Noname manuscript No.
(will be inserted by the editor)

Joint Algorithm-Architecture Optimization of CABAC

Vivienne Sze · Anantha P. Chandrakasan

Received: date / Accepted: date

Abstract This paper uses joint optimization of both the algorithm and architecture to enable high coding efficiency in conjunction with high processing speed and low area cost. Specifically, it presents several optimizations that can be performed on Context Adaptive Binary Arithmetic Coding (CABAC), a form of entropy coding used in H.264/AVC, to achieve the throughput necessary for real-time low power high definition video coding. The combination of syntax element partitions and interleaved entropy slices, referred to as Massively Parallel CABAC, increases the number of binary symbols that can be processing in a cycle. Subinterval reordering is used to reduce the cycle time required to process each binary symbol. Under common conditions using the JM12.0 software, the Massively Parallel CABAC, increases the bins per cycle by 2.7 to 32.8x at a cost of 0.25% to 6.84% coding loss compared with sequential single slice H.264/AVC CABAC. It also provides a 2x reduction in area cost, and reduces memory bandwidth. Subinterval reordering reduces critical path by 14% to 22%, while modifications to context selection reduces memory requirement by 67%. This work illustrates that accounting for implementation cost during video coding algorithms design can enable higher processing speed and reduce hardware cost, while still delivering high coding efficiency in the next generation video coding standard.

This work was funded by Texas Instruments. Chip fabrication was provided by Texas Instruments. The work of V. Sze was supported by the Texas Instruments Graduate Women's Fellowship for Leadership in Microelectronics and NSERC.

V. Sze, A. P. Chandrakasan
Microsystems Technology Laboratories, Massachusetts Institute of Technology, Cambridge, MA 02139, USA (e-mail: sze@alum.mit.edu; anantha@mtl.mit.edu)

Keywords CABAC · Arithmetic Coding · H.264/AVC · HEVC · Video Coding · Architecture · Entropy Coding

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 minimal coding penalty. An increase in processing speed 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 Adaptive Binary Arithmetic Coding (CABAC) [8], a form of entropy coding used in H.264/AVC, which is a known throughput 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 HEVC, the successor to H.264/AVC. CABAC has been adopted into the HEVC test model [15].

Several joint algorithm and architecture optimizations of CABAC are proposed to enable parallelism for increased throughput with minimal coding loss. Specifically, three forms of parallelism will be exploited which enable multiple arithmetic coding engines to run in parallel as well as enable parallel operations within the arithmetic coding engine itself. In addition, optimiza-

tions are also discussed that reduce hardware area and memory requirements.

This paper is organized as follows: Section 2 provides an overview CABAC. Section 3 describes existing approaches to addressing the CABAC bottleneck. Section 4 describes how syntax element partitions and interleaved entropy slices can enable multiple arithmetic coding engines to run in parallel. Section 5 describes how subinterval reordering can enable parallel operations within the arithmetic coding engine. Section 6 describes how memory requirements can be reduced. Section 7 discusses the combined throughput impact of all techniques described in this work. Finally, Section 8 present a summary of the benefits of the proposed optimizations.

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 (e.g. motion vectors, coefficients, etc). 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 better coding efficiency than the Huffman-based Context Adaptive Variable Length Coding (CAVLC) [8].

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.

Arithmetic coding is based on recursive interval division; this recursive nature, contributes to the serial nature of the CABAC. The size of the subintervals are determined by multiplying the current interval by the probability 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.

Context modeling is used to generate an estimate of the bin's probability. In order to achieve optimal compression efficiency, an accurate probability must be used to code each bin. Accordingly, context modeling is highly adaptive and one of 400+ contexts (probability models) is selected depending on the type of syntax element, binIdx, luma/chroma, neighboring information, etc. A context switch can occur after each bin. Since the probabilities are non-stationary, the contexts are updated after each bin.

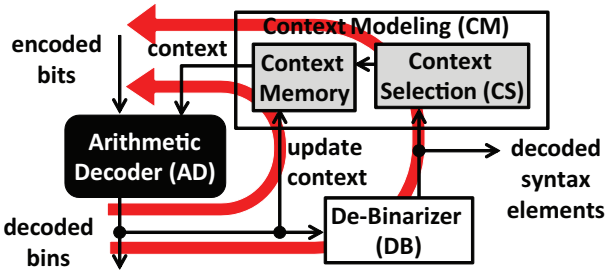

Fig. 1: Feedback loops in the CABAC decoder.

These data dependencies in CABAC result in tight feedback loops, particularly at the decoder, as shown in Fig. 1. Since the range and contexts are updated after every bin, the feedback loops are tied to bins; thus, the goal is to increase the overall bin-rate (bins per second) of the CABAC. In this work, two approaches are used to increase the bin-rate:

1. running multiple arithmetic coding engines in operate parallel (increase bins per cycle)
2. enabling parallel operations within the arithmetic coding engine (increase cycles per second)

2.1 Throughput Requirements

Meeting throughput (bin-rate) requirements is critical 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. 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 [1]. The bin-rate 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 [4] only go up to 210 MHz (in 90-nm CMOS process); 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*
3.1	30	4.0	17.5	121
3.2	60	4.0	25	242
4.0	30	9.2	25	275
4.1	30	17.6	50	527
4.2	60	17.6	50	1116
5.0	72	17.6	169	1261
5.1	26.7	17.6	300	2107

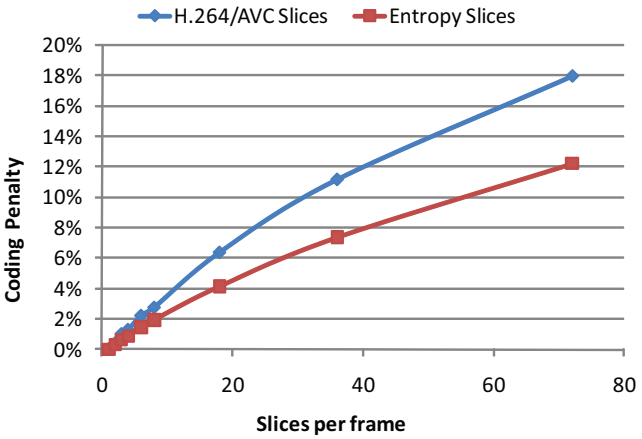

Fig. 2: Coding penalty versus slices per frame. Sequence bigships, QP=27, under common conditions [14].

3 Related Work

There are several methods of either reducing the peak bin-rate requirement or increasing the bin-rate of CABAC; however, they come at the cost of decreased coding efficiency, increased power consumption and/or increased latency. This section will discuss approaches that are both standard compliant and non-compliant.

3.1 Standard compliant approaches

Workload averaging across frames can be used to reduce the bin-rate requirements to be within the range of the maximum bit-rate at the cost of increased latency and storage requirements. For low-delay applications such as video conferencing, an additional delay of several frames may not be tolerated. The buffer also has implications on the memory bandwidth.

Bin parallelism is difficult to achieve due to the discussed data dependencies in CABAC. H.264/AVC CABAC implementations [3,4,16,17] need to use speculative computations to increase bins per cycle; however, speculative computations result in increased computations and consequently increased power consumption. Furthermore, the critical path delay increases with each additional bin, since all computations cannot be done entirely in parallel and thus the bin-rate increase that can be achieved with speculative computations is limited.

In H.264/AVC, frames can be broken into slices that can be encoded and decoded completely independently for each other. 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. 2 shows how the coding penalty increases with more H.264/AVC slices per frame.

3.2 Entropy Slices (non-standard compliant)

As shown in the previous section, increasing the performance of the CABAC is challenging when constrained by the H.264/AVC standard. An alternative is to modify the algorithm itself. In recent years, several new CABAC algorithms have been developed that seek to address this critical problem [6,18,19]. These algorithms looked at various ways of using a new approach called entropy slices to increase parallel processing for CABAC. Entropy slices are similar to H.264/AVC slices in that contiguous macroblocks are allocated to different slices. However, unlike H.264/AVC slices, which are completely independent of one another, some dependency is allowed for entropy slices. While entropy slices do not share information for entropy (de)coding (to enable parallel processing), motion vector reconstruction and intra prediction are allowed across entropy slices, resulting in better coding efficiency than H.264/AVC slices (Fig. 2). However, entropy slices still suffer coding efficiency penalty versus H.264/AVC with single slice per frame. This penalty can be attributed to a combination

of three key sources: no context selection across entropy slices, start code and header for each entropy slice, and reduced context training.

As described in Section 2, 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.

With ordered entropy slices, macroblocks are processed in zig-zag order within a slice to minimize memory bandwidth costs from syntax element buffering [6]. Furthermore, it allows for context selection dependencies across entropy slices which improves coding efficiency. However, the zig-zag order results in increased latency and does not provide a favorable memory access pattern necessary for effective caching.

4 Parallelism *across* arithmetic coding engines

In this section, we propose a parallel algorithm called Massively Parallel CABAC (MP-CABAC) that enables multiple arithmetic coding engines to run in parallel with an improved tradeoff between coding efficiency and throughput [9]. It can also be easily implemented in hardware and with low area cost. The MP-CABAC leverages a combination of two forms of parallelism. First, it uses syntax element parallelism, presented in Section 4.2, by simultaneously processing different syntax element partitions, allowing the context training to be performed across all instances of the elements, thus improving the coding efficiency. Second, macroblock/slice parallelism is achieved by simultaneously processing interleaved entropy slices, presented in Section 4.3, with simple synchronization and minimal impact on coding efficiency. Note that the MP-CABAC can also be combined with bin parallelism techniques previously described in Section 3.1.

4.1 Improving Tradeoffs

The goal of this work is to increase the throughput of the CABAC at minimal cost to coding efficiency and area. Thus, the various parallel CABAC approaches (H.264/AVC Slices, Entropy Slices, Ordered Entropy Slices, MP-CABAC) are evaluated and compared across two important metrics/tradeoffs:

- Coding Efficiency vs. Throughput
- Area Cost vs. Throughput

It should be noted that while throughput is correlated with degree of parallelism, they are not equal. It depends strongly on the workload balance between the parallel engines. If the workload is not equally distributed, some engines will be idle, and the throughput is reduced (i.e. N parallel hardware blocks will not result in an Nx throughput increase). Thus, we chose throughput as the target objective rather than degree of parallelism.

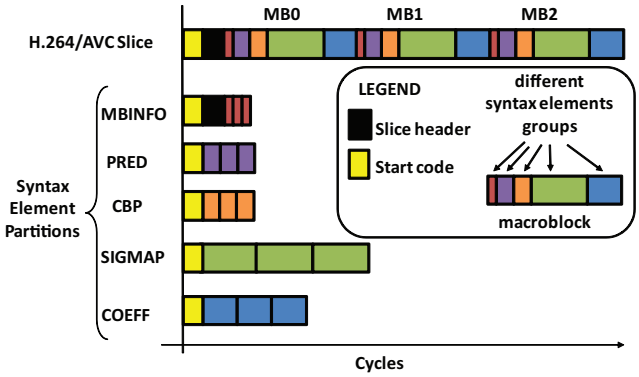

Fig. 3: Concurrency with syntax element partitioning.

4.2 Syntax Element Partitions (SEP)

Syntax element partitions enables syntax elements to be processed in parallel in order to avoid reducing the training [13]. In other words, bins are grouped based on syntax element and placed in different partitions which are then processed in parallel (Fig. 3). 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 probability estimate and eliminate the coding penalty from reduced training. Table 2 shows the five different syntax element partitions. The syntax elements were assigned to partitions based on the bin distribution in order to achieve a balanced workload. A start code prefix for demarcation is required at the beginning of each partition.

4.2.1 Coding Efficiency and Throughput

The syntax element partitions approach was evaluated using JM12.0 reference software provided by the standards body, under common conditions [14]. The coding efficiency and throughput were compared against

Table 2: Syntax Element Partitions.

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, mb_qp_delta, 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 BD-rate against H.264/AVC with single slice per frame. The speed up was computed relative to serial 1 bin/cycle decoding. The area cost was computed based on the increased gate count relative to a serial 1 bin/cycle CABAC.

	H.264/AVC Slices		Entropy Slices		Syntax Element Partitions	
Area Cost	3x		3x		1.7x	
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

H.264/AVC slices and entropy slices (Table 3). The coding efficiency is measured with the Bjøntegaard ΔBitrate (BD-rate) [2]. To account for any workload imbalance, the partition with the largest number of bins in a frame was used to compute the throughput. An average throughput speed up of $\sim$2.7x can be achieved with negligible impact (0.06% to 0.37%) on coding efficiency [13]. To achieve similar 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 partitions provides 2 to 4x reduction in coding penalty relative to these other approaches.

4.2.2 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 FSM 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. FIFOs need to be included to synchronize the partitions. Overall the SEP engine area is approximately 70% larger than the estimated H.264/AVC CABAC area [12]. 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 partitions only increase the area by 70%.

Note that the area cost for SEP may be even less than 70% if we account for storage of the last line data. If the last line data is stored in an on-chip cache, then it also needs to be replicated for the H.264/AVC and entropy slices approach which results in significant additional area cost. Alternatively, the last line data can be stored off-chip but this will increase the off-chip memory bandwidth. SEP does not require this cache to be replicated, which either reduces area cost or off-chip memory bandwidth.

4.3 Interleaved Entropy Slices (IES)

To achieve additional throughput improvement, SEP (as well as bin parallelism) can be combined with slice parallelism such as entropy slices. As mentioned in Section 3.2, entropy slices can undergo independent entropy decoding in the 'front-end' of the decoder. However, to achieve better coding efficiency than fully independent slices (i.e. H.264/AVC slices), there remains dependencies between the entropy slices for spatial and motion vector prediction in the 'back-end' of the decoder.

In the entropy slice proposals [6, 18, 19], the spatial location of the macroblocks allocated to each entropy slice is the same as in H.264/AVC (Fig. 4), i.e. contiguous groups of macroblocks. Due to the existing dependencies between entropy slices, back-end processing of slice 1 in Fig. 4 cannot begin until the last line of slice 0 has been fully decoded when using regular entropy slices. As a result, the decoded syntax elements of slice 1 need to be buffered as shown in Fig. 5, which increases latency and adds to memory costs - on the order of several hundred megabytes per second for HD. In this work, we propose the use of *interleaved* entropy slices where macroblocks are allocated as shown in Fig. 6, i.e. for two slices, even rows are assigned to one slice, while odd rows are assigned to the other [5]. Within each slice, the raster scan order processing is retained. Benefits of interleaved entropy slices include cross slice context selection, simple synchronization, reduction in

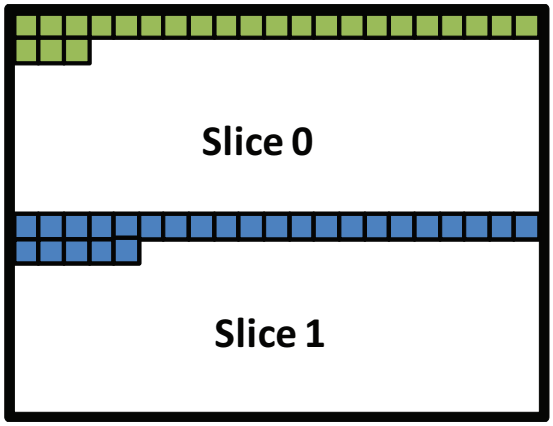

Fig. 4: Macroblock allocation for entropy slices [6, 18, 19].

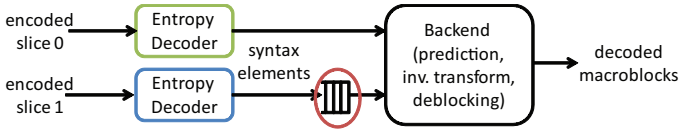

Fig. 5: Decoded syntax elements need to be buffered.

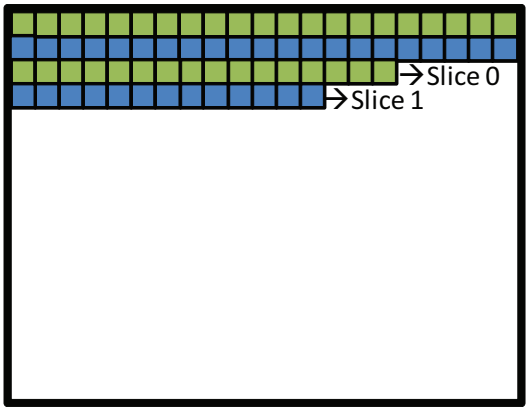

Fig. 6: Macroblock allocation for interleaved entropy slices.

memory bandwidth, low latency and improved workload balance.

In interleaved entropy slices, as long as the slice 0 is one macroblock ahead of slice 1, the top-macroblock dependency is retained, which enables cross slice context selection during parallel processing (i.e. spatial correlation can be utilized for better context selection) resulting in improved coding efficiency [9]. This is not possible with regular entropy slices.

Synchronization between entropy slices can easily be implemented through the use of FIFO between the slices (Fig. 7) [10]. Furthermore, both the front-end entropy processing and the back-end prediction processing can be done in this order (i.e. the entire decoder

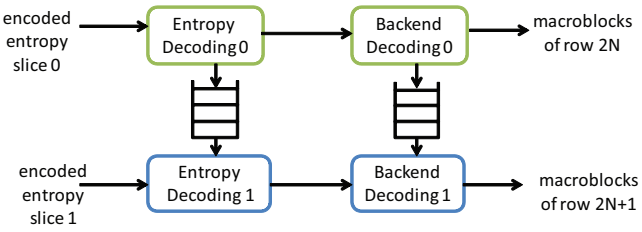

Fig. 7: Interleaved Entropy Slices architecture example for 2x parallel decoding. Note that the *entire* decode path can be parallelized.

path is parallelized), which allows the decoded syntax elements to be immediately processed by the back-end. Consequently, no buffering is required to store the decoded syntax elements, which reduces memory costs. This can have benefits in terms of reducing system power and possibly improving performance (by avoiding read conflicts in shared memory). No buffering also reduces latency which makes interleaved entropy slices suitable for low latency applications (e.g. video conferencing).

The number of accesses to the large last line buffer is reduced for interleaved entropy slices [12]. In Fig. 6, the last line buffer (which stores an entire macroblock row) is only accessed by slice 0. Since slice 0 is only several macroblocks ahead of slice 1, slice 1 only needs to access a small cache, which stores only a few macroblocks, for its last line (top) data. Thus out of N slices, N-1 will access small FIFO for the last line data, and only one will access the large last line buffer. If the last line buffer is stored on-chip, interleaved entropy slices reduces area cost since it does not need to be replicated for every slice as with H.264/AVC and entropy slices. Alternatively, if the last line buffer is stored off-chip, the off-chip memory bandwidth for last line access is reduced by 1/N. Note that the depth of the FIFO affects how far ahead slice 0 can be relative to slice 1. A deeper FIFO means that slice 0 is less likely to be stalled by slice 1 due to a full FIFO. For instance, increasing the FIFO depth from 4 to 8 macroblocks gives a 10% increase in throughput. However, increasing the FIFO depth also increases the area cost. Thus the depth of the FIFO should be selected based on both the throughput and area requirements.

Unlike ordered entropy slices, interleaved entropy slices retains raster scan order processing within each entropy slice which provides a favorable memory access pattern for caching techniques that enable further bandwidth reduction. Finally, in interleaved entropy slices the number of bins per slice tends to be more equally balanced; consequently, a higher throughput can be achieved for the same amount of parallelism.

The concept of using IES to enable wavefront parallel processing has been extended in [7].

A video sequence should be encoded with a certain number of IES based on the coding efficiency, throughput and area requirements. A minimum number of IES per frame should be included as part of the level definition that determines the bin-rate throughput requirement in order to ensure that the requirement can be met.

4.3.1 Coding Efficiency and Throughput

We measured the throughput of interleaved entropy slice alone as well as in combination with syntax element partitions, which we call the MP-CABAC. Note that syntax element partitions can also be combined with any of the other entropy slice approaches. Fig. 9 compares their coding efficiency and throughput against regular and ordered entropy slices as well as H.264/AVC slices. Table 4 shows the coding efficiency across various sequences and prediction structures for throughput increase (speed up) of around 10x over serial 1 bin/cycle H.264/AVC CABAC. MP-CABAC offers an overall average 1.2x, 3.0x, and 4.1x coding penalty (BD-rate) reduction compared with ordered entropy slices, entropy slices, and H.264/AVC respectively [9]. Data was obtained across different degrees of parallelism and plotted in Fig. 9. The throughput provided in Fig. 9 is averaged across five sequences, prediction structures (Ionly, IPPP, IBBP) and QP (22, 27, 32, 37). The sequences were also coded in CAVLC for comparison purposes; the coding penalty should not exceed 16% since there would not longer be any coding advantage of CABAC over CAVLC.

To account for any workload imbalance, the slice with the largest number of bins in a frame was used to compute the throughput. The BD-rates for entropy slices and ordered entropy slices are taken directly from [6]. Since the macroblock allocation for these proposals are the same, the workload imbalance should also be the same. The workload imbalance was measured based on simulations with JM12.0. The number of bins for each macroblock and consequently each entropy slice was determined and the throughput was calculated from the entropy slice in each frame with the greatest number of bins. Total number of bins processed by MP-CABAC, interleaved entropy slices, entropy slices and ordered entropy slices are the same; however, the number of bins for the H.264/AVC slices increases since the prediction modes and consequently syntax elements are different; this impact is included in the throughput calculations.

It should be noted that the coding efficiency for entropy slices and ordered entropy slices was obtained from implementations on top of the KTA2.1 software, which includes next generation video coding tools, while their throughput and the coding efficiency/throughput of interleaved entropy slices and MP-CABAC were obtained from implementations on top of the JM12.0 software, which contains only H.264/AVC tools. This accounts for the slight discrepancy in coding efficiency between the ordered entropy slices and interleaved entropy slices at 45x parallelism. In theory, they should be an exact match in terms of both coding efficiency and throughput.

For interleaved entropy slices, the slice overhead, due to the start code bits and the slice header, accounts for a significant portion of the BD-rate penalty. Each interleaved entropy slice has a 32-bit start code that enables the decoder to access the start of each slice as well as a slice header. In Table 4, 12 interleaved entropy slices are used per frame in order to achieve a speed up of around 10x. Given the fixed slice overhead, I only encoded sequences experience less BD-rate penalty than IPPP and IBBP since the slice data bits for I only is more than IPPP which is more than IBBP. H.264/AVC slices, entropy slices and ordered entropy slices have a more imbalanced workload than interleaved entropy slices, and therefore require 15 slices per frame to achieve the same 10x speed up. This increases the fixed slice overhead per frame. Note for slice parallelism techniques, the balance of bins per slices is unaffected by the prediction structure; thus all prediction structures experience similar speed up improvements for a given technique.

For MP-CABAC results in Table 4, interleaved entropy slices and syntax element partitions are combined in the same bitstream as shown in Fig. 8. Five syntax elements partitions, each with at 32-bit start code, are embedded in each interleaved entropy slices and the slice header information, such as slice type (I, P, B), slice quantization, etc., is inserted at the beginning of the MBINFO partition. Four interleaved entropy slices, each with 5 syntax element partitions, were used per frame in order to achieve a speed up of around 10x. Despite having more start code bits than that other techniques, MP-CABAC has lower BD-rate penalty due to improved context training of syntax element partitions, enabling context selection across slices, and fewer slice headers.

4.3.2 Area Cost

As in the case of the entropy slices and ordered entropy slices, the area of the entire CABAC (including the context memory) must be replicated for IES. Thus the total CABAC area increases linearly with paral-

Table 4: A comparison of the coding efficiency penalty (BD-rate) versus throughput for parallel CABAC approaches. Speed up and BD-rates are measured against serial 1 bin/cycle one slice per frame H.264/AVC CABAC.

		H.264/AVC slices		Entropy Slices		Ordered Entropy Slices		IES only		MP-CABAC (IES + syntax element partitioning)	
	Video Sequence	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up
Ionly	bigships	3.29	8.49	1.21	8.61	0.38	8.61	1.04	10.73	0.51	9.03
	city	3.02	11.89	0.63	12.12	-0.01	12.12	0.81	10.28	0.43	8.88
	crew	8.47	9.29	2.05	9.48	0.24	9.48	1.80	9.32	0.97	10.24
	night	4.10	9.66	0.68	9.83	-0.09	9.83	0.62	10.65	0.37	9.62
	shuttle	7.55	8.80	1.97	9.17	0.84	9.17	3.34	9.58	1.81	11.42
	Average	**5.29**	**9.62**	**1.31**	**9.84**	**0.27**	**9.84**	**1.52**	**10.11**	**0.82**	**9.85**
IPPP	bigships	5.65	9.95	5.69	10.31	2.01	10.31	2.37	9.91	1.91	9.77
	city	9.01	10.67	4.86	11.69	1.27	11.69	2.19	9.93	1.99	9.73
	crew	10.00	10.01	12.96	10.63	8.49	10.63	2.34	9.91	1.90	10.07
	night	4.87	8.27	2.14	8.50	0.56	8.50	1.67	10.02	1.30	10.44
	shuttle	1.95	8.47	9.63	8.84	2.93	8.84	5.06	9.96	3.99	10.70
	Average	**8.09**	**9.48**	**7.06**	**9.99**	**3.05**	**9.99**	**2.72**	**9.95**	**2.22**	**10.14**
IBBP	bigships	7.50	10.32	2.88	10.59	3.30	10.59	2.76	9.89	2.15	9.79
	city	11.31	11.53	5.69	12.20	1.67	12.20	2.73	10.16	2.40	9.83
	crew	11.01	10.28	4.86	10.79	6.00	10.79	2.58	10.13	1.99	10.49
	night	5.50	8.51	12.96	8.72	1.91	8.72	2.27	10.13	1.61	11.14
	shuttle	13.36	8.99	13.41	9.23	5.94	9.23	5.68	10.09	4.68	10.39
	Average	**9.73**	**9.93**	**8.75**	**10.30**	**3.76**	**10.30**	**3.21**	**10.08**	**2.57**	**10.33**

Table 5: A comparison of features of parallel CABAC approaches

	H.264/AVC slices	Entropy Slices	Ordered Entropy Slices	IES only	MP-CABAC (IES + syntax element partitioning)
Reference	Anchor	[19]	[6]	[9]	[9]
Software	JM12.0	KTA2.1	KTA2.1	JM12.0	JM12.0
Average BD-rate	7.7%	5.71%	2.36%	2.48%	1.87%
Area Cost	15x	15x	15x	12x	6x
Context Selection across Entropy Slices	No	No	Yes	Yes	Yes
Syntax Element Buffering	No	Yes	No	No	No

lelism as shown in Fig. 10. In Fig. 10 coding efficiency and throughput are averaged across prediction structures, sequences and quantization. Note that the area cost versus throughput tradeoff for the entropy slices and ordered entropy slices are the same since they have the same throughput. For the same throughput, interleaved entropy slices require less area increase since it needs fewer parallel engines due to its better workload balance. Table 5 shows that for a 10x throughput increase over serial 1 bin/cycle CABAC, interleaved entropy slices reduced area cost by 20%, while MP-CABAC reduces area cost by 60%. Furthermore, no buffering is required to store syntax elements and easy synchronization can be performed with FIFO between the interleaved entropy slices. The simplicity of this approach allows the whole decoder to be parallelized. Note that a significant area cost reduction is achieved for MP-CABAC, when interleaved entropy slices are combined with syntax element partitions. As mentioned earlier, if the last line buffer is stored on-chip, IES provides additional area savings since the buffer does not need to be replicated for every slice [10, 12].

5 Parallelism *within* arithmetic coding engines

In this section, we propose an optimization that increases parallel operations within the arithmetic coding engine to reduce the critical path delay, increasing the cycles per second that can be achieved by the CABAC [11]. 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

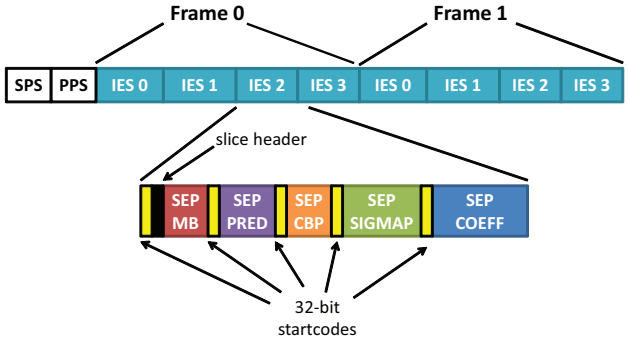

Fig. 8: MP-CABAC data structure. In this example, there are four IES per frame and five SEP per IES.

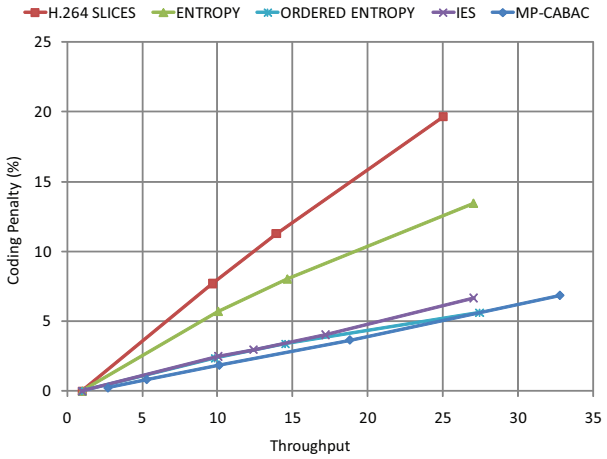

Fig. 9: Tradeoff between coding efficiency and throughput for various parallel CABAC approaches.

(LPS) and most probable symbol (MPS). The range of MPS (rMPS) is compared to the offset to determine whether the bin is MPS or LPS. rMPS is computed by first obtaining range of LPS (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.

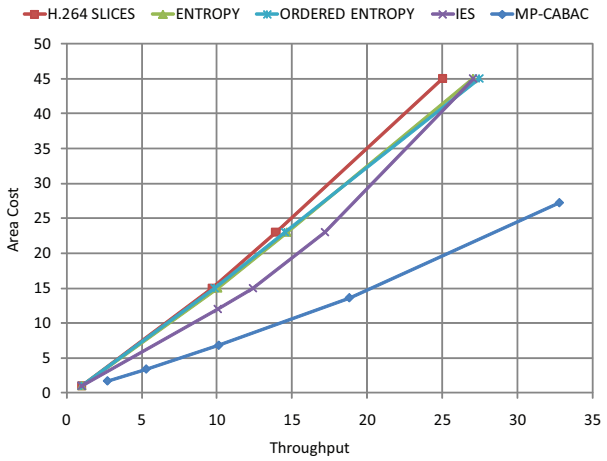

Fig. 10: Area cost versus throughput tradeoff for the various parallel CABAC approaches.

If the offset was compared to rLPS rather than rMPS, then the comparison and subtraction to compute rMPS can occur in parallel. 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. 11 shows the difference between the subinterval order of H.264/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 JM12.0 under common conditions. An arithmetic decoder was also implemented in RTL for each subinterval ordering and 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 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. 12. The lower bound can also be updated earlier since it depends on rLPS rather than rMPS.

6 Reduction in Memory Requirement

To leverage spatial correlation of neighboring data, context selection can depend on the values of the top and left blocks as shown in Fig. 13. The top dependency requires a line buffer in the CABAC engine to store

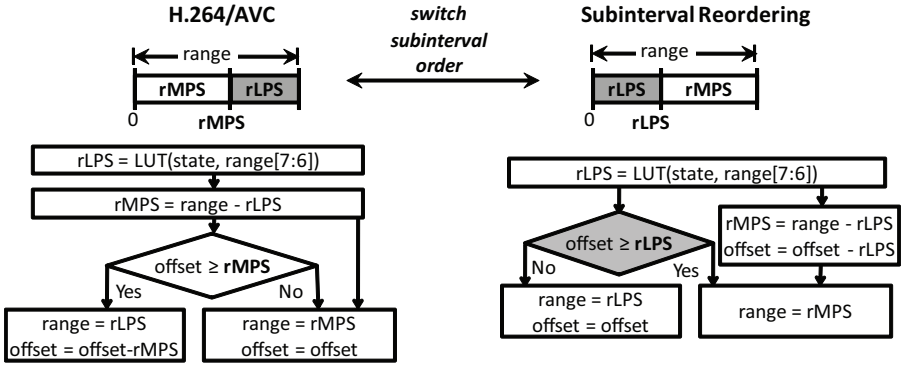

Fig. 11: Impact of subinterval reordering for CABAC *decoding*.

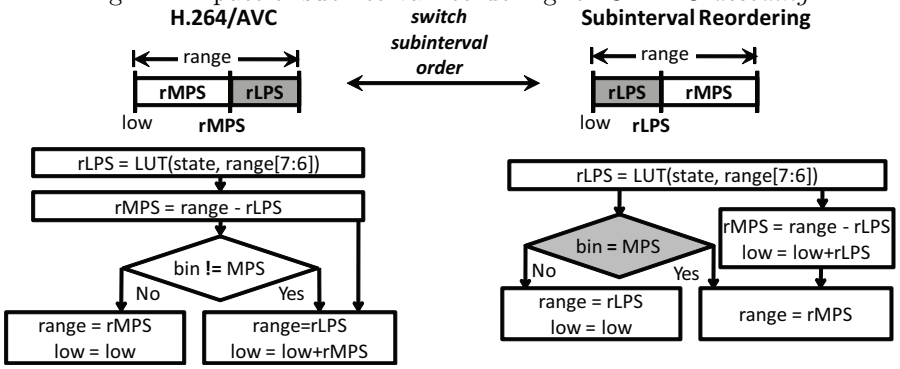

Fig. 12: Impact of subinterval reordering for CABAC *encoding*.

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 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. Table 6 shows the bits requires to be stored in the line buffer for context selection when processing a 4096 pixel wide video sequence. We propose reducing the bit-width of this data to reduce the overall line buffer size of the CABAC.

Majority of the data stored in the line buffer is for the context selection of *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.

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

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. 13.

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

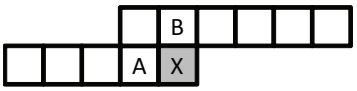

Fig. 13: For position X, context selection is dependent on A and B (4x4 blocks for *mvd*); a line buffer is required to store the previous row of decoded data.

increment, χ_{mvd}, is computed in two steps [8]:
Step 1: Sum the absolute value of neighboring *mvd*

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

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

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

Since the upper threshold is 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 $6\times2\times2\times1024$ =24,576 bits are required for *mvd* storage.

Table 6: Context selection line buffer storage requirements for a 4096 pixel wide video sequence.

Syntax Element (SE)	Frequency of signaling	SE/ macroblock	Bits/ SE (H.264/AVC)	Bits for 4kx2k (H.264/AVC)	Bits/ SE (proposed)	Bits for 4kx2k (proposed)
mb_type	per MB	1	1	256	1	256
mb_skip_flag	per MB	1	1	256	1	256
refIdx_l0	per 8x8	2	1	512	1	512
refIdx_l1	per 8x8	2	1	512	1	512
mvd_l0 (vertical)	per 4x4	4	6	6144	1	1024
mvd_l0 (horizontal)	per 4x4	4	6	6144	1	1024
mvd_l1 (vertical)	per 4x4	4	6	6144	1	1024
mvd_l1 (horizontal)	per 4x4	4	6	6144	1	1024
intra_chroma_pred_mode	per MB	1	1	256	1	256
intra_16x16	per MB	1	1	256	1	256
coded_block_flag (luma DC)	per MB	1	1	256	1	256
coded_block_flag (luma)	per 4x4	4	1	1024	1	1024
coded_block_flag (chroma DC)	per 8x8	2	1	512	1	512
coded_block_flag (chroma)	per 4x4	4	1	1024	1	1024
coded_block_pattern (luma)	per 8x8	2	1	512	1	512
coded_block_pattern (chroma)	per 8x8	2	1	512	1	512
transform_8x8_mode_flag	per MB	1	1	256	1	256
Total				30720		10240

To reduce the memory size, we propose performing comparisons for each component *before* summing their results. In other words,

Step 1: Compare 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 results thresh_A and thresh_B from Step 1

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

With this change, only single bit is required to be stored per component per 4x4 block; the size of the line buffer for *mvd* is reduced to $1 \times 2 \times 2 \times 1024 = 4{,}096$ bits. In H.264/AVC, the overall line buffer size of the CABAC required for all syntax elements is 30,720 bits. The modified *mvd* context selection reduces the memory size by 67%, from 30,720 bits to 10,240 bits as shown in Table 6. The average coding penalty of this approach, was verified across common conditions to be $\leq 0.02\%$.

7 Overall Impact

Three forms of parallelism have been presented in this work. First, syntax element partitions which enables processing different syntax elements in parallel. Second, interleaved entropy slices which enables processing macroblocks in parallel. Finally, subinterval reordering enables parallel operations within the arithmetic coding engine. Fig. 14 shows how all techniques are integrated together. The slices engines are connected using FIFOs to process IES in parallel. Each slice engine contains five arithmetic decoder (AD) to process the five SEP in parallel. Note that IES FIFOs are only need to connect AD for MBINFO, PRED and CBP since only syntax elements in those partitions use top macroblock information for context selection. The last line buffer can be viewed as a large FIFO that connects slice engine 0 and 3. This can be stored on-chip (increases area cost) or off-chip (increases memory bandwidth). Within these AD, subinterval reordering has been applied to enable parallel operations that speed up the overall throughput.

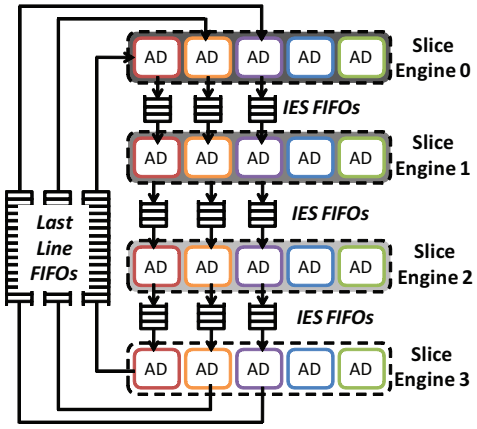

Fig. 14: Overall architecture to support multiple forms of parallelism presented in this work.

Both syntax element partitions and interleaved entropy slices increase the number of bins processed per cycle. Their impact on throughput varies depending on the properties of a video sequence which affect the balance of bins across slices. For instance, with two interleaved entropy slices per frame, the BigShips sequence, encoded using a QP=32 with IBBP achieves a 5.94x throughput increase with a bit-rate increase of 1.3%; the ShuttleLaunch sequence, encoded using a QP=32 with IBBP achieves 5.02x throughput increase with a bit-rate increase of 2.3%.

In contrast, subinterval reordering reduces the critical path of arithmetic coding engine which provides throughput increase across all video sequences. For example, a given area cost in a 45-nm CMOS process, a H.264/AVC CABAC arithmetic coding engine can run at 270 MHz, whereas using subinterval reordering, the arithmetic coding engine can run at 313 MHz. This 16% increase in frequency translates to an increase in throughput for all sequences.

The overall throughput is calculated as follows:

$$\text{bin-rate} = \text{bins/cycle x cycles/second}$$

Thus, subinterval reordering has an added throughput impact on top of the IES and SEP approaches. Assuming a initial serial H.264/AVC CABAC of one bin per cycle, which has a throughput of 270 Mbins/s, the techniques presented in this paper could increase the throughput to process BigShips by 5.94 x (313/270) = 6.9x for a bin-rate of 1859 Mbins/s. Similarly, throughput to process ShuttleLaunch is increased by 5.02 x (313/270) = 5.8x for a bin-rate of 1571 Mbins/s. Since subinterval reordering has negligible impact on coding efficiency, the coding loss would remain as 1.3% and 2.3% respectively, as described earlier.

8 Summary and Conclusions

In this work, several joint algorithm and architecture optimizations were proposed for CABAC for increased throughput with minimal coding efficiency cost. Parallelism is achieved across multiple arithmetic coding engines to increase bins per cycle as well as within the arithmetic coding engine to increase cycles per second for an overall increase in bin-rate (bins per second).

Across arithmetic coding engines, MP-CABAC, which is a combination syntax element and slice parallelism, can be used. MP-CABAC involved reorganizing the data (syntax elements) in an encoded bitstream such that the bins (workload) can be distributed across different parallel processors and multiple bins can be decoded simultaneously without significant increase in coding penalty and implementation cost.

Benefits of the MP-CABAC include

1. high throughput
2. low area cost
3. good coding efficiency
4. reduced memory bandwidth
5. simple synchronization and implementation
6. low latency
7. enables full decoder parallelism

For a 2.7x increase in throughput, syntax element partitions were shown to provide between 2 to 4x reduction in coding penalty when compared to slice parallel approaches, and close to 2x reduction in area cost. When combined with interleaved entropy slices to form the MP-CABAC, additional throughput improvement can be achieved with low coding penalty and area cost. For a 10x increase in throughput, the coding penalty was reduced by 1.2x, 3x and 4x relative to ordered entropy, entropy and H.264/AVC slices respectively. Over a 2x reduction in area cost was achieved.

Additional optimizations within the arithmetic coding engine using subinterval reordering increases processing speed by 14 to 22% with no coding penalty. Finally, to address memory requirement, the context selection can be modified to reduce memory size by 50% with negligible coding efficiency impact ($\leq$0.02%). Details on an implementation of the MP-CABAC with these optimizations can be found in [10, 12].

This work 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.

Acknowledgements The authors would like to thank Madhukar Budagavi and Daniel Finchelstein for valuable feedback and discussions.

References

1. Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services. Tech. rep., ITU-T (2003)
2. Bjøntegaard, G.: VCEG-M33: Calculation of Average PSNR Differences between RD curves. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2001)
3. Chen, J.W., Lin, Y.L.: A high-performance hardwired CABAC decoder for ultra-high resolution video. IEEE Trans. on Consumer Electronics **55**(3), 1614 –1622 (2009)
4. Chuang, T.D., Tsung, P.K., Pin-Chih Lin, L.M.C., Ma, T.C., Chen, Y.H., Chen, L.G.: A 59.5 Scalable/Multi-View Video Decoder Chip for Quad/3D Full HDTV and Video Streaming Applications. In: IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, pp. 330–331 (2010)

5. Finchelstein, D., Sze, V., Chandrakasan, A.: Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders. IEEE Trans. on Circuits and Systems for Video Technology **19**(11), 1704–1713 (2009)
6. Guo, X., Huang, Y.W., Lei, S.: VCEG-AK25: Ordered Entropy Slices for Parallel CABAC. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2009)
7. Henry, F., Pateux, S.: JCTVC-E196: Wavefront Parallel Processing. Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 (2011)
8. Marpe, D., Schwarz, H., Wiegand, T.: Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. IEEE Trans. on Circuits and Systems for Video Technology **13**(7), 620– 636 (2003)
9. Sze, V., Budagavi, M., Chandrakasan, A.: VCEG-AL21: Massively Parallel CABAC. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2009)
10. Sze, V., Chandrakasan, A.: A highly parallel and scalable cabac decoder for next generation video coding. In: IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, pp. 126 –128 (2011)
11. Sze, V., Chandrakasan, A.: Joint Algorithm-Architecture Optimization of CABAC to Increase Speed and Reduce Area Cost. In: IEEE Inter. Conf. on Acoustics, Speech and Signal Processing, pp. 1577–1580 (2011)
12. Sze, V., Chandrakasan, A.: A highly parallel and scalable cabac decoder for next generation video coding. IEEE Journal of Solid-State Circuits **47**(1), 8–22 (2012)
13. Sze, V., Chandrakasan, A.P.: A High Throughput CABAC Algorithm Using Syntax Element Partitioning. In: IEEE Inter. Conf. on Image Processing, pp. 773–776 (2009)
14. Tan, T., Sullivan, G., Wedi, T.: VCEG-AE010: Recommended Simulation Common Conditions for Coding Efficiency Experiments Rev. 1. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2007)
15. Tan, T.K., Sullivan, G., Ohm, J.R.: JCTVC-C405: Summary of HEVC working draft 1 and HEVC test model (HM) (2010)
16. Yang, Y.C., Guo, J.I.: High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications. IEEE Trans. on Circuits and Systems for Video Technology **19**(9), 1395 –1399 (2009)
17. Zhang, P., Xie, D., Gao, W.: Variable-bin-rate CABAC engine for H.264/AVC high definition real-time decoding. IEEE Trans. on Very Large Scale Integration (VLSI) Systems **17**(3), 417–426 (2009)
18. Zhao, J., Segall, A.: COM16-C405: Entropy slices for parallel entropy decoding. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2008)
19. Zhao, J., Segall, A.: VCEG-AI32: New Results using Entropy Slices for Parallel Decoding. ITU-T SG. 16 Q. 6, Video Coding Experts Group (VCEG) (2008)