Prediction-based Real-time CABAC Decoder for High Definition H.264/AVC

WonHee Son and In-Cheol Park
School of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
E-mail: wson@eeinfo.kaist.ac.kr, icpark@ee.kaist.ac.kr

Abstract— **This paper proposes a prediction scheme to decode in real-time H.264/AVC bitstream coded in Context-based Adaptive Binary Arithmetic Coding (CABAC). The proposed scheme predicts the subsequent syntax element type, leading to a significant reduction in cycles invoked by the syntax element switching overhead that degrades the decoding performance significantly. In addition, a new pipelined architecture combined with the proposed scheme is presented for hardware implementation. The simulation results show that the proposed scheme achieves a decoding performance of 1.2 cycles/bin, which releases the CABAC parsing from a bottleneck in decoding H.264 bitstream.**

I. INTRODUCTION

Context-based Adaptive Binary Arithmetic Coding (CABAC) is an essential entropy coding block used in the main profile of H.264/AVC being widely adopted as the next generation video coding standard [1][2]. The large coding gain of H.264/AVC is known to significantly depend on CABAC. Compared with Context Adaptive Variable Length Coding (CAVLC) which is adopted for the basic profile of H.264/AVC, CABAC leads to about 15% reduction in the size of encoded data [5]. Such an efficient compression makes it inevitable to adopt CABAC in high definition (HD) video stream which is rapidly becoming the mainstream format.

The promising adoption of CABAC especially in HD video stream has stimulated many researchers to endeavor to design a high-performance CABAC decoder. However, its intrinsic sequential nature has imposed challenges on the design of high-throughput hardware units which can be obtained by adopting a pipelined or parallel architecture. Until the pipeline hazards were resolved in [3], few real-time CABAC decoders that are fast enough to decode a HD video clip were implemented by adopting such techniques as exploiting parallelism, which implies difficulties in implementing a high-throughput CABAC decoder.

Previous works such as [3][5] have focused on resolving the inter-bin dependency which is considerable in decoding multiple-bin syntax elements (SEs) to achieve a fast decoding of the bitstream. Although multiple-bin SEs occupy the large portion of the bitstream and deserve to be spotlighted, there is another overlooked critical factor that degrades the decoding performance. In this paper, the factor is named syntax element switching overhead (SESO). Assuming a pipelined architecture, SESO is defined to be the pipeline stall owing to SE switching. Without resolving the significant degradation caused by SESO, it is hard to implement a real-time decoder appropriate for HD video stream. To our best knowledge, this paper is the first paper that points out the significance of switching overhead in performance degradation and resolves it.

In this paper, we propose a prediction scheme to resolve the SESO that has been overlooked in previous pipelined architectures. Based on the predictor, a new architecture is presented to exploit the characteristics of successive SEs in resolving the switching overhead, achieving a real-time decoding of CABAC-coded bitstream.

The rest of this paper is organized as follows. Section II describes the fundamentals of CABAC and previous pipelined architectures. In Section III, the properties of SESO are analyzed and the proposed prediction scheme is described in detail. Section IV presents a novel pipelined architecture combined with the proposed scheme. Simulation results are briefly explained in Section V, and concluding remarks are made in Section VI.

II. CABAC DECODING

The input of CABAC decoder is a sequence of syntax elements (SE), where an SE is an element of data represented in the bitstream [1]. In other words, H.264/AVC bitstream is a chain of various syntax elements. The decoded value of an SE is used for actual decoding of the video sequence. Let *S* denote the entire set of SEs which are encoded or decoded by using CABAC. Fig. 1 summarizes all the SEs contained in *S*.

Basically, CABAC is a binary arithmetic coding of which the encoding and decoding are context-dependent. The context information is modeled as a pair of probability state index (pStateIdx) and the value of the currently most probable symbol (valMPS) which is either 0 or 1, and is denoted by context model (CM). CM provides the decoder with the prior-probability of an encoded bin. CABAC decoding can be

This work was supported by IC Design Education Center (IDEC).

decomposed into context loading (CL), context selection (CS), binary arithmetic decoding (BAD), and binarization matching (BM) [3]. Let s denote the current SE. In CL, $C(s)$, an array of context models (CM) of which the base is common, denoted as ctxIdxOffset in [1], is fetched from the context memory. CS determines which CM of the array is to be actually used for binary arithmetic decoding. In this stage, ctxIdxInc [1], an index increment relative to ctxIdxOffset, is used to select the desired CM from the array fetched in CL. As pointed out in [3], the calculation of ctxIdxOffset and ctxIdxInc can be parallelized, which enables to implement the decoder in a pipelined architecture. BAD decodes the current bin in the given bitstream by utilizing the current CM determined in CS. The decoded bin in BAD is de-binarized in BM to determine the decoded value of s, $V(s)$. If the decoded bin stream is valid, BM outputs $V(s)$ declaring the end of decoding s.

III. PREDICTION SCHEME

This section explains syntax element switching overhead (SESO) which significantly degrades the decoding performance, and proposes a prediction scheme to resolve the SESO problem. In addition, the SE flow in H.264 is analyzed to design the prediction scheme. Prior to describing SESO and the proposed scheme, let us define some notations and symbols to be used hereafter.

$s = [b_0 \mid b_1 \mid ... \mid b_{n-1}],\ s \in S$

: SE s is a chain of bins b_0,b_1, .., b_{n-1} when $|s| = n$

$T(s)$: SE type of s where $s \in S$

$V(s)$: decoded value of s where $s \in S$

$binVal(b_i)$: decoded bin value of b_i

Transition: $s_1 \rightarrow s_2$: SE transitions from s_1 to s_2 where $s_1, s_2 \in S$ (i.e., s_2 succeeds s_1)

Prediction: $P(s_1) = T(s_2)$ where $s_1 \rightarrow s_2$

Dependency: $T(t) \sim V(s)$ indicates that the SE type of t is determined by using the value of s where $s, t \in S$.

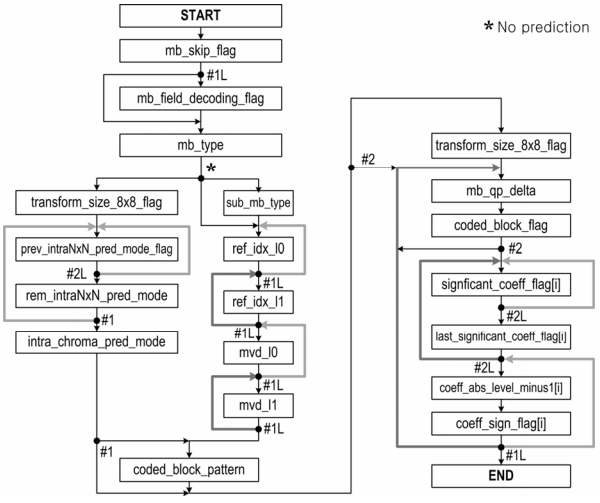

Figure 1. SE flow and branch points

A. Syntax Element Switching Overhead (SESO)

Whenever all the bins of an SE are decoded, there is an SE transition. Suppose that $s_i \rightarrow s_{i+1}$. $T(s_i)$ and $T(s_{i+1})$ can be either equal or different. If $T(s_i) \neq T(s_{i+1})$, this transition is called *SE switching* hereinafter. As shown in Table I, SE switching occupies over 60% of the entire SE transitions, which implies the frequent changes in SE type. Assuming a pipelined architecture, the resulting pipeline stall leads to a drastic degradation in decoding performance. As an example, [3] achieves the decoding performance up to nearly 1 cycle/bin if only the intra-SE decoding is considered, while it degrades to about 4 cycles/bin when inter-SE switching is taken into account, showing the significance of SESO in decoding performance. Such a huge gap is resulted from the fact that an SE switching requires at least three cycles for $T(s_{i+1})$ determination, CU and CL. If $T(s_{i+1})$ can be predicted before s_i is decoded, the decoder does not need to wait for $T(s_{i+1})$ to be determined, which leads to one cycle saving in SE switching. In addition, CL and CU stages, which consume two cycles when SE type changes, can be skipped in cases that $s_i \rightarrow s_{i+1}$ is not a switching but merely a transition. Hence, the prediction of $T(s_{i+1})$ is a key factor in reducing the performance degradation resulting from SESO.

B. SE Flow and Branches

To design a predictor for $T(s_{i+1})$, the parsing process of CABAC-coded SEs is first analyzed. SE Parsing, which has been assumed to be executed by an external processor in previous works, can be morphed into SE flow as depicted in Fig. 1. The SE flow is heavily dependent on conditional branches. Investigation on the SE flow shows that those branches can be classified into two types: Type 1 and Type 2.

Suppose that $s_i \rightarrow s_{i+1}$. In Type 1 branch, which is annotated as #1 in Fig. 1, $V(s_i)$ does not affect the determination of $T(s_{i+1})$. $T(s_{i+1})$ only depends on $T(s_i)$. Branches marked by #1L are loops of SEs $\{s_0, s_1, \ldots, s_{n-1}\}$ with $T(s_0)=T(s_1) = ...= T(s_{n-1})$ while the iteration is not terminated in the loop. Fortunately, most of #1L branches are statically determined, which enables it to predict $T(s_i)$ of #1

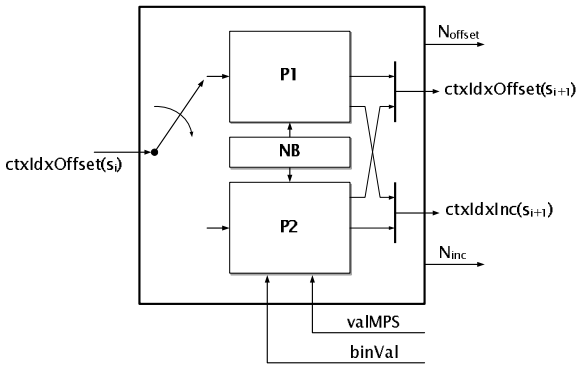

Figure 2. SE Predictor

and #1L with ease.

In contrast, Type 2 branches need not only $T(s_i)$ but also $V(s_i)$ to determine $T(s_{i+1})$. In most cases, $T(s_i)$ on a Type 2 branch is a flag type, that is, $s_i = [b_0]$, and thus the prediction of $binVal(b_0)$ can determine $T(s_{i+1})$. Assuming that CM(i), CM for s_i, is available, we can determine $V(s_i)$ by using valMPS of CM(i) before b_0 is actually decoded in the BAD stage. How to take advantage of valMPS in predicting $binVal(b_0)$ is to be explained later.

C. SE Predictor

As illustrated in Fig. 2, the proposed SE Predictor is composed of two parts: P1 for Type 1 branches and P2 for Type 2. The selection of P1 and P2 is determined by $T(s_i)$. The block denoted by NB provides both P1 and P2 with SE values which were already decoded in neighbor blocks. In addition to predicting $T(s_{i+1})$, therefore $\text{ctxIdxOffset}_{i+1}$, P1 and P2 predicts ctxIdxInc for the first bin of s_{i+1}. To reduce the mis-prediction penalty, SE predictor outputs N_{offset} and N_{inc} which are ctxIdxOffset and ctxIdxInc for the branches predicted not to be taken.

P1 is implemented by using a table whose inputs are $T(s_i)$ and the loop index. As Type 1 branches are not dependent on the $V(s_i)$, the table merely onto-maps $T(s_i)$ to $T(s_{i+1})$ according to conditions which are already calculated. Most of them are easily obtained by the neighbor context information provided by the NB block. Loop index are used only for #1L branches. Static transitions, not annotated as either Type 1 or Type 2 in Fig. 1, are also tabularized in P1.

The targets of type 2 branches are predicted by P2. In most cases, $T(s_i)$ on a Type 2 branch is a flag type, i.e. $s_i = [b_0]$, and the prediction of $binVal(b_0)$ leads to determination of $T(s_{i+1})$. P2 consists of a mapping table and a two-bit predictor. The two-bit predictor predicts $binVal(b_0)$ based on the current state which is updated by the actual value of the previously decoded bin. The table of P2 also maps $T(s_i)$ to $T(s_{i+1})$ by predicting the branch result based on $binVal(b_0)$ which is predicted by the two-bit predictor.

Since the MPS is not always equal to actual $binVal$, whether to use the MPS or the LPS, least probable symbol, should be decided for $binVal(b_0)$. To enhance the accuracy of the determination, a two-bit predictor based on CM(j) is proposed. It has four stages of MPS_11, MPS_10, LPS_01, and LPS_00. Based on the current state, it is determined whether to use the MPS or the LPS. If the current state is either MPS_11 or MPS_10, b_j is determined to be equal to valMPS(j), the MPS value of the CM(j). The state transition occurs whenever a bin is decoded in the BAD module. Fig. 3 depicts the proposed CM-based two-bit predictor contained in P2. Compared with a simple predictor which does not exploit the prior probability information, the proposed MPS-based predictor can achieve up to 20% higher prediction accuracy at the cost of slight increase in area.

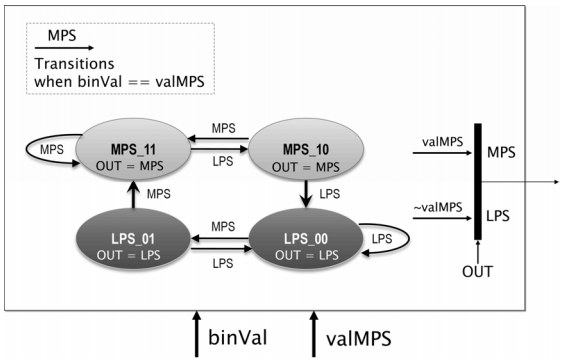

Figure 3. The proposed MPS-based two-bit predictor in P2

IV. Prediction-based Pipelined Architecture

In general, to achieve high-throughput, parallel architecture and pipelined architecture are considered. For fast CABAC decoding, there have been suggested parallel architectures capable of decoding multiple bins at the same time [6][7]. However, those architectures lack consideration of the computational overhead of context model loading (CL) or update (CU) which imposes heavy limitations on the enhancement of decoding performance. Furthermore, the considerable enhancement in performance is limited to only several syntax elements at the cost of large area caused by the inevitable redundancies of the parallel architecture. As a result, those multi-bin decoding parallel architectures are hard to be applied to actual decoding schemes. Practically, the pipelined architecture is the most plausible for the CABAC decoder implementation.

There have been some pipelined architectures for CABAC coding [3][5]. In [3]-[5], the syntax parsing and decoding are separated. The separated architecture enables it to concentrate on CABAC decoding only, leaving the parsing to another processor. In this architecture, the SE type determined by the companion processor ignites the decoding process. Though such architecture has the advantage of relatively simple implementation, the separation of parsing and decoding invokes significant performance degradation due to SESO which has not been considered with concern in previous studies.

The statistics of SE type reveals that flag-type SEs occupy a considerable portion in the entire set of SEs, which indicates that latency can be a main obstacle in adopting a pipelined architecture for CABAC decoding. Considering the frequent occurrence of flag-type SEs and the decoding performance

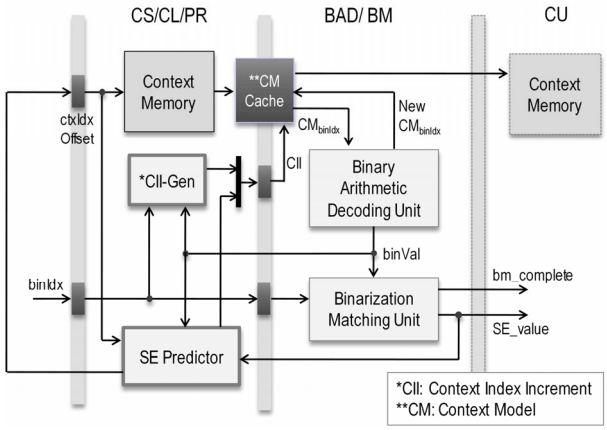

Figure 4. Proposed architecture

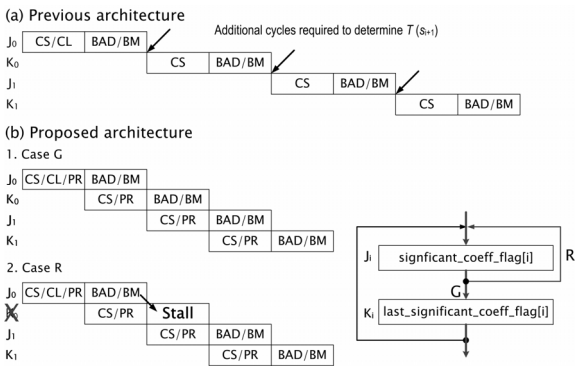

Figure 5. Pipeline scheduling comparison (a) previous architecture (b) proposed architecture

degradation caused by SESO, the number of pipeline stages should be as small as possible. Hence, two-stage pipeline can be the optimal solution that meets the conflicting requirements of short latency and high-throughput while occupying relatively less area than a parallel architecture.

The proposed architecture is depicted in Fig. 4. Since CABAC parsing is hardly intervened, it is possible to embed the parsing function into the CABAC decoder. SE predictor executes the parsing function. For Type 1 branches, SE predictor determines the next SE type without stalling the pipeline. In Type 2 branches, SE predictor predicts $T(s_{i+1})$. When the prediction turns out to be wrong, which is judged by examining the actual value of *binVal* and the predicted one, the BAD stage stalls when s_{i+1} is to enter the BAD/BM stage until the correct SE type is determined. With the proposed scheme, the mis-prediction penalty is merely one cycle as illustrated in Fig. 5. To parse non-CABAC coded SEs before entering the CABAC-coded portion of the bitstream, software processing or another companion parsing unit can be adopted.

V. Simulation Results

Table I summarizes the simulation results. All the test video streams are HD 1080p (1920x1080) having 4:2:0 chroma format and have the frame rate of 25 fps. All of them are encoded by using the JM encoder ver. 12.2 in Main Profile at Level 5.0.

As shown in Table I, the new architecture combined with the proposed SE predictor achieves a decoding performance of about 1.2 cycles/bin, which is sufficient to decode HD1080p H.264/AVC bitstream in real-time. Compared with [3], the number of cycles needed to decode a bin has reduced by over 60%. Such a dramatic enhancement is resulted from the effort minimizing the cycle consumption invoked by SESO.

VI. Conclusion

In this paper, we have presented a prediction scheme for CABAC decoding. We have shown that the proposed scheme enables to achieve nearly 1 cycle/bin decoding performance on the slice level in which the decoding performance of a conventional architecture would be significantly degraded by the SESO. To design an efficient pipelined architecture, a thorough investigation on the transition pattern of context models has been conducted as well as some modifications of the entire two-stage pipeline structure. The simulation results show that the proposed scheme achieves such a high performance that is enough to decode HD H.264 bitstream in real-time.

References

[1] ITU-T Recommendation H.264 (ITU-T Rec. H.264 ISO/IEC 14496-10), Mar. 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. Circuits and Syst. Video Technol.*, vol. 13, no. 7, pp. 620 – 636, July 2003

[3] Y. Yi and I. C. Park, "High-Speed H.264/AVC CABAC decoding," *IEEE Trans. Circuits and Syst. Video Technol.*, vol. 17, no. 4, pp. 490-494, Apr. 2007

[4] R. R. Osorio and J. D. Bruguera, "Arithmetic coding architecture for H.264/AVC CABAC compression system," in *Proc. IEEE EUROMICRO Symp. Dig. Syst. Design (DSD 2004)*, Aug.-Sep. 2004, pp.62-69

[5] L. Li, Y. Song, T. Ikenaga, and S. Goto, "A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile," in *Proc. IEEE Asia Pacific Conf. on Circuits and Syst. (APCCAS 2006)*, Dec. 2006, pp. 761 – 764

[6] C.H. Kim and I.C. Park, "High Speed Decoding of Context-based Adaptive Binary Arithmetic Codes Using Most Probable Symbol Prediction," in *Proc. IEEE Int'l Conf. on Circuits and Syst. (ISCAS 2006)*, May. 2006, pp. 1707-1710

[7] P. Zhang, W. Gao, D. Xie, and D. Wu, "High-Performance CABAC Engine for H.264/AVC High Definition Real-Time Decoding," in *Digest of Tech Papers Int'l Conf. on Consum. Elec. (ICCE 2007)*, Jan. 2007, pp. 1-2

TABLE I. Comparison of Decoding Performance with a Previous Architecture

Video stream	Decoded bins	Decoded SEs	Switching ratio[a] (%)	Hit ratio[b] (%)	Average decoding performance (cycle/bin)		Speedup (times)
					Previous[c]	*Proposed*	
Pedestrian area	37011903	30095189	66.8	72.0	3.44	1.23	2.80
Riverbed	73355259	62223028	72.6	71.0	3.54	1.25	2.83
Rush hour	21686743	16461407	66.6	67.8	3.28	1.24	2.65
Station2	31150513	25117991	66.7	72.1	3.42	1.22	2.80
Sunflower	41651890	32342607	69.1	73.4	3.33	1.21	2.75
Tractor	62034903	50654481	71.0	72.2	3.45	1.23	2.80
Average					3.41	1.23	2.77

a. Switching ratio = (number of SE switching) / (number of SE transition)

b. Hit ratio = (number of SE prediction hit) / (number of SE transition)

c. Separated architecture with no SE prediction (the same as in [3])