CONFIGURABLE VARIABLE LENGTH CODE FOR VIDEO CODING

Ngai-Man Cheung and Yuji Itoh

E-mail: cheung@ti.com, yitoh@ti.com

Tsukuba Research & Development Center, Texas Instruments Japan Ltd.

ABSTRACT

The variable length code (VLC) tables in the MPEG-1/2/4 and H.263 are fixed and optimized for a limited range of bit-rates, and they cannot handle a variety of applications. The universal variable length code (UVLC) is a new scheme to encode syntax elements and has some configurable capabilities. It is also being considered in the ITU-T H.26L. However, the configurable feature of the UVLC has not been well explored. In this paper we propose configuring the UVLC with the additional code configuration (ACC). The ACC is used to adapt UVLC to different symbol distributions by adjusting the partitioning of the symbols into different categories, and the code size assignment to different categories. Experimental results show that the UVLC with ACC outperforms the current proposed scheme in H.26L and the VLC tables of existing standards, while drastically simplifying the encoding and decoding process, and is applicable to a variety of applications.

1. INTRODUCTION

The variable length coding is a statistical coding technique that assigns symbols to code words based on the occurrence frequency of the symbols. Symbols that occur more frequently are assigned short code words while those that occur less frequently are assigned long code words. Compression is achieved by the fact that overall the more frequent shorter code words dominate. The variable length code (VLC) tables in MPEG-1/2/4 and H.263 are fixed and optimized for a limited range of bit-rates, and they cannot handle a variety of applications. For example, in the MPEG-2, the VLC tables for DCT coefficients are designed for broadcast quality video applications, and they cannot readily handle low bit-rate applications. Therefore, it is desirable to have a type of configurable VLC that can handle a wide range of applications and coding conditions. The universal variable length code (UVLC) was proposed in [1] as a new scheme to encode syntax elements and offers such advantage. The UVLC tables are constructed by a dozen of *coarse codes* and *additional codes*. The UVLC can handle a wide range of applications and different syntax elements by changing the construction rules. Figure 1 depicts the UVLC for DCT coefficients coding. The UVLC also drastically simplifies the encoding and decoding process. Furthermore, a look-up table is not essential for either the encoder or decoder. With these advantages, the UVLC is being considered in the ITU-T H.26L [2] to code all the syntax elements, and will potentially become an important technique in video coding. Details about UVLC can be found in [1].

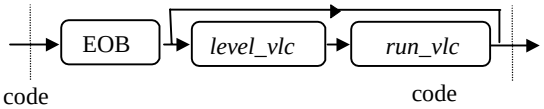
Although the UVLC can be adapted to handle a variety of applications and coding conditions, the configurable feature has not been well explored. In [3], we presented some works using the code length of the end of block to adapt the UVLC to different bit-rate applications. In [4] the Dynamic Symbol Reordering (DSR) method was proposed to automatically re-configure the UVLC. The DSR uses the probability of each symbol to construct a mapping table that re-orders the assignments of symbols to code words. The method is suitable for coding syntax elements which have only a few different symbols, e.g. the macroblock type in H.26L, which has only 9 different symbols. It is not practical for syntax elements like transform coefficients or motion vectors, which consist of a lot of different symbols. Since the transform coefficients and motion vectors make up a significant portion of the total encoded bits, it is foremost important to be able to encode these syntax elements optimally in different types of applications.

In this paper we propose configuring UVLC with the additional code configuration. The method is applicable to code the syntax elements with many different symbols like the transform coefficients and motion vectors. Section 2 of this paper describes the method. The additional code configuration is used to tune the UVLC to different symbol rates and symbol types. The additional code configuration can be determined on the fly during video encoding, or off-line during training of code tables. Section 3 presents the experimental results, and shows that the method can achieve very good coding efficiency while drastically simplifying the encoding and decoding process, and is widely applicable. Finally, we conclude the work in Section 4.

2. CONFIGURABLE VARIABLE LENGTH CODE

In this section we describe configuring UVLC using the additional code configuration. As shown in Figure 1, the universal variable length coding divides the symbol into different categories, and assigns different coarse code to each category. Within a category, the additional code identifies individual symbol. The length of the end of block (EOB) can be used to adjust the UVLC tables for different bit-rates applications. For example, it is found that for the luminance level symbols, we should set EOB length=2 for inter picture and EOB length=3 for intra picture, since intra picture has more symbols per block and EOB happens relatively less often. In [3], the code sizes of the additional codes are fixed for all bit-rates.

Suppose we try to divide the symbols into L categories. The additional code configuration, ACC, is a one-dimensional array of L integers, $[r_k : k=0 \text{ to } L-1]$, where r_k is the code size of the additional code for the k*th* category. For example, the ACC for the runs in Figure 1 is [0,0,1,2,3,4,5]. Instead of having a fixed ACC, we propose to have several different ACCs for different bit-rates and symbol types.



run_vlc coarse code	additional code	code size	value of run
1		1	0
01		2	1
001	x_0	4	"x_0"+2 (2:3)
0001	x_1x_0	6	"x_1x_0"+4 (4:7)
00001	$x_2x_1x_0$	8	"$x_2x_1x_0$"+8 (8:15)
000001	$x_3x_2x_1x_0$	10	"$x_3x_2x_1x_0$"+16 (16:31)
0000001	$x_4x_3x_2x_1x_0$	12	"$x_4x_3x_2x_1x_0$"+32 (32:63)

level_vlc coarse code	additional code	code size	absolute value of level
1	s	2	1
01		2	EOB
001	x_0s	5	"x_0"+2 (2:3)
0001	x_1x_0s	7	"x_1x_0"+4 (4:7)
00001	$x_2x_1x_0$s	9	"$x_2x_1x_0$"+8 (8:15)
000001	$x_3x_2x_1x_0$s	11	"$x_3x_2x_1x_0$"+16 (16:31)
0000001	$x_4x_3x_2x_1x_0$s	13	"$x_4x_3x_2x_1x_0$"+32 (32:63)
00000001	$x_5x_4x_3x_2x_1x_0$s	15	"$x_5x_4x_3x_2x_1x_0$"+64(64:127)
's' denotes the sign of level. '0' for positive and '1' for negative			

UVLC table for level configured with EOB (end of block) length = 2.

Figure 1. The UVLC for DCT coefficient coding.

The coding performance of the UVLC can be improved with ACC as follows. First of all, the universal variable length coding should try to optimally divide the symbols into different categories. As the symbols within a category have the same code size, they should have similar occurrence probabilities, $2^{(-\text{code size})}$, for optimal performance. In other words, the universal variable length coding should try to partition the symbols such that symbols within a category have similar occurrence probabilities. Since the symbol distributions at different bit-rates may be quite different, and the same symbol may have quite different occurrence probabilities across different bit-rates, so a fixed partition will not work optimally for all bit-rates. Using ACC we can adjust the partition of symbols to fit different symbol distributions. Let integer j, *MIN_J*, *MAX_J* denote the value, the minimum and the maximum of the symbols respectively, i.e., $MIN\_J \leqq j \leqq MAX\_J$, and t_0, t_1, …, t_{L-1} denote the boundary values of the categories. The k*th* category is $[t_k, t_{k+1}-1]$. The range of each category is 2^{r_k}, and the boundary values are expressed as

$$t_{k+1} = t_k + 2^{r_k} \quad (1)$$

$$t_0 = MIN\_J \quad (2)$$

$$t_k = MIN\_J + 2^{r_0} + 2^{r_1} + \ldots + 2^{r_{k-1}} \quad (3)$$

Also we have

$$r_{L-1} = ceil(\log_2(MAX\_J - MIN\_J - (2^{r_0} + 2^{r_1} + \ldots + 2^{r_{L-2}}) + 1)) \quad (4)$$

So we can use the ACC, $[r_k]$, to adjust the boundary values t_k, and hence the way we partition the symbols, according to the symbol distributions.

Moreover, the universal variable length coding should try to assign optimal code size to each category. If the average occurrence probability of the symbols in the k*th* category is p_k, then the code size of symbols in that category should be $-\log_2(p_k)$ for optimal performance. As the occurrence probabilities of symbols may be different for different bit-rates, we should adjust the code sizes correspondingly, instead of having fixed code sizes for all bit-rates. Let c_k, cs_k be the coarse code size and the code size of the k*th* category respectively. As r_k is the code size of the additional code, we have

$$cs_k = c_k + r_k \quad (5)$$

We assign short coarse codes to small symbols, which appear more frequently in general. Let $c_k=k+1$, then

$$cs_k = r_k + k + 1 \quad (6)$$

For optimal code size assignment, the ACC, $[r_k]$, and p_k is related by

$$r_k = -(\log_2 p_k + k + 1) \quad (7)$$

The above issues are not independent, since how we assign the code sizes to the categories will affect the way we partition the symbols. This makes the problem of determining the optimal ACC difficult. In this paper we determine the ACC by examining the symbol distributions. Let N_j be the number of occurrences of the symbol j. The total bits used to encode the symbols, *B*, is

$$B = \sum_{k=0}^{L-1} (cs_k \times \sum_{j=t_k}^{t_{k+1}-1} N_j) \quad (8)$$

An optimal ACC should minimize *B*.

Figure 2 shows the UVLC with ACC. The proposed coding scheme has a very regular structure, and requires very simple encoding and decoding process. The coding scheme can assume different probability distributions and be instantiated to several popular coding schemes. For example, when $[r_k] = [0,1,2,3,4,\ldots,k,\ldots,L-1]$, then it resembles the Elias Gamma code, which is suitable for proportionally decreasing symbol distributions. When $[r_k] = [1,1,1,1,\ldots,1]$, then it resembles the Golomb codes G(2). In general, for any integer $\psi \geqq 0$, when $[r_k] = [\psi,\psi,\psi,\ldots,\psi]$, then the UVLC resembles Golomb codes $G(2^{\psi})$. Figure 3 shows several examples.

(a) Additional Code Configuration:

ACC: $[r_k]$ $r_k \geqq 0$ k:0 to L-1

(b) Regular Codeword Structure:

	codeword: coarse code	additional code	*code size:*	*value:*
k*th* category:	00…01	$x_{r_{k-1}}\ldots x_1x_0$	$k+1+r_k$	$[t_k,t_{k+1}-1]$
number of bits:	k+1	r_k		

$$t_0 = MIN\_J; t_{k+1} = t_k + 2^{r_k}$$

Figure 2. The UVLC with ACC.

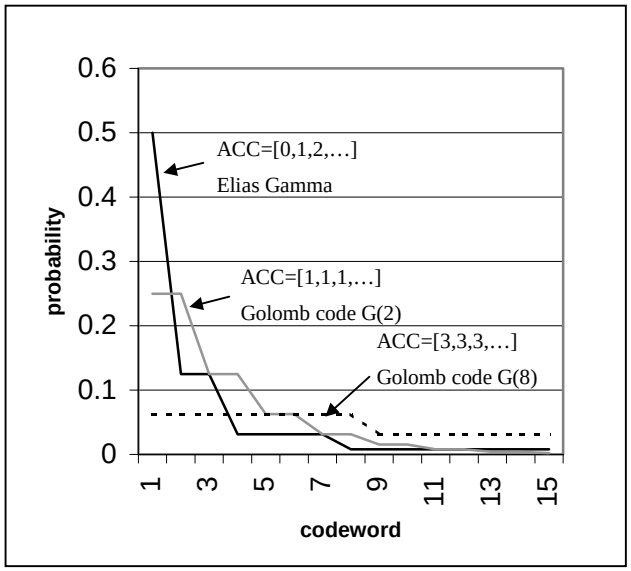


Figure 3. The UVLC with ACC can assume different probability distributions and be instantiated to several popular coding schemes.

The value of L can vary from 1 to the number of different values of symbols. For example, L can vary from 1 to 64 for run. Consider the two extreme cases. When L=1,

$$B = \sum_j cs_0 \times N_j \qquad (9)$$

where cs_0=6 by Equations (4) and (5), and we do not need a coarse code. This is not optimal, as we should assign shorter codes (instead of cs_k=6) to the small runs, which occur more frequently in general. When L=64,

$$B = \sum_j (j+1) \times N_j \qquad (10)$$

as we need different coarse codes for each symbol. This is also not optimal, as the long code words for large runs (can up to 64) will significantly increase B. In the experiment we choose some values of L that are good compromise between the above issues.

3. EXPERIMENTS

In this section we presented the comparison results between the proposed coding scheme and the entropy coding scheme in H.26L TML-4 [2], and the VLC tables in MPEG-1/2.

The current test model in H.26L uses an entropy coding scheme based on the UVLC. A single code structure is used to code all syntax elements. The codewords are constructed by interleaving fixed-length codes (FLC) into symmetric variable length code [2]. The codewords are numbered from 0 and upwards. For each type of syntax element, table is used to map each symbol into the codeword number. It is found that while the coding performance of the current scheme at middle range quantization scales (QP) is very competitive, there is still room for improvement at low and high QP, in particular for the transform coefficients corresponding to the TCOEFF_Luma_SimpleScan syntax element, which makes up a significant portion of the total encoded bits [5].

We performed some simulations using the UVLC with ACC to code the TCOEFF_Luma_SimpleScan syntax element, using the Telenor software TML4.3. We did not change the mapping from symbol to codeword number, but only the codeword itself. Figure 4 shows the results for the 'foreman' sequence (QCIF 10fps) and the 'mobile' sequence (CIF 30fps). The redundancy is calculated by dividing the total number of bits by the total number of symbols. The improvement is calculated by the formula

$$\text{Improvement} = \frac{\text{Bits used by H.26L UVLC - Bits used by UVLC with ACC}}{\text{Bits used by H.26L UVLC}} \times 100\%$$

We used different ACCs for different QP determined by examining the symbol distributions, and the same ACC throughout the sequence. The number of categories L was set to 10. As shown in the figure, the UVLC with ACC outperforms the current H.26L UVLC in every QP, and by as much as 12.77% in some case. It is expected that we can further improve the results by changing the ACC throughout the sequence.

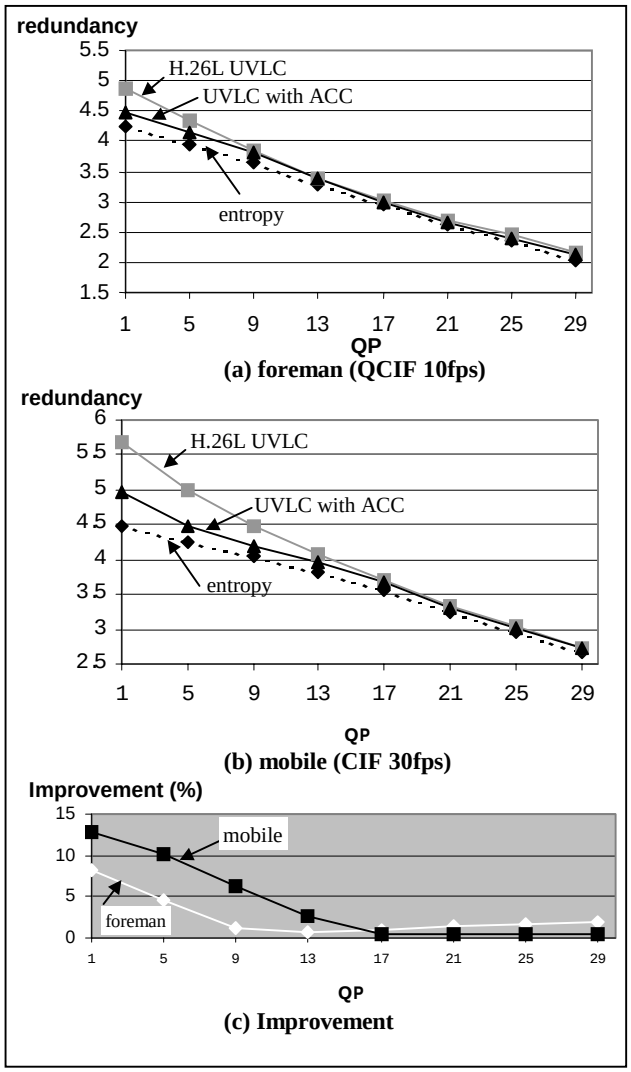


Figure 4. Comparing the UVLC with ACC to the H.26L UVLC.

To investigate how the UVLC with ACC can outperform the H.26L UVLC, we examine the probability distributions of the UVLC with ACC, H.26L UVLC, and the distribution of symbols. The results are shown in Figure 5. As shown in Figure 5(a), the UVLC with ACC can effectively match the symbol distribution, especially at the region with small codewords, thanks to the improved flexibility in symbol partition and code size assignment. Also the UVLC with ACC can readily adapt to a different symbol distribution as shown in Figure 5(b), which has a much lower bit-rate.

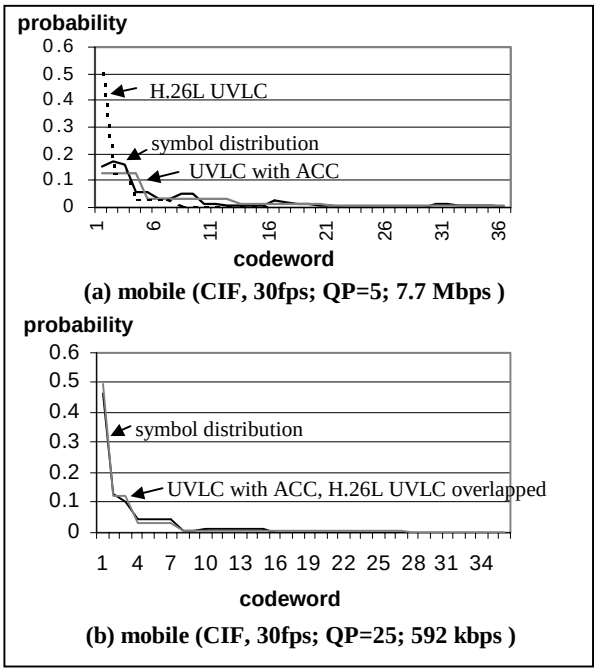


Figure 5. Probability distributions of the H.26L UVLC, UVLC with ACC, and symbols, at different bit-rates.

We also compared the UVLC with ACC to the VLC tables in MPEG-1/2 [6]. We compared the bits used in coding DCT coefficients. For the UVLC with ACC, we used the symbol-to-codeword mappings similar to Figure 1, i.e., the runs and levels are coded separately. Figure 6 shows the improvement. As shown in the figure the UVLC with ACC outperforms the MPEG-1/2 VLC in every case when coding DCT coefficients, and by as much as 6.77% in some case. Also the UVLC with ACC outperforms the one with only EOB length configuration as in [3].

4. CONCLUSIONS

We have proposed a configurable variable length coding scheme based on adjusting the UVLC with ACC. The ACC is used to adapt the UVLC to different symbol distributions by adjusting the partitioning of the symbols into different categories, and the code size assignment to different categories. The method is applicable to code the syntax elements with a lot of different symbols like the transform coefficients and motion vectors. We have presented the experimental results and showed that the method consistently outperforms the entropy scheme in H.26L TML-4 [2], and the VLC tables in MPEG-1/2, and by as much as 12.77% in some case. The UVLC with ACC can achieve very good coding efficiency while drastically simplifies the encoding and decoding process, and is applicable to a variety of applications. With these advantages it will be very useful for video coding.

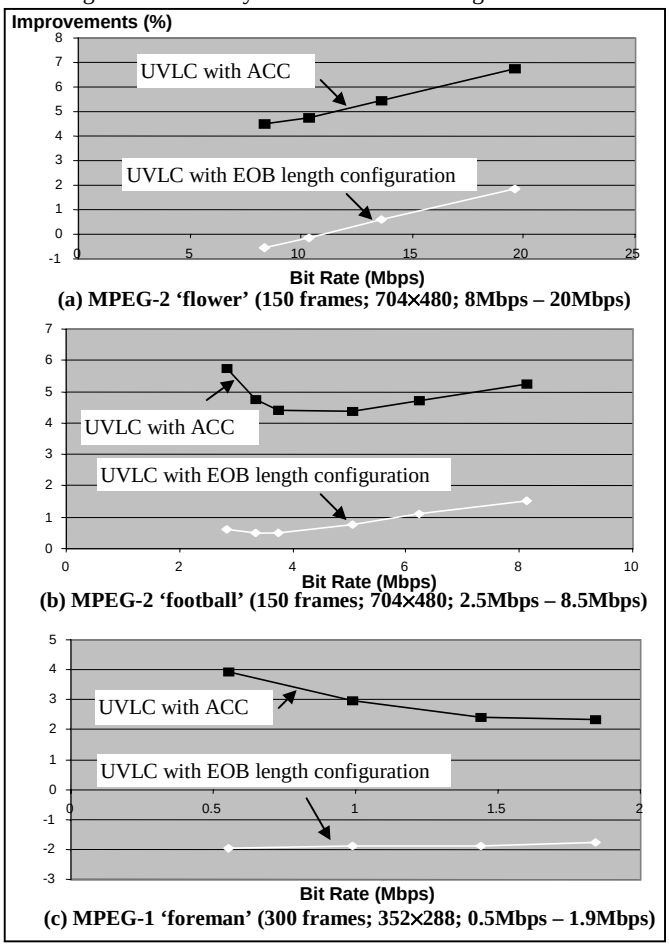


Figure 6. Comparing UVLC with ACC to the VLC tables in MPEG-1/2.

5. REFERENCES

[1] Yuji Itoh, "Bi-directional motion vector coding using universal VLC," *Signal Processing: Image Communication*, Vol. 14, pp. 541-557, May 1999.

[2] G. Bjontegaard, "H.26L Test Model Long Term Number 4 (TML-4)," *ITU-T Q.15/16*, Doc. #Q15-J72, June 2000.

[3] Yuji Itoh and N.-M. Cheung, "Universal variable length code for DCT coding," in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, Vancouver, Canada, Sept. 10-13, 2000.

[4] K.-Y. Yoo, B.-S. Choi and Y.-Y. Lee, "Improvements to the Telenor proposal for H.26L: Preliminary results on Dynamic Symbol Reordering (DSR) method for Universal VLC encoding/decoding," *ITU-T Q.15/16*, Doc. #Q15-H19, August, 1999.

[5] Louis Kerofsky, "Entropy coding of transform coefficients," *ITU-T Q.15/16*, Doc. #Q15-K45, August 2000.

[6] ISO/IEC 13818-2, "Generic coding of moving pictures and associated audio information – Part 2: Video," November 1994.