PARALLEL CABAC FOR LOW POWER VIDEO CODING

Vivienne Sze
Anantha P. Chandrakasan
Massachusetts Institute of Technology
Cambridge, MA

Madhukar Budagavi
Minhua Zhou
Texas Instruments
Dallas, TX

ABSTRACT

With the growing presence of high definition video content on battery-operated handheld devices such as camera phones, digital still cameras, digital camcorders, and personal media players, it is becoming ever more important that video compression be power efficient. A popular form of entropy coding called Context-Based Adaptive Binary Arithmetic Coding (CABAC) provides high coding efficiency but has limited throughput. This can lead to high operating frequencies resulting in high power dissipation. This paper presents a novel parallel CABAC scheme which enables a throughput increase of N-fold (depending on the degree parallelism), reducing the frequency requirement and expected power consumption of the coding engine. Experiments show that this new scheme (with N=2) can deliver ~2x throughput improvement at a cost of 0.76% average increase in bit-rate or equivalently a decrease in average PSNR of 0.025dB on five 720p resolution video clips when compared with H.264/AVC.

Index Terms— video coding, arithmetic coding, parallel processing, CABAC.

1. INTRODUCTION

The use of video is becoming ever more pervasive on battery-operated handheld devices such as camera phones, digital still cameras, personal media players, etc. Annual shipment of such devices already exceeds a hundred million units and continues to grow. As a result, it is increasingly important that video compression become power efficient since the battery life is limited by the size, weight and cost of the portable devices. Voltage scaling is an effective technique to reduce power and energy (Equation 1).

$$Power \propto Capacitance \times (Voltage)^2 \times Frequency \quad (1)$$

The tradeoff is that at lower voltages circuits become slower, and the maximum operating frequency is reduced. This is an issue for applications such as real-time video coding where a frame must be processed by a given deadline. Specifically, the codec must operate at a frequency that allows it to meet its performance requirement (based on resolution, frame rate and bit-rate). It is important that the codec be clocked at a frequency high enough to meet not just the average workload but the worst case workload. In the near future, portable video devices are expected to support high-definition (HD) resolution video coding, which requires high performance codecs and consequently high operating frequencies (potentially in the GHz range). In addition to consuming high power, such high frequency circuits are difficult to design and implement. The frequency can be lowered by performing operations in parallel to maintain the same overall performance.

This work focuses on the arithmetic coding (AC) engine found in many compression algorithms due to its high compression efficiency. For instance, in H.264/AVC [1], the Context-Based Adaptive Binary Arithmetic Coding (CABAC) provides a 9-14% improvement over the Huffman-based Context-Adaptive Variable Length Coding [2]. Arithmetic coding is used for a wide variety of applications, and is found in standards such as H.264/AVC, H.263 and China AVS for video; JPEG-2000 and JPEG-LS for image; and MPEG-4 SNHC for 3D animation.

While arithmetic coding provides significant improvement in compression efficiency, its main drawback is its limited throughput (symbols/cycle). Arithmetic coding is inherently serial due to strong data dependencies, and typically only a single symbol is coded at a time. Consequently, the AC engine is often the bottleneck in the codec, requiring high frequency (cycles/sec) to achieve the desired performance (symbols/sec). For HD video coding, symbols need to be coded at very high rates, which further increase the frequency requirement. Operating at high frequencies limits our ability to voltage scale and, as shown in Equation 1, results in significant power consumption which is undesirable for battery operated devices.

In this paper, we present a parallelized form of the CABAC engine that improves throughput and lowers frequency at both the encoder and decoder without sacrificing coding efficiency. In single clock domain designs, lowering the frequency of the CABAC engine reduces the power of the entire codec, making it an attractive technique to be used in today's high performance energy-constrained environment. As such, this approach addresses the desired features in future coding standards such as low power consumption and enhanced parallelism support [3].

2. CONTEXT-BASED ADAPTIVE BINARY ARITHMETIC CODING (CABAC)

Binary arithmetic coding is based on recursive interval subdivision. The sizes of the subintervals are determined by multiplying the current interval by the probability of the binary symbol ('bin'). At the encoder, a subinterval is selected based whether the bin is a least probable symbol (LPS) or most probable symbol (MPS). At the decoder, the value of the bin (LPS/MPS) depends on which subinterval the offset is located. The range of the current interval has a limited bit-precision, so renormalization is required whenever the range falls below a certain value to prevent underflow.

In order to achieve optimal compression efficiency the correct probabilities must be used to code each bin. All bins of the same type, with the same probability distribution and characteristics, are grouped together and use the same model known as a *context*. Accordingly, the context of a bin dictates the probability with which it is coded (Table 1). Note that context switching can occur at every bin. The probabilities used for each context must be accurately modeled; this process of determining the probability of a bin is called source modeling. Since the bins have non-stationary distributions, the probabilities are continuously updated by the context modeler making the engine *adaptive*.

Bin	Probability
0	$Pr_{context}(0)$
1	$Pr_{context}(1) = (1-Pr_{context}(0))$

Table 1. Probability table for binary arithmetic coding.

3. THROUGHPUT OF H.264/AVC CABAC

The performance requirement of the arithmetic coding engine, and thus operating frequency, is dictated by the rate of the bins that need to be encoded and decoded, and not the bits of the compressed data. For high definition (Level 3.1 to 4.2) in H.264/AVC, the maximum bin rate, averaged across a coded picture, ranges from 121 Mbins/s up to 1.12 Gbins/s [1]. Depending on the architecture of the system, the instantaneous bin rate could be even higher. For real-time encoding and decoding, the CABAC engine must meet this performance.

H.264/AVC CABAC [2] is done serially due to inherent data dependency stemming from the fact that the context can change for every bin. In many cases the context of the next bin is not known until the current bin is decoded. This serial nature limits the throughput. We could reduce the serial dependence by reducing the amount of context switches. However, context adaptability is responsible for high coding efficiency. For instance, we found that if we do not adapt the context for the first bin of `coeff_abs_level_minus1` in H.264/AVC we get a 5% penalty. To account for possible context switching at every bin, the H.264/AVC CABAC operates serially, with a 1 bin/cycle throughput, requiring very high operating frequencies to meet the given performance requirements stated above.

In the next section, we propose a method that enables us to process multiple bins per cycle while accounting for changing contexts to maintain coding efficiency.

4. PARALLEL CABAC

This work proposes a scheme (Fig. 1) that can perform CABAC on N-bins (symbols) per cycle. At the encoder, the context modeler keeps track of the probabilities of each bin type and a probability table is constructed for each group of bins to be encoded. This table is passed to the encoding engine along with the bins to generate an encoded bitstream. At the decoder, based on the previously decoded bins, the context modeler and decision tree determine the possible contexts of the next set of bins and generate a probability table which is passed to the decoding engine. The next two sections describe the encoder and decoder in detail.

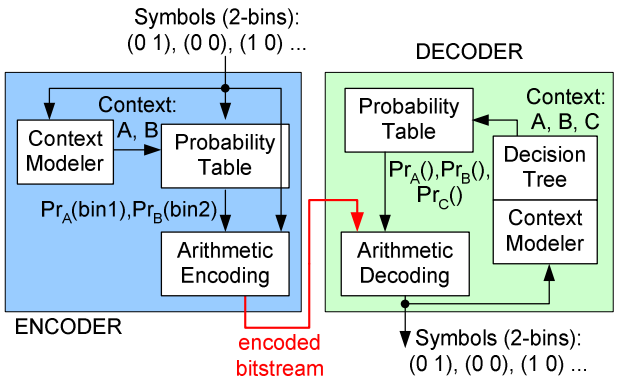

Fig. 1. Block diagram of parallel CABAC (N=2).

4.1. Encoder

At the encoder, since the sequence of bins to be compressed are known a priori, the contexts to be used for each bin are also known and multiple (N) bins can be encoded in parallel. Rather than using the Table 1 to encode a single bin at time (N=1), if we know the next two bins belong to context A and B, we can encode N=2 bins at the same time by constructing Table 2. Effectively, by knowing which probabilities to use for both bins, the alphabet can be expanded from two symbols to four.

1st Bin	2nd Bin	Probability
0	0	$Pr_A(0)\times Pr_B(0)$
0	1	$Pr_A(0)\times Pr_B(1)$
1	0	$Pr_A(1)\times Pr_B(0)$
1	1	$Pr_A(1)\times Pr_B(1)$

Table 2. Probability table of 2-bin encoder in Fig. 1.

4.2. Decoder

The main challenge of parallel CABAC occurs at the decoder. Typically the context to be used on a bin depends on the *value* of the previous bin. Thus, to decode two bins in parallel, the context for the second bin is unknown until the first bin has been decoded. Consequently, it is a significant challenge to decode the second bin at the same time as the first since it is necessary to know which context to use in order to correctly decode the compressed data.

Other techniques for parallel decoding either don't account for the varying context for each bin [4]; utilize a predictive scheme to guess the context of the second bin, which would still have limited throughput in the worst case [5]; or concatenates two single bin engines and use pipelining to increase throughput [6]; however there has yet to be an approach that is fundamentally parallel that can deterministically decode several (N) bins with different contexts at the same time.

The algorithm proposed in this work achieves deterministic decoding that accounts for the different contexts through the use of *conditional* probabilities. At any given time, there are only two possible outcomes for the first bin (one or zero) and thus there are only two contexts that could be used for the second bin based on the value of the first bin; let's assume they are contexts B and C respectively (Fig. 2). B and C are determined using a decision tree (Fig. 1). Table 3 can be used to decode two bins simultaneously, with

$$Pr_B(2^{nd}\ Bin) = Pr\ (2^{nd}\ Bin \mid 1^{st}\ Bin = 0)$$
$$Pr_C(2^{nd}\ Bin) = Pr\ (2^{nd}\ Bin \mid 1^{st}\ Bin = 1).$$

1st Bin	2nd Bin	Probability
0	0	$Pr_A(0) \times Pr_B(0)$
0	1	$Pr_A(0) \times Pr_B(1)$
1	0	$Pr_A(1) \times Pr_C(0)$
1	1	$Pr_A(1) \times Pr_C(1)$

Table 3. Probability table of 2-bin decoder in Fig. 1.

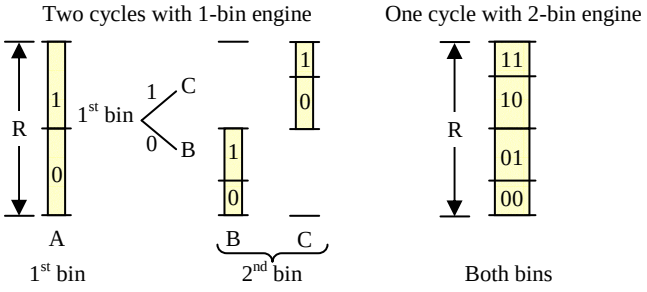

Fig. 2. Comparison of 1-bin and 2-bin (parallel) decode.

4.3. Further Modifications

Several additional modifications were made to the CABAC engine to facilitate the 2-bins/cycle operation. First, binary arithmetic coding is implemented using multipliers rather than the look up tables used in H.264/AVC. The probability for each bin is quantized to 6-bits, and the probabilities are multiplied by 2-bits of the range. Second, renormalization occurs only every two bins, which requires the size of the range to be increased from 9-bit to 14-bits in order to prevent underflow errors.

Finally, in the cases where the number of bins is odd, a dummy bin is inserted. For better coding efficiency, the dummy bin is coded with a fixed skewed probability of 0.015625, which is the smallest value based on the 6-bit quantization of the probabilities. This N-bins/cycle approach can be applied to all bins or it can be selectively applied to certain bins that dominate the workload of the engine. Dummy bins allow certain syntax elements to be coded at 2-bins/cycle and other elements at 1-bin/cycle. For instance, if syntax element A is to be coded at 2-bin/cycle followed by syntax element B at 1-bins/cycle, and element A is comprised of an odd number of bins, a dummy bin is inserted before transitioning to element B .

4.4. Complexity Impact

Note that the total number of contexts remains the same regardless of the degree of parallelism. Contexts can be cached to enable concurrent updates of multiple contexts.

The parallel CABAC can be extended to N bins potentially giving an Nx increase in throughput, and reducing the required frequency by 1/N. The additional cost is an increase in the number of multiplications per bin and increased complexity in the decision tree. For N=1, there is one multiplication in the encoder and one in the decoder, resulting in 2 mult/bin. For N=2, there are 3 multiplications in the encoder and 4 in the decoder, resulting in 3.5 mult/bin. The multiplications can be performed in parallel, and do not increase the critical path of the engine. For software implementation, the algorithm can run on a vector-based processor. For hardware implementation, using parallel operations at a lower frequency and voltage reduces the power consumed by each multiplication. This counteracts the increase in number of multiplications (i.e. switched capacitance). However, since the number of multiplications grows exponentially with N, in practice N would remain around 2 or 3.

Furthermore, the additional comparisons are required since the offset could be located on one of four subintervals rather than two. Again, these comparisons can be done mostly in parallel at a lower frequency and voltage, each consuming less power than if they ran serially.

5. SIMULATION RESULTS

The 2-bins/cycle parallel CABAC was incorporated into the encoder and decoder of the JM 12.0 software. It was applied to the following groups of syntax elements:

```
significant_coeff_flag, last_significant_coeff_flag,
coeff_abs_level_minus1, coeff_sign_flag, mvd_l0_0,
mvd_l1_0, mvd_l0_1, mvd_l1_1, ref_idx_l0, ref_idx_l1,
mb_type, coded_block_pattern, mb_qp_delta,
prev_intra4x4_pred_mode_flag, rem_intra4x4_pred_mode,
prev_intra8x8_pred_mode_flag, rem_intra8x8_pred_mode
```

Experiments were performed using common conditions specified in [7]. Table 4 lists the Bjontegaard ΔBitrate and ΔPSNR when using 2-bins/cycle parallel CABAC (with all modifications in Section 4.3) versus the 1-bin/cycle CABAC in H.264/AVC. The QP values used to calculate the Bjontegaard ΔBitrate and ΔPSNR were 22, 27, 32, and 37.

Sequence	ΔBitrate (BD delta)	ΔPSNR (BD delta)
BigShips	1.29%	-0.039
City	1.27%	-0.041
Crew	1.35%	-0.037
Night	1.16%	-0.044
ShuttleStart	1.46%	-0.054
Average	**1.30%**	**-0.043**

Table 4. Rate-distortion performance of 2-bins/cycle parallel CABAC versus 1-bin/cycle H.264/AVC CABAC.

The average throughput improvement (i.e. cycle count reduction) across the 5 video sequences and different QP values is shown in Table 5. The throughput improvement is calculated for a macroblock (MB) with average and worst case number of bins. Since the average and worst case MB have different bin distributions, and only a subset of syntax elements are coded at 2-bin/cycle, they have different throughput improvements.

QP	Average Case MB	Worst Case MB
22	1.79×	1.98×
27	1.64×	1.97×
32	1.49×	1.98×
37	1.37×	1.97×

Table 5. Throughput improvement of 2-bins/cycle CABAC vs. 1-bin/cycle CABAC in H.264/AVC.

For the results shown in Table 4 and 5, the CABAC engine transitioned between 1-bin/cycle and 2-bin/cycle for the selected group of syntax elements and consequently dummy bins were inserted when certain syntax elements were binarized to an odd number of bins. However, if all elements were encoded at 2-bin/cycle, the decision tree would take transitions across these syntax elements into account to avoid insertion of dummy bins between elements which would reduce ΔBitrate and increase ΔPSNR. To remove the impact of the dummy bins in our measurement, we compare the 2-bins/cycle parallel CABAC with a 1-bin/cycle CABAC that also encodes dummy bins. Table 6 lists the Bjontegaard ΔBitrate and ΔPSNR excluding the impact of dummy bins for 2-bins/cycle parallel CABAC versus the 1-bin/cycle CABAC. Furthermore, by removing the dummy bins and applying the 2-bins/cycle engine to all syntax elements, the throughput for the average case in Table 5 can be increased to 2x.

Sequence	ΔBitrate (BD delta)	ΔPSNR (BD delta)
BigShips	0.67%	-0.020
City	0.73%	-0.024
Crew	0.80%	-0.022
Night	0.80%	-0.031
ShuttleStart	0.81%	-0.029
Average	**0.76%**	**-0.025**

Table 6. Rate-distortion performance of parallel CABAC engine excluding the impact of dummy bins.

6. CONCLUSION

A novel CABAC algorithm that can encode and decode any arbitrary number of bins in parallel with coding efficiency comparable to H.264/AVC was presented. Increasing the throughput of the engine lowers the required operating frequency, and enables voltage scaling to reduce the expected power consumption of video coding which is necessary for high performance mobile applications especially at HD resolutions. The parallel CABAC is a promising technique for future low power video coding.

7. REFERENCES

[1] "ITU-T Recommendation H.264," March 2005.

[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 Circuits and Systems for Video Tech.*, vol.13, no.7, pp.620-636, July 2003.

[3] Texas Instruments, Nokia, Polycom, Samsung AIT, Tandberg, "Desired features in future video coding standards", Document T05-SG16-C-0215, June 2007.

[4] J.-H. Lin and K. K. Parhi, "Parallelization of Context-Based Adaptive Binary Arithmetic Coders," *IEEE Trans. on Signal Processing*, vol. 54, no. 10, pp. 3702-3711, October 2006.

[5] C.-H. Kim and I.-C. Park, "Parallel Decoding of Context-Based Adaptive Binary Arithmetic Codes Based on Most Probably Symbol Prediction," *IEICE – Trans. on Information and Systems*, vol. E90-D, no. 2, pp. 609-612, February 2007.

[6] W. Yu and Y. He, "A High Performance CABAC Decoding Architecture," *IEEE Trans. on Consumer Electronics*, vol. 51, no.4, November 2005.

[7] TK Tan, G. Sullivan, and T. Wedi, "Recommended Simulation Common Conditions for Coding Efficiency Experiments Revision 1," ITU–T Standardization Sector, Document VCEG-AE010, January 2007.