, \\ -\log_2 (1 - \hat{p}_i(w_k)), \text{ if } x_i = 0, \end{cases} \tag{15}$$

and window length w^* is assigned by equation

$$w^* = \arg\min_k \hat{R}(w_k). \tag{16}$$

Thus, compression gain is reached by assigning specific window length selected by statistical properties of corresponding source. Therefore, Virtual Sliding Window provides better compression efficiency [14] in comparison to adaptation mechanism in H.264/AVC standard [15].

3 Computation Complexity Analysis

From Algorithm 1 follows, that lines 1–8 are used for each input binary symbol. On the other hand, amount of using of lines 9–25 is in direct proportion to number of bits in the output bit stream. Let use define N as a number of the

input binary symbols, R as a size of the output bit stream. Therefore encoding time for binary arithmetic coder T_{arith} includes two main parts:

$$T_{arith} = \alpha_{arith} \cdot N + \beta_{arith} \cdot R, \tag{17}$$

where α_{arith} is the computation complexity of lines 1–8 per one input binary symbol, β_{arith} is the computation complexity of lines 9–25 per one output binary symbol.

Using the reasoning described above, encoding time for binary range coder (see Algorithm 2) can be written as:

$$T_{range} = \alpha_{range} \cdot N + \beta_{range} \cdot \frac{1}{8} \cdot R, \tag{18}$$

where α_{range} is the computation complexity of lines 1–8 per one input binary symbol, β_{range} is the computation complexity of lines 9–14 per one output byte.

It is known [20], that redundancy of integer implementation of arithmetic encoder depends on the number of bits for probabilities representation τ and bit size v of registers *low* and *range*. Therefore, the size of the output bit stream

$$R \approx N \cdot \left(h(p) + 2 \cdot (\tau + \log e) \cdot 2^{-(v-2)} \right), \tag{19}$$

where $h(p)$ is entropy of binary memoryless source with ones probabilities p, $v \geq \tau + 2$.

Equations (17), (18) and (19) show, that if probability of ones is known, then for given arithmetic coder implementation encoding time is the linear function of a number of input binary symbols and source entropy $h(p)$.

Values α_{arith}, β_{arith}, α_{range} and β_{range} depend on processor architecture. For simplification let us assume that $\alpha_{arith} \approx \beta_{arith} \approx \alpha_{range} \approx \beta_{range}$ and $v \gg \tau$, then from (17), (18) and (19) follows that

$$\frac{T_{arith} - T_{range}}{T_{arith}} \approx \frac{\frac{7}{8} \cdot h(p)}{1 + h(p)} \in [0, ..., 0.4375]. \tag{20}$$

Figure 1 shows the encoding time for 10^8 input binary symbols using Processor Intel Core 2 DUO, 3GHz. These results show that byte-oriented renormalization allows to decrease encoding time up to 40% in comparison with bit-oriented renormalization. In addition this figure shows that proposed linear model is fits for encoding time representation for Algorithms 1-2.

In real applications the probability of ones is not known. In this case for input binary symbol x_i the probability estimation of ones $\hat{p}_i$ is calculated and used in line 1 of Algorithms 1-2 instead p. In this case, the size of output bit stream can be calculated as

$$R \approx \sum_{i=0}^{N-1} r_i, \tag{21}$$

where

$$r_i = \begin{cases} -\log_2 \hat{p}_i, \text{ if } x_i = 1, \\ -\log_2 (1 - \hat{p}_i), \text{ if } x_i = 0, \end{cases} \tag{22}$$

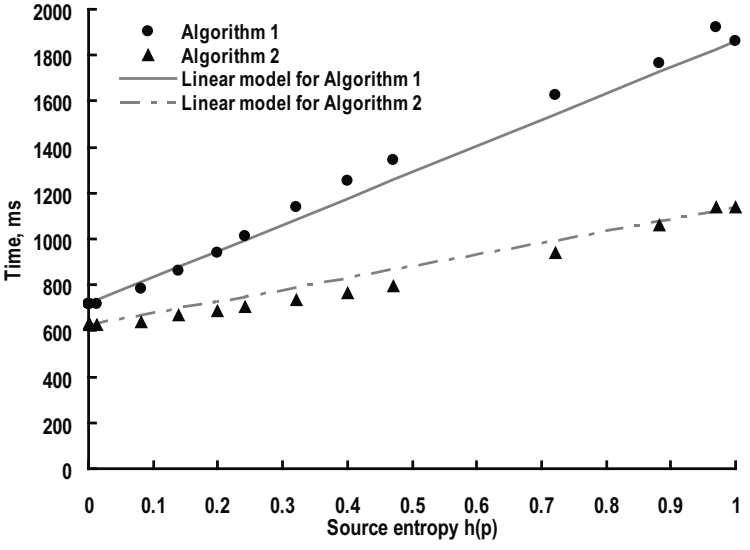

Fig. 1. Encoding time for $N = 10^8$ in case when probability is known

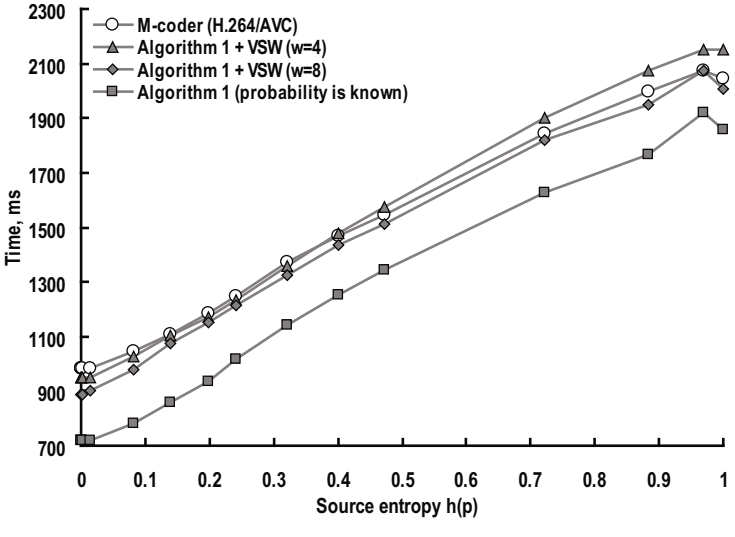

Fig. 2. Encoding time for $N = 10^8$ in case of binary arithmetic encoder with probability estimation using state machine and "Virtual Sliding Window"

Equations (17), (18) and (21) show, that the encoding time for adaptive binary arithmetic coder is the linear function of a number of input binary symbols and the output bit rate which depends on precision of the probability estimation $\hat{p}_i$.

Figure 2 shows the encoding time for Algorithm 1 in case of binary arithmetic encoder with probability estimation using state machine (as in H.264/AVC standard) and "Virtual Sliding Window" with parameters $w = 4$ and $w = 8$. This figure shows that both probability estimation algorithms require additional computation complexity. At the same time, "Virtual Sliding Window" allows to decrease the encoding time up to 10% (for $w = 8$) in comparison to probability estimation algorithm from H.264/AVC standard.

4 Conclusion

In this paper we have proved that in case of binary memoryless sources with known probability distribution encoding time for binary arithmetic encoder is a linear function of a number of input binary symbols and source entropy. We have shown that the adaptive binary arithmetic encoder implementation based on byte-oriented renormalization and probability estimation using "Virtual Sliding Window" has significantly less computational complexity than binary arithmetic encoder from H.264/AVC standard. Therefore, it is more preferable as an entropy encoding method for future video compression standards or state of the art codecs.

Acknowledgements. This work was supported by the Academy of Finland (project no. 213462, Finnish Programme for Centres of Excellence in Research 2006-2011), by the Russian Foundation for Basic Research (project 10-08-01071-a) and by the project of NSFC International Young Scientists.

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 14496-10, AVC (2009)
4. Eeckhaut, H., Schrauwen, B., Christiaens, M., Campenhout, J.: Speeding up Dirac's entropy coder. In: Proc. 5th WSEAS Int. Conf. on Multimedia, Internet and Video Technologies, pp. 120–125 (2005)
5. Pennebaker, W.B., Mitchel, J.L., Langdon, G.G., Arps, R.B.: An overview of the basic principles of the q-coder adaptive binary arithmetic coder. IBM J. Research and Development 32, 717–726 (1988)
6. Belyaev, E.: Low bit rate video coding based on three-dimensional discrete pseudo cosine transform. In: International Conference on Ultra Modern Telecommunications (2010)

7. Said, A.: Comparative analysis of arithmetic coding computational complexity. Hewlett-Packard Laboratories Report, HPL-2004-75 (2004)
8. High Efficiency Video Coding, http://www.h265.net/
9. Lu, X., Wang, Y., Erkip, E.: Power efficient H.263 video transmission over wireless channels. In: International Conference on Image Processing, pp. 533–536 (2002)
10. He, Z., Liang, Y., Chen, L., Ahmad, I., Wu, D.: Power-rate-distortion analysis for wireless video communication under energy constraints. IEEE Transactions on Circuits and Systems for Video Technology 15, 645–658 (2005)
11. Ryabko, B.: Imaginary sliding window as a tool for data compression. Problems of Information Transmission, 156–163 (1996)
12. Krichevski, E., Trofimov, V.: The performance of universal encoding. IEEE Transactions on Information Theory IT-27, 199–207 (1981)
13. Leighton, T., Rivest, R.L.: Estimating a probability using finite memory. IEEE Transactions on Information Theory IT-32, 733–742 (1986)
14. Belyaev, E., Gilmutdinov, M., Turlikov, A.: Binary arithmetic coding system with adaptive probability estimation by Virtual Sliding Window. In: Proceedings of the 10th IEEE International Symposium on Consumer Electronics, pp. 194–198 (2006)
15. Marpe, D., Schwarz, H., Wiegand, T.: Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Transactions on Circuits and Systems for Video Technology 7, 620–636 (2003)
16. Schindler, M.A.: Byte oriented arithmetic coding. In: Proceedings of Data Compression Conference (1998)
17. Vatolin, D.: Data compression methods. Dialog-MIFI Publisher, Moscow (2002) (in Russian)
18. Lindstrom, P., Isenburg, M.: Fast and Efficient Compression of Floating-Point Data. IEEE Transactions on Visualization and Computer Graphics 12(5), 1245–1250 (2006)
19. Subbotin, D.: Carryless Rangecoder (1999), http://search.cpan.org/src/SALVA/Compress-PPMd-0.10/Coder.hpp
20. Ryabko, B.Y., Fionov, A.N.: An efficient method for adaptive arithmetic coding of sources with large alphabets. Problems of Information Transmission 35(4), 95–108 (1999)