CONTEXT-BASED ADAPTIVE BINARY ARITHMETIC CODING IN JVT/H.26L

Detlev Marpe, Heiko Schwarz, Gabi Blättermann, Guido Heising, and Thomas Wiegand

Image Processing Department, Heinrich-Hertz-Institute, 10587 Berlin, Germany
[marpe,hschwarz,blaetter,heising,wiegand]@hhi.de

ABSTRACT

In this paper a new adaptive entropy coding scheme for video compression is presented. It utilizes an adaptive arithmetic coding technique to better match the first order entropy of the coded symbols and to keep track of non-stationary symbol statistics. In addition, remaining symbol redundancies will be exploited by context modeling to further reduce the bit-rate. A novel approach for coding of transform coefficients and a table look-up method for probability estimation and arithmetic coding is presented. Our new approach has been integrated in the current JVT test model (JM) to demonstrate the performance gain, and it was adopted as a part of the current JVT/H.26L draft.

1. INTRODUCTION

In our prior work [1], we demonstrated that the context-based adaptive binary arithmetic coding (CABAC) approach along with new techniques for fast adaptation is well suited for the JVT/H.26L video codec [2] and that large bit-rate reductions compared to the UVLC-based entropy coding in H.26L are achievable. Thus, this scheme was adopted for H.26L. In this paper, we present new methods of adaptive entropy coding, which significantly improve the performance both with respect to coding efficiency and computational complexity.

First, we focus on an alternative scheme for coding of transform coefficients within the CABAC entropy coding framework. This approach further improves the coding efficiency compared to the CABAC version of JVT/H.26L Working Draft 2 (WD2) [2], and its main features can be summarized as follows:

- Instead of using counts of significant, *i.e.* non-zero coefficients and run-length coding for signaling of insignificant coefficients, a one-bit coded block pattern symbol *CBP4* is transmitted for each block of transform coefficients along with a significance map
- All significant levels are encoded in reverse scanning order
- New context models are designed for all syntax elements related to transform coefficient encoding

In the second part of this paper we tackle the problem of low-complexity binary arithmetic coding within the scope of JVT/H.26L. Computationally less demanding variants of binary arithmetic coding such as the Q-Coder [3] are well known and have been successfully established in other image coding standards. However, simply applying such coding engines to JVT/H.26L may result in a significant degradation of coding efficiency. In this paper, we will present a novel low-complexity method of arithmetic coding and probability estimation, which is well suited to the CABAC entropy coding framework and which can be shown to have minor impact on coding efficiency. Our proposed method has three distinguishing properties:

- Probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 representative probability states
- The process of interval subdivision is simplified by a pre-quantization of the interval range and a subsequent table look-up operation
- A separate encoding and decoding bypass for syntax elements or parts thereof having an approximately uniform probability distribution has been established

The paper is organized as follows. In the next section, novel techniques for an improved coding of transform coefficients are presented. In Section 3, our fast binary arithmetic coding scheme is described in more details. Simulation results in Section 4 validate the efficiency of the new techniques.

2. IMPROVED CODING OF TRANSFORM COEFFICIENTS

The new coding scheme for transform coefficients is a three-step process. First, a one-bit symbol called *CBP4* is transmitted for each block of transform coefficients unless the coded block pattern (CBP) on macroblock (16x16) level indicates that the regarded block has no non-zero coefficients. The *CBP4* symbol is set to one, if there are any significant, *i.e.* non-zero coefficients inside the corresponding block. If it is zero, no further information is transmitted for the block; otherwise, in a second coding step, a significance map specifying the positions of significant coefficients is encoded. Finally, the absolute value as well as the sign is encoded for each significant transform coefficient. These values are transmitted in reverse scanning order. A more detailed description is given in the subsequent sections.

Table 1: Basic block types with number of coefficients and their related context types.

Block types	# of coeff.	Context type
Y-DC, INTRA16x16	16	**0:** Y-Intra16-DC
Y-AC, INTRA16x16	15	**1:** Y-Intra16-AC
Y-4x4, INTRA	16	**2:** Y-4x4
Y-4x4, INTER	16	
U-DC, INTRA	4	**3:** C-DC
V-DC, INTRA	4	
U-DC, INTER	4	
V-DC, INTER	4	
U-AC, INTRA	15	**4:** C-AC
V-AC, INTRA	15	
U-AC, INTER	15	
V-AC, INTER	15	

2.1. Description of the Encoding Process for Transform Coefficients

If the *CBP4* symbol indicates that a block has significant coefficients, a significance map is encoded. For each coefficient in scanning order, a one-bit symbol *SIG* is transmitted. If the *SIG* symbol is one, that is, if a non-zero coefficient exists at this scanning position, a further one-bit symbol *LAST* is sent. This symbol indicates if the current significant coefficient is the last one inside the block or if further significant coefficients follow.

Note that the significance information (*SIG, LAST*) for the last scanning position of a block is never transmitted. If the last scanning position is reached and the significance map encoding was not already terminated by a *LAST*-symbol of one, it is obvious that the last coefficient has to be significant.

The encoded significance map determines the positions of all significant coefficients inside a block of quantized transform coefficients. The values of the significant coefficients (levels) are encoded by two coding symbols: *ABS* (representing the absolute value), and *SIGN* (representing the sign). For encoding the absolute values of the coefficients, a unary binarization scheme, as already proposed in [1] is used. The levels are transmitted in reverse scanning order (beginning with the last significant coefficient of the block); this allows the usage of more reasonable contexts, as will be described in the next section.

2.2. Context Modeling

In JVT/H.26L coding, there are 12 different types of transform coefficient blocks with different statistics of transform coefficients (left column of Table 1). However, for most sequences and coding conditions some of the statistics are very similar. To reduce the dimensionality of the modeling space used for coefficient coding, the block types are classified into 5 categories: three for luminance (Y) and two for chrominance (C) data, as specified in the right column of Table 1. For each of these categories, a separate set of context models is used. For instance, encoding of the *CBP4* bit requires four different context models for each of the five categories specified in Table 1.

Coefficients	14	0	-5	3	0	0	-1	0	1							
ctx_abs_1bit	4		4	2			1		0							
ctx_abs_rbits	2		1	0												

Coefficients	18	-2	-1	6	4	-5	1	-1	0	1	0	0	1	0	0	1
ctx_abs_1bit	4	4	4	4	4	3	3	3		2			1			0
ctx_abs_rbits	4	3		2	1	0										

Figure 1: Two examples of context determination for encoding the absolute value of significant coefficients

The specific choice of the model *ctx_cbp4* for encoding the *CBP4* bit of a given block *C* is done as follows:

$$ctx\_cbp4\ (C) = CBP4\ (A) + 2 \times CBP4\ (B), \quad (1)$$

where *A* and *B* represent the corresponding blocks of the same type to the left and on the top of the regarded block *C*. Note that only blocks of the same type (left column of Table 1) are used for context determination, *i.e.* if no neighboring block *X* of the same type exists the corresponding *CBP4(X)* value in Equation (1) is replaced by a default value.

For encoding the significance map, up to 15 context models (depending on the block type category) are used for both the SIG and the LAST symbols. Here the choice of the context model is determined by the corresponding scanning position, *i.e.* for a coefficient *coeff[i]*, which was scanned at the i-th position, the related context models *ctx_sig* and *ctx_last* for the SIG and LAST symbol, respectively, are chosen as follows:

$$ctx\_sig\ (coeff[i]) = ctx\_last\ (coeff[i]) = i.$$

We use two different sets of context models for encoding of the absolute value *ABS* of a significant coefficient: one for the first bin and another one for all remaining bins of the binarized absolute value. The context model for the first bin of *ABS* is determined by the number of successive coefficients (in reverse scanning order) having an absolute value of 1 with saturation at the number of three preceding coefficients having an absolute value of 1. When a coefficient with an absolute value greater 1 is encoded, a separate context model is used for the first bin of all remaining coefficients of the regarded block, as shown for two examples in Figure 1.

All remaining bins of the absolute value are encoded using the context model, which is determined by the number of transmitted coefficients having an absolute value greater than 1 (in reverse scanning order) such that a maximum number of four preceding coefficients with absolute value greater than 1 is taken into account. Figure 1 shows two examples of the context determination for encoding the absolute values of significant coefficients.

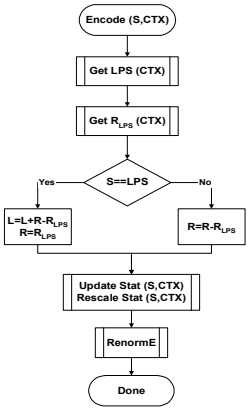

Figure 2: Flowchart of JVT/H.26L WD2 arithmetic encoder

3. FAST BINARY ARITHMETIC CODING

3.1. Review of the Conventional Binary Arithmetic Coding Engine in JVT/H.26L WD2

Figure 2 illustrates the arithmetic encoding process for a given symbol *S* in context *CTX*, as it is currently defined in JVT/H.26L WD2 [2][4]. It consists of a sequence of 5 elementary steps, where two of them (GetLPS, UpdateStat and RescaleStat) are related to probability estimation and where the remaining 3 steps are done in order to modify the internal state of the coding engine according to the given symbol and its estimated probability.

The internal state of the coding engine is characterized by two 16-bit quantities R and L: the range R and the base L (lower endpoint) of the current subinterval. Encoding of a binary symbol $S \in \{0,1\}$ is done as follows: First, in module *GetLPS(S,CTX)* a probability estimation for the given context model *CTX* is performed such that the value of the *least probable symbol* (*LPS*) and its estimated probability P_{LPS} is determined. Then, in a second step the corresponding range R_{LPS} of the *LPS* subinterval is specified ($GetR_{LPS}$). Given the *LPS* subinterval range R_{LPS}, the third step of the encoding process corresponds to the calculation of the values L and R of the new subinterval according to the given symbol S. In the fourth step, the probability estimation is updated using the encoded symbol S. Finally, in module *RenormE* the new code interval is rescaled to the range $(2^{14}, 2^{15}]$ and bits are outputted.

Regarding computational complexity, the main shortcoming of this conventional binary arithmetic coding engine is the need for two multiplicative operations. One of these multiplications is required for probability estimation based on a scaled-count estimator [5]:

$$P_{LPS} = c_{LPS} * FRAC(c_{tot}), \text{ where } FRAC(c_{tot}) = \left\lfloor \frac{2^{16}}{c_{LPS} + c_{MPS}} + 0.5 \right\rfloor, \quad (2)$$

where c_{LPS} und c_{MPS} denote the cumulative frequency counts of the *LPS* and the *most probable symbol* (*MPS*), respectively. To avoid the more costly operation of a division, tabulated 16-bit values $FRAC(c_{tot})$ corresponding to the reciprocals of the total cumulative frequency count $c_{tot} = c_{LPS} + c_{MPS}$ are used for the calculation of a 16-bit representation of the estimated *LPS* probability in Eq. (2):

$$P_{LPS} = c_{LPS} / c_{tot}.$$

A second multiplication together with an additional shift operation is required in the current WD2 binary arithmetic coder to determine the subinterval range R_{LPS} for the *LPS* symbol in module $GetR_{LPS}$ (see Fig. 3):

$$R_{LPS} = (P_{LPS} * R) >> 16. \quad (3)$$

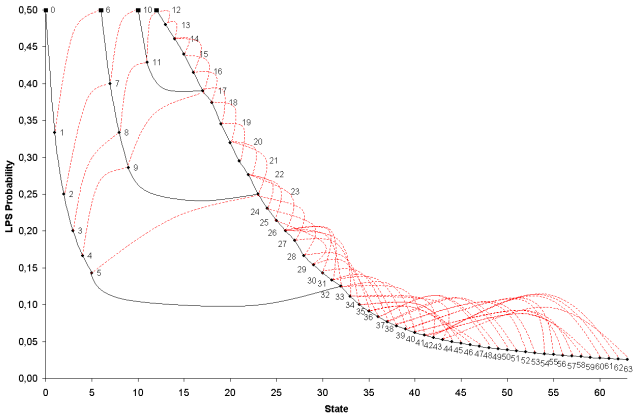

Figure 3: Probability states and their corresponding LPS probabilities and transition rules for LPS (red dashed lines, upwards) and MPS (black lines, downwards)

3.2. Table-driven Probability Estimation

To get rid of the multiplication in Eq. (2), we propose a table-driven method of probability estimation. The proposed estimator is realized by a finite-state machine (FSM) consisting of a set of representative probability states $\{P_k \mid 0 \le k < N_{max}\}$ together with some appropriately defined state transition rules. In our presented approach, we use a FSM with $N_{max} = 64$ states, where the states and transition rules have been chosen based on empirical observations.

Figure 3 illustrates the *LPS* probabilities of the individual states together with the transition rules for adapting to a *MPS* or *LPS* decision. Note however that in our proposed coding engine no explicit probability values are used. Each state is only addressed by the current value of the *MPS* and its state index, which is appropriately changed to a new state index after the encoding of a *MPS* or *LPS* symbol. In the case, where the current state corresponds to a probability value of 0.5 (state indices=0,6,10,12) and a *LPS* symbol is observed, the meaning of *MPS* and *LPS* has to be interchanged.

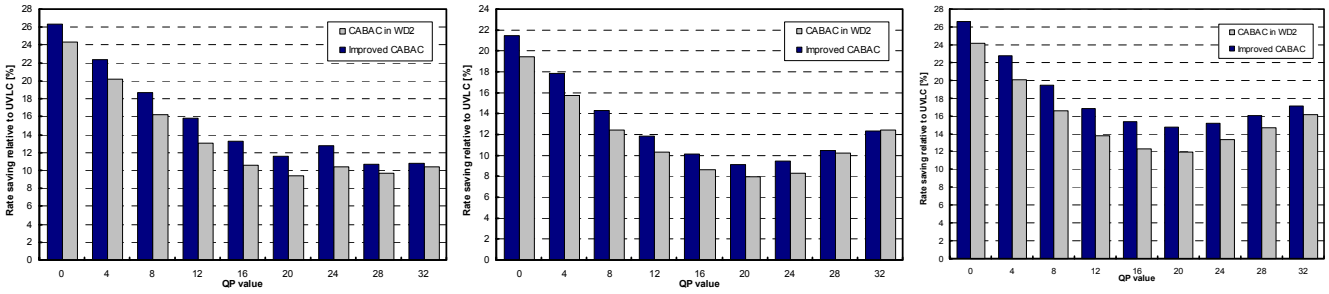

Figure 2: Average bit-rate reduction relative to UVLC (in percent) over quantization parameter (QP): (left) intra frame coding (progressive test set); (middle) inter frame coding (progressive test set); (right) overall coding performance for interlaced test set

3.2. Table-based Interval Subdivision

For substituting the multiplication in Eq. (3) that is required for interval subdivision, we propose another approximation. Given the current interval range R in a 16-bit representation, we first map R to a quantized value Q using a small set of μ quantized representations of the range of $R \in (2^{14}, 2^{15}]$. We propose to use $\mu=4$, which was found to provide a good trade-off between coding efficiency and the resulting table size. The corresponding mapping of R to Q is then realized by

$$Q=(R-0x4001)>>12.$$

Finally, given the (probability) state index *state* and the quantized range Q, the (approximate) *LPS* subinterval range R_{LPS} can be determined by

$$R_{LPS}=RTAB[state][Q], \qquad (4)$$

where the table *RTAB* has $N_{max} \times \mu = 64 \times 4$ entries containing all pre-computed product values $Q \times P_k$ in 8-bit precision.

3.3. Bypass for Equiprobable Coding Decisions

For syntax elements or binary decisions with an approximately uniform probability distribution, *i.e* $p(0)=1-p(1) \approx 0.5$, we propose to further simplify the encoding and decoding process. In this special case, which, for instance, applies to encoding of the sign information related to motion vectors or transform coefficients, probability estimation is obsolete and interval subdivision reduces to the simple operation $R \leftarrow (R>>1)$.

4. EXPERIMENTAL RESULTS

Three sets of experiments have been conducted. For this purpose, our methods have been integrated into the JVT test model, version JM 1.9, which also served as a reference system. The first set of simulations was performed using a test set of QCIF- and CIF-sequences such that the first picture of each sequence was coded as an I-picture and all successive pictures being coded as P-pictures. The graphs on the left and in the middle of Fig. 4 show the average gain for all sequences relative to UVLC coding for pure I-frame coding and coding of whole sequences, respectively. For the former case, a performance gain in the range of 1-3% has been observed in favor of the proposed transform coefficient coding scheme when compared to the WD2 CABAC version, while additional bit-rate savings of approx. 0-2% could be obtained for coding of whole sequences. In a second experiment, the performance of the new coding method was analyzed for a set of interlaced sequences in field coding mode with a regular I-field refresh every 500 milliseconds. As can be seen from Fig. 4, about 1-3% of the corresponding UVLC bit-rate can be additionally saved on average in that case. Finally, an experiment was performed to evaluate the performance degradation by using the fast arithmetic coding engine instead of the original WD2 engine. The corresponding results in Table 2 show that a negligible amount of 0.4% average loss in bit-rate has been observed.

Table 2: Average bit-rate gains in % using the QP-range = 16, 20, 24, 28 for coding of I-frames (first row) and whole sequences

Cont. QCIF	Fore. QCIF	News QCIF	Sile. QCIF	Paris CIF	Mob. CIF	Tem. CIF	**Avg.**
-0.05	-0.04	-0.22	-0.41	-0.54	-1.11	-0.41	**-0.40**
0.11	-0.41	-0.33	-0.60	-0.50	-0.49	-0.32	**-0.36**

5. CONCLUSIONS

Comparing the new context-based adaptive binary arithmetic coding method for JVT/H.26L with the UVLC entropy coding mode, average improvements of 11-26% and 9-21% bit-rate savings can be obtained for I-frame and P-frame coding, respectively. For coding of interlaced material, average bit-rate savings are in the range from 15% to 27% in comparison to UVLC. At the expense of minor increases in bit-rate, a table-based low-complexity arithmetic coding engine has been incorporated in H.26L.

6. REFERENCES

[1] Marpe, D., Blättermann, G., Heising, G., Wiegand, T., "Video Compression Using Context-Based Adaptive Arithmetic Coding", *Proc. IEEE ICIP*., pp. 558-561, 2001.

[2] Wiegand, T., "JVT Working Draft 2", JVT-B118r7, April 2002.

[3] Pennebaker, W.B., Mitchell, J.L., Langdon, G.G., Arps, R.B., "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder", IBM J. Res. Dev., Vol. 32, pp. 717-726, 1988.

[4] Moffat, A., Neal, R.M., and Witten, I.H., "Arithmetic Coding Revisited", *Proc. IEEE Data Comp. Conf.*, pp. 201-211, 1995.

[5] Duttweiler, D.L., Chamzas, Ch., "Probability Estimation in Arithmetic and Adaptive-Huffman Entropy Coders*", IEEE Trans. on Image Processing*, Vol. 4, pp. 237- 246, 1995.