Multimed Tools Appl (2016) 75:7351–7370
DOI 10.1007/s11042-015-2651-0

Arithmetic coding using hierarchical dependency context model for H.264/AVC video coding

Min Gao[1] · Qiang Wang[1] · Debin Zhao[1] · Wen Gao[1]

Received: 8 July 2014 / Revised: 20 April 2015 / Accepted: 22 April 2015
Published online: 7 May 2015

Abstract In this paper, a hierarchical dependency context model (HDCM) is firstly proposed to exploit the statistical correlations of DCT (Discrete Cosine Transform) coefficients in H.264/AVC video coding standard, in which the number of non-zero coefficients in a DCT block and the scanned position are used to capture the magnitude varying tendency of DCT coefficients. Then a new binary arithmetic coding using hierarchical dependency context model (HDCMBAC) is proposed. HDCMBAC associates HDCM with binary arithmetic coding to code the syntax elements for a DCT block, which consist of the number of non-zero coefficients, significant flag and level information. Experimental results demonstrate that HDCMBAC can achieve similar coding performance as CABAC at low and high QPs (quantization parameter). Meanwhile the context modeling and the arithmetic decoding in HDCMBAC can be carried out in parallel, since the context dependency only exists among different parts of basic syntax elements in HDCM.

Keywords Arithmetic coding · Context modeling · DCT coefficient coding · H.264/AVC

✉ Min Gao
mgao@hit.edu.cn

Qiang Wang
qwanghitcs@gmail.com

Debin Zhao
dbzhao@hit.edu.cn

Wen Gao
wgao@jdl.ac.cn

[1] Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, People's Republic of China

1 Introduction

Recent video coding standards, such as HEVC [14], H.264/AVC [21] and AVS [26], are all based on the motion-compensated hybrid coding framework. In such a framework, intra and inter predictions are first used to remove the spatial and temporal correlations, generating the prediction residual. Then the prediction residual is processed by a two dimensional Discrete Cosine Transform (DCT) followed by quantization (dividing the DCT coefficients by a quantization step size selected using QP). Finally, the quantized transform coefficients are coded using entropy coding methods. Entropy coding is used for data compression by removing the statistical correlations.

It is well known that for the best entropy coding method, the number of output bits is -log p for encoding a symbol whose probability of occurrence is p. If we can provide an accurate context model for the probability of occurrence of each symbol, arithmetic coding [10] allows us to encode a symbol that actually occurs with probability p using the number of bits approximating -log p.

One of essential parts in arithmetic coding is the context model, which is used to exploit statistical correlations behind the symbols or underlying mechanisms that drive to generate the symbols. A more accurate context model could predict more exactly, based on past observed symbols, how the symbols will occur next, and the bits for coding the upcoming symbols will be reduced. A context modeling method [11] for universal data coding was proposed by Rissanen, which is a dynamic tree-based context modeling and can theoretically approach the bound of the minimal length based on the concept of stochastic complexity [19].

In recent image/video entropy coding, context modeling has been widely used for achieving high coding efficiency. However, Rissanen's context modeling is not favored in these instances because of its unsuitability to image/video data, such as not being natural to adapt to an image's two-dimensional data feature [19, 25] and being prone to context dilution over an image's large alphabet size [19, 22]. So a priori domain knowledge is usually used to guide context modeling for image/video coding. More specifically, for image coding, texture patterns are used as heuristic information to drive context modeling such as CALIC [23] and LOCO-I [20] for JPEG-LS, and EBCOT [17] for JPEG2000. In addition, texture patterns are also used in [24] to solve the context dilution and context quantization problem in context modeling process. For video coding, DCT coefficients' statistical behaviors are usually used to drive context modeling. For example, along the zig-zag scan path of DCT blocks, non-zero coefficients show a statistical decreasing tendency in magnitude and the run-length of successive zero coefficients shows a statistical increasing tendency. This a priori domain knowledge is used to drive the context modeling in Context-Based Adaptive Binary Arithmetic Coding (CABAC) [8] for H.264/AVC and Context-based Binary Arithmetic coding(CBAC) [27] for AVS. These context models effectively exploit the statistical correlations and make image/video arithmetic coding achieve high efficiency.

With the development of the video codec to handle the demand for higher resolution and frame rates, the data throughput of entropy coding also needs to be considered in addition to high coding efficiency, since data throughput has direct influence on the processing speed of a video codec. To improve the data throughput of entropy coding in H.264/AVC, some coding engines have been proposed in [15] and [9] to process multiple binary symbols (bins) in one cycle; in addition, some thread-level parallel techniques for CABAC have also been proposed in [3] and [6] to take advantage of multi-core platforms. During the standardization of transform coefficient coding for HEVC [12], various techniques to improve the data throughput have been introduced and detailed in [16]. These techniques include reducing the number of context coded bins, grouping bypass coded bins, grouping the bins with the same context, reducing the context dependency, and reducing total bins, which are summarized as follows:

1) *Reduce the number of context coded bins*: For each context coded bin, its probability is updated based on the value of the previously coded bins that are coded with the same context, so the data throughput is limited for the context coded bins because of the data dependencies. For bypass coded bins, they do not have data dependencies, since they are coded with the equal probability of 0.5. So it is easier to process the bypass coded bins in parallel compared with context coded bins. Thus, the data throughput can be improved by using bypass coded bins instead of context coded bins to reduce the number of context coded bins.
2) *Group the bypass coded bins*: Multiple bypass coded bins can be processed in one cycle if they occur consecutively within the bit stream, thus the throughput can be improved by grouping together the bypass coded bins.
3) *Group the bins with the same context*: The speculative computation is required in context selection to process multiple context coded bins in one cycle. The number of speculative computations increases if bins with different contexts are interleaved, because lots of combinations should be considered. Thus, the throughput can be improved by reducing the speculative computations if bins with the same context are grouped together.
4) *Reduce the context dependencies*: The speculative computation is also required for decoding multiple bins in one cycle, if there are context dependencies in context selection. Thus, the throughput can be improved by reducing the number of speculative computations required to process multiple bins in parallel, if the context dependencies are removed from the context selection.
5) *Reduce total bins*: A straight approach to improve the data throughput is to reduce the number of total bins. This can be achieved by inferring the values of some bins, transmitting higher level flag and so on.

Although the above context models in [8] and [27] provide high coding efficiency, the context dependencies in these context models make it challenging to exploit the parallelism for high data throughput. As mentioned in [16], if the context selection of a bin depends on the value of another bin being decoded in parallel, the speculative computations are required to pre-calculate the context, which limits the throughput that can be achieved. Therefore, to reduce the context dependency in the context model for DCT coefficients, we proposed a variable length coding (VLC) with parallel orientation in [18] for DCT coefficients while keeping similar coding efficiency as CAVLC [2]. Later in [5], we also proposed a parallel context model for level information in CABAC.

To further reduce the context dependency in the context model for DCT coefficients, in this work, an alternative context model for DCT coefficients namely hierarchical dependency context model (HDCM) is first proposed, in which the number of non-zero coefficients in DCT block and the scanned position are used to estimate the statistics. Then, a binary arithmetic coding scheme based on HDCM (HDCMBAC) is proposed to code the DCT blocks, where the syntax elements consist of the number of non-zero coefficients, significant flag and level information. Since the context dependency in HDCM only exists among different syntax elements, the context modeling and the arithmetic decoding in HDCMBAC can be carried out in parallel. Meanwhile, experimental results demonstrate that HDCMBAC can achieve similar coding performance as CABAC at low and high QPs.

The main difference between this work and our previous works in [27], [18] and [5] is shown as follows. First, the context used for DCT coefficients in [27] is the maximum of all previously coded magnitudes of coefficients in the current DCT block. It means that the context selection for the current coefficient depends on the value of the previously coded coefficient. Obviously, there are data dependencies among the same syntax elements in this context model. In this work, the number of non-zero coefficients in DCT block and the scanned position are adopted as the contexts of DCT coefficients. This context model can break the above mentioned context dependency and allow the context modeling to be carried out in parallel with the arithmetic decoding. Second, the coding scheme in [18] is proposed for variable length coding, in which $(run, level)$ pairs are used as the syntax elements, where $level$ denotes the magnitude of non-zero coefficient and run denotes the number of successive zero coefficients before a $level$. The proposed coding scheme in this work is devised for binary arithmetic coding, in which significant flag and level information are used as the syntax elements. Finally, only level information was taken into account in the work in [5], while the significant flag is also considered in this work in addition to level information. Moreover, the number of non-zero coefficients in DCT block is introduced as a new syntax element to break the context dependency between significant flag and last significant flag.

As shown in [4], the context models are generally devised for a specific coding scheme by considering the coding engine, transform strategy, prediction tools and so on. The proposed coding scheme is now specifically devised for DCT coefficients in H.264/AVC and it cannot be directly applied for HEVC or AVS, since the coding tools in H.264/AVC are different from those in HEVC and AVS. However, the proposed coding scheme can be extended to HEVC and AVS, because DCT is also adopted in these video coding standards. To achieve this, the impact of coding tools used in HEVC and AVS, e.g., the coding structure, prediction tools, and quantization strategy, on the proposed coding coding scheme should be first significantly investigated. Then, some modifications will be required to be made on the proposed coding scheme, such as refining the context model for more accurate estimation of transform coefficients' distribution. In our future work, we will extend the proposed coding scheme to HEVC and AVS.

The rest of this paper is organized as follows. Section 2 presents the overview of the context modeling process of CABAC in H.264/AVC. Section 3 describes the hierarchical dependency context model (HDCM) for DCT coefficients based on their statistical information. Section 4 presents the implementation of HDCMBAC. In Section 5, the extensive experimental results are provided. Finally, Section 6 concludes this paper.

2 Overview of context modeling of CABAC in H.264/AVC

The syntax elements used by CABAC in H.264/AVC consist of *significant_flag*, *last_significant_flag* and level information. The one-bit symbol *significant_flag* is used to indicate whether a coefficient at each scanned position is zero. If the *significant_flag* is one, it means that a non-zero coefficient exists at this scanned position, and a further one-bit symbol *last_significant_flag* is transmitted to indicate whether the current non-zero coefficient is the last non-zero coefficient of the DCT block. For each non-zero coefficient, its magnitude (*level*) is coded by level information, which consists of *coeff_abs_level_minus1* (represent the absolute value of level minus 1) and *coeff_sign_flag* (the sign of the level).

The context models used for *significant_flag* and *last_significant_flag* depend on the scanning position, i.e., for a coefficient *coeff[P]* with the scanned position P, the context indexes are determined as follows:

$$C_{SIG}(coeff[P]) = C_{LAST}(coeff[P]) = P \tag{1}$$

where $C_{SIG}(coeff[P])$ and $C_{LAST}(coeff[P])$ are the context indexes for *significant_flag* and *last_significant_flag*, respectively.

Since *coeff_abs_level_minus1* is non-binary valued symbol, it needs to be mapped into a string of bins by UEG0 [8] binarization scheme before context modeling. For coding *coeff_abs_level_minus1*, there are two context models. One is for the first bin with bin index equal to 0, denoted as $bin0$. The other is for the bins with bin index between 1 and 13, denoted as $bin13$.

Let $NumT1(P)$ denote the accumulated number of the previously encoded levels with absolute value equal to one, and $NumLgt1(P)$ denote the accumulated number of the previously encoded levels with absolute value greater than one, where P represents the scanned position of the current transform coefficient. It is worthwhile to note that the coding of levels is under reverse scanning order, and both counters are initialized with the value of 0 at the beginning of the reverse scanning. Then the context index $C_{L^P_{bin0}}$ for $bin0$ is determined as follows:

$$C_{L^P_{bin0}} = \begin{cases} 4, & \text{if} NumLgt1(P) > 0 \\ min(3, NumT1(P)), & \text{otherwise} \end{cases} \tag{2}$$

The context index $C_{L^P_{bin13}}$ for $bin13$ is determined as follows:

$$C_{L^P_{bin13}} = 5 + min(4, NumLgt1(P)) \tag{3}$$

It can be seen that *last_significant_flag* is encoded or decoded only when *significant_flag* is equal to 1. This implies that the decision regarding coding of *last_significant_flag* is dependent on the value of *significant_flag*. If it is required to process the *significant_flag* and *last_significant_flag* for multiple coefficients in parallel, their contexts should be calculated in advance. The speculative computation tree required is shown in Fig. 1, in which 0 and 1 represent the values of *significant_flag* and *last_significant_flag*, B represents the DCT block size, P is the scanned position and EOB is the end of the DCT block. We can see that the number of speculative computations increases exponentially with the number of the *significant_flag* and *last_significant_flag* to be processed in parallel.

According to (2), one can see that the context index for $bin0$ of *coeff[P]* can be calculated only after its closest past symbol $bin0$ of *coeff[P+1]* is decoded, since $bin0$ of *coeff[P+1]* involves in the computation of $NumT1(P)$ and $NumLgt1(P)$. It implies that the context

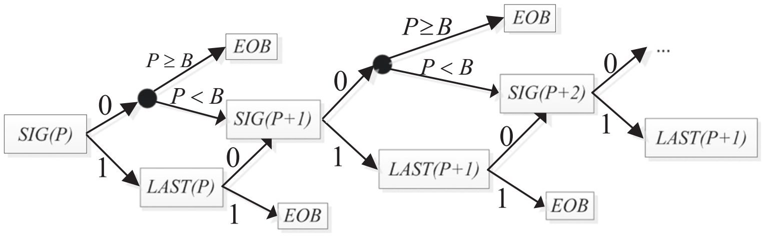


Fig. 1 Speculative computation tree required to process *significant_flag* and *last_significant_flag* for multiple coefficients in parallel in H.264/AVC

index for $bin0$ of the following coefficient is different when the $bin0$ of the current coefficient has different values. Thus the speculative computation is also required to pre-calculate the contexts for processing multiple $bin0$s in parallel. Figure 2 illustrates the speculative computation tree required, in which 0 and 1 denote the values of $bin0$ of *coeff[P_i]*, and $C_{L^{P\_i}_{bin0}}$ is the context index for $bin0$ of *coeff[P_i]*. Here it is assumed that the absolute value of *coeff[P_i]* is equal to 1, if its $bin0$ is 0; otherwise, the absolute value of *coeff[P_i]* is greater than 1. It can be seen that the number of speculative computations also increases with the number of $bin0$s to be processed in parallel.

3 Hierarchical dependency context model for DCT coefficients

In this section, *significant_flag*, $bin0$ and $bin13$ in CABAC are still used as the syntax elements to be modeled.

The context modeling for *significant_flag* is defined as follows:

$$C_{SIG}(P, N) = P + (N - 1) \times B \tag{4}$$

where P is the scanned position, N is the number of non-zero coefficients in DCT block, and B is a constant denoting the size of DCT block, e.g., $B = 16$ for 4x4 DCT block.

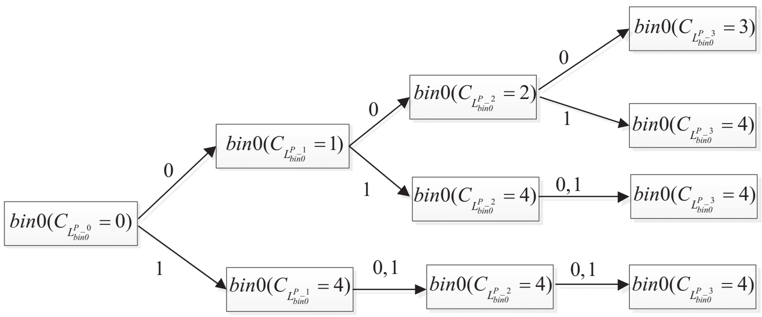
The context modeling for $bin0$ of *coeff_abs_level_minus1* at scanned position P is defined as follows:

$$C_{L^{P}_{bin0}}(P, N) = P + (N - 1) \times B \tag{5}$$

where N and B are the same as those in the above equation.

When $bin0$s of all levels are decoded, $NumLgt1$ used in the context model of $bin13$ can be calculated according to Section 2. This means that the context selection for $bin13$ only depends on $bin0$, and there are no context dependencies between $bin13$s of different levels. Thus, the original context model in CABAC is used for $bin13$, which is shown in (3).

Context modeling in (4) and (5) are inspired by the statistical behavior of DCT coefficients. The magnitude varying tendency of levels is a normal inference of the behavior that different DCT sub-bands fall into statistical distributions with different variances. So it is obviously reasonable to utilize sub-band position to model the level [7]. Sub-band position P roughly captures the behavior of levels, but not accurately enough. When the number of non-zero coefficients N in a DCT block is different, even in the same sub-band position the level may still have diverse distributions. Figure 3 shows the probability distributions of level when $N = 4$ and $N = 10$ at $P = 0$. It can be seen the distributions in a sub-band



bin0(C_{L^{P_0}_{bin0}} = 0)
0
1
bin0(C_{L^{P_1}_{bin0}} = 1)
bin0(C_{L^{P_1}_{bin0}} = 4)
bin0(C_{L^{P_2}_{bin0}} = 2)
bin0(C_{L^{P_2}_{bin0}} = 4)
bin0(C_{L^{P_3}_{bin0}} = 3)
bin0(C_{L^{P_3}_{bin0}} = 4)
0,1

Fig. 2 Speculative computation tree required to process multiple $bin0$s in parallel in H.264/AVC

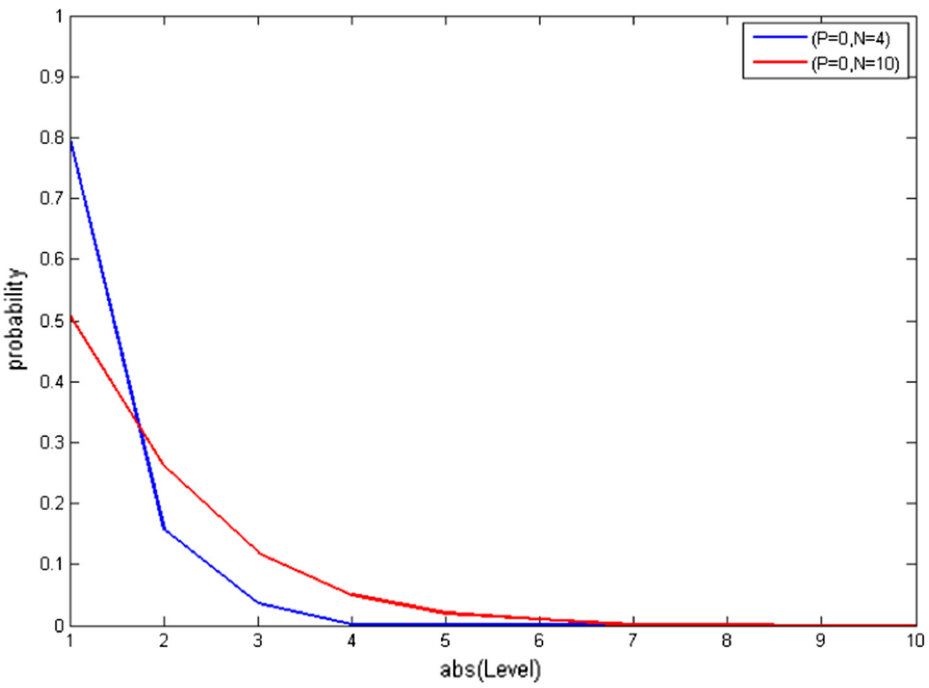


Fig. 3 Probability distributions of *Levels* magnitude when $N = 4$ and $N = 10$ in $Mobile\&Calendar\ CIF$ video at $P = 0$

position are quite diverse when N is different. Actually, N partially reflects the influence to DCT coefficients' distributions due to some important factors, such as quantization step size, local texture complexity, and prediction techniques. So P and N are used for modeling DCT coefficients' magnitude more precisely.

Figure 4 illustrates the context dependency among the syntax elements, where SIG_i, $bin0_i$ and $bin13_i$ denote the *significant_flag*, $bin0$ and $bin13$ for DCT coefficient at scanned position i, respectively. It can be seen that the context dependency in the proposed context model only exists among different syntax elements, which means *significant_flag* depends on N; $bin0$ depends on N and *significant_flag*, and $bin13$ depends on $bin0$. We call the context model as *hierarchical dependency context model* (HDCM), since its context dependency is hierarchical.

4 Binary arithmetic coding using HDCM: HDCMBAC

4.1 Implementation of HDCMBAC

To separate the coding of *significant_flag* from the coding of *last_significant_flag* in CABAC, HDCMBAC uses the number of non-zero coefficients (N) in the DCT block to indicate the position of the last non-zero coefficient instead of *last_significant_flag*. Therefore, the syntax elements in HDCMBAC consist of N, *significant_flag*, $bin0$ and $bin13$, and their coding order is $N \rightarrow$ *significant_flag* $\rightarrow bin0 \rightarrow bin13$ due to the context dependency in HDCM.

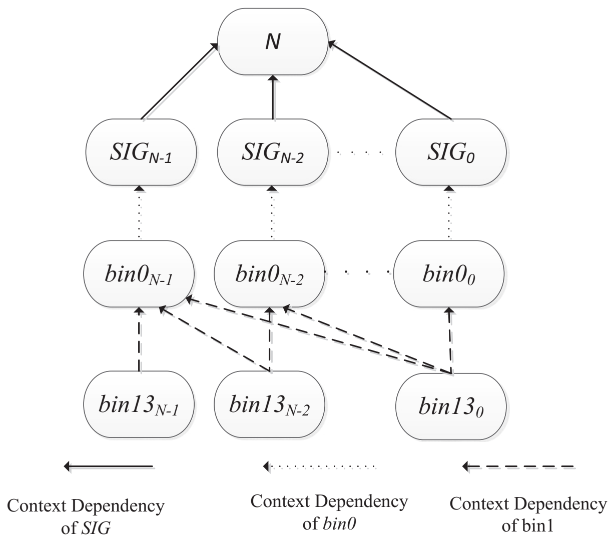


Fig. 4 Context dependency in the hierarchical dependency context model

For coding N, it should be mapped into a string of bins by unary code before context modeling, since it is non-binary symbol. For each bin of N, the context index C_{N_bin} is determined as follows:

$$C_{N\_bin}(binIdx, \hat{N}) = binIdx + (\hat{N} - 1) \times B \tag{6}$$

where $binIdx$ is the bin index, B is a constant denoting the size of DCT block, and $\hat{N}$ is defined as $\frac{N_U+N_L}{2}$, in which N_U and N_L are the number of non-zero coefficients of its two neighboring blocks on the top and on the left of the current DCT block.

According to the context modeling process for bins of N, *significant_flag* and $bin0$, the number of contexts used for each syntax element is up to B^2 for a DCT block with block size equal to B. Take 4x4 DCT block as an example, there are 256 different contexts for each syntax element. So many contexts will increase the model cost and may cause the context dilution problem. To balance the model accuracy against the model cost and avoid the context dilution problem, the contexts with similar probability distribution should be merged into one context. To achieve this, we employ the Lloyd-max algorithm [24] to classify the context set into K clusters, and then the contexts in the same cluster are merged together.

After context quantization, the context index for N is determined as follows:

$$C_{N\_bin}(binIdx, \hat{N}) = k_N[\Delta_{\hat{N}}][binIdx] \tag{7}$$

where $k_N[\Delta_{\hat{N}}][binIdx]$ is a table to store the context index, and $\Delta_{\hat{N}}$ is determined by $\hat{N}$ according to the following equation.

$$\Delta_{\hat{N}} = \begin{cases} 0, & \text{if } \hat{N} < 2 \\ 1, & \text{if } \hat{N} < 4 \\ 2, & \text{if } \hat{N} < 8 \\ 3, & \text{otherwise} \end{cases} \tag{8}$$

The determination of the context index for *significant_flag* is similar to that for N, which is described as follows:

$$C_{SIG}(P, N) = k_{SIG}[P][\Delta_N] \tag{9}$$

where $k_{SIG}[P][\Delta_N]$ is a table to store the context index, and Δ_N is determined by N according to the following equation.

$$\Delta_N = \begin{cases} 0, & \text{if } N < 3 \\ 1, & \text{if } N < 5 \\ 2, & \text{if } N < 10 \\ 3, & \text{otherwise} \end{cases} \tag{10}$$

The context index for $bin0$ is determined as follows:

$$C_{L^P_{bin0}}(P, N) = k_{bin0}[P][N] \tag{11}$$

where $k_{bin0}[P][N]$ is a table to store the mapping relationship between (P, N) and context index, which is $\{(P, N)\} \to \{0, 1, 2, 3\}$ with $0 \le P \le 15$ and $1 \le N \le 16$.

After determination of the context index, the syntax element is coded by the binary arithmetic coding engine M-coder [8]. In M-coder, the probability of a symbol to be coded is represented by (p_{LPS}, V_{MPS}), in which p_{LPS} is the probability of the least probable symbol (LPS), and V_{MPS} is the value of the most probable symbol (MPS); and the probability range associated with LPS is projected into a set of representative values $S_p = \{p_0, p_1, ..., p_{63}\}$. Each representative probability p_σ, $0 \le \sigma \le 63$, is implicitly addressed by its index σ in M-coder. Therefore, the context index should be mapped to (σ, V_{MPS}) pair, which is shown as follows:

$$C_X \longrightarrow \{(\sigma, V_{MPS}) | 0 \le \sigma \le 63, V_{MPS} \in \{0, 1\}\} \tag{12}$$

where C_X represents the context index for bins of N, *significant_flag* or $bin0$. In the implementation, the initial values of (σ, V_{MPS}) pairs for each context index of N and $bin0$ are determined over typical videos under typical bit-rates, while the initial values of (σ, V_{MPS}) pairs for each context index of *significant_flag* are the same as those in CABAC.

The complete description for encoding a DCT block using HDCMBAC is illustrated in Fig. 5. Table 1 provides an example of the context derivation in HDCMBAC. In this example, it is assumed that the coefficients at scanning positions between 5 and 15 are all zero.

To better understand the proposed algorithm and describe the analysis to the proposed algorithm in the rest of paper, the summary of some important notations is provided in Table 2.

4.2 Potential parallelism between context modeling and arithmetic decoding in HDCMBAC

The context dependency in HDCM can be viewed as data dependency, since the current instruction e.g. context modeling, refers to the data of a preceding instruction e.g. arithmetic coding. However, the data dependency only exists among different syntax elements, thus the closest past symbol will not involve in the context modeling of the symbol to be coded. For example, different from CABAC, the value of the $bin0$ for *coeff[P+1]* is not used in the context modeling of $bin0$ for *coeff[P]*, this is because the context indexes for all $bin0$s can be obtained when N and *significant_flag* are decoded. Therefore, the context modeling and arithmetic decoding in HDCMBAC could be carried out in parallel by appropriate organization. Here, we present a parallel organization for context modeling and arithmetic decoding.

```
Algorithm: HDCMBAC
Inputs:    N, significant_flag and coeff_abs_level_minus1

For each bin in bin string of N, where 1 <= N <= 16
    Calculate $C_{N\_bin}$ according to (7);
    Encode bin with $C_{N\_bin}$.
End for

Coded_Sig_Num = 0;
For i = 0; i < B; i++
    Calculate $C_{SIG}$ according to (9);
    Encode significant_flag[i] with $C_{SIG}$.
    If significant_flag[i] == 1
        Coded_Sig_Num = Coded_Sig_Num + 1;
    End if
    If Coded_Sig_Num == N
        break;
    End if
End for

For i = B - 1; i >= 0; i--
    If significant_flag[i] == 1
        Calculate $C_{bin0}$ according to (11);
        Encode bin0 with $C_{bin0}$;
    End if
End for

For i = B - 1; i >= 0; i--
    If coeff_abs_level_minus1[i] > 1
        Calculate $C_{bin13}$ according to (3);
        Encode bin13 with $C_{bin13}$;
    End if
End for
```

Fig. 5 HDCMBAC encoding scheme for a DCT block

Since the bin index is used as the context for bins of N, the value of the current bin is used to determine whether the decoding of N is finished. For the parallel processing between context modeling and arithmetic decoding, the context index for each bin of N is always input to the arithmetic decoder before it is finished. When the value of the current bin is decoded, we can reset the state of the arithmetic decoder if the decoding of N is finished and start to calculate the context index for *significant_flag* at the same time.

Table 1 Example for determination of context index in HDCMBAC

Scanning position P	0	1	2	3	4
DCT coefficient	9	0	3	−1	1
Significant flag	1	0	1	1	1
C_{SIG}	$k_{SIG}[0][1]$	$k_{SIG}[1][1]$	$k_{SIG}[2][1]$	$k_{SIG}[3][1]$	$k_{SIG}[4][1]$
C_{bin0}	$k_{bin0}[0][4]$		$k_{bin0}[2][4]$	$k_{bin0}[3][4]$	$k_{bin0}[4][4]$
C_{bin13}	6		5		

Table 2 Important notations in this paper

Notation	Description
SIG	significant_flag
$LAST$	*last_significant_flag*
$bin0$	the first bin with bin index equal to 0 of *coeff_abs_level_minus1*
$bin13$	bins with bin index between 1 and 13 of *coeff_abs_level_minus1*
N	the number of the non-zero coefficients in DCT block
P	the scanned position of a DCT coefficient
B	DCT block size
σ	the index of each representative probability for LPS (least probable symbol)
V_{MPS}	the value of MPS (most probable symbol)

From the encoding scheme of HDCMBAC, the value of *significant_flag* is used to determine whether the decoding of level information starts. For the parallel processing between context modeling and arithmetic decoding, the context index of *significant_flag* is always input to the arithmetic decoder before decoding level information. When the value of the current *significant_flag* is decoded, we can reset the arithmetic decoder to the starting state if the upcoming syntax element is level information, and start to calculate the context index for level information.

Here we provide an example to better illustrate the parallelism between context modeling and arithmetic decoding in HDCMBAC. It is assumed that there are two significant coefficients in a DCT block, and both of them are greater than one. For decoding such a DCT block, the parallel organization of context modeling and arithmetic decoding for N is illustrated in Fig. 6. The parallel organization for *significant_flag*, $bin0$ and $bin13$ is shown in Fig. 7. The $R\&D$ module in Fig. 6 and Fig. 7 executes reading a code-word from a code stream and decoding the code-word into a bin. The $Undo$ module executes resetting the arithmetic decoder.

From Fig. 6 and Fig. 7, it can be seen that the context index can be calculated in parallel with $R\&D$. It seems that there is no context modeling process when parsing the coding stream. For example, when $R\&D$ circled by red square is executed for $bin0$ of *coeff_abs_level_minus1[1]*, the calculation of $C_{L^0_{bin0}}$ circled by green square for $bin0$ of *coeff_abs_level_minus1[0]* can be carried out at the same time because the prerequisite parameters like N and P_0 are available at that moment.

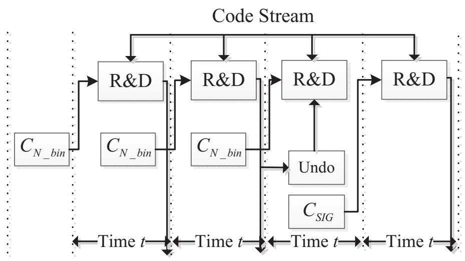


Fig. 6 Parallel organization of context modeling and arithmetic decoding for N in HDCMBAC

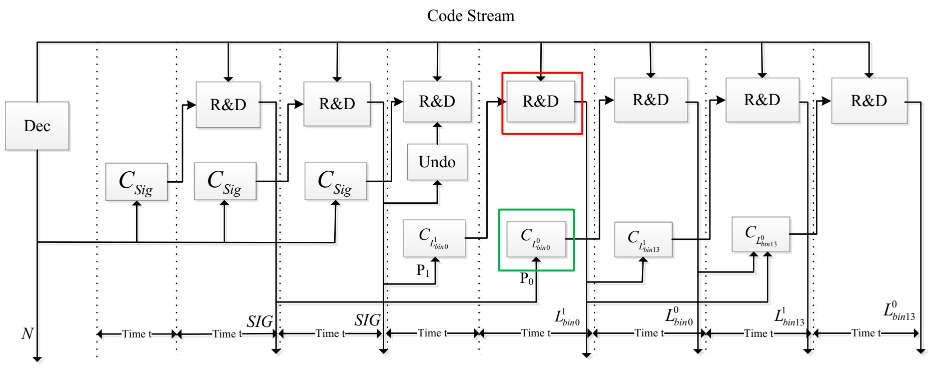


Fig. 7 Parallel organization of context modeling and arithmetic decoding for significant map and level information in HDCMBAC

4.3 Comparisons between HDCMBAC and transform coefficient coding in HEVC

HDCMBAC has some distinctions over the transform coefficient coding scheme [12] in HEVC. First, HDCMBAC uses the number of the significant coefficients in the block for separating the coding of *significant_flag* from the coding of *last_significnat_flag*, while HEVC uses the position of the last significant coefficient. Second, the context for *significant_flag* and the context for $bin0$ of level (corresponding to the syntax element *coeff_abs_level_greater1_flag* in HEVC) in HDCMBAC depend on the number of the significant coefficients and its scanned position. This context model for *significant_flag* and $bin0$ of level in HDCMBAC is simple to implement and allows the potential parallelism between context modeling and arithmetic coding. The last one is that the proposed context model could be embedded into the transform coefficient coding scheme in HEVC to further improve the data throughput by taking advantages of all methods (including mode dependent coefficient scanning method, Rice binarization for *coeff_abs_level_remaining*, and sign data hiding).

5 Experimental results

In this section, HDCMBAC is compared with CABAC to evaluate its coding performance, computational complexity and space complexity. The following experiments adopt H.264/AVC reference software Jm15.1 as the test platform, and the 4x4 DCT defined in H.264/AVC is used. The coding conditions for CABAC and HDCMBAC consist of 5 reference frames, 1/4-pixel motion vector resolution, +/-32 pixel motion search range, and R-D optimization enabled. The test videos include $Mobile$, $Foreman$, and $Paris$ in CIF@30Hz; $City$, $Crew$ and Ice in $4CIF$@30Hz; and $City$, $Bigships$ and $Cyclist$ in $720p$@30Hz; $BasketballDrive$@50HZ, $BQTerrace$@60HZ and $Cactus$@50HZ in 1920x1080; $PeopleOnStreet$ and $Traffic$ in 2560x1600@30HZ. The QPs 16, 20, 24, 28, 32 and 36 are used.

5.1 Coding performance of HDCMBAC

To evaluate the coding performance, the experiments are conducted under three configurations, which include all $Intra$, $IPPP$ and $IBBP$ coding structures. The bit rate difference (*BD-Rate*) is used to measure the coding performance, which can be calculated according to [1]. The *BD-Rate* under each configuration is shown in Table 3, where *BD-Rate* in $LowQP$ column is calculated at QPs 16, 20, 24 and 28, while *BD-Rate* in $HighQP$ column is calculated at QPs 24, 28, 32 and 36. A negative value of *BD-Rate* means that HDCMBAC requires less bits than CABAC to achieve the same video quality, while a positive value of *BD-Rate* means HDCMBAC requires more bits than CABAC to achieve the same video quality.

As shown in Table 3, the coding performance of HDCMBAC is similar as that of CABAC on average. But it can be seen that the *BD-Rate* is different for different configurations at both $LowQP$ and $HighQP$ columns. More specifically, HDCMBAC results in a little coding loss for $IPPP$ and $IBBP$ configurations, while it has some coding gain for all $intra$ configuration. In addition, the coding loss and coding gain in $HighQP$ column are less than those in $LowQP$ column.

To explain the reason for that, we use the mutual information $I(X;Y)$ of the syntax element X and its context Y to evaluate the efficiency of the context model, which is defined as follows:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) \tag{13}$$

Table 3 Bit-rate difference [%] of HDCMBAC relative to CABAC

Sequences	$BD-Rate$[%]					
	$LowQP$			$HighQP$		
	Intra	*IPPP*	*IBBP*	*Intra*	*IPPP*	*IBBP*
Mobile@CIF	−0.5	0.2	0.1	−0.2	0.3	0.4
Paris@CIF	−1.2	−0.7	−0.8	−0.7	0.1	0.0
Foreman@CIF	−0.2	−0.1	0.0	0.0	−0.2	0.2
City@4CIF	−0.2	0.4	0.4	−0.1	−0.2	0.1
Crew@4CIF	−0.3	0.2	0.3	−0.5	−0.2	0.0
Ice@4CIF	−1.1	−0.7	−0.5	−0.1	0.0	0.1
City@720p	−0.1	0.4	0.4	0.1	0.0	0.4
Bigships@720p	−0.1	0.7	0.6	0.1	0.2	0.2
Cyclist@720p	0.0	−0.1	0.1	0.0	−0.3	0.1
BasketballDrive@1920x1080	0.2	0.3	0.5	0.0	−0.1	0.0
BQTerrace@1920x1080	−0.5	0.9	0.9	−0.7	0.3	0.2
Cactus@1920x1080	0.0	0.8	0.7	0.0	0.1	−0.1
PeopleOnStreet@2560x1600	−0.2	−0.3	−0.3	−0.1	−0.1	0.0
Traffic@2560x1600	−0.6	−0.5	−0.5	−0.2	0.1	0.0
Average	−0.343	0.107	0.136	−0.171	0.000	0.114

where $p(x, y)$ is joint probability distribution function of X and Y, $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y, respectively.

As we know, the mutual information $I(X; Y)$ is larger, it means that the context model is more efficient, thus the number of bits to code the syntax element X using Y is smaller. Therefore, we compare the efficiency of the context model in HDCMBAC with that in CABAC by calculating the mutual information on some test sequences, which are shown in Table 4. According to the comparison, we find that the context model for *last_significant_flag* in CABAC is more efficient than that for N in HDCMBAC, the context model for *significant_flag* in HDCMBAC is more accurate than that in CABAC, and the accuracy of the context model for $bin0$ in HDCMBAC is similar as that in CABAC.

Because there are only a few non-zero coefficients in DCT block in $IPPP$ and $IBBP$ configurations, the number of bits saved by coding *significant_flag* cannot complement the number of bits consumed by coding N; whereas the number of bits saved by coding *significant_flag* can complement the number of bits consumed by coding N in all $intra$ configuration, since there are more non-zero coefficients in a DCT block. So, HDCMBAC results in a little coding loss for all $IPPP$ and $IBBP$ configurations, and has some coding gain for all $intra$ configuration. In addition, the number of the non-zero coefficients in DCT block will be decreased when the QP increases, thus the coding loss and coding gain of HDCMBAC will be decreased with QP. In addition to the context model, the rate-distortion technique [13] for video compression and the frame content have also some influences on the coding performance of HDCMBAC.

5.2 Analysis of potential parallelism in HDCMBAC

If the context modeling and arithmetic decoding are organized according to the method described in Section 4, it seems that there is no context modeling process in HDCMBAC when parsing the coding stream. Since HDCMBAC is a coefficient-level parallel technique, it only requires two processors to achieve the parallelism in HDCMBAC. However, it can be adopted by some parallel entropy coding algorithms [3, 6] related to multi-core platform to further improve the throughput; e.g. replacing CABAC with HDCMBAC. Not limited on the multi-core platform, HDCMBAC can also be implemented on other platforms supporting parallel processing, such as GPU, hardware architecture and so on, since the parallelism of HDCMBAC results from the reduction of the context dependencies between syntax elements, and it is not designed for a specific platform.

Table 4 Mutual information of syntax elements and their contexts in CABAC and HDCMBAC

Sequences	Mutual Information					
	$CABAC$			$HDCMBAC$		
	LAST	*SIG*	*bin0*	*N*	*SIG*	*bin0*
Mobile@CIF	0.282	0.102	0.278	0.164	0.285	0.287
City@4CIF	0.322	0.127	0.202	0.233	0.257	0.224
Bigships@720p	0.240	0.236	0.181	0.199	0.262	0.187
BQTerrace@1920x1080	0.311	0.067	0.176	0.188	0.208	0.196
PeopleOnStreet@2560x1600	0.157	0.146	0.306	0.202	0.212	0.271
Average	0.2624	0.1356	0.2286	0.1972	0.2448	0.2330

In our experiments, there is only one binary arithmetic coding engine M-coder, this means that the arithmetic decoding for all syntax elements is serial due to the strict data dependencies introduced by M-coder. In this case, we focus on the decoding process of the syntax elements designed for DCT coefficients in CABAC and HDCMBAC to obtain their execution time, i.e. N, *significant_flag* and *coeff_abs_level_minus1* in HDCMBAC, and *significant_flag*, *last_significant_flag* and *coeff_abs_level_minus1* in CABAC.

According to the parallel organization applied to HDCMBAC in Section 4, the execution time in HDCMBAC is the maximum one between context modeling and arithmetic decoding, while the execution time in CABAC is the sum of context modeling and arithmetic decoding due to the context dependency between syntax elements.

The execution time for HDCMBAC and CABAC is measured with the number of clock cycles consumed by the basic instructions in assembly language, since HDCMBAC and CABAC can be implemented through the basic instructions in assembly language. The basic instructions involved consist of *Add*, *Subtract*, *Shift*, *Compare*, *Logical* and *Move*. If the operands for *Add*, *Subtract*, *Shift*, *Compare* and *Logical* are all in registers, their execution time is one clock cycle for $Pentium$ processors. If one operand for *Move* is in memory and the other is in register, its execution time is also one clock cycle.

To quantitatively analyze the parallelism of HDCMBAC, the speedup of HDCMBAC relative to CABAC is used, which is defined as follows:

$$S_p = \frac{T_{CABAC}}{T_{HDCMBAC}} \tag{14}$$

where p is the number of processors; and T_{CABAC} and $T_{HDCMBAC}$ are the number of clock cycles consumed by CABAC and HDCMBAC, respectively. In our experiments, two processors are used to achieve the parallelism of HDCMBAC, thus $p = 2$.

Table 5 presents the speedup of HDCMBAC relative to CABAC under $IBBP$ coding structure. The average speedup of HDCMBAC relative to CABAC is 1.184 $\sim$ 1.153 at

Table 5 Speedup of HDCMBAC Relative to CABAC under $IBBP$ coding structure

QP / Sequences	16	20	24	28	32	36
Mobile@CIF	1.17	1.18	1.18	1.18	1.18	1.18
Paris@CIF	1.17	1.17	1.17	1.15	1.18	1.13
Foreman@CIF	1.16	1.16	1.18	1.18	1.17	1.18
City@4CIF	1.19	1.19	1.18	1.17	1.17	1.17
Crew@4CIF	1.19	1.18	1.16	1.15	1.14	1.13
Ice@4CIF	1.19	1.17	1.17	1.17	1.17	1.16
City@720p	1.19	1.19	1.18	1.17	1.16	1.15
Bigships@720p	1.19	1.18	1.18	1.17	1.17	1.15
Cyclist@720p	1.18	1.17	1.16	1.15	1.16	1.13
BasketballDrive@1920x1080	1.19	1.18	1.18	1.16	1.16	1.16
BQTerrace@1920x1080	1.18	1.19	1.19	1.19	1.17	1.16
Cactus@1920x1080	1.19	1.19	1.18	1.16	1.15	1.16
PeopleOnStreet@2560x1600	1.18	1.15	1.12	1.13	1.12	1.15
Traffic@2560x1600	1.19	1.18	1.18	1.17	1.16	1.15
Average	1.184	1.177	1.172	1.166	1.163	1.153

different QPs. As can be seen, the speedup is not significant, and it is nearly the same at different QPs, this is because the arithmetic decoding of all syntax elements is serial due to the strict data dependency introduced by M-coder, and the arithmetic decoding dominates the decoding process of HDCMBAC and CABAC. According to the collected data, the time cost by arithmetic coding is nearly four times more than that cost by context modeling. If the execution time of arithmetic decoding could be reduced, the speedup would become larger. For example, if we use the parallel binary arithmetic coding engine [15] to decode the syntax elements, and the speedup will be 1.423.

5.3 Computational complexity and memory requirement of HDCMBAC

The number of operations e.g. *Table lookup*, $Compare$ and Add, is used to measure the computational complexity of the context models in HDCMBAC and CABAC.

In HDCMBAC, both N and *significant_flag* require 1 $Compare$ and 1 *Table lookup* to get the (σ, V_{MPS}) pairs; for $bin0$, it first requires 1 *Table lookup* to get the context index and then 1 *Table lookup* to get the (σ, V_{MPS}) pair. In CABAC, both *significant_flag* and *last_significant_flag* require 1 *Table lookup* to get the (σ, V_{MPS}) pair, and 1 $Compare$ is still required for *last_significant_flag* to decide decoding it or not; for $bin0$, 1 $Compare$ and 1 Add are required to calculate $NumT1$ and $NumLgt1$, then 1 or 2 $Compare$ is required to get the context index, finally 1 *Table lookup* is used to get (σ, V_{MPS}) pair. Table 6 lists the number of operations consumed by the context modeling process in HDCMBAC and CABAC. It can be seen that the computational complexity of the context modeling process in HDCMBAC is lower than that in CABAC.

There are 5 block types in HDCMBAC and CABAC, which are *Luma_DC* (for *DC* coefficients in intra16x16 prediction block), *Luma_AC* (for *AC* coefficients in intra16x16 prediction block), *Luma_*4x4 (for coefficients in intra 4x4 or inter prediction block), *Chroma_DC* (for *DC* coefficients in chroma block) and *Chroma_AC* (for *AC* coefficients in chroma block). HDCMBAC needs the memory for storing the contexts of N, *significant_flag*, $bin0$ and $bin13$ for each block type, and the mapping relationship of (11) for $bin0$. CABAC spends memory to store the contexts of *significant_flag*, *last_significant_flag*, $bin0$ and $bin13$ for each block type. Since it requires 7 bits to represent one (σ, V_{MPS}) pair (6 bits for σ and 1 bit for V_{MPS}) and 2 bits to represent one entry of the mapping relationship of (11), the memory consumption for HDCMBAC and CABAC is listed in Table 7, in which $TableSize$ represents the number of bits consumed by the context table for a corresponding syntax element. The total size of the memory required by HDCMBAC is 5307 bits, while the total size of the memory required by CABAC is 1470 bits.

Table 6 The number of operations cost by context models in HDCMBAC and CABAC in H.264/AVC

	Syntax Elements	$Operations$		
		Table lookup	*Compare*	*Add*
HDCMBAC	*N*	1	1	
	significant_flag	1	1	
	bin0	2		
CABAC	*significant_flag*	1		
	last_significant_flag	1	1	
	bin0	1	3 or 2	1

Table 7 Memory Requirement for HDCMBAC and CABAC in H.264/AVC

	Objects	Table Size	Memory Requirement
HDCMBAC	*N*	$5 \times 4 \times 16 \times 7$	5307 bits
	significant_flag	$5 \times 4 \times 16 \times 7$	
	bin0	$16 \times 16 \times 2 + 5 \times 4 \times 7$	
	bin13	$5 \times 5 \times 7$	
CABAC	*significant_flag*	$5 \times 16 \times 7$	1470 bits
	last_significant_flag	$5 \times 16 \times 7$	
	bin0	$5 \times 5 \times 7$	
	bin13	$5 \times 5 \times 7$	

6 Conclusion

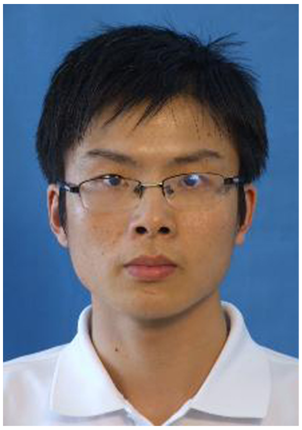
In this paper, a hierarchical dependency context model (HDCM) is firstly proposed for DCT block based on its statistical information, in which the number of non-zero coefficients in DCT block and the scanned position are used to capture the magnitude varying tendency of DCT coefficients. Then a new arithmetic coding scheme namely HDCMBAC is proposed to code the DCT block based on HDCM. HDCMBAC uses HDCM with binary arithmetic coding to code the number of non-zero coefficients, significant flag and level information of DCT block. Experimental results demonstrate that HDCMBAC can achieve similar coding performance as CABAC at low and high QPs. Meanwhile, the context modeling and the arithmetic decoding in HDCMBAC can be carried out in parallel, and a parallel organization scheme between them is proposed. Based on this parallel organization scheme, HDCMBAC can parse the coding stream almost like CABAC without context modeling.

Acknowledgments This work was supported in part by the Major State Basic Research Development Program of China (973 Program 2015CB351804) and the National Science Foundation of China (NSFC) under grant 61272386.

References

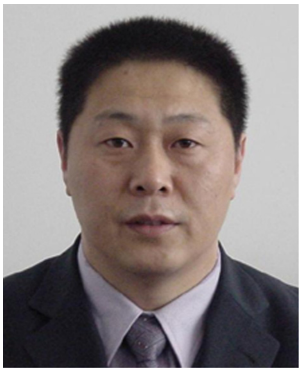
1. Bjontegaard G (2001) Calculation of average PSNR differences between RD-curves. In: Doc. ITU-T VCEG (VCEG-M33). Austin, Texas, USA
2. Bjontegaard G, Lillevold K (2002) Context-adaptive VLC (CVLC) coding of coefficients. In: Doc. joint video team ofISO/IEC MPEG & ITU-T VCEG (JVT-C028)
3. Chen S, Chen S, Sun S (2010) P3-CABAC: A nonstandard tri-thread parallel evolution of CABAC in the many core era. IEEE Trans Circuits Syst Video Technol 20(6):920–924
4. Francesc A (2014) Entropy-based evaluation of context models for wavelet-transformed images. IEEE Trans Image Process 24(1):1778–1791
5. Gao M, Fan X, Wang Q, Zhao D, Gao W (2011) A parallel context model for level information in CABAC, visual communications and image processing
6. Kim W, Cho K, Chung K (2011) Multi-threaded syntax element partitioning for parallel entropy decoding. IEEE Trans Consumer Electronics 57(2):897–905
7. Lam Y, Goodman W (2000) A mathematical analysis of the DCT coefficient distributions of images. IEEE Trans Image Process 9(10):1661–1666
8. Marpe D, Schwarz H, Wiegand T (2003) Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Trans Circuits Syst Video Technol 13(7):620–636
9. Marpe D, Schwarz H, Wiegand T (2010) Entropy coding in video compression using probability interval partitioning. In: Proceedings picture coding symposium

10. Moffat A, Neal R, Witten I (1995), Arithmetic coding revisited. IEEE international conference on data compression conference
11. Rissanen J (1983) A universal data compression system. IEEE Trans Inform Theory 29(5):656–664
12. Sole J, Joshi R, Nguyen N, Ji T, Karczewicz M, Clare G, Henry F, Duenas A (2012) Transform coefficient coding in HEVC. IEEE Trans Circuits Syst Video Technol 22(12):1765–1777
13. Sullivan G, Wiegand T (1998) Rate-distortion optimization for video compression. IEEE Signal Process Mag 15(6):74–90
14. Sullivan G, Ohm J, Han W, Wiegand T (2012) Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol 22(12):1649–1668
15. Sze V, Budagavi M (2008) Parallel CABAC for low power video coding. In: Proceedings IEEE international conference on image process
16. Sze V, Budagavi M (2012) High throughput CABAC entropy coding in HEVC. IEEE Trans Circuits Syst Video Technol 22(12):1778–1791
17. Taubman D (2000) High performance scalable image compression with EBCOT. IEEE Trans Image Process 9(7):1158–1170
18. Wang Q, Zhao D, Gao W, Ma S (2005) High efficiency context-based variable length coding with parallel orientation, Pacific Rim conference on multimedia
19. Weinberger M, Rissanen J, Arps B (1996) Applications of universal context modeling to lossless compression of gray-scale images. IEEE Trans Image Process 5(4):575–586
20. Weinberger M, Seroussi G, Sapiro G (2000) The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. IEEE Trans Image Process 9(8):1309–1324
21. Wiegand T, Sullivan G, Bjontegaard G, Luthra A (2003) Overview of the H.264/AVC video coding standard. IEEE Trans Circuits Syst Video Technol 13(7):560–576
22. Wu X (1997) Lossless compression of continuous-tone images via context selection, quantization, and modeling. IEEE Trans Image Process 6(5):656–664
23. Wu X, Memon N (1997) Context-based, adaptive, lossless image coding. IEEE Trans Commun 45(4):437–444
24. Wu J, Xu Z, Jeon G, Zhang X, Jiao L (2013) Arithmetic coding for image compression with adaptive weight-context classification. Signal Process Image Commun 28(7):727–735
25. Xu M, Wu X, Franti P (2006) Context quantization by kernel fisher discriminant. IEEE Trans Image Process 15(1):169–177
26. Yu L, Chen S, Wang J (2009) Overview of AVS-video coding standards. Signal Process Image Commun 24(4):247–262
27. Zhang L, Wang Q, Zhang N, Zhao D, Wu X, Gao W (2009) Context-based entropy coding in AVS standard. Signal Process Image Commun 24(4):263–276



Min Gao received his BS and MS degrees in Department of Computer Science and Technology from Harbin Institute of Technology (HIT) in 2009 and 2011, respectively. Now he is pursuing the Ph.D. degrees in the same university. His research interests include image/video coding and transmission.

Qiang Wang received the B.S., M.S. and Ph.D. degrees in Department of Computer Science and Technology from Harbin Institute of Technology (HIT), Harbin, China, in 2001, 2003 and 2009, respectively. Since 2002, he has been with the Joint R&D Lab (JDL), Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His research interests mainly focus on image and video compression.



Debin Zhao received the B.S., M.S., and Ph.D. degrees in computer science all from Harbin Institute of Technology (HIT), China, in 1985, 1988, and 1998, respectively. Now he is a professor of HIT and Institute of Computing Technology, Chinese Academy of Sciences. He has been a research fellow at Dept. of Computer Science, City University of Hong Kong. His research interests include multimedia compression. He has authored or co-authored over 100 publications.

Wen Gao received the Ph.D. degree in electronics engineering from the University of Tokyo, Japan, in 1991. Now he is a professor of computer science at Peking University, China. He has published extensively including five books and over 600 technical articles in refereed journals and conference proceedings in the areas of image processing, video coding and communication, pattern recognition, multimedia information retrieval, multimodal interface, and bioinformatics.

Dr. Gao served or serves on the editorial board for several journals, such as IEEE Trans. Circuits and System for Video Technology, IEEE Trans. Multimedia, IEEE Trans. Autonomous Mental Development, EURASIP Journal of Image Communications, Journal of Visual Communication and Image Representation. He chaired a number of prestigious international conferences on multimedia and video signal processing, such as IEEE ICME and ACM Multimedia, and also served on the advisory and technical committees of numerous professional organizations.