A HIGH THROUGHPUT CABAC ALGORITHM USING SYNTAX ELEMENT PARTITIONING

Vivienne Sze, Anantha P. Chandrakasan

Massachusetts Institute of Technology

ABSTRACT

Enabling parallel processing is becoming increasingly necessary for video decoding as performance requirements continue to rise due to growing resolution and frame rate demands. It is important to address known bottlenecks in the video decoder such as entropy decoding, specifically the highly serial Context-based Adaptive Binary Arithmetic Coding (CABAC) algorithm. Concurrency must be enabled with minimal cost to coding efficiency, power, area and delay. This work proposes a new CABAC algorithm for the next generation standard in which binary symbols are grouped by syntax elements and assigned to different partitions which can be decoded in parallel. Furthermore, since the distribution of binary symbols changes with quantization, an adaptive binary symbol allocation scheme is proposed to maximize throughput. Application of this next generation CABAC algorithm on five 720p sequences shows a throughput increase of up to 3x can be achieved with negligible impact on coding efficiency (0.06% to 0.37%), which is a 2 to 4x reduction in coding penalty compared with H.264/AVC and entropy slices. Area cost is also reduced by 2x. This increased throughput can be traded-off for low power consumption in mobile applications.

Index Terms— Arithmetic Coding, Parallel Processing

1. INTRODUCTION

Arithmetic coding is a form of entropy coding found in many compression algorithms that has high coding efficiency since it can map symbols to non-integer length codewords. The coding efficiency depends strongly on the use of accurate probabilities. Accordingly, in H.264/AVC, context-adaptive coding is used such that different contexts (probability models) are applied for different syntax elements. Furthermore, since the probabilities are non-stationary, the contexts are also constantly updated. The adaptive context increases the serial dependencies in the entropy decoding. In H.264/AVC CABAC, the contexts are assigned on a binary symbol ('bin') basis which leads to data dependencies at the bin level and consequently requires bin level serial processing; this causes

a severe bottleneck particularly for high performance applications. Since the next generation standard is looking to support very high resolutions and frame rates (e.g. 4kx2k, 120 fps), there is a critical need for increased concurrency in CABAC to achieve higher throughput.

Video is becoming increasingly prevalent on handheld battery operated devices. Voltage scaling reduces energy consumption by a quadratic factor at the cost of slower circuits. Parallelism can be used to lower the clock rate and compensate for the speed reduction. This approach has shown to be extremely effective in reducing the power consumption of video decoding [1]. Higher throughput allows for greater voltage scaling and consequently energy savings. Increasing concurrency in CABAC can enable significant power reduction for video codecs used in mobile applications.

2. THROUGHPUT REQUIREMENTS

It is important to understand the throughput requirements for real-timing decoding applications such as video conferencing. To achieve real-time low-delay decoding, the processing deadline is dictated by the time required to decode each frame to achieve a certain frames per second (fps) performance. The throughput of CABAC is dictated by the bin-rate rather than bit-rate. Table 1 shows the peak bin-rate requirements for a frame to be decoded instantaneously based on the specifications of the H.264/AVC standard [2]. They are calculated by multiplying the maximum number of bins per frame by the frame rate for the largest frame size. For Level 5.1, the peak bin-rate is in the Gbins/s; without concurrency, decoding 1 bin/cycle requires multi-GHz frequencies, which leads to high power consumption and is difficult to achieve even in an ASIC. Existing H.264/AVC CABAC hardware implementations such as [3] only go up to 149 MHz; the maximum frequency is limited by the critical path, and thus parallelism is necessary to meet next generation performance requirements.

Table 1: Peak bin-rate requirements for real-time decoding of worst case frame at various high definition levels.

Level	Max Frame Rate	Max Bins per picture	Max Bit Rate	Peak Bin Rate
	fps	*Mbins*	*Mbits/sec*	*Mbins/sec*
4.0	30	9.2	25	275
5.1	26.7	17.6	300	2107

3. EXISTING PARALLEL APPROACHES

There are several methods of either reducing the peak throughput requirement or increasing the throughput of CABAC; however, they come at the cost of decreased coding efficiency, increased power consumption and/or increased delay.

3.1. Frame Level Workload Averaging

Averaging the workload of several frames (i.e. on the order of the rate control buffer size, 30+) can decrease the peak bin-rate requirements to be within the range of the average maximum bit-rate. However, buffering results in increased delay and storage costs. For low-delay applications such as video conferencing, an additional delay of several frames may not be tolerated.

3.2. Bin Level Parallel Processing

Due to the strong data dependencies from bin to bin, speculative computation is required for parallelism at the bin level. This approach has been proposed in papers for both H.264/AVC compliant [3, 4] and non-compliant [5, 6] high throughput CABAC solutions. Speculation requires additional computations which may increase power consumption. Furthermore, the critical path increases with each additional bin, since all computations cannot be done entirely in parallel (e.g. each bin needs to wait for 'codIRangeLPS' from the previous bin). This reduces the overall throughput improvement that can be achieved. The reported bin-rates for these approaches are in the low hundreds of Mbins/s. Additional parallelism is needed to reach the Gbins/s required for 4kx2k.

3.3. Frame and/or Slice Level Parallel Processing

Parallelism can be applied at the slice level since CABAC parameters such as range, offset and context states are reset every slice. Each frame has a minimum of one slice, so at the very least parallelism can be achieved across several frames. However, frame level parallelism leads to increased latency and needs additional buffering, as inter-frame prediction prevents several frames from being fully decoded in parallel.

The storage and delay costs can be reduced if there are several slices per frame. However, increasing the number of slices per frame reduces the coding efficiency since it limits the number of macroblocks that can be used for prediction, reduces the training period for the probability estimation, and increases the number of slice headers and start code prefixes. Fig. 1 shows how the coding efficiency penalty increases with more H.264/AVC slices per frame.

3.4. Entropy Slices

Entropy slices have been proposed for the next generation standard in [7]. They are similar to H.264/AVC slices in that macroblocks are allocated to different slices. While entropy slices do not share info for entropy (de)coding (which enables parallel processing), motion vector reconstruction and
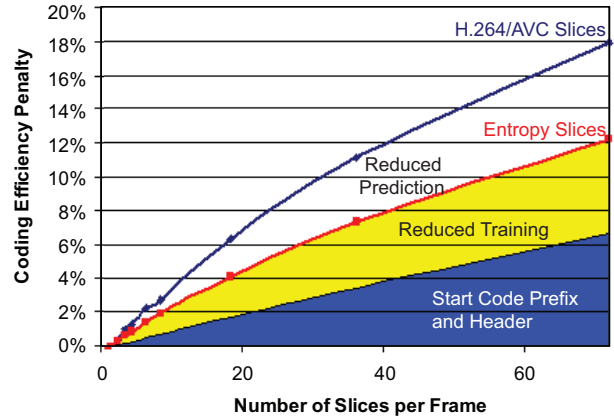

Fig. 1: Coding efficiency penalty vs. slices per frame. Sequence 'BigShips' (QP=27) under common conditions [8].

intra prediction are allowed across entropy slices, resulting in better coding efficiency than H.264/AVC slices (Fig. 1). However, entropy slices still suffer coding efficiency penalty versus H.264/AVC with single slice per frame. A significant portion of the coding penalty can be attributed to the reduction in context training (Fig. 1).

One of the features that gives CABAC its high coding efficiency is that the contexts are adaptive. While encoding/decoding, the contexts undergo training to achieve an accurate estimate of the syntax element probabilities. A better estimate of the probabilities results in better coding efficiency. A drawback of breaking up a picture into several entropy slices is that there are fewer macroblocks, and consequently fewer syntax elements, per slice. Since the entropy engine is reset every entropy slice, the context undergoes less training and can results in a poorer estimate of the probabilities.

4. SYNTAX ELEMENT PARTITIONING

This work propose a different method distributing the bins across parallel entropy engines. To avoid reducing the training, rather than processing various macroblocks/slices in parallel, syntax elements are processed in parallel. In other words, rather than grouping bins by macroblock and placing them in different entropy slices, bins are grouped based on syntax element and placed in different partitions which are then processed in parallel (Fig. 2). As a result, each partition contains all the bins of a given syntax element, and the context can then undergo the maximum amount of training (i.e. across all occurrences of the element in the frame) to achieve the best possible estimate and eliminate the coding efficiency penalty from reduced training. Table 2 shows the five different groups of syntax elements. The syntax elements were assigned to groups based on the bin distribution for a balanced workload. Each group of elements can be assigned
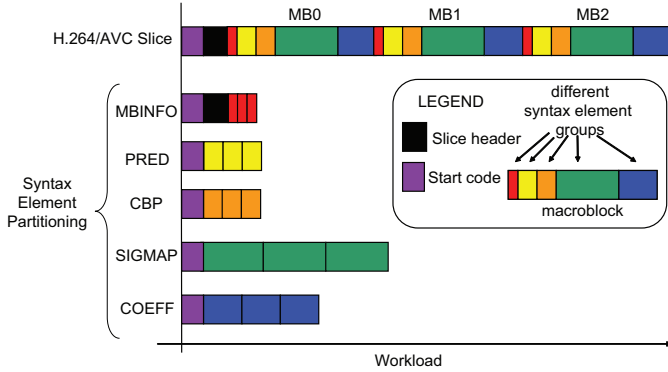
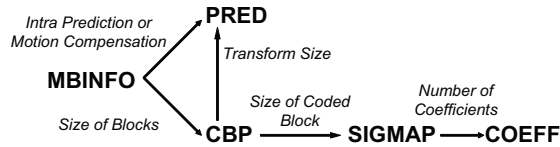**Fig. 2**: Concurrency with syntax element partitioning.

Fig. 3: Dependencies between groups.

to a different partition. A start code prefix for demarcation is required at the beginning of each partition.

This syntax element partitioning scheme is similar to slice data partitioning in the extended profile of H.264/AVC. However, slice data partitioning in H.264/AVC is limited to Context-based Adaptive Variable Length Coding (CAVLC) and is done primarily for error resilience purposes. Syntax element partitioning for CABAC can also benefit in terms of error resilience, however, it is done primarily to increase throughput and the partitions are assigned accordingly.

Dependencies between each group are shown in Fig. 3. For instance, the MBINFO group for a given macroblock (MB0) must be decoded before the PRED group for the same macroblock (MB0). However, the MBINFO group of the next macroblock (MB1) can be decoded in parallel with the PRED group of MB0. Thus, the processing of each partition must be synchronized. Synchronization can be done using data driven FIFOs between engines, similar to the ones used in [1] between processing units.

4.1. Coding Efficiency and Throughput

The syntax element partitioning approach was evaluated using JM12.0 software under common conditions [8]. The coding efficiency and throughput were compared against H.264/AVC slices and entropy slices (Table 3). To account for any workload imbalance, the partition with the largest number of bins per frame was used to compute the throughput. An average throughput speedup of ~2.7x can be achieved with negligible impact (0.06% to 0.37%) on coding efficiency. To achieve similar effective throughput requires at least three H.264/AVC or entropy slices per frame which have coding penalty of 0.87% to 1.71% and 0.25% to 0.69% respectively. Thus, syntax element partitioning provides 2 to 4x reduction in coding penalty relative to these other approaches.

Table 2: Syntax Element Groups.

Group	Syntax Element
MBINFO	mb_skip_flag, mb_type, sub_mb_type, mb_field_decoded_flag, end_of_slice_flag
PRED	prev_intra4x4_pred_mode_flag, rem_intra4x4_pred_mode, prev_intra8x8_pred_mode_flag, rem_intra8x8_pred_mode, intra_chroma_pred_mode, ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1
CBP	transform_size_8x8_flag coded_block_pattern, coded_block_flag
SIGMAP	significant_coeff_flag, last_significant_coeff_flag
COEFF	coeff_abs_level_minus1, coeff_sign_flag

Table 3: Comparison of various parallel processing techniques. The coding efficiency was computed by evaluating the Bjontegaard ΔBitrate against H.264/AVC with single slice per frame. Throughput was computed relative to serial 1 bin/cycle decoding. Results are averaged across BigShips, City, Crew, Night, and Shuttle.

	H.264/AVC Slices		Entropy Slices		Syntax Element Partitioning	
Area Cost	3x		3x		1.5x	
Prediction Structure	BD-rate	speed up	BD-rate	speed up	BD-rate	speed up
Ionly	0.87	2.43	0.25	2.43	0.06	2.60
IPPP	1.44	2.42	0.55	2.44	0.32	2.72
IBBP	1.71	2.46	0.69	2.47	0.37	2.76

4.2. Area Cost

Implementations for parallel H.264/AVC slices and entropy slices processing require that the entire CABAC be replicated which can lead to significant area cost. An important benefit to syntax element parallelism is that the area cost is quite low since the finite state machine used for context selection, and the context memory do not need to be replicated. Only the arithmetic coding engine needs to be replicated, which accounts for a small percentage of the total area. To achieve the throughput in Table 3, H.264/AVC slices and entropy slices require a 3x replication of the CABAC area, whereas syntax element partitioning has only a 50% area increase.

4.3. Adaptive Bin Allocation for Varying Quantization

In the previous analysis, each syntax element group was assigned to a different partition. Certain syntax element groups can be allocated to the same partition, such that only three partitions are used instead of five. This reduces the overhead of the start code prefix and the number of arithmetic engines (i.e. area). The overall throughput depends on how well the number of bins per partition are balanced. The distribution of the bins per syntax element group changes depending on the quantization (QP) as shown in Fig. 4. To maximize throughput for varying QP, the allocation of groups to each partition should be adaptive.
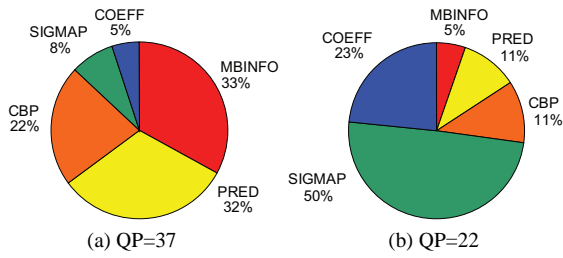
(a) QP=37 (b) QP=22

Fig. 4: Average bin distribution per frame.

Table 4: Group allocation to partitions.

Mode	MBINFO	PRED	CBP	SIGMAP	COEFF
High QP	0	0	0	1	2
Low QP	0	1	2	2	2

A threshold QP is used to distinguish the two QP modes of bin allocation. Table 4 shows which partition (0,1,2) each group is allocated to for a given QP mode. Adaptive QP is only necessary for P frames. In I frames, SIGMAP and CO-EFF tend to dominate regardless of QP, and thus the high QP mode is always used. The QP threshold can be different for each sequence and transmitted in the sequence parameter set. It can be selected by the encoder either using a two-pass approach or based on the number of non-zero coefficients.

Fig. 5 shows the throughput impact of adaptive syntax element partitioning over serial H.264/AVC CABAC. The throughput is greatly increased with adaptive QP. For most of the sequences, the QP threshold should be set somewhere between 27 and 32. However, for 'ShuttleStart', the QP threshold should be between 22 and 27.

4.4. Achieving Additional Parallelism

To reach Gbins/s, syntax element partitioning can be combined with the other approaches presented in Section 3 to achieve additional throughput at lower cost. For instance, a 6x throughput increase can be achieved by combining syntax element partitioning with 4 entropy slices, which results in better coding efficiency and lower area costs than just using 8 entropy slices or 8 H.264/AVC slices as shown in Fig. 6.
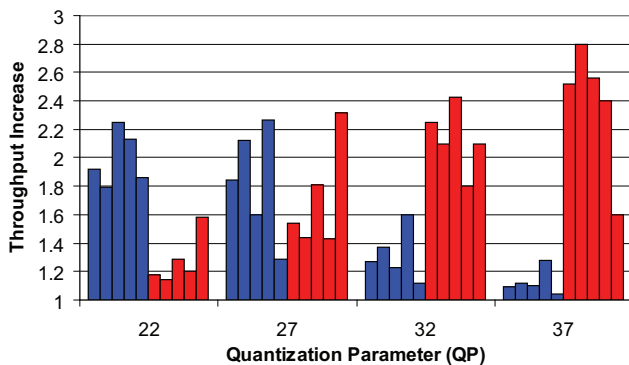

Fig. 5: High QP (blue); Low QP (red). Sequences (left to right): BigShips, City, Crew, Night, ShuttleStart.
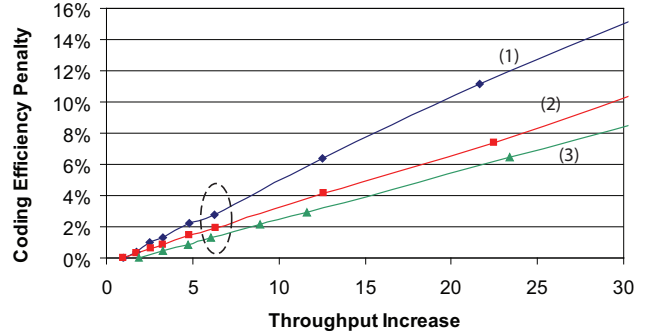

Fig. 6: Coding efficiency vs. throughput for (1) H.264/AVC slices, (2) entropy slices and (3) entropy slices with syntax element partitioning. Sequence 'BigShips' (QP=27) [8].

The slice and/or partition that has the most number of bins in a frame dictates the number of cycles required to decode that frame. Thus, the throughput improvement in Fig. 6 is determined by comparing the total number of bins in the frame with the slice and/or partition that has the most number of bins in that frame.

5. CONCLUSION

This work presents a new CABAC algorithm for the next generation standard that has increased concurrency by processing the bins of different syntax elements in parallel. A throughput increase of up to 3x can be achieved without sacrificing coding efficiency, power, or delay. The area overhead is minimal since the context memory does not have to be replicated. This approach can be combined with other approaches such as entropy slices for further throughput increase for an improved trade-off between coding efficiency and throughput/power.

6. REFERENCES

[1] D. Finchelstein, *et al.*, "A Low-Power 0.7-V H.264 720p Video Decoder," in *IEEE Asian Solid State Circuits Conf.*, November 2008.

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

[3] W. Yu *et al.*, "A High Performance CABAC Decoding Architecture," *IEEE Trans. on Consumer Electronics*, vol. 51, no. 4, pp. 1352–1359, November 2005.

[4] C.-H. Kim *et al.*, "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.

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

[6] V. Sze, *et al.*, "Parallel CABAC for Low Power Video Coding," in *IEEE International Conf. on Image Processing*, October 2008.

[7] J. Zhao *et al.*, "C0405: Entropy slices for parallel entropy decoding," ITU-T SG16/Q6, April 2008.

[8] T. Tan, *et al.*, "VCEG-AE010: Recommended Simulation Common Conditions for Coding Efficiency Experiments Rev. 1," ITU-T SG16/Q6, January 2007.