

Complexity Modeling of H.264/AVC CAVLC/UVLC Entropy Decoders

Szu-Wei Lee and C.-C. Jay Kuo

Ming Hsieh Department of Electrical Engineering and Signal and Image Processing Institute

University of Southern California, Los Angeles, CA 90089-2564, USA

E-mails: suzweile@usc.edu and cckuo@sipi.usc.edu

Abstract—A complexity model for context-based adaptive variable length coding (CAVLC) and universal variable length coding (UVLC) in the H.264/AVC decoder is proposed. CAVLC and UVLC are used to code source data, *i.e.*, quantized transformed coefficients (QTCs), and header data, respectively. CAVLC and UVLC are important in the baseline profile of the H.264/AVC codec since this profile targets at low complexity and low cost applications so that only CAVLC and UVLC are allowed to be the entropy coding tools. This model is verified experimentally. Possible applications of this model are also described. Since the entropy decoding complexity becomes significant in the high bit rate video due to more non-zero QTCs and motion vectors (MVs), one application scenario is to integrate this model into the H.264 encoder so that the encoder can estimate the possible decoding complexity and then select proper coding modes to generate the decoder-friendly bit streams, which are easy to decode for a particular decoding platform.

I. INTRODUCTION

One possible solution to reduce the decoding complexity is that the encoder can generate the decoder-friendly bit streams, which are easy to decode for a particular decoding platform. Under such a scenario, the decoding complexity model must be included into the encoder so that it can estimate the possible decoding complexity and then choose the best coding mode to balance the trade off between rate-distortion (RD) and decoding complexity. In H.264/AVC, there are two entropy coding modes. One is the context-based adaptive binary arithmetic coding (CABAC), which can be used to code all syntax elements in H.264/AVC. The other mode consists of two entropy coding tools: context-based adaptive variable length coding (CAVLC) and universal variable length coding (UVLC). Unlike CABAC, CAVLC is only used to encode quantized transformed coefficients (QTCs) while UVLC is adopted to encode header data, such as motion vectors (MVs), macro-block (MB) types, intra prediction type, and other flags. In this paper, we consider the decoding complexity model of CAVLC/UVLC.

There are several reasons for us to study this topic. First, it has been observed that entropy decoding demands a higher computational complexity for high bit rate video streams (*e.g.*, high definition contents) due to more non-zero QTCs and MVs. Second, entropy decoding actually becomes the most computationally expensive process while other decoding processes, such as motion compensation and de-blocking filtering, are implemented in the graphic processor unit (GPU) [1], in which entropy decoding becomes the main bottleneck

and its complexity reduction is critical. Furthermore, although CABAC can provide better RD performance, the H.264/AVC baseline profile is limited to CAVLC/UVLC only since it targets at the low complexity and low delay applications. In other words, this profile is particularly suitable for portable devices. When the video bit stream is decoded in portable devices, it is desirable to reduce the decoding complexity for power saving.

To estimate the complexity of variable length decoding (VLD), a complexity model was reported in [2]. The sum of magnitudes of non-zero QTCs and the sum of run lengths of zero QTCs are first estimated. Then, the entropy decoding complexity is modeled as a function of these two parameters. This model is however not suitable for CAVLC/UVLC in H.264/AVC for two reasons. First, CAVLC is more complicated than conventional variable length coding (VLC) schemes and, thus, cannot be well captured by existing complexity models. Second, H.264/AVC adopts CAVLC and UVLC to encode QTCs and header data, respectively. This flexibility makes the decoding complexity modeling more sophisticated as compared with that in [2]. More recently, a CABAC decoding complexity model was proposed in [3]. Since CABAC is different from CAVLC and UVLC, this model cannot be applied to estimate the CAVLC/UVLC decoding complexity. To address above issues, a novel complexity model for CAVLC/UVLC entropy decoding in H.264/AVC will be proposed in this paper.

The rest of this paper is organized as follows. A CAVLC/UVLC decoding complexity model is proposed in Sec. II. This model is verified experimentally and the results are shown in Sec. III. Possible applications of this complexity model and its integration into the H.264/AVC encoder are discussed in Sec. IV. Finally, concluding remarks are given in Sec. V.

II. DECODING COMPLEXITY MODELS FOR CAVLC AND UVLC

A. CAVLC Encoding and Decoding

The CAVLC encoding process is based on several properties of QTCs in one block [4], [5]. First, QTCs in one block are usually sparse (*i.e.*, the 2-D array contains lots of zeros) so that the run length coding is used to encode these repeating zeros. Second, non-zero QTCs of the current block are correlated

with those of neighboring blocks, and the magnitudes of non-zero QTCs tend to decrease from low to high frequencies. The two properties of non-zero QTCs are used in the VLC table selection, and the coding of each non-zero QTC is implemented by table look-up. In other words, the selection of the VLC table and its codewords are context adaptive. Third, after being converted into a 1-D array using the zig-zag scan order, high frequency non-zero QTCs are usually equal to ± 1 with equal probabilities. This property is exploited by CAVLC, where trailing ones are treated as a different case from normal non-zero QTCs so as to reduce the number of coding bits for non-zero QTCs furthermore.

The CAVLC encoding process is described below. QTCs in one block is first mapped into a 1-D array in the zig-zag scan order. Then, the number of non-zero QTCs and the number of trailing ones (*i.e.*, high frequency QTCs with their magnitude equal to 1) are encoded in the bit stream, where the number of trailing ones is at most three. If the number of trailing ones is larger than three, the remaining QTCs with value equal to ± 1 are treated as normal non-zero QTCs. Then, the sign bits of trailing ones are stored into the bit stream in the reverse zig-zag scan order, which is the order from high to low frequencies. After that, the value of each non-zero QTC, which is called the level, is encoded by VLC table look-up, where the selection of the VLC table depends on neighboring blocks and the value of the non-zero QTC. This implies that the VLC table selection is context adaptive. Finally, the total number of zeros before the last non-zero QTC and the number of zeros before each non-zero QTC (called the “run before”) are encoded. In CAVLC, both the level and the “run before” are encoded in the reverse zig-zag scan order.

Based on the above discussion, the CAVLC encoding process includes the following syntax elements: the number of non-zero QTCs, the number of trailing ones, the sign bits of trailing ones, the level, the total number of zeros, and “run before”.

The CAVLC decoding process is shown in Fig. 1. The number of non-zero QTCs and that of trailing ones are first decoded. Then, non-zero QTCs with value equal to ± 1 are reconstructed with their corresponding sign bits. After that, non-zero QTCs and zeros before each non-zero QTC are decoded. Finally, non-zero QTCs are stored into a 1-D array in the zig-zag scan order.

B. CAVLC Decoding Complexity Model

As shown in Fig. 1, the CAVLC decoding process consists of four modules. The first and the second modules are used to decode the sign bits of trailing ones and the level, respectively. Then, the third and the fourth modules are used to decode the “run before” and reconstruct QTCs in the zig-zag scan order, respectively. In terms of program implementation, the four modules are implemented as four loops.

Clearly, the CAVLC decoding complexity depends on the loop number of these four modules. Thus, the CAVLC decoding complexity is modeled as a function of the number of non-skipped MB (N_{mb}), the number of CAVLC executions

```

readCoeff4x4_CAULC ()
{
    [numcoeff, numtrailingones] = NumCoeffTrailingOnes();

    // decode sign of trailing ones
    sign_of_trailingones = sign_U VLC(&currSE, dP->bitstream);
    for (k=numcoeff-1; k>numcoeff-1-numtrailingones; k--)
    {
        if ((sign_of_trailingones[k]&1)
            level[k] = -1;
        else
            level[k] = 1;
    }

    // decode levels
    for (k = numcoeff - 1 - numtrailingones; k >= 0; k--)
        level[k] = Level_U VLC();

    // decode total zeros
    total_zeros = TotalZeros_U VLC(&currSE, dP);

    // decode run before each coefficient
    zerosleft = totzeros;
    i = numcoeff-1;
    while (zerosleft != 0 && i != 0);
    {
        run[i] = Run_U VLC(&currSE, dP);
        zerosleft -= run[i];
        i--;
    }

    // reconstruct QTC in zig-zag scan order
    for(i=k-0; k<numcoeff; k++)
    {
        if (level[k])
        {
            i+=run[k];
            coeff[i]=level[k];
        }
    }
}

```

Fig. 1. The CAVLC decoding process.

(N_{cavlc}), the number of trailing ones (N_{one}), the number of remaining non-zero QTCs (N_{qtc}), and the number of run executions (N_{run}). Mathematically, the CAVLC decoding complexity, C_{cavlc} , is expressed by

$$C_{cavlc} = \omega_{mb} \cdot N_{mb} + \omega_{cavlc} \cdot N_{cavlc} + \omega_{one} \cdot N_{one} + \omega_{qtc} \cdot N_{qtc} + \omega_{run} \cdot N_{run}, \quad (1)$$

where ω_{mb} , ω_{cavlc} , ω_{qtc} , ω_{one} , and ω_{run} are weights for these five decoding complexity coefficients, respectively. Please note that the number of trailing ones is limited to be three in CAVLC.

C. UVLC Encoding and Decoding

UVLC is used to code header data, such as MVs, the MB type, inter and intra prediction modes, and so on. The UVLC coding process uses the Exp-Golomb (EG) code. The EG code for an unsigned integer syntax element of value C has the following bit fields:

$$[M \text{ zeros}][1][INFO] \quad (2)$$

where $M = \text{floor}(\log_2[C + 1])$, $INFO = C + 1 - 2^M$, and the separation bit between $M \text{ zeros}$ and $INFO$ is one. In other words, the EG code is a variable length code. A signed integer syntax element of value SC is first mapped into an unsigned integer value, UC , via

$$UC = 2 \cdot |SC|, \quad \text{if } SC \leq 0, \\ UC = 2 \cdot SC - 1, \quad \text{otherwise.} \quad (3)$$

Then, UC is coded by the EG code. The EG code decoding process is simple. First, M zeros are read until one is reached, and then M -bit $INFO$ is read from the bit stream. Finally,

the unsigned integer syntax element of value C is constructed by $C = INFO + 2^M - 1$.

D. UVLC Decoding Complexity Model

To model the UVLC decoding complexity, we examine the complexity of the EG decoding process, which consists of reading M zeros and M -bit $INFO$ in the EG code bit field. However, the implementation of EG code bit field reading might be different among decoders. For example, a simple decoder can read one bit a time and, therefore, $2M + 1$ times of reading bits are required to decode a syntax element. On the other hand, a sophisticated decoder can read eight bits (*i.e.*, one byte) a time and then examine if the one-byte data is zero or not. If the one-byte data is non-zero, it implies that the separation bit between M zeros and $INFO$ is found. If the one-byte data is zero, the same operation is performed again until the separation bit is identified. The execution time of the above operations are $2 \cdot \lceil \frac{M}{8} \rceil$. As a result, the decoding complexity model that considers the length of the EG code bit field is not suitable for all decoders.

Here, the UVLC decoding complexity is modeled as a function of the number of skipped MBs ($N_{skipped}$), the number of intra blocks (N_{intra}), the number of MVs (N_{mv}), and the number of reference frames (N_{ref}). These four decoding complexity coefficients are used to model the decoding complexities for MB types, MB subtypes, intra prediction types, MVs, and reference frame indices, respectively. Then, the UVLC decoding complexity, C_{uvlc} , can be written as

$$C_{uvlc} = \omega_{skipped} \cdot N_{skipped} + \omega_{intra} \cdot N_{intra} + \omega_{mv} \cdot N_{mv} + \omega_{ref} \cdot N_{ref}, \quad (4)$$

where $\omega_{skipped}$, ω_{intra} , ω_{mv} , and ω_{ref} are weights of these four decoding complexity coefficients, respectively.

The above decoding complexity model is not built upon a specific decoder implementation, and it is suitable for variant H.264/AVC decoders. Note that the number of intra blocks depends on the intra prediction mode. For example, there are 16, 4, and 1 intra blocks for I4MB, I8MB and I16MB, respectively. The number of intra blocks is used to estimate the complexity of decoding intra prediction directions. The number of reference frames depends on inter prediction modes. For example, there are 1, 2, 2 and 4 reference frames for the MB coded by P16x16, P16x8, P8x16 and P8x8 inter prediction modes, respectively. The number of reference frames in one MB is at most four because those blocks within one 8x8 block share the same reference frame index.

The weights in the above model can be obtained as follows. First, several pre-encoded bit streams are selected, and the Intel Vtune performance analyzer is used to measure the number of clock ticks in CAVLC and UVLC decoding, which gives the measures of C_{cavlc} and C_{uvlc} . Second, the numbers of these decoding complexity coefficients are counted individually for those pre-encoded bit streams. Finally, a constrained least square method is used to find the weights. Note that the weights in the proposed model vary among different decoding platforms. Thus, they must be trained for a particular decoding

platform first so that the proposed model can estimate the possible CAVLC/UVLC decoding complexity. The proposed CAVLC/UVLC decoding complexity model will be verified experimentally in Sec. III.

III. EXPERIMENTAL RESULTS

We conducted experiments to verify the proposed CAVLC/UVLC decoding complexity model on the PC platform. The CPU was Pentium mobile 1.7 GHz CPU with 512 Mb RAM and the operating system was Windows XP. The reference JM9.4 decoder was optimized by the Intel MMX technology. We selected Foreman and Mobile CIF sequences as training sequences and pre-encoded 40 training bit streams. Each bit stream file contained 270 frames and was encoded by different quantization parameters (QPs), *i.e.*, QP=2, 4, 6, ...40. The Intel Vtune performance analyzer 8.0 was used to measure the CAVLC/UVLC decoding complexities for all pre-encoded bit streams. The numbers of clock-ticks measured by the Intel Vtune were divided by $1.7 \cdot 10^7$ to get CAVLC and UVLC decoding time in milli-seconds. Then, the proposed CAVLC/UVLC decoding complexity model counted the numbers of all decoding complexity coefficients for those pre-encoded bit streams.

The above information was used to train the weights for CAVLC decoding complexity model, *i.e.*, ω_{mb} , ω_{cavlc} , ω_{qtc} , ω_{one} , and ω_{run} , and those for UVLC decoding complexity model, *i.e.*, $\omega_{skipped}$, ω_{intra} , ω_{mv} , and ω_{ref} . The constrained least square method was used to determine all weights. The weights for the CAVLC decoding complexity model are:

$$\begin{aligned} \omega_{mb} &= 6.566 \cdot 10^{-5} & \omega_{cavlc} &= 6.327 \cdot 10^{-5} \\ \omega_{qtc} &= 3.476 \cdot 10^{-5} & \omega_{one} &= 2.867 \cdot 10^{-5} \\ \omega_{run} &= 1.644 \cdot 10^{-5} \end{aligned} \quad (5)$$

The weights for the UVLC decoding complexity model are:

$$\begin{aligned} \omega_{skipped} &= 1.154 \cdot 10^{-4} & \omega_{intra} &= 3.22 \cdot 10^{-5} \\ \omega_{mv} &= 8.15 \cdot 10^{-5} & \omega_{ref} &= 2.82 \cdot 10^{-4} \end{aligned} \quad (6)$$

Next, these weights were used in the decoding complexity model to estimate the CAVLC and UVLC decoding complexities of the following four HD (1920x1080) bit streams: Blue Sky, Toy and Calendar, Sunflower, and Rush Hour. Figures 2 and 3 give the comparisons between the estimated decoding complexities based on the proposed complexity model and the actual decoding complexities measured by the Intel Vtune for CAVLC and UVLC, respectively. We see that the proposed complexity model provides good estimation results for these test bit streams. The errors are within 7%.

IV. APPLICATIONS OF THE PROPOSED DECODING COMPLEXITY MODEL

The application of the proposed decoding complexity model is discussed in this section. A more thorough treatment will be given in our future work. Consider the following two scenarios. First, an H.264/AVC encoder generates a single bit stream for different decoding platforms without any decoding complexity model. Second, the encoder generates several bit

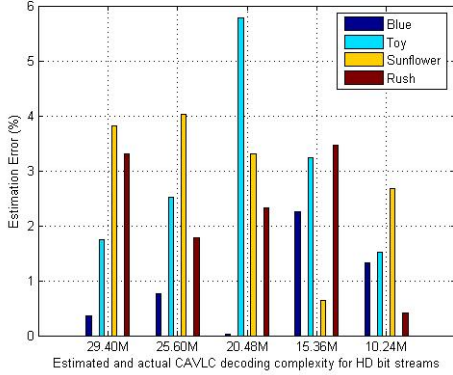


Fig. 2. Comparison between actual and estimated CAVLC decoding complexities for Blue Sky, Toy and Calendar, Sunflower, and Rush Hour bit streams

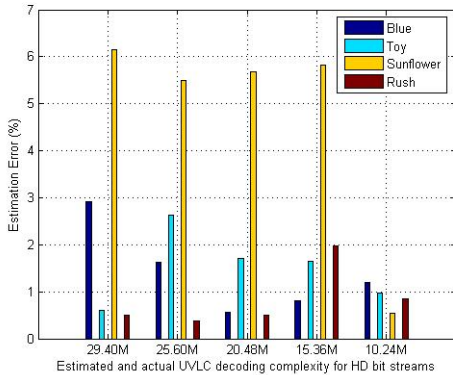


Fig. 3. Comparison between actual and estimated UVLC decoding complexities for Blue Sky, Toy and Calendar, Sunflower, and Rush Hour bit streams

streams for different decoding platforms separately according to their computational power so that the resultant bit stream is easy to decode for a particular platform. For the latter case, the decoding complexity models must be incorporated in the H.264/AVC encoder so that the encoder can estimate the possible decoding complexity and then generate the decoder-friendly bit streams.

The rate-distortion optimization (RDO) process in the conventional H.264/AVC encoder is used to decide the best inter or intra prediction mode, which consists of the following steps. First, since different inter prediction modes have a different number of MVs, the RDO process performs the motion estimation (ME) to find the best MV if the current MB is to be coded by inter prediction modes. On the other hand, the RDO process finds the best intra prediction direction if intra prediction modes are adopted for this MB. Second, the RDO process performs actual encoding (e.g. the spatial domain transform, quantization and entropy encoding) and decoding tasks to get the reconstructed video frame so as to determine the associated bit rate and distortion. Then, the RDO process

evaluates the RD cost function given by

$$J_{rd}(blk_i|QP, m) = D(blk_i|QP, m) + \lambda_m \cdot R(blk_i|QP, m), \quad (7)$$

where $D(blk_i|QP, m)$ and $R(blk_i|QP, m)$ are the distortion and the bit rate of block blk_i for a given coding mode m and quantization parameter (QP), respectively. Finally, the RDO process finds the best mode that yields the minimal RD cost. The minimization of the RD cost function in (7) implies that the RDO process decides the best mode that minimizes the distortion $D(blk_i|QP, m)$ while meeting the rate constraint (i.e., $R(blk_i|QP, m) \leq R_{st,i}$). Note that the Lagrangian multiplier, λ_m , in (7) is used to control the bit rate. Thus, it depends on QP.

The original RD optimization problem can be extended to consider the joint problem of RD and decoding complexity optimization (RDC). In other words, not only the rate constraint but also the decoding complexity constraint are considered to minimize the distortion in the RDO process. We can introduce the decoding complexity cost into the original RD cost function (7) via

$$J_{rdc}(blk_i|QP, m) = D(blk_i|QP, m) + \lambda_m \cdot R(blk_i|QP, m) + \lambda_c \cdot C(blk_i|QP, m), \quad (8)$$

where $C(blk_i|QP, m)$ is the decoding complexity of block blk_i for a given coding mode m and QP. Being similar to λ_m for rate control, the Lagrangian multiplier, λ_c , is used to control the decoding complexity. The algorithm to select proper λ_c for a given CABAC decoding complexity constraint (i.e., $C(blk_i|QP, m) \leq C_{st,i}$) was proposed in [3]. In our future work, this algorithm will be extended so as to select proper λ_c for the CAVLC/UVLC decoding complexity control.

V. CONCLUSION

The CAVLC and UVLC decoding complexity models were presented. Since CAVLC and UVLC are used to encode source and header data, respectively, the proposed decoding complexity model consists of source and header parts. Both decoding complexity models were verified experimentally. Possible applications of this model were also described. The H.264 encoder integrated with the proposed complexity model can select proper coding modes to minimize the distortion while satisfying the rate and decoding complexity constraints.

REFERENCES

- [1] G. Shen, G. P. Gao, S. Li, H. Y. Shum, and Y. Q. Zhang. Accelerate video decoding with generic GPU. *IEEE Trans. on Circuits and Systems for Video Technology*, 5:685–693, May 2005.
- [2] Y. Andreopoulos and M. van der Schaar. Complexity-constrained video bitstream shaping. *IEEE Trans. on Signal Processing*, 55:1967–1974, May 2007.
- [3] S. W. Lee and C.-C. Jay Kuo. Complexity modeling for context-based adaptive binary arithmetic coding (CABAC) in H.264/AVC decoder. In *Conference on Applications of Digital Image Processing, SPIE Optics and Photonics (SPIE 2007)*, Aug. 2007.
- [4] G. Bjontegarrd and K. Lillevold. Context-adaptive VLC coding of coefficients. In *JVT document*, volume JVT-C028, May 2002.
- [5] Iain E. G. Richardson. H.264 and MPEG-4 video compression - video coding for next generation multimedia. In *John Wiley and Sons*, pages 198–207, 2003.