Philipp Habermann, Chi Ching Chi, Mauricio Álvarez-Mesa, Ben Juurlink

Optimizing HEVC CABAC decoding with a context model cache and application-specific prefetching

Suggested Citation

Habermann, Philipp; Chi, Chi Ching; Álvarez-Mesa, Mauricio; Juurlink, Ben: Optimizing HEVC CABAC decoding with a context model cache and application-specific prefetching. - In: 2015 IEEE International Symposium on Multimedia : ISM. - New York, NY [u.a.]: IEEE, 2015. - ISBN: 978-1-5090-0379-2. - pp. 429–434. - DOI: 10.1109/ISM.2015.97. (*Postprint version is cited, available at http://dx.doi.org/10.14279/depositonce-5749, page numbers differ.*)

Optimizing HEVC CABAC Decoding with a Context Model Cache and Application-specific Prefetching

Philipp Habermann, Chi Ching Chi, Mauricio Alvarez-Mesa and Ben Juurlink
Embedded Systems Architecture Group
Technische Universität Berlin
Berlin, Germany
Email: {p.habermann, chi.c.chi, mauricio.alvarezmesa, b.juurlink}@tu-berlin.de

***Abstract*—Context-based Adaptive Binary Arithmetic Coding is the entropy coding module in the most recent JCT-VC video coding standard HEVC/H.265. As in the predecessor H.264/AVC, CABAC is a well-known throughput bottleneck due to its strong data dependencies. Beside other optimizations, the replacement of the context model memory by a smaller cache has been proposed, resulting in an improved clock frequency. However, the effect of potential cache misses has not been properly evaluated. Our work fills this gap and performs an extensive evaluation of different cache configurations. Furthermore, it is demonstrated that application-specific context model prefetching can effectively reduce the miss rate and make it negligible. Best overall performance results were achieved with caches of two and four lines, where each cache line consists of four context models. Four cache lines allow a speed-up of 10% to 12% for all video configurations while two cache lines improve the throughput by 9% to 15% for high bitrate videos and by 1% to 4% for low bitrate videos.**

***Keywords*-HEVC, H.265, CABAC, Cache, Prefetching**

I. Introduction

High Efficiency Video Coding (HEVC/H.265, [1]) is the most recent video coding standard developed by the Joint Collaborative Team on Video Coding (JCT-VC). It allows the compression of videos with the same perceptive quality as its predecessor H.264/AVC ([2]) while requiring only half the bitrate. Context-based Adaptive Binary Arithmetic Coding (CABAC, [3], [4]) has been a throughput bottleneck in AVC due to its sequential nature, and it still is in HEVC. CABAC operates on the bitstream and decodes binary symbols (bins) which are connected to form syntax elements that are used to control the remaining decoding steps. If possible, context models are used to estimate the probabilities that bins have a specific value. Context-coded bins are associated with context models to exploit statistical properties and thereby increase the compression rate. However, sometimes the probabilities cannot be accurately predicted. Because of that, the decoding of these so called bypass-coded bins goes without context models.

Strong bin-to-bin dependencies make low-level parallelization of CABAC decoding very challenging. Although high-level parallelization is possible in HEVC, it has to be enabled in the encoder which is not mandatory. Beside many other optimizations, the replacement of the context model memory in the data path by a smaller cache has been proposed [5], [6]. This aims to shorten the critical path and increase the clock frequency and throughput. Unfortunately, the effect of potential cache misses has not been properly evaluated. Cache misses result in a performance degradation that might nullify a lot of the throughput improvements reached by the introduction of the cache. Prefetching has been proposed to address this issue [7], but a quantitative evaluation is also missing. In this paper we evaluate both, cache miss rate without and with prefetching, to justify if these are gainful optimizations.

Our work makes the following contributions:

- an optimized cache architecture (as a result of an extensive evaluation of different configurations)
- an efficient context model memory layout regarding spatial locality and prefetching efficiency, as well as the corresponding prefetching algorithm
- an evaluation of prefetching efficiency for different cache configurations

The remaining paper is structured as follows. An overview of related work is provided in Section II. The proposed decoder architecture with a context model cache and a prefetching module is described in Section III. Afterwards, Section IV quantitatively evaluates the cache miss rate and prefetching efficiency for different cache configurations. Finally, our work is concluded in Section V.

II. Related Work

Many implementations of CABAC hardware decoders have been proposed. Although most of them cover AVC, the general ideas are also applicable for HEVC. Additionally, HEVC CABAC is designed to allow higher throughput by various optimizations in the standard, such as a reduced fraction of context-coded bins, grouping of bypass-coded bins, a reduction of the total number of bins as well as more relaxed parsing and context selection dependencies [4].

Two architectural optimizations have proven themselves to be effective and at least one of them can be found in almost every proposed CABAC hardware decoder. The first one is the parallel decoding of multiple bins per clock cycle

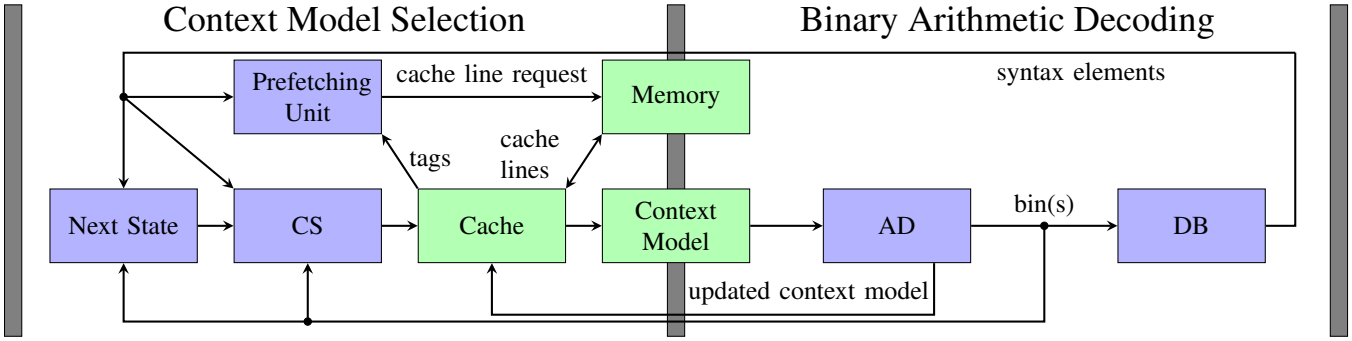

Figure 1: Two-stage pipeline of the proposed decoder architecture.

[8], [9], [10]. It is well-suited for bypass-coded bins because their decoding process is simple and does not require context models. The second optimization is pipelining which is used to overlap the decoding of consecutive bins and thereby increase the throughput. Most proposals agree on a conceptual four-stage pipeline for CABAC decoding: context selection (CS) $\rightarrow$ context load (CL) $\rightarrow$ arithmetic decoding (AD) $\rightarrow$ de-binarization (DB). However, often neighboring stages are merged because the efficient implementation of deep pipelining is very complex for CABAC due to strong bin-to-bin dependencies. For example, CS depends on the results of AD and DB, which might lead to a flush of the complete pipeline. Decoders with three pipeline stages include [5] and [9]. Yu-Hsin Chen et al. [8] propose a very deep pipeline. A fifth pipeline stage is added at the beginning to compute a binary decision tree that is used for state prefetching in the remaining stages and thereby reduces pipeline stalls. Additionally, their implementation decodes up to two bypass-coded bins per cycle, resulting in a throughput of up to 2 Gbin/s which represents the state-of-the-art.

The CL stage can be shortened when the context model memory is replaced by a small cache. The stage might be even removed when the cache fits into an adjacent stage. The shorter pipeline might result in less pipeline stalls. Cached designs have been proposed [5], [6], [7], however the potential performance degradation due to cache misses has not been evaluated. Prefetching was used to reduce the cache miss rate [7], but results for this optimization were also not provided. Our work evaluates both, the cache miss rate and the effectiveness of prefetching.

III. Architecture

The proposed decoder architecture is implemented with a two-stage pipeline (see Figure 1). The Context Selection Stage computes the next state of the decoder control state machine, calculates the index of the required context model (CS) and accesses the context model cache. The cache is clocked with a phase-shifted clock signal to allow the access in the end of the pipeline stage. In the Binary Arithmetic Decoding stage one context-coded bin or up to two bypass-coded bins are decoded by the arithmetic decoder (AD). The decoded bins are fed back to the first stage and the context model is updated and written back to the Cache. Finally, de-binarization (DB) is performed to build syntax elements from the decoded bins.

A. Context Model Cache

A non-cached version of the decoder has been implemented as a reference where the context model memory is directly accessed. The memory contains sixteen memory sets of context models and is capable of fast in-memory copies. This allows the maintenance of multiple context model memory sets and thus supports efficient CABAC decoding when high-level parallelization tools (wavefront processing, tiles) are used. A cache can be used to replace the context model memory in the critical path and thereby allow a higher clock frequency. The cache fetches context model sets (CMS's) from the memory and writes them back when they have to be replaced. In our implementation a cache miss results in a miss penalty of two clock cycles while the missing CMS is loaded from memory. CMS replacement is handled by a least recently used (LRU) policy. The cache is fully-associative and contains a generic number of cache lines (1 to 64) each storing one CMS.

Table I shows the optimized context model memory layout. Mostly, context models for the same type of syntax element are grouped to exploit spatial locality, e.g. $last\ sig\ x/y\ prefix$ (ll. 0-9), $sig\ flags$ (ll. 16-27) and $absG1\ flags$ (ll. 32-37). However, in some cases context models that are not logically connected are put in the same CMS (e.g. ll. 10, 11, 29). The purpose is to have the required context model for sure if the decoder state machine can go to different states. A CMS contains four context models (4×7 bit) because in most cases not more than four different context models are needed to decode a syntax element or a group of consecutive syntax elements of the same type. There are only five cases where more than four context models are needed ($last\ sig\ x\ prefix\ Y\ 32x32$, $last\ sig\ y\ prefix\ Y\ 32x32$, $inter\_pred\_idc$, $sig\ Y\ 4x4$,

0	last sig x prefix Y 4x4 1	last sig x prefix Y 4x4 2	last sig y prefix Y 4x4 1	last sig y prefix Y 4x4 2
1	last sig x prefix Y 4x4 3	transform_skip_flag Y	last sig y prefix Y 4x4 3	sig DC Y
2	last sig x prefix Y 8x8 1	last sig x prefix Y 8x8 2	last sig y prefix Y 8x8 1	last sig y prefix Y 8x8 2
3	last sig x prefix Y 8x8 3	last sig x prefix Y 32x32 5	last sig y prefix Y 8x8 3	last sig y prefix Y 32x32 5
4	last sig x prefix Y 16x16 1	last sig x prefix Y 16x16 2	last sig y prefix Y 16x16 1	last sig y prefix Y 16x16 2
5	last sig x prefix Y 16x16 3	last sig x prefix Y 16x16 4	last sig y prefix Y 16x16 3	last sig y prefix Y 16x16 4
6	last sig x prefix Y 32x32 1	last sig x prefix Y 32x32 2	last sig y prefix Y 32x32 1	last sig y prefix Y 32x32 2
7	last sig x prefix Y 32x32 3	last sig x prefix Y 32x32 4	last sig y prefix Y 32x32 3	last sig y prefix Y 32x32 4
8	last sig x prefix Cb/Cr 1	last sig x prefix Cb/Cr 2	last sig y prefix Cb/Cr 1	last sig y prefix Cb/Cr 2
9	last sig x prefix Cb/Cr 3	transform_skip_flag Cb/Cr	last sig y prefix Cb/Cr 3	sig DC Cb/Cr
10	csb Y 1	csb Y 2	absG2 Y other 1	absG2 Y other 2
11	csb Cb/Cr 1	csb Cb/Cr 2	absG2 Cb/Cr 1	absG2 Cb/Cr 2
12			absG2 Y top-left 1	absG2 Y top-left 2
13	cbf_chroma 1	cbf_chroma 2	cbf_luma 1	cbf_luma 2
14	split_transform_flag 1	split_transform_flag 2	split_transform_flag 3	rqt_root_cbf
15	cbf_chroma 3	cbf_chroma 4	cu_qp_delta_abs 1	cu_qp_delta_abs 2
16	sig Y 8x8 diag top-left 1	sig Y 8x8 diag top-left 2	sig Y 8x8 diag top-left 3	
17	sig Y 8x8 diag other 1	sig Y 8x8 diag other 2	sig Y 8x8 diag other 3	
18	sig Y 8x8 hor/ver top-left 1	sig Y 8x8 hor/ver top-left 2	sig Y 8x8 hor/ver top-left 3	
19	sig Y 8x8 hor/ver other 1	sig Y 8x8 hor/ver other 2	sig Y 8x8 hor/ver other 3	
20	sig Y 4x4 1	sig Y 4x4 2	sig Y 4x4 3	sig Y 4x4 4
21	sig Y 4x4 5	sig Y 4x4 6	sig Y 4x4 7	sig Y 4x4 8
22	sig Y 16x16/32x32 top-left 1	sig Y 16x16/32x32 top-left 2	sig Y 16x16/32x32 top-left 3	
23	sig Y 16x16/32x32 other 1	sig Y 16x16/32x32 other 2	sig Y 16x16/32x32 other 3	
24	sig Cb/Cr 4x4 1	sig Cb/Cr 4x4 2	sig Cb/Cr 4x4 3	sig Cb/Cr 4x4 4
25	sig Cb/Cr 4x4 5	sig Cb/Cr 4x4 6	sig Cb/Cr 4x4 7	sig Cb/Cr 4x4 8
26	sig Cb/Cr 8x8 1	sig Cb/Cr 8x8 2	sig Cb/Cr 8x8 3	
27	sig Cb/Cr 16x16 1	sig Cb/Cr 16x16 2	sig Cb/Cr 16x16 3	
28	cu_skip_flag 1	cu_skip_flag 2	cu_skip_flag 3	pred_mode_flag
29	part_mode 1	part_mode 2	merge_flag	merge_idx
30	inter_pred_idc 1	inter_pred_idc 2	inter_pred_idc 3	inter_pred_idc 4
31	mvp_l0/1_flag	inter_pred_idc 5	part_mode 3	part_mode 4
32	absG1 Y top-left 1.1	absG1 Y top-left 1.2	absG1 Y top-left 1.3	absG1 Y top-left 1.4
33	absG1 Y top-left 2.1	absG1 Y top-left 2.2	absG1 Y top-left 2.3	absG1 Y top-left 2.4
34	absG1 Y other 1.1	absG1 Y other 1.2	absG1 Y other 1.3	absG1 Y other 1.4
35	absG1 Y other 2.1	absG1 Y other 2.2	absG1 Y other 2.3	absG1 Y other 2.4
36	absG1 Cb/Cr 1.1	absG1 Cb/Cr 1.2	absG1 Cb/Cr 1.3	absG1 Cb/Cr 1.4
37	absG1 Cb/Cr 2.1	absG1 Cb/Cr 2.2	absG1 Cb/Cr 2.3	absG1 Cb/Cr 2.4
38	sao_merge_flag	sao_type_idx	prev_intra_luma_pred_flag	intra_chroma_pred_mode
39	split_cu_flag 1	split_cu_flag 2	split_cu_flag 3	cu_transquant_bypass_flag
40	abs_mvd_greater0_flag	abs_mvd_greater1_flag	ref_idx_l0/1 1	ref_idx_l0/1 2

Table I: Context Model Memory Layout (Y: luma, Cb/Cr: chroma, SAO, coding quadtree, CU and intra PU, inter PU, transform tree, last_sig_coeff_x/y_prefix, significance flags (sig: sig_coeff_flag, csb: coded_sub_block_flag), coefficient level flags (absG1/2: coeff_abs_level_greater1/2_flag, transform skip flags.

$sig\ Cb/Cr\ 4x4$). In all other cases the required context models fit in a CMS. With a smaller CMS size, the required context models for 4×4 transform sub-blocks do not fit because at least three sig_coeff_flags and four $coeff\_abs\_level\_greater1\_flags$ are potentially needed. This is a critical issue as transform block decoding contains a high fraction of the decoded bins. Bigger CMS sizes require that context models for different types of syntax elements are merged to keep the memory overhead small. Unfortunately, the context models that are used for the decoding of consecutive groups of equal syntax elements often depend on different parameters. For example, sig_coeff_flags depend on the transform block size and scan pattern while $coeff\_abs\_level\_greater1\_flags$ depend on the decoded bins in the previous 4×4 sub-block.

B. Context Model Prefetching

Application-specific context model prefetching can significantly reduce the miss rate in HEVC CABAC. Admittedly, the required context model often depends on the results of the previously decoded bin and cannot be prefetched early enough. However, most of the time one can be sure that the required context model is contained in a specific set of context models. If this set is available in the cache, it is not necessary to know the exact context model, but a hit is still guaranteed.

The prefetching module reads the current state of the control state machine, the decoder settings, some decoded syntax elements and the currently decoded bin. Based on this information, it selects up to two CMS candidates that are likely to be needed soon. As the module keeps track

of the CMS's in the cache, it can see if they are already present. If one is not, a read request is sent to the context model memory. The first candidate has a higher priority than the second, so the second is only fetched when the first is already available. Unfortunately this can lead to a behavior where the first line is available and will be replaced by the second because it is the next to be replaced according to LRU. A refresh mechanism is implemented to avoid this behavior. This is done by resetting the LRU index of the first context model set candidate if it is already in the cache. As a result, it will not be replaced next. The prefetching strategy depends on the number of available cache lines (CLs). The strategy for at least four CLs fetches CMS's that are likely to be used soon, while for smaller caches only the CMS's are fetched that will be used for sure. Prefetching with one CL can only be used when no CMS is currently in use or if it is known when it will not be needed anymore. Also the second candidate is not used by the strategy for one CL.

IV. Evaluation

A hybrid HEVC decoder has been realized as hardware-software co-design on the Xilinx Zynq-7045 SoC to validate the functionality of the proposed CABAC hardware decoder. The CABAC decoder is implemented in the FPGA while the remaining parts are executed on the ARM CPU. The highly optimized HEVC software decoder developed by the Embedded Systems Architecture Group at TU Berlin is used [11]. Five test sequences from the JCT-VC class B test set (1080p) serve for evaluation. They are encoded in all-intra (AI) and random-access (RA) mode with quantization parameters (QP) of 14, 22, 30 and 38. Wavefront Parallel Processing is enabled so that the same context models are used for a row of thirty CTUs. In the remaining paper the arithmetic mean of the results for the test sequences is shown.

The remaining evaluation section is structured as follows. First, the impact of different cache sizes on the clock frequency is shown to provide an upper boundary for the overall speed-up. Afterwards, the cache miss rate without prefetching is presented to illustrate that a cache does not improve the overall throughput without further measures. Finally, it is demonstrated that the miss rate can be significantly reduced when application-specific context model prefetching is used, resulting in different overall speed-ups depending on the cache size.

A. Clock Frequency

The purpose of replacing the context model memory in the data path by a smaller cache is to shorten the critical path and thereby increase the achievable clock frequency and throughput. The proposed design has been synthesized with Xilinx Synthesis Technology 14.6 (optimization goal: speed). Both, the memory and the cache, are forced to be synthesized with the same FPGA resources to get a fair comparison that is not only valid for FPGAs. The influence of the cache size on the maximum clock frequency of the decoder can be seen in Figure 2. It is increased by 20.1% for a single CL, by 17.3% for two CLs and by 12.3% for four CLs. While there is no significant improvement for eight CLs (2.8%), the clock frequency is reduced for bigger caches. The rapid clock frequency reduction comes from the LRU implementation and CL selection which are not well suited for bigger caches with full associativity. It should also be noted that these results can vary for different implementations, e.g. shorter pipeline stages can lead to greater relative improvement.

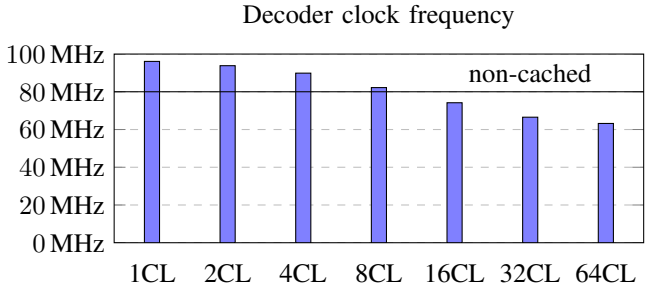

Figure 2: Decoder clock frequency for a different number of cache lines. The horizontal line shows the clock frequency for the non-cached design.

While the improved clock frequency accelerates the decoding of all bins, only context-coded bins can result in cache misses. The fraction of context-coded bins in the test sequences varies from 64% to 77%. However, as up to two bypass-coded bins can be decoded in parallel, the decoding time fraction for context-coded bins is slightly increased (74% to 83%).

B. Performance without Prefetching

Unfortunately the improved clock frequency comes at the cost of potential cache misses. They lead to stalls in the decoder pipeline and thereby reduce the overall throughput. Figure 3 (top) presents the cache miss rate for different cache sizes and video modes. In general, the miss rate grows with higher QPs. This is due to the fact that smaller QPs result in more bins because of less quantization. As bins of the same syntax elements are grouped, temporal and spatial locality can be better exploited when accessing the required context models in the cache. A significant miss rate reduction can be observed for all video modes when the number of CLs is increasing. However, there is no noticeable improvement with 64 CLs where all required context models for the decoding of a specific CTU fit in the cache. This means that all resulting cache misses are cold misses during the first access. Often not all context models are used during the decoding of a CTU, especially if only few bins are decoded as in the RA QP38 configuration. In this case 32 CLs are also sufficient and lead to the same results as 64 CLs.

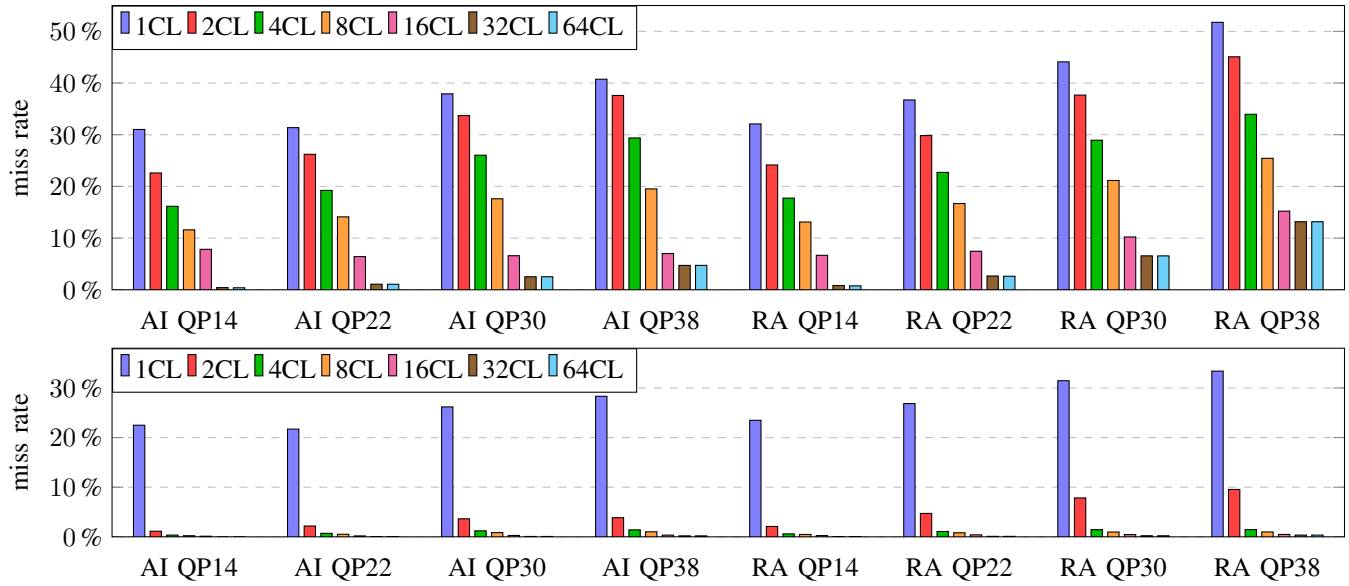

Figure 3: Cache miss rate without prefetching (top) and with prefetching (bottom).

As there are high miss rates for few CLs and reduced clock frequencies for more than eight CLs, an overall performance improvement cannot be reached without prefetching. For example with an AI QP14 video and two CLs (74% context-coded bin cycle fraction, 17.3% higher clock frequency, 22.6% miss rate) the overall performance is only 88% of a non-cached decoder.

C. Performance with Prefetching

Our prefetching algorithm significantly reduces the cache miss rate (see Figure 3 (bottom)). The miss rate with only one CL is still not acceptable as it is greater than 20% for all configurations because of the restricted prefetching opportunities. Two CLs already result in significant improvements that depend on the video mode. For all AI modes and for the low QP RA modes the miss rate is less than 5%, but it almost reaches 10% for the high QP RA modes. With four and more CLs the miss rate is less than 1.5% for all modes. As a result the decoder performance is not noticeably affected by the cache miss rate anymore (less than 2.5% performance reduction) and almost the full gain of the clock frequency improvement remains.

Figure 4 shows the speed-up over the non-cached design, but only for the cache sizes where an improvement was achieved. The miss rate for a single CL is too high to result in an overall speed-up (10% to 23% reduction depending on the video mode) while more than eight CLs cannot improve the throughput due to the reduced clock frequency. 16, 32 and 64 CLs result in a throughput reduction of 8%, 17% and 21%. The cache with eight CLs combines a miss rate of at most 1% and a small clock frequency improvement of 2.8%, leading to a 1.0% to 2.5% throughput improvement.

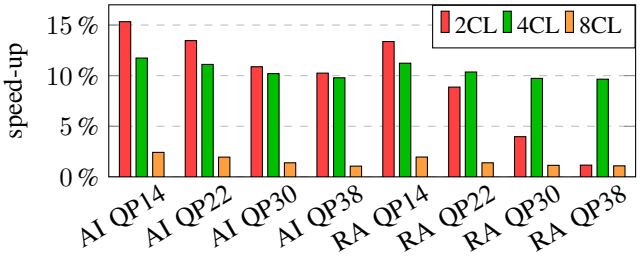

Figure 4: Speed-up with prefetching (compared to non-cached decoder).

	AI				RA			
QP	**14**	**22**	**30**	**38**	**14**	**22**	**30**	**38**
2CL	105.0	100.3	93.7	88.4	97.4	90.7	84.3	78.4
4CL	101.7	98.1	92.9	87.7	95.5	91.7	88.3	83.8

Table II: Decoder throughput in Mbins/s.

Four CLs allow a consistent speed-up of 9.7% to 11.7% as the miss rate is only negligibly higher than with eight CLs but the clock frequency allows up to 12.3% speed-up. Two CLs allow even higher throughput, but only for AI modes (10.3% to 15.3%) and RA QP14 mode (13.4%). For the other RA modes the miss rate is too high and leads to an overall improvement of 1.2% to 8.9%. However, CABAC throughput is not critical for the decoding of high QP videos as only few bins are processed. That is why the configuration with two CLs can still be the preferred option for a general hardware decoder. Table II presents the absolute throughput for the designs with two and four CLs. It can be seen that the designs are capable of decoding 90 to 105 Mbins/s for high bitrate bitstreams.

Design	Registers	LUTs	BRAMs	DSPs
non-cached	3,055	7,542	15	1
(area opt)	0.23%	1.15%	2.75%	0.11%
cached 2cl	3,240	8,070	15	1
(area opt)	0.25%	1.23%	2.75%	0.11%
cached 4cl	3,391	8,280	15	1
(area opt)	0.26%	1.26%	2.75%	0.11%
cached 2cl	3,234	8,819	15	1
(speed opt)	0.25%	1.34%	2.75%	0.11%
cached 4cl	3,398	8,811	15	1
(speed opt)	0.26%	1.34%	2.75%	0.11%
available	1,311,600	655,800	545	900

Table III: Resource utilization on the Xilinx Zynq-7045 SoC.

D. Resource Utilization

Table III compares the resource utilization of the non-cached CABAC decoder with the cached designs with two and four CLs. Synthesis has been performed with area optimization to get meaningful results for a comparison between the different designs. Results with speed optimization are also provided to allow a fair comparison with other implementations. Two main conclusions can be drawn from the results. First, the cached designs (2CL and 4CL) require only 6% and 11% more Registers, as well as 7% and 10% more LUTs compared to the non-cached design. Second, less than 3% of the FPGA resources are needed to implement the CABAC decoder including the processor interface.

V. Conclusions

A quantitative performance analysis of an HEVC CABAC decoder has been conducted in this paper. We focused on the evaluation of the miss rate when the context model memory is replaced by a smaller cache. While this replacement results in significant clock frequency improvements for small cache sizes, the emerging cache misses nullify the effect. However, the cache miss rate can be effectively reduced with a well-designed context model memory layout and the corresponding prefetching strategy.

The configurations with two and four CLs are most promising. Four CLs result in a speed-up of 10% to 12% due to effective prefetching and a solid clock frequency improvement. Two CLs allow even higher clock frequencies but the miss rate is also higher, especially for high QP RA videos. As a result the design with two CLs outperforms the design with four CLs for high bitrate videos (9% to 15% speed-up) but not for low bitrates (1% to 4% speed-up). However, as CABAC throughput is not critical for the latter, the design with two CLs can still be the preferred option.

Despite the direct throughput improvement due to the enhanced clock frequency, other designs might remove the pipeline stage that performs the context model memory access when the cache can be shifted to an adjacent stage. The shortened pipeline might also significantly improve the throughput as the strong dependencies in CABAC decoding make deep pipelining inefficient.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 645500 (Film265).

References

[1] G. J. Sullivan, J. Ohm, W.-J. Han and T. Wiegand, *"Overview of the High Efficiency Video Coding (HEVC) Standard"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 22, Issue 12, pp. 1649-1668, September 2012

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, *"Overview of the H.264/AVC Video Coding Standard"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 13, Issue 7, pp. 560-576, July 2003

[3] D. Marpe, H. Schwarz and T. Wiegand, *"Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 13, Issue 7, pp. 620-636, July 2003

[4] V. Sze and M. Budagavi, *"High Throughput CABAC Entropy Coding in HEVC"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 22, Issue 12, pp. 1778-1791, October 2012

[5] Y. Yi and I.-C. Park, *"High-Speed H.264/AVC CABAC Decoding"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 17, Issue 4, pp. 490-494, April 2007

[6] Y.-C. Yang and J.-I. Guo, *"High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 19, Issue 9, pp. 1395-1399, September 2009

[7] Y. Hong, P. Liu, Z. Hang, Z. You, D. Zhou and S. Goto, *"A 360 Mbin/s CABAC Decoder for H.264/AVC Level 5.1 Applications"*, 2009 IEEE International SoC Design Conference (ISOCC 2009), pp. 71-74, Busan, South Korea, November 2009

[8] Y.-H. Chen and V. Sze, *"A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-tier Applications"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 25, Issue 5, p. 856-868, May 2015

[9] C.-H. Kim and I.-C. Park, *"High Speed Decoding of Context-based Adaptive Binary Arithmetic Codes using Most Probable Symbol Prediction"*, IEEE International Symposium on Circuits and Systems (ISCAS 2006), pp. 1707-1710, Island of Kos, Greece, May 2006

[10] P.-C. Lin, T.-D. Chuang and L.-G. Chen, *"A Branch Selection Multi-symbol High Throughput CABAC Decoder Architecture for H.264/AVC"*, IEEE International Symposium on Circuits and Systems (ISCAS 2009), pp. 365-368, Taipei, Taiwan, May 2009

[11] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, T. Schierl, *"SIMD Acceleration for HEVC Decoding"*, IEEE Transactions on Circuits and Systems for Video Technology, Volume 25, Issue 5, p. 841-855, May 2015