JOINT ALGORITHM-ARCHITECTURE OPTIMIZATION OF CABAC TO INCREASE SPEED AND REDUCE AREA COST

Vivienne Sze, Anantha P. Chandrakasan
Massachusetts Institute of Technology

ABSTRACT

To address the increasing demand for higher resolution and frame rates, processing speed (i.e. performance) and area cost need to be considered in the development of next generation video coding. Accordingly, both algorithm and architecture should be taken into account during video codec design. This paper proposes joint optimization of both the algorithm and architecture to ensure that high coding efficiency can be achieved in conjunction with high processing speed and low area cost. Specifically, it presents two optimizations that can be performed on Context-based Adaptive Binary Arithmetic Coding (CABAC), a form of entropy coding in H.264/AVC. First, subinterval reordering is proposed for the arithmetic decoder to increase the processing speed by 14 to 22% with no cost to coding efficiency. Second, modification of the *motion vector difference (mvd)* context selection is proposed to reduce memory requirements (i.e. area cost) by 50% with negligible coding efficiency impact ($\leq$0.02%). These joint algorithm and architecture optimizations are non-standard compliant and thus are well suited to be used in High Efficiency Video Coding (HEVC), the successor to H.264/AVC.

Index Terms— Arithmetic Coding, Video Coding, Architecture

1. INTRODUCTION

Traditionally, the focus of video coding development has been primarily on improving coding efficiency. However, as processing speed requirements and area cost continue to rise due to growing resolution and frame rate demands, it is important to address the architecture implications of the video coding algorithms. In this paper, we will show that modifications to video coding algorithms can provide speed up and reduce area cost with little to no effect on the coding efficiency. An increase in processing speed (i.e. performance) can also translate into reduced power consumption using voltage scaling, which is important given the number of video codecs that reside on battery operated devices. The approach of jointly optimizing both architecture and algorithm is demonstrated on Context-based Adaptive Binary Arithmetic Coding (CABAC), a form of entropy coding in H.264/AVC, which is a known performance bottleneck in the video codec, particularly the decoder. These optimizations render the algorithm non-standard compliant and thus are well suited to be used in the next generation video coding standard High Efficiency Video Coding (HEVC), the successor to H.264/AVC. CABAC has been adopted into the HEVC test model [1].

Funding for this research was provided by Texas Instruments and an NSERC fellowship. The authors would like to thank Madhukar Budagavi for valuable feedback and discussions. The authors are with the Microsystems Technology Laboratories, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: sze@alum.mit.edu; anantha@mtl.mit.edu)

2. OVERVIEW OF CABAC

Entropy coding delivers lossless compression at the last stage of video encoding (and first stage of video decoding), after the video has been reduced to a series of syntax elements. Arithmetic coding is a type of entropy coding that can achieve compression close to the entropy of a sequence by effectively mapping the symbols (i.e. syntax elements) to codewords with non-integer number of bits. In H.264/AVC, the CABAC provides a 9 to 14% improvement over the Huffman-based Context-based Adaptive Variable Length Coding (CAVLC) [2].

CABAC involves three main functions: binarization, context modeling and arithmetic coding. Binarization maps syntax element to binary symbols (bins). Context modeling estimates the probability of the bins and arithmetic coding compresses the bins. This paper focuses on reducing the critical path delay of the arithmetic coding engine as well as reducing the area cost of context modeling.

2.1. Arithmetic Coding

Arithmetic coding is based on recursive interval division. Binary arithmetic coding refers to the case where the alphabet of the symbol is restricted to zero and one (i.e. binary symbols (bins)). The size of the subintervals are determined by multiplying the current interval by the probabilities of the bin. At the encoder, a subinterval is selected based on the value of the bin. The range and lower bound of the interval are updated after every selection. At the decoder, the value of the bin depends on the location of the offset. The offset is a binary fraction described by the encoded bits received at the decoder. The range and lower bound of the current interval have limited bit-precision, so renormalization is required whenever the range falls below a certain value to prevent underflow. Fig. 1 shows a flowchart of the arithmetic decoder.

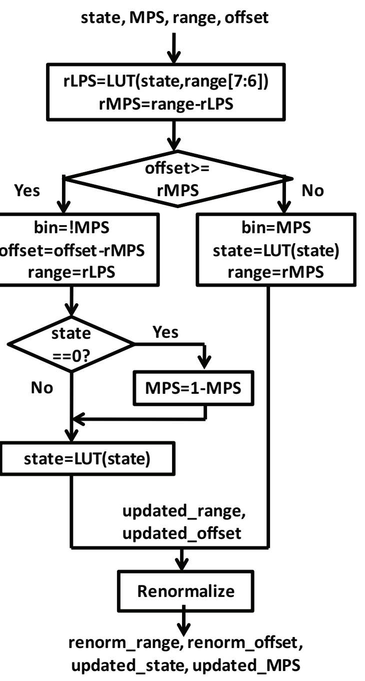


Fig. 1: Flowchart of binary arithmetic decoder in H.264/AVC CABAC [4].

The CABAC engine used in H.264/AVC leverages a modulo coder (M coder) to calculate the range of a subinterval based on the product of the current range and the probability of the bin. The M coder involves using a look up table (LUT) rather than a true multiplier to reduce implementation complexity [3].

The arithmetic coding engine typically contains the critical path in H.264/AVC CABAC. Section 3.1 will discuss various optimizations that can be used to reduce the delay of this critical path.

2.2. Context Modeling

In order to achieve optimal compression efficiency, an accurate probability must be used to code each bin. For High profile in H.264/AVC, CABAC uses over 400 different probability models to achieve the significant coding gains over CAVLC. All bins of the same type (i.e. with the same probability distribution and characteristics) are grouped together in a context and use the same model. Accordingly, the context of a bin dictates the probability with which it is coded.

Since distributions in neighboring macroblocks are correlated, the value of the syntax elements of the macroblocks (or blocks) located to the top and left impact the context selection. For instance, bins of *motion vector difference (mvd)* syntax elements that have neighbors with large *mvds*, use the same context. Using information from the top and left neighbor requires additional memory. Section 3.2 will describes how to reduce the memory size and consequently the area cost of context modeling.

3. JOINT ALGORITHM-ARCHITECTURE OPTIMIZATIONS

Two joint algorithm and architecture optimizations are presented which increase the speed and reduce the area cost of CABAC. The optimizations were implemented on the H.264/AVC reference software [5] and HEVC test model [6] to measure their coding efficiency impact under common conditions specified by the Video Coding Experts Group (VCEG) [7] and Joint Collaborative Team on Video Coding (JCT-VC) [8] standards bodies, respectively.

3.1. Subinterval Reordering to increase speed

In the arithmetic decoder of H.264/AVC CABAC, the interval is divided into two subintervals based on the probabilities of the least probable symbol (LPS) and most probable symbol (MPS). The range of the MPS subinterval (rMPS) is compared to the offset to determine whether the bin is MPS or LPS. rMPS is computed by first obtaining range of the LPS subinterval (rLPS) from a 64x4 LUT (using bits [7:6] of the current 9-bit range and the 6-bit probability state from the context) and then subtracting it from the current range. Depending on whether an LPS or MPS is decoded, the range is updated with their respective subintervals. To summarize, the interval division steps in the arithmetic decoder are

1. obtain rLPS from the 64x4 LUT
2. compute rMPS by subtracting rLPS from current range
3. compare rMPS with offset for bin decoding decision
4. update range based on bin decision.

If the offset was compared to rLPS rather than rMPS, then the comparison and subtraction to compute rMPS can occur at the same time. Furthermore, the updated offset is computed by subtracting rLPS from offset rather than rMPS. Since rLPS is available before rMPS, this subtraction can also be done in parallel with range-offset comparison. Fig. 2 shows the difference between the subinterval order of H.246/AVC CABAC and subinterval reordering. The two orderings of the subintervals are mathematically equivalent in arithmetic coding; thus changing the order has no impact on coding efficiency. This was verified with simulations of the modified reference software and test model under common conditions [7, 8]. An arithmetic decoder was implemented in RTL for each subinterval ordering. Both implementations were synthesized to obtain their area-delay trade-off in a 45-nm CMOS process. For the same area, subinterval reordering reduces the critical path delay by 14 to 22%.

Subinterval reordering has similar benefits for the arithmetic *encoder* of H.264/AVC CABAC. Rather than comparing offset to rLPS or rMPS, the bin to be encoded is compared to MPS. Depending on whether the bin equals MPS, the range

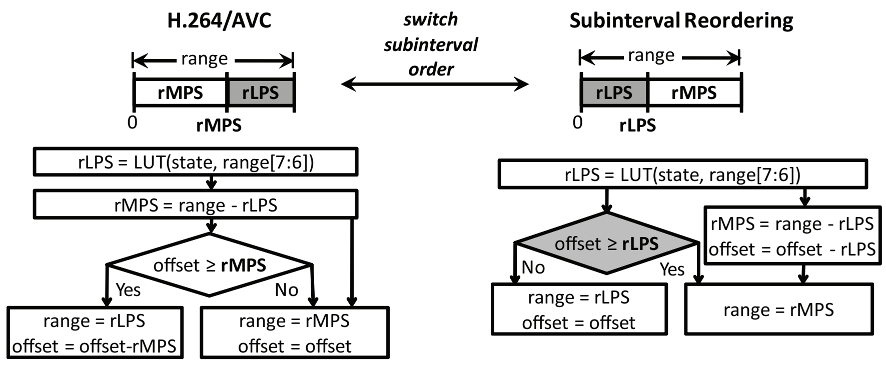


Fig. 2: Impact of subinterval reordering for CABAC *decoding*.

is updated accordingly. Reversing the order of subintervals allows the bin-MPS comparison to occur in parallel with the rMPS subtraction in the CABAC encoder as shown in Fig. 3. The lower bound can also be updated earlier since it depends on rLPS rather than rMPS.

3.2. Modified *mvd* Context Selection to reduce area cost

To make use of the spatial correlation of neighboring data, context selection can depend on the values of the top and left blocks as shown in Fig. 4. Consequently, a line buffer is required in the CABAC engine to store information pertaining to the previously decoded row. The depth of this buffer depends on the width of the frame being decoded which can be quite large for high resolution (e.g. 4kx2k) sequences. The bit-width of the buffer depends on the type of information that needs to be stored per block or macroblock in the previous row. We propose reducing the bit-width of this data to reduce the overall line buffer size of the CABAC.

Specifically, we propose modifying the context selection for *motion vector difference (mvd)*. *mvd* is used to reduce the number of bits required to represent motion information. Rather than transmitting the motion vector, the motion vector is predicted from its neighboring 4x4 blocks and only the difference between motion vector prediction (mvp) and motion vector (mv), referred to as *mvd*, is transmitted.

$$mvd = \text{mv} - \text{mvp}$$

A separate *mvd* is transmitted for the vertical and horizontal components. The context selection of mvd depends on neighbors A and B as shown in Fig. 4.

In H.264/AVC, neighboring information is incorporated into the context selection by adding a context index increment (between 0 to 2 for *mvd*) to the calculation of the context index. The *mvd* context index increment, χ_{mvd}, is computed in two steps [2]:

Step 1: Sum the absolute value of neighboring *mvds*

$$e(A,B,cmp)=|mvd(A,cmp)|+|mvd(B,cmp)|$$

where *A* and *B* represent the left and top neighbor and *cmp* indicates whether it is a vertical or horizontal component.

Step 2: Compare e(A,B,cmp) to thresholds of 3 and 32

$$\chi_{mvd}(\text{cmp}) = \begin{cases} 0, & \text{if e(A,B,cmp)}<3 \\ 1, & \text{if } 3\leq\text{e(A,B,cmp)}\leq 32 \\ 2, & \text{if e(A,B,cmp)}>32 \end{cases}$$

Fig. 5a illustrates how the above equation maps the *mvd* of A and B to different χ_{mvd}. In a given slice, all blocks surrounded by large *mvds* will use the same probability model (χ_{mvd}=2). Blocks surrounded by small *mvds* will use another probability model (χ_{mvd}=0 or χ_{mvd}=1).

With the upper threshold set to 32, a minimum of 6-bits of the *mvd* has to be stored per component per 4x4 block in the line buffer. For 4kx2k, there are $(4096/4)$ =1024 4x4 blocks per row, which implies $6 \times 2 \times 1024 =$ 12,228 bits are required for *mvd* storage.

To reduce the memory size, rather than summing the components and then comparing to a threshold, we propose separately comparing each component to a threshold and summing their results. In other words,

Step 1: Compare the components of *mvd* to a threshold

$$\text{thresh}_A(\text{cmp})=|mvd(\text{A,cmp})| > 16$$
$$\text{thresh}_B(\text{cmp})=|mvd(\text{B,cmp})| > 16$$

Step 2: Sum the results thresh$_A$ and thresh$_B$ from Step 1

$$\chi_{mvd}(\text{cmp})=\text{thresh}_A(\text{cmp})+\text{thresh}_B(\text{cmp})$$

Fig. 5b illustrates how the above equation maps the *mvd* of A and B to different χ_{mvd}. A single threshold of 16 is used. Consequently, only a single bit is required to be stored per component per 4x4 block; the size of the line buffer for *mvd* is reduced to $1 \times 2 \times 1024$ =2,048 bits. In H.264/AVC, the overall line buffer size of the CABAC required for all syntax elements is 20,480 bits. The modified *mvd* context selection reduces the memory size by 50%, from 20,480 bits to 10,240 bits. The average coding penalty of this approach, measured using BD-rate [9], was verified across common conditions to be 0.02% for H.264/AVC and 0% in the HEVC test model (Table 1).

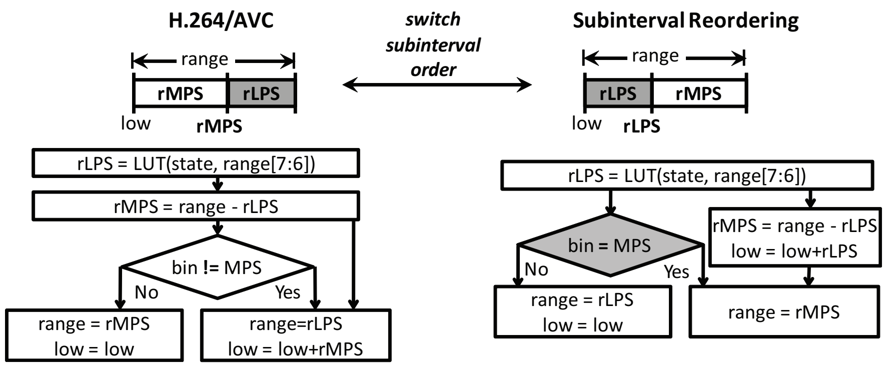


Fig. 3: Impact of subinterval reordering for CABAC *encoding*.

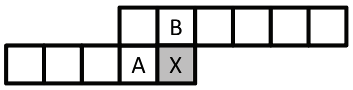
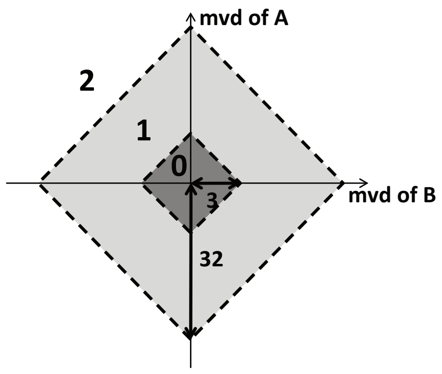
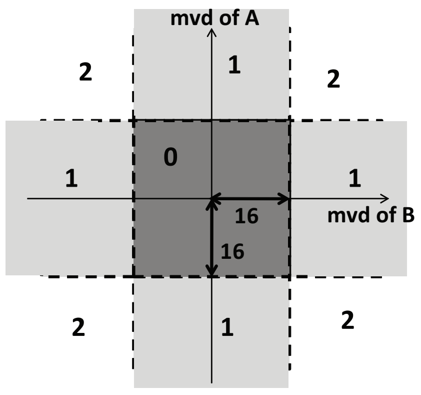


Fig. 4: For position X, context selection is dependent on A and B (4x4 blocks for *mvd*); a line buffer required to store the previous row of decoded data.



(a) H.264/AVC CABAC



(b) 1-bit per component per 4x4.

Fig. 5: Context increments χ_{mvd} for different *mvd* in top (A) and left (B) neighboring 4x4 blocks.

Table 1: Coding penalty due to modification of *mvd* context selection. Measured using Bjontegaard ΔBitrate against H.264/AVC (JM12.0) and HEVC test model (HM2.0). Results are averaged across sequences defined in the common conditions.

(%)	Intra	Low Delay	Random Access
JM12.0	0.00	0.02	0.02
HM2.0	0.00	0.00	0.00

4. CONCLUSION

This paper presents two joint algorithm and architecture optimizations of the CABAC engine that increase processing speed by 14 to 22% and reduce memory size by 50% with negligible coding efficiency impact ($\leq$0.02%). It demonstrates the benefits of accounting for implementation cost when designing video coding algorithms. We recommend that this approach be extended to the rest of the video codec to maximize processing speed and minimize area cost, while delivering high coding efficiency in the next generation video coding standard.

5. REFERENCES

[1] T. K. Tan, G. Sullivan, and J.-R. Ohm, "JCTVC-C405: Summary of HEVC working draft 1 and HEVC test model (HM)," Oct. 2010.

[2] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. on CSVT*, vol. 13, no. 7, pp. 620– 636, July 2003.

[3] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," in *IEEE Inter. Conf. on Image Processing,*, Sept. 2003, vol. 2, pp. II – 263–6 vol.3.

[4] "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," Tech. Rep., ITU-T, 2003.

[5] "H.264/AVC Reference Software, JM 12.0," http://iphome.hhi.de/suehring/tml/.

[6] "HEVC Test Model, HM 2.0," https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-2.0/.

[7] T.K. Tan, G. Sullivan, and T. Wedi, "VCEG-AE010: Recommended Simulation Common Conditions for Coding Efficiency Experiments Rev. 1," Jan. 2007.

[8] F. Bossen, "JCTVC-D600: Common test conditions and software reference configurations," Jan. 2011.

[9] G. Bjøntegaard, "VCEG-M33: Calculation of Average PSNR Differences between RD curves," April 2001.