High-Throughput HEVC CABAC Decoding

vorgelegt von
M.Sc.
Philipp Habermann

an der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Giuseppe Caire, Ph.D.
Gutachter:	Prof. Dr.-Ing. Thomas Sikora
Gutachter:	Prof. Dr.-Ing. Benno Stabernack
Gutachter:	Dr.-Ing. Jens Brandenburg

Tag der wissenschaftlichen Aussprache: 02.11.2020

Berlin 2020

Abstract

Video applications have emerged in various fields of our everyday life. They have continuously enhanced the user experience in entertainment and communication services. All this would not have been possible without the evolution of video compression standards and computer architectures over the last decades. Modern video codecs employ sophisticated algorithms to transform raw video data to an intermediate representation consisting of syntax elements, which allows enormous compression rates before reconstructing the video with minimal objective quality losses compared to the original video. Modern computer architectures lay the foundation for these computationally intensive tasks. They provide multiple cores and specialized vector architectures to exploit the massive amount of parallelism that can be found in video applications. Customized hardware solutions follow the same principles. Parallel processing is essential to satisfy real-time performance constraints while optimizing energy efficiency, the latter being the most important design goal for mobile devices.

One of the main tasks in modern video compression standards implements a highly sequential algorithm and lacks data-level parallelism in contrast to all other compute-intensive tasks: Context-based Adaptive Binary Arithmetic Coding (CABAC). It is the entropy coding module in the state-of-the-art High Efficiency Video Coding (HEVC) standard and also its successor Versatile Video Coding. Its purpose is the compression and decompression of the intermediate video representation by exploiting statistical properties, thus achieving minimal bitrates. CABAC is one of the main throughput bottlenecks in video coding applications due to the limited parallelization opportunities, especially for high-quality videos. Close-distance control and data dependencies make CABAC even more challenging to implement with modern computer architectures. This thesis addresses the critical CABAC decoding throughput bottleneck by proposing multiple approaches to uncover new parallelization opportunities and to improve the performance with architectural optimizations.

First of all, we quantitatively verify the severity of the CABAC decoding throughput bottleneck by evaluating the HEVC decoding performance for various workloads using a representative selection of state-of-the-art computer architectures. The results show that even the most powerful processors cannot provide real-time performance for several high-quality workloads. The profiling results clearly show that CABAC decoding is the main reason for that in most cases.

Wavefront Parallel Processing (WPP) is a well-established high-level parallelization technique used in video coding and other applications. It can lead to a high degree of parallelism, however, it suffers from inefficiencies due to the dependencies between consecutive rows in a frame. We present three WPP implementations for HEVC CABAC

decoding with improved parallel efficiency. The WPP versions based on more fine-grained dependency checks allow speed-ups up to 1.83× at very low implementation cost.

We also present a bitstream partitioning scheme for future video compression standards. It enables additional parallelism in CABAC decoding by distributing syntax elements among eight bitstream partitions. Besides the parallelization opportunities, this allows specialization of the subdecoders responsible for the processing of their corresponding partitions as they have to process fewer types of syntax elements. This leads to further improvements in clock frequency and significant hardware savings compared to a full replication of the CABAC decoder as it is required for approaches such as WPP. Decoding speedups up to 8.5× at the cost of only 61.9% extra hardware area and less than 0.7% bitstream overhead for typical Full High Definition videos make this technique a promising candidate for use in future video compression standards.

Furthermore, a cache-based architectural optimization is presented. It replaces the context model memory – a critical component in the CABAC decoder pipeline – by a smaller cache, thus increasing the achievable clock frequency. An application-specific adaptive prefetching algorithm is used together with a context model memory layout optimized for spatial and temporal locality. We perform a design space exploration of different cache configurations, finding that a cache of 2×8 context models provides the best performance. It allows for a 17% increase in clock frequency and miss rates of less than 2%, resulting in performance improvements up to 16.7%.

We also propose techniques for more efficient CABAC decoding on general-purpose processors. Frequent hardly predictable branches lead to very inefficient implementations with these processors. Using more complex but linear arithmetic functions for the parallel decoding of binary symbols provides a speedup of up to 2.04×. A separate bitstream partition for this type of binary symbol even allows speedups up to 2.45× at the cost of not more than 0.2% higher bitrate for typical Full High Definition videos.

Finally, we provide recommendations for future video compression standards and computer architectures as well as further research ideas for video coding in general and CABAC in particular. The research conducted in this thesis shows multiple approaches that can substantially improve the performance of CABAC decoding, thereby addressing one of the most critical throughput bottlenecks in modern video coding applications.

Zusammenfassung

Videoanwendungen haben sich in vielen Bereichen unseres täglichen Lebens etabliert und dabei die Nutzererfahrung in den Bereichen Unterhaltung und Kommunikation zunehmend verbessert. Das wäre ohne die ständige Weiterentwicklung von Videokompressionsstandards und Computerarchitekturen nicht möglich gewesen. Moderne Videocodecs nutzen komplexe Algorithmen, um rohe Videodaten in eine aus Syntaxelementen bestehende Zwischenrepräsentation zu transformieren, was enorme Kompressionsraten erlaubt. Die anschließende Rekonstruktion der Videodaten kann mit minimalen Qualitätsverlusten im Vergleich zum Originalvideo durchgeführt werden. Modern Computerarchitekturen legen die Grundlage für diese rechenintensiven Prozesse. Sie stellen zahlreiche Rechenkerne und spezialisierte Vektorarchitekturen zur Verfügung, welche die zahlreichen Parallelisierungsmöglichkeiten in Videoanwendungen ausnutzen. Die parallele Datenverarbeitung ist essenziell, um die Echtzeitfähigkeit zu gewährleisten und gleichzeitig die Energieeffizienz zu optimieren, was insbesondere für Mobilgeräte eines der wichtigsten Entwicklungsziele darstellt.

Context-based Adaptive Binary Arithmetic Coding (CABAC) ist das Entropiekodierungsverfahren im aktuellen High Efficiency Video Coding (HEVC) Standard, sowie in dessen Nachfolger Versatile Video Coding. CABAC ist für die Kompression und Dekompression der Zwischenrepräsentation eines Videos unter Ausnutzung statistischer Gegebenheiten verantwortlich, wodurch minimale Bitraten erreicht werden können. Dafür wird ein sequentieller Algorithmus verwendet, der CABAC im Vergleich zu allen anderen rechenintensiven Komponenten aktueller Videokompressionsstandards keine Ausnutzung von Datenparallelität ermöglicht. Durch die mangelnden Parallelisierungsmöglichkeiten ist CABAC eine der kritischsten Komponenten, welche die Gesamtleistung eines Videodekoders beschränken. Das gilt insbesondere für Videos mit hoher Qualität und dementsprechend hohen Bitraten. Außerdem stelle eine Vielzahl an Steuer- und Datenabhängigkeiten in CABAC moderne Computerarchitekturen vor große Herausforderungen. Das Ziel dieser Doktorarbeit ist die Verbesserung der Leistung des CABAC-Dekoders, da er die Gesamtleistung aktueller Videodekoder maßgeblich beeinflusst. Wir stellen dafür verschiedene Ansätze vor, die einerseits neue Parallelisierungsmöglichkeiten schaffen und andererseits durch architekturelle Optimierungen effizientere Implementierungen ermöglichen.

Zuerst verifizieren wir quantitativ, dass CABAC für den Dekodierungsprozess in HEVC eine kritische Komponente ist. Dafür analysieren wir die Dekodierleistung einer repräsentativen Auswahl aktueller Computersysteme für verschiedene typische Videoanwendungen. Die Ergebnisse zeigen, dass selbst die performantesten Prozessoren nicht für alle Anwendungen echtzeitfähig sind. Weitere Untersuchungen bestätigen deutlich, dass CABAC in den meisten Fällen dafür hauptverantwortlich ist.

Anschließend beschäftigen wir uns mit der Optimierung von Wavefront Parallel Processing (WPP). Dabei handelt es sich um eine weit verbreitete Parallelisierungstechnik, die in der Videokodierung und vielen anderen Anwendungen verwendet wird. WPP erlaubt ein hohes Maß an Parallelisierung, erleidet aber wegen der Abhängigkeiten zwischen benachbarten Bildbereichen Einbußen in seiner Effizienz. Wir stellen drei Implementierungsvarianten vor, die die Effizienz der Parallelisierung mit WPP für CABAC in HEVC deutlich verbessern. Dies wird durch eine feingranularere Prüfung von Abhängigkeiten im Vergleich zu konventionellen WPP-Implementierungen erreicht. So kann die Dekodierung von Videos um einen Faktor von bis zu 1.83× beschleunigt werden, während die Implementierung nur unwesentlich komplexer wird.

Dann stellen wir ein Bitstreampartitionierungsschema für zukünftige Videokompressionsstandards vor, welches zusätzliche Parallelisierungsmöglichkeiten schafft. Dies wird durch die Aufteilung aller Syntaxelemente unter Berücksichtigung ihrer Abhängigkeiten auf acht Partitionen erreicht. Zusätzlich ermöglicht dies deutliche Erhöhungen der Taktfrequenz eines Hardwaredekoders, da die spezialisierten Teildekoder für die verschiedenen Partitionen weitaus weniger verschiedene Syntaxelemente bearbeiten müssen. Die reduzierte Komplexität der Teildekoder erlaubt außerdem drastische Hardwareeinsparungen, vor allem im Vergleich zu Techniken wie WPP, die eine vollständige Replikation des CABAC-Dekoders erfordern. Der vorgestellte Dekoder erlaubt eine Beschleunigung um bis zu 8.5× bei lediglich 61.9% zusätzlichen Hardwarekosten und einer Erhöhung der Bitrate um maximal 0.7% bei typischen Full-HD-Videos.

Außerdem stellen wir einen Cache-basierten CABAC-Dekoder vor. Dieser ersetzt den Context-Model-Speicher durch einen kleineren Cache und ermöglicht somit den Betrieb mit höheren Taktfrequenzen, da der Speicherzugriff den kritischen Pfad beeinflusst. Die auftretenden Fehlzugriffe auf den Cache werden mit einem optimierten Speicherlayout und einem adaptiven Vorhersagealgorithmus effektiv reduziert. Die Untersuchung verschiedener Cache-Architekturen zeigt, dass ein 2×8 Context-Model-Cache die beste Leistung liefert. Durch die Erhöhung der Taktfrequenz um 17% und eine Fehlzugriffsrate von maximal 2% kann der Durchsatz des Dekoders um bis zu 16.7% erhöht werden.

Die letzte vorgestellte Optimierung behandelt die Software-CABAC-Dekodierung. Der Algorithmus beinhaltet viele schwer vorhersagbare Verzweigungen im Steuerfluss, was für aktuelle Prozessoren eine große Herausforderung darstellt und zu ineffizienten Implementierungen führt. Der Einsatz komplexer arithmetischer Instruktionen zur parallelen Dekodierung führt zu einer Beschleunigung bis zu 2.04×. Die Nutzung von zwei Bitstreampartitionen für verschiedene Arten von binären Symbolen ermöglicht es sogar, die Dekodierung einer davon ohne Rechenaufwand durchzuführen. Folglich ist eine noch höhere Beschleunigung bis zu 2.45× bei höchstens 0.2% höherer Bitrate möglich.

Abschließend sprechen wir Empfehlungen für die Entwicklung zukünftiger Videokompressionsstandards und Computerarchitekturen aus. Weitere Forschungsideen für Videokodierung im Allgemeinen und CABAC im Besonderen werden ebenfalls diskutiert. Die dieser Arbeit zugrunde liegende Forschung demonstriert bereits einige vielversprechende Ansätze, welche die Leistung von CABAC und damit des gesamten Dekoders deutlich erhöhen können. Durch die Behandlung dieser kritischen Komponente leisten wir einen wichtigen Beitrag zur Verbesserung vieler aktueller und zukünftiger Videoanwendungen.

Acknowledgement

This thesis is the result of many years of hard work. It would not have been possible without the help of a couple of people.

First of all, I want to thank my supervisor Prof. Dr. Ben Juurlink for inspiring me as a student which led me to start a research career. I am also very grateful for his scientific guidance during my time as a research assistant in the Embedded Systems Architecture group at TU Berlin.

I also want to thank Dr. Mauricio Alvarez-Mesa and Chi Ching Chi for their support even after they left our group to found their own company. They brought me in touch with video coding which turned out to be a perfect match. Both are experts in this field and guided me to acquire high technical proficiency as well as a high-quality research methodology.

Thank you to all my colleagues in the Embedded Systems Architecture group. It was the nice working atmosphere and the mutual support that enabled our accomplishments and the persistence in our long-term goals. Special thanks go to Angela Pohl and Matthias Göbel with whom I have walked the long road towards a Ph.D. together from the beginning. We went through similar struggles and always supported each other. I am happy and proud that we all made it.

Last and most importantly I want to thank my parents Birgit and Heiko, and my brother Paul for their support during the stressful and frustrating phases while pursuing a Ph.D.. I want to thank my parents for raising me the way they did, for their unconditional love no matter how much I challenged them, and for supporting me in all the things I aspired in life. It is an insufferable loss that my mother unexpectedly passed away not even a year ago and will not be able to witness so many good things in life that she deserves, the completion of this thesis just being one of them. I dedicate this thesis to her to honor her life's work – our family – which made this accomplishment possible in the first place.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objectives . . . 3
1.3 Contributions . . . 4
1.4 Thesis Organization . . . 6

2 Background **9**
2.1 Video Coding . . . 9
2.1.1 High Efficiency Video Coding . . . 10
2.1.2 Other Video Coding Standards . . . 15
2.1.3 Parallelism in Video Coding . . . 16
2.2 Performance Evaluation . . . 18
2.2.1 Experimental Setup . . . 20
2.2.2 Evaluation . . . 22
2.2.3 Conclusions and Recommendations . . . 26
2.3 Context-based Adaptive Binary Arithmetic Coding . . . 27
2.3.1 Binary Arithmetic Coding . . . 27
2.3.2 Context Modeling and Adaptation . . . 30
2.3.3 Binarization . . . 31
2.3.4 HEVC CABAC . . . 32
2.3.5 Other Entropy Coding Methods . . . 34

3 Related Work **37**
3.1 CTU-level Parallelism . . . 37
3.2 Bin-level Parallelism . . . 39
3.3 Architectural Optimizations . . . 41
3.3.1 Pipelining . . . 42
3.3.2 Context Model Cache . . . 43
3.3.3 Data Path Optimizations . . . 43
3.4 Summary . . . 44

4 Wavefront Parallel Processing **45**
4.1 Dependency Analysis . . . 46
4.2 Fine-grained WPP . . . 47
4.3 Evaluation . . . 48
4.3.1 Decoding Speedup . . . 48
4.3.2 Parallel Efficiency . . . 50
4.4 Conclusions . . . 51

5 Bin-based Bitstream Partitioning **53**
5.1 Bitstream Format . . . 54
5.2 Parallel Decoder Architecture . . . 57
5.2.1 CC Bin Subdecoder . . . 59
5.2.2 BC Bin Subdecoder . . . 59
5.2.3 Improvements over Sequential Decoding . . . 61
5.2.4 Design Challenges . . . 63
5.3 Evaluation . . . 64
5.3.1 Parallel Decoding Speedup . . . 64
5.3.2 Bitstream Overhead . . . 65
5.3.3 Hardware Cost . . . 70
5.4 Conclusions . . . 70

6 Context Model Cache and Prefetching **73**
6.1 Decoder Architecture . . . 74
6.1.1 Context Model Cache and Memory Layout . . . 74
6.1.2 Adaptive Prefetching . . . 79
6.2 Evaluation . . . 81
6.2.1 Clock Frequency . . . 81
6.2.2 Performance without Prefetching . . . 82
6.2.3 Performance with Prefetching . . . 85
6.2.4 Real-time Decoding . . . 89
6.2.5 Resource Utilization . . . 91
6.2.6 Energy Efficiency . . . 92
6.3 Conclusions . . . 92

7 CABAC Decoding on GPPs **95**
7.1 HEVC BC Bin Decoding on GPPs . . . 95
7.2 Parallel BC Bin Decoding . . . 97
7.2.1 Bypass-Bin Reservoir . . . 98
7.2.2 Division by Inverse Multiplication . . . 99
7.2.3 Bitstream Partitioning . . . 100
7.3 Evaluation . . . 101
7.4 Conclusions . . . 102

8 Conclusions **105**
8.1 Summary 105
8.2 Recommendations and Future Work 108

List of Figures **112**

List of Tables **113**

List of Listings **115**

List of Abbreviations **117**

Bibliography **119**
Publications 119
Standards, Software, Testsets 119
Video Coding 121
Entropy Coding 122
Miscellaneous 125

CHAPTER 1

Introduction

Video is widespread in many applications on a variety of computing systems. The emergence of video-on-demand services, IP television, video chatting and conferencing, and others has pervaded many types of digital systems. 75% of all internet traffic has been video data in 2017 according to the Cisco Visual Networking Index and it is predicted to increase up to 82% in 2022 [76]. Video coding standards define a format to represent video data efficiently, such that its storage and transmission over a network becomes feasible. Only five minutes of raw video data can be stored on a 50 GB dual-layer Blu-ray disc when Full High Definition resolution (FHD, 1920×1080 samples), 50 frames per second and a basic color format are used. This is reduced to a few seconds with higher resolutions and color depths. Consequently, video compression becomes inevitable to realize all kinds of entertainment applications. The ever increasing demand for higher video resolutions (4K/8K) and higher quality levels makes video compression a very important research field now and in the future.

1.1 Motivation

Video coding standards have evolved in the last decades and constantly improved their compression efficiency (see Figure 1.1). High Efficiency Video Coding (HEVC/H.265 [8] [9]) is the latest video coding standard developed by the Joint Collaborative Team on Video Coding (JCT-VC). It provides a 50% decrease in bitrate compared to its predecessor H.264/AVC [14] at the same subjective quality [26]. The potential for another 40% bitrate reduction for HEVC's in-development successor Versatile Video Coding (VVC [17]) was confirmed after the Joint Video Exploration Team studied the performance of new coding tools [36]. Modern video codecs provide compression rates between 100× and 1,000× without a noticeable difference in subjective quality compared to the uncompressed video. This is mainly achieved by sophisticated algorithms that remove spatial and temporal redundancies in similar frame areas. It is also exploited that the human visual system is insensitive to specific information, which can be removed without affecting the perceived quality. However, these algorithms have a very high computational complexity and push even modern high-performance computer architectures to their limits. Video coding applications and computer architectures

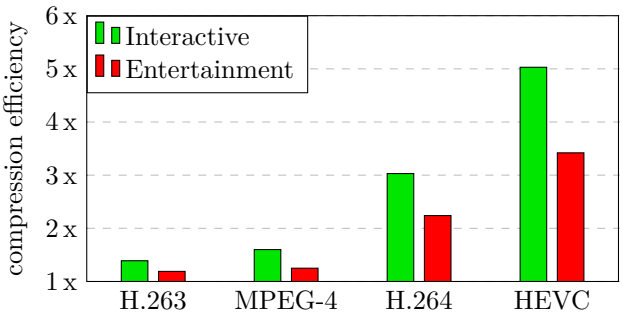


Figure 1.1: Average improvement in compression efficiency for interactive and entertainment applications of previous video coding standards compared to MPEG-2. Results derived from [26].

have mutually influenced each other during the last decades. Modern video compression standards such as HEVC employ multiple high-level parallelization tools to exploit the capabilities of today's multi- and many-core architectures. It is inevitable to make use of these thread-level parallelism (TLP) opportunities to implement high-performance and energy-efficient video applications. Furthermore, many video compression algorithms are designed to allow high-throughput processing by relaxing some data and control dependencies even though this might introduce small losses in compression efficiency. On the other hand, modern general-purpose processors (GPPs) provide wide vector extensions to efficiently process the massive amount of data-level parallelism (DLP) that can be found in video processing applications. Modern GPPs show real-time decoding capabilities for high-quality 4K video material, however, with poor energy efficiency. Therefore, dedicated video encoding and decoding hardware is used in many computing systems, especially mobile devices, to drastically reduce their energy consumption. These mobile devices heavily rely on maximized energy and compression efficiency while real-time performance is a minimum requirement.

One component in HEVC and also in other video compression standards is often a throughput bottleneck, especially for high-quality videos: context-based adaptive binary arithmetic coding (CABAC [44]). The CABAC decoder extracts syntax elements from the compressed bitstream and controls all other decoder modules with this information. It implements a strictly sequential algorithm with close-distance data and control dependencies which makes optimizations very challenging. In contrast to all prediction, transform and filtering modules in HEVC, it does not contain any DLP. As a result, CABAC can neither benefit from the vector extensions of modern GPPs nor from vectorization approaches in customized hardware. This strictly limits the overall speedup that can be achieved with vectorization. Amdahl's Law suggests that the maximum application speedup is determined by the non-parallelizable fraction [77]. CABAC decoding can easily account for 20% to 40% of the runtime for high-quality videos in a non-vectorized decoder, in extreme cases even more than 80%. Consequently, the overall vectorization speedup is limited to 5×, 2.5× and 1.25× respectively, although much higher improvements can be reached for the other components.

The performance of video decoding is more critical than video encoding in most applications. Especially in video-on-demand services, playback from Blu-ray discs, etc., a video needs to be encoded only a few times while decoding can easily be done billions of times. The performance of the encoding process does not matter too much as it can be done offline. On the other hand, real-time performance is essential on the decoder side and energy efficiency is also of utmost importance considering that the number of mobile devices used for video playback is rapidly increasing. Applications with real-time encoding requirements, such as digital cameras and video communication, can always aim for a lower compression rate to reduce the performance requirements. This tradeoff makes insufficient encoder performance less critical than at the decoder side. Furthermore, the inefficient compression increases the workload for the decoder. As a matter of fact, the efficient implementation of CABAC is more challenging in the decoder in general as the currently decoded syntax element determines which one needs to be decoded next. Compared to that, the next syntax elements are always known in the CABAC encoder which allows more efficient pipelining strategies. This thesis is focused on addressing the critical CABAC decoding throughput bottleneck that limits the performance of many video applications.

1.2 Objectives

In this thesis, we develop and evaluate different optimization strategies for HEVC CABAC decoding. We also propose modifications in the HEVC standard that allow more efficient CABAC decoding in future video coding standards.

Objective 1: Identify main performance bottlenecks in video decoding.
The performance of the overall video decoder determines the quality of experience in video-based applications. Therefore, an analysis of the work distribution between the different decoder modules is essential to identify bottlenecks that need to be addressed. Video coding applications are especially challenging in this regard as the work distribution highly depends on the video content, characteristics and quality.

Objective 2: Improve the performance and efficiency of CABAC decoding.
Based on the results of *Objective 1*, CABAC decoding has been identified as the main performance bottleneck for high-quality video decoding. Consequently, we will focus on optimizing CABAC decoding as high-quality video decoding is most challenging for modern computer architectures. We aim at implementing the existing HEVC CABAC algorithm as efficiently as possible on a variety of today's computing systems.

Objective 3: Enable new parallelization opportunities in CABAC decoding.
CABAC decoding implements a sequential arithmetic coding algorithm which, unlike all other major decoder modules, does not allow the use of DLP techniques for performance improvement. While modern computer architectures – especially graphics

processing units (GPUs) and GPPs – constantly improve their support for exploiting DLP, CABAC is the only module in state-of-the-art video codecs that does not benefit from this development. Finding alternative parallelization opportunities is essential to compensate for the lack of DLP. Modifications in the HEVC standard might be necessary to achieve this goal. Therefore, the developed techniques can be seen as proposals for future video coding standards.

By addressing the CABAC decoding throughput bottleneck, the data-parallel capabilities of modern computer architectures become also more effective as the sequential fraction in the video decoding process is reduced.

1.3 Contributions

The research conducted in this thesis is supposed to improve the performance and efficiency of modern video decoders in hardware and software implementations. Due to the versatility of video coding applications, we provide a comprehensive performance analysis of the HEVC decoder. The CABAC decoder is identified as the main performance bottleneck for high-quality videos in this analysis. Consequently, we develop and evaluate strategies to optimize the CABAC decoding process on different levels, e.g. frame, block and symbol level. Furthermore, we aim at enabling new parallelization opportunities for use in future video coding standards. The main contributions are as follows:

- We provide a quantitative performance evaluation of the HEVC software decoder. Therefore, we employ multiple processors to represent a wide range of computing devices, from smartphones to high-performance desktop computers. We also use testsets that are typical for different video coding applications and evaluate them in a broad range of quality levels. Based on this analysis, the main decoder bottlenecks and architectural limitations can be identified. We further provide recommendations for future video coding standards and computer architectures to better match the capabilities and requirements of each other.
- Wavefront Parallel Processing (WPP) is an established tool to exploit intra-frame parallelism by processing consecutive rows of pixel blocks, however, it suffers from parallel inefficiencies. When analyzing the dependencies that lead to the common horizontal offset between consecutive rows, it can be found that they are most relaxed for CABAC. To exploit that, we propose three methods that improve the parallel efficiency of WPP for CABAC decoding when it is decoupled from the reconstruction process. The methods differ in the granularity at which dependency checks are performed.
- We propose a modified bitstream format for future video coding standards that enables additional parallelization opportunities and substantial speedups for CABAC decoding. A common HEVC bitstream is divided into eight partitions that can be

processed simultaneously with few dependencies. The fixed partitioning scheme works best for high-quality videos when CABAC throughput is most critical. Furthermore, the bitstream overhead and hardware cost are much smaller than with the existing high-level parallelization tools in HEVC.

- Architectural improvements for CABAC hardware decoding result in higher achievable clock frequencies and consequently higher throughput. We present a context model cache architecture with an application-specific memory layout. An adaptive prefetching algorithm is used to reduce the miss rate to a negligible level. The use of this cache architecture leads to improvements in performance and energy efficiency with low hardware overhead.

- Although many devices have dedicated video coding hardware, software is often used as a fallback solution, e.g. when advanced video features are used. We present an improved technique for parallel bypass-bin decoding on GPPs. Although it is based on simple integer arithmetic, common CABAC decoding is inefficient on GPPs due to frequent hardly predictable branches. We achieve substantial decoding speedups by replacing such code by a branch-free version based on more complex arithmetic instructions.

The contributions of this thesis are based on previous works that have been published as follows:

1. **P. Habermann**, *"Design and Implementation of a High-Throughput CABAC Hardware Accelerator for the HEVC Decoder*, Lecture Notes in Informatics - Seminars, Informatiktage 2014, pp. 213-216, Potsdam, Germany, March 2014

2. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Optimizing HEVC CABAC Decoding with a Context Model Cache and Application-specific Prefetching"*, Proceedings of the 11th IEEE International Symposium on Multimedia (ISM 2015), pp. 429-434, Miami, FL, USA, December 2015, **Best Student Paper Award**

3. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Application-Specific Cache and Prefetching for HEVC CABAC Decoding"*, IEEE Multimedia, volume 24, issue 1, pp. 72-85, January 2017

4. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Syntax Element Partitioning for high-throughput HEVC CABAC Decoding"*, Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017), pp. 1308-1312, New Orleans, LA, USA, March 2017

5. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Improved Wavefront Parallel Processing for HEVC Decoding"*, Proceedings of the 13th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES 2017), pp. 253-256, Fiuggi, Italy, July 2017

6. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"A Bin-Based Bitstream Partitioning Approach for Parallel CABAC Decoding in Next Generation Video Coding"*, Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2019), pp. 1053-1062, Rio de Janeiro, Brazil, May 2019

7. **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Efficient Wavefront Parallel Processing for HEVC CABAC Decoding"*, Proceedings of the 28th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2020), pp. 339-343, Västerås, Sweden, March 2020

The author of this thesis developed the research ideas of all publications. He performed the implementation work, designed and executed the evaluations, and wrote the papers. The co-authors assisted in technical discussions, by providing software that was used as a starting point for the implementation (publications 1-3), and by performing reviews.

1.4 Thesis Organization

Chapter 2 provides a brief overview of the fundamentals of video coding with a focus on the HEVC standard. We also present the results of the conducted performance evaluation to identify the main throughput bottlenecks in the HEVC decoder. Finally, we provide a more detailed description of CABAC as it is the main decoder module covered in this thesis. Related work is discussed in Chapter 3. The analysis of existing work is divided into multiple parts that cover parallelization approaches on different levels, as well as architectural optimizations. Based on the results of the analysis we propose multiple optimizations which are covered in the following chapters.

Three WPP implementations for CABAC decoding are presented in Chapter 4. They perform more fine-grained dependency checks and improve the parallel efficiency compared to conventional WPP. The proposed techniques yield best results for low-delay applications which also benefit the most from the improved parallel efficiency within a frame. Afterwards, we propose a bitstream partitioning approach in Chapter 5 that enables additional parallelism and significantly improves the performance of CABAC decoding for future video coding standards. At the same time, the hardware cost and bitstream overhead are substantially smaller than for the HEVC high-level parallelization approaches because these metrics have been specifically targeted in the design process of the bitstream partitioning scheme. Chapter 6 shows architectural enhancements with an application-specific context model cache architecture, the corresponding adaptive prefetching algorithm, and an optimized context model memory layout. A design space exploration has been performed to find the best cache configuration in terms of performance, energy efficiency and hardware cost. Different techniques for improved CABAC decoding on GPPs are presented in Chapter 7. The replacement of frequent hardly predictable branches by more complex but branch-free arithmetic instructions leads to substantial speedups when exploiting the bypass-bin grouping in HEVC.

Finally, the thesis is concluded and promising future work is discussed in Chapter 8. We also provide recommendations for developers of video compression standards and computer architectures based on the findings in this thesis.

CHAPTER 2

Background

This chapter provides the background information to which we will refer in the thesis. Section 2.1 covers information about video coding, especially the HEVC standard. Furthermore, the results of a performance analysis of the HEVC decoder are presented in Section 2.2 to quantitatively motivate the need for further optimizations in CABAC decoding. Finally, a more detailed description of HEVC CABAC is provided in Section 2.3 as it is the main topic of this thesis.

2.1 Video Coding

Most modern video coding standards employ a hybrid approach with prediction techniques and a transformed residual signal. Intra- and inter-picture prediction are used to remove spatial and temporal redundancies in similar frame areas. As these predictions do not yield perfect results, a transformed and quantized version of the difference between the real frame area and its prediction is also transmitted. In this way, video data can be reconstructed based on previously decoded information. The post-processing with image filters and the compression of the prediction and residual information with an entropy coding method complete the majority of state-of-the-art video coding standards.

Sampling is applied to videos to transform them into a time-discrete fixed spatial resolution. The time-discrete sampling determines the number of frames per second (fps) a video consists of. 20 to 60 fps are typically used in most applications to allow smooth transitions between frames. The spatial resolution determines the amount of details that can be observed in video data. Full High Definition (FHD, 1920×1080 samples) is a widespread resolution that is used in many entertainment applications. Table 2.1 shows more video resolutions that are typically used.

Video data is commonly represented in the YCbCr color scheme. It consists of luminance information (luma, Y) and two chrominance (chroma) components that represent the deviation from gray towards blue (Cb) and red (Cr). This decorrelation of the color components is beneficial for the compression efficiency. Furthermore, chroma subsampling is often applied to reduce the amount of required data, exploiting that the human visual system is more sensitive to luminance than to chrominance. Only one Cb and

Table 2.1: Common video resolutions for different aspect ratios (4:3, 16:9, 17:9).

4:3		**16:9**		**17:9**	
VGA	640×480	HD	1280×720		
SVGA	800×600	FHD	1920×1080	2K	2048×1080
UXGA	1600×1200	UHD-1	3840×2160	4K	4096×2160
QXGA	2048×1536	UHD-2	7680×4320	8K	8192×4320

Table 2.2: Chroma subsampling modes with chroma resolutions relative to an $N \times N$ luma block.

Mode	4:2:0	4:4:4	4:2:2	4:0:0
Chroma resolution (width × height)	$\frac{N}{2} \times \frac{N}{2}$	$N \times N$	$\frac{N}{2} \times N$	–

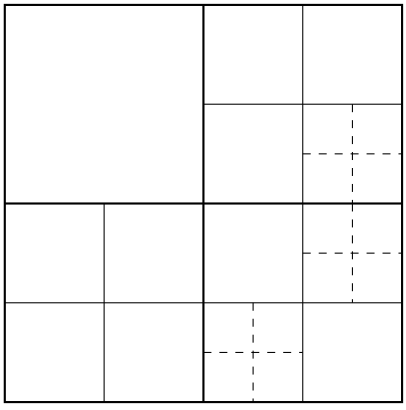
Cr sample are used together with a block of 2×2 Y samples in the most common 4:2:0 mode, which is equivalent to half the horizontal and vertical resolution for Cb and Cr compared to Y. A list of common chroma subsampling modes can be seen in Table 2.2.

2.1.1 High Efficiency Video Coding

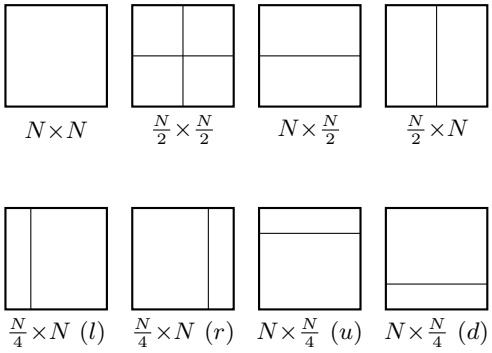
The HEVC standard has been released in 2013. Since then, it has been updated to the current version of November 2019 [9]. Multiple extensions have been specified to extend its applicability to more areas, e.g. range extensions [11], scalable video coding [12], as well as multiview and 3D extensions [13]. The HEVC Test Model (HM [10]) is the corresponding reference software that provides an implementation of the encoder and decoder. The following description of the HEVC standard is based on the work of Sullivan et al. [8].

Each frame is divided into coding tree units (CTUs) in the HEVC standard (see Figure 2.1). CTUs are square blocks of 64×64, 32×32 or 16×16 samples. The CTU size is fixed for a video and can be selected to allow the best compression efficiency (64×64) or lowest processing latency (16×16). A coding quadtree per CTU can recursively divide the area into square coding units (CUs) (see Figure 2.2a). The largest possible CU size is equal to the CTU size while the smallest size is 8×8 samples. The decision about the prediction mode (intra- or inter-picture) is made on the CU level. Every CU is associated with one, two or four prediction units (PUs) and a (potentially empty) transform tree. The area of a CU can be divided into PUs with a variety of shapes (see Figure 2.2b) to better match the structure of the corresponding frame content and allow more efficient prediction. While only square shapes are available for intra-picture prediction, all shapes can be used for inter-picture prediction as long as the resulting PUs have at least a size of 4×8/8×4 samples. The transform tree represents the residual signal, i.e. the difference between the real block and the predicted block.

Figure 2.1: Partitioning of a frame into 64×64 CTUs. (Cactus test sequence from JCT-VC common test conditions [23])



(a) Recursive quadtree splitting



(b) PU shapes

Figure 2.2: Block partitioning in HEVC.

The transform tree can also be recursively divided into square transform units (TUs) with a minimum size of 4×4 samples. TUs consist of multiple transform blocks (TBs) for the color components Y, Cb and Cr.

An overview of the most important modules in the HEVC decoder is presented in Figure 2.3. First, the entropy coding module extracts syntax elements – a non-sample-based representation of a frame – from the compressed input bitstream. HEVC uses CABAC as the only entropy coding method. The extracted syntax elements are then used to control other decoding steps, such as prediction and filters. Intra prediction exploits spatial similarities of neighboring blocks within a frame to remove redundant data and decrease the required data rate. Inter prediction targets the same goal by exploiting temporal similarities of blocks in consecutive frames. In the encoder, the residual is transformed to the frequency domain and quantized before compression. Inverse quantization and inverse transforms are applied to the transform coefficient syntax elements in the decoder to restore the residual data which is added to the predicted block before filters are applied. The deblocking filter (DBF) is used to reduce compression artifacts that arise due to the block-wise processing of video data. Afterwards, the sample adaptive offset (SAO) filter reduces sample distortion by adding specific offsets to different sample categories. The main components are described in more detail in the following.

CABAC is the only entropy coding method in HEVC. It employs binary arithmetic coding which is known to provide high compression performance at high computational complexity. Context modeling is used to compress binary symbols (bins) efficiently and to enable an adaptive entropy coding method that can work well with very different video characteristics. A more detailed description of CABAC will be provided in Section 2.3.4 as it is the main decoder component discussed in this thesis.

Intra Prediction predicts the signal of a block based on the adjacent pixels in the top-left, top, top-right, left and bottom-left blocks. It is predominantly used to exploit spatial similarities in regular structures and homogeneous regions in a frame. HEVC intra prediction supports a DC mode, a planar mode, and 33 directional modes for prediction. Neighboring samples are extrapolated in the respective prediction direction when a directional mode is used. The DC mode fills the whole predicted block with an average value of all neighboring samples. The planar mode creates a gradient that provides a smooth transition from the adjacent samples to the predicted area. Intra prediction is performed on 4×4, 8×8, 16×16 or 32×32 blocks.

Inter Prediction exploits temporal similarities in different frames to predict a block of samples from previously decoded frames. This method is particularly effective as consecutive frames in a video are most often very similar. As a result, inter prediction is one of the most important techniques being responsible for the very high compression rates in modern video compression standards. A motion vector is signaled to indicate the location of the reference block in a previous frame compared to the predicted block. Motion vectors compensate for the movement of objects or the camera in consecutive frames. Further information is coded to identify the reference frame that should be used. An eight-tap interpolation filter is used for luma inter prediction (four-tap for chroma) to achieve quarter-pixel (eighth-pixel for chroma) accuracy. Bi-prediction is used to perform a weighted interpolation between two different reference frames.

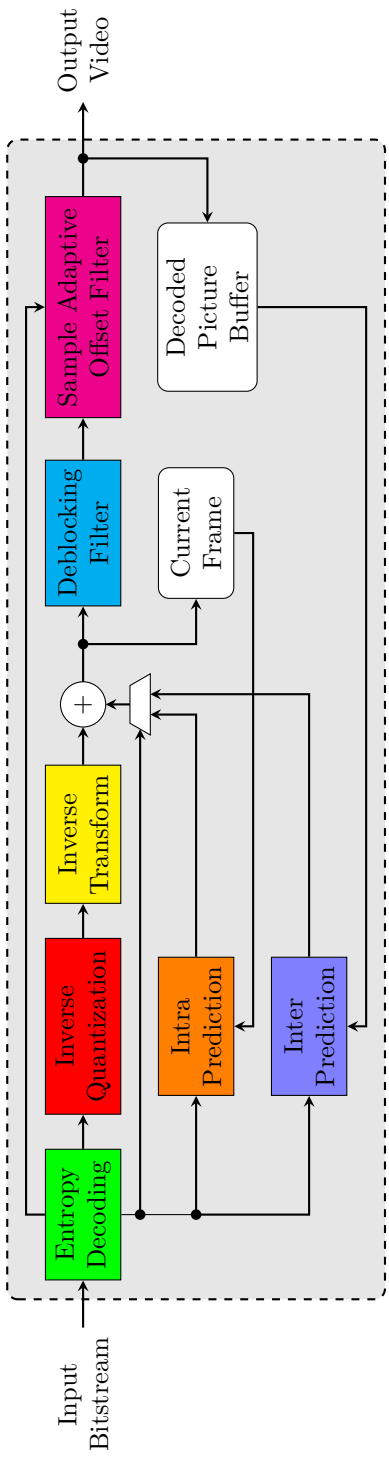


Figure 2.3: Block diagram of the HEVC decoder.

Inverse Quantization is applied to undo the quantization in the encoder. The quantization process divides the transformed residual signal by a specific value to reduce the amount of data that has to be coded. There are losses in precision when undoing the quantization step due to the use of integer arithmetic. The quantization parameter (QP) determines the extent of data loss.

Inverse Transform is performed after the inverse quantization process. Discrete cosine transforms and discrete sine transforms are applied to the residual signal in the encoder to transform it into the frequency domain. As the human visual system is less sensitive to specific parts of the frequency spectrum, they can be removed without affecting the noticeable quality. The inverse process is performed in the decoder to restore the initial residual signal, however, it might be inaccurate due to the quantization that was performed in between. HEVC specifies 4×4, 8×8, 16×16 and 32×32 discrete cosine transforms and a 4×4 discrete sine transform.

The **DBF** is applied after the residual signal was added to the predicted block. The main purpose of the filter is the reduction of artifacts that arise due to the block-based coding. The DBF is only applied to PU and TU boundaries on an 8×8 grid. Different filter strengths are used. The strongest is applied if at least one of the involved blocks is intra-predicted. A weak filter is applied when either of the blocks has non-zero transform coefficients or when the inter prediction data differs from each other.

The **SAO Filter** is the final step in the decoder pipeline. It can enhance the video quality in smooth areas and around edges by providing two modes that conditionally manipulate all samples based on parameters that are encoded in the compressed bitstream. The band offset mode adds specific offsets to all samples whose values are within a specific range while the edge offset mode manipulates samples if their horizontal, vertical or diagonal neighbors fulfill certain requirements that make the samples local minima/maxima or edge samples.

The work distribution among the described decoder modules depends on the frame content and especially on the video quality. The QP is the most important parameter to determine the quality of the residual signal and, consequently, also the quality of the reconstructed video. Lower QPs result in higher video quality and higher bitrates. Typical QPs for low quality are 32 and 37, while 22 and 27 are used for high-quality videos. QPs of 12 and 17 are used for very high video quality in special applications.

Different modes are commonly used in video coding. All-intra (AI) does not use inter prediction, thereby removing dependencies between frames at the cost of much higher bitrates. Random-access (RA) makes use of the effective inter prediction for better compression efficiency. Intra frames are typically used every second to provide access points. This is the common mode for most applications. Low-delay (LD) makes use of inter prediction but limits the maximum decoding delay by restricting the available reference frames.

2.1.2 Other Video Coding Standards

HEVC is by far not the only existing video coding standard. In this section, we will briefly describe the main differences to HEVC's predecessor H.264 and its in-development successor VVC. We further provide an overview of the emerging AV1 standard.

H.264/AVC [14] [15] was released in 2003 and has been the widely established standard before the introduction of HEVC. It requires about twice the bitrate as HEVC for compressing videos at the same subjective quality. A major reason for that is HEVC's support of 64×64 CTUs. This allows a much more efficient compression than with the 16×16 macroblocks in H.264. Furthermore, HEVC intra prediction supports 33 instead of eight directional modes in H.264, thereby allowing a much more fine-grained directional prediction. The inter prediction interpolation precision was improved with seven- and eight-tap filters compared to a six-tap filter in H.264. The H.264 DBF was applied on a 4×4 grid. The simplified filter decision on an 8×8 block granularity in HEVC makes the filter easier to parallelize. The SAO filter did not exist at all in H.264. Finally, H.264 supported two entropy coding methods. In addition to CABAC, context-based adaptive variable length coding (CAVLC) could also be selected. It has a lower computational complexity but does not reach the coding efficiency of CABAC. Furthermore, the CABAC design has been significantly improved to allow higher throughput.

VVC [17] aims to reduce the bitrate by another 50% in comparison to HEVC. The Joint Video Experts Team (JVET) is currently supervising the development process and the release is planned for 2020. An important step in this direction is the increase of the CTU size to 256×256 samples. The nesting of a multi-type tree into the coding quadtree allows a better mapping of coding blocks to objects in a frame by also allowing binary and ternary splits. The number of directional intra modes was increased from 33 to 65 for more fine-grained prediction. Transform sizes have been increased from 32×32 up to 128×128. Another new feature is the introduction of the adaptive loop filter (ALF) which was also part of HEVC but was removed before its release. It applies a Wiener filter to minimize the mean square errors between original samples and reconstructed samples from SAO and thereby allows significant bitrate reductions at the same quality level. While the CABAC implementation is very similar as in HEVC, some refinements have been applied, e.g. adaptive context model initialization and multi-hypothesis probability estimation. The VVC Test Model [18] is the corresponding reference software.

AV1 (AOMedia Video 1, [19]) is an open-source video coding standard developed by the Alliance for Open Media. It has been released in 2018 [20] and succeeds Google's VP9 standard [21] [22]. Both aimed at providing a royalty-free alternative to H.264 and HEVC video coding standards and implement the same hybrid approach, however, there are many differences in the coding tools. AV1 specifies 128×128 superblocks compared to the 64×64 CTUs in HEVC. The partitioning of the coding quadtree is more flexible and allows smaller block sizes. 56 directional modes are available for intra prediction and an intra block copy allows to fully reuse previously decoded blocks in the same frame. Interesting techniques in AV1 inter prediction include overlapped block motion compensation, warped motion compensation, and advanced compound

prediction. Besides the DBF, a constrained directional enhancement filter is applied for deringing and one of two loop restoration filters can be used. AV1 supports frame super-resolution which offers coding gains at low bitrates by processing a frame at a lower resolution and upscaling it afterwards. AV1 employs a multi-symbol entropy coding scheme compared to the binary arithmetic coder in HEVC. 15-bit precision is used for the probabilities of all potential symbols to enable improved accuracy, especially for infrequent symbols.

A comparison of the coding efficiency of HEVC, H.264 and AV1 has been provided by Grois et al. [37]. They found 32.8% and 38.4% bitrate reductions of HEVC compared to H.264 and AV1 respectively. The results have been partly confirmed by Laude et al. [38] who measured the same bitrate increase of AV1 compared to HEVC. Furthermore, they have shown an average 27.7% and 44.6% bitrate reduction in the RA configuration of VVC over HEVC and AV1 respectively. They also found a 9.7× and 32.6× increase in encoding time of VVC and AV1 over HEVC. This might limit the applicability of the new standards in applications with timing constraints or lead to performance reductions due to necessary tradeoffs between runtime and coding efficiency. In general, the performance comparison between different video codecs is a challenging task. The results vary significantly depending on the encoder configuration, the evaluated metrics (objective/subjective) and especially the employed testsets. An established metric for the comparison of the compression performance of different encoders at the same objective quality is described in [40].

2.1.3 Parallelism in Video Coding

Video coding applications contain a massive amount of parallelism on different levels. High-quality real-time video applications can only be efficiently implemented by exploiting these parallelization opportunities.

Frame-level Parallelism refers to the simultaneous processing of multiple frames. This is generally possible in HEVC as every frame is coded in a separate slice, however, some inter-frame dependencies need to be considered. Slices are data structures that allow independent processing of the corresponding frames or frame areas. A frame consists of at least one slice and can contain multiple slices, which also enables parallel processing within a frame. HEVC supports I-, P- and B-frames. I-frames use only intra-prediction and consequently do not depend on other frames. That is why they are mainly used as access points in a video. P-frames can be inter-predicted from previously decoded frames. B-frames can make use of bi-prediction to be reconstructed from two other frames. Both P- and B-frames can also use intra-prediction. The decision for the prediction mode is made at CU granularity.

Figure 2.4 shows a typical structure of a group of pictures (GOP) and the dependencies between them. The dependencies and the resulting decoding order are chosen to make the best use of the effective bi-prediction. Every video sequence starts with an I-frame (index 0). Afterwards, a P-frame (index 8) can be predicted from the I-frame. The B-frame at index 4 can be decoded as soon as both the I- and the P-frame are decoded.

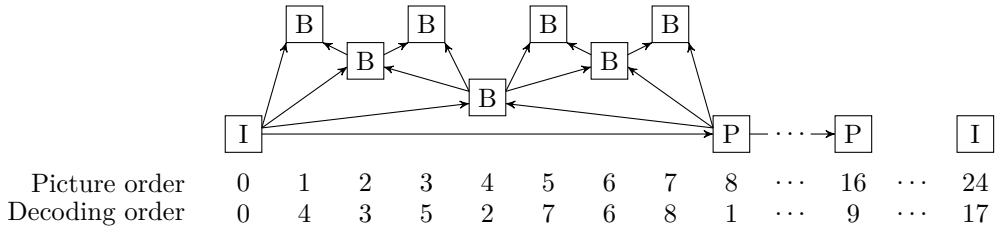
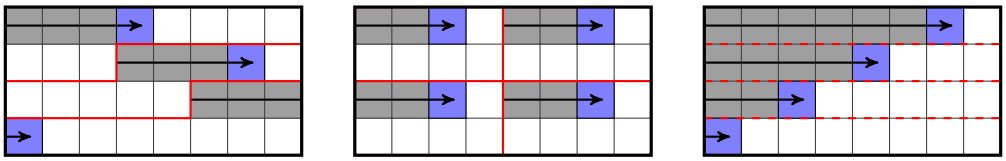


Figure 2.4: GOP structure with one I-frame every 24 frames.

The first frame-level parallelization opportunities within a group of nine frames arise when frames 2 and 6 can be decoded in parallel. After that, even four frames can be processed simultaneously (1, 3, 5, 7). In addition to the parallelism inside a group of nine frames, multiple of these groups can also be processed at the same time, i.e the I-frame at index 24 (and all other multiples of 24) in parallel with the I-frame at index 0, and the P-frame at index 16 in parallel with the B-frame at index 4. Overall, many frames can be processed in parallel, however, due to delay constraints and limitations in computational and memory resources, the number of parallel frames is very limited in practice.

Block-level Parallelism refers to the parallel processing of multiple blocks within a frame. These are most commonly CTUs, however, the approach can also be applied to PUs, TUs, and even smaller blocks of samples, especially when performing operations for the different color components. Three high-level parallelization techniques for parallel CTU processing are specified in the HEVC standard: Slices [33], Tiles [34] and WPP [35]. A frame can consist of multiple slices whose corresponding areas can be processed independently (see Figure 2.5a). Slices contain a set of CTUs which are processed row by row. A frame can also be divided into rectangular tiles for independent parallel processing (see Figure 2.5b). The frame partitioning approaches prevent some prediction opportunities at tile boundaries and thereby reduce the compression efficiency. WPP allows the parallel processing of multiple rows of CTUs (see Figure 2.5c). The decoding of a CTU row can be started as soon as the second CTU in the above row has been processed because the CABAC context information needs to be forwarded. These dependencies lead to a wavefront-like progression of multiple threads over the frame. WPP can also lead to a reduced compression efficiency compared to sequential decoding as the CABAC learning process is interrupted. Furthermore, a ramp-up and -down in the number of active parallel threads limits the efficiency of WPP within a single frame, however, this problem becomes negligible as soon as multiple frames can be processed in parallel. Slices, Tiles and WPP depend on even load balancing for best parallel scalability. Unfortunately, the load is most often concentrated in specific frame areas. Differently-sized slices and tiles can adapt to the load-intensive areas and reach a better work distribution.

An analysis of the parallel scalability and efficiency of HEVC parallelization approaches has been provided by Chi et al. [30]. One of the main contributions of this work is a technique called Overlapped Wavefront (OWF) which extends WPP to multiple frames.



(a) Slices **(b)** Tiles **(c)** WPP

Figure 2.5: High-level parallelization tools in HEVC.

This three-dimensional parallelization approach substantially improves the scalability and efficiency of WPP.

Task pipelining is also an effective technique at the CTU level. Instead of performing entropy decoding, inverse transform and quantization, prediction, and filtering for a CTU after each other before processing the next CTU, these steps can be pipelined. A single thread can perform the entropy decoding for CTU N. When the work is done, it can directly proceed with the entropy decoding for CTU $N + 1$. At the same time, another thread can execute all inverse transform and quantization operations for CTU N while a third thread performs the prediction for CTU $N - 1$ and a fourth thread executes the DBF and SAO filter for CTU $N - 2$. Load balancing is also the main limiting factor for performance improvements, as with Slices, Tiles and WPP. Task pipelining can also be applied at the frame-level, however, it induces immense memory requirements as information for multiple frames needs to be stored for the next thread. This is also not optimal in terms of data locality.

Sample-level Parallelism can be found in all prediction, transform and filtering tasks in the HEVC decoder. The same operations are most often applied to all samples in a block. The vector extensions of modern processors can be used to process many of these samples in parallel and with reduced overhead for instruction fetch, decode, etc.. The same vectorization approaches can also be implemented in customized hardware. Vectorization is greatly responsible for the existence of high-performance and energy-efficient real-time video coding applications as it provides significant speedups up to 5× for the entire HEVC decoder [31]. This work by Chi et al. provides an overview of the vectorization techniques applied to all components of the HEVC decoder that contain significant DLP. The evaluation covers many different processors with a variety of vector extensions. It has also been shown that the improvements in energy-efficiency grow very similar to the speedup achieved by vectorization [32].

2.2 Performance Evaluation

The HEVC decoder is a complex application consisting of multiple components (see Figure 2.3) that differ in their computational and memory requirements. The performance of the whole decoder is limited if only one component does not deliver the required throughput for real-time decoding. It is necessary to identify these bottlenecks and

address them with specific optimizations to improve the overall decoder performance in the most efficient way. Therefore, we perform an analysis of the HEVC software decoder providing useful general insights that also apply to hardware decoding as the same operations are possible in hardware, however, most often more efficiently. The resulting higher computational performance in dedicated hardware might bring new bottlenecks to light, especially limitations of the memory system. Nevertheless, this can also be mitigated with application-specific cache architectures or customized data layouts for improved data reuse.

An overview of complexity-related aspects in the HEVC standardization process has been provided by Bossen et al. [27] shortly before the publication of the standard in 2013. Although the main focus is a comparison to H.264, the performance results can be used as a reference. An optimized HEVC software decoder has been used for the evaluation that evolved from the one used in [29]. Intel SSE 4.1 and ARM Neon vector extensions are used to exploit the significant amount of DLP. The authors evaluated single-threaded decoding performance in RA mode with QPs of 27 and 32. An Intel Core i7-3720QM mobile CPU (central processing unit) can process all JCT-VC class B testsequences (1920×1080 samples) while an embedded 1 GHz ARM Cortex-A9 CPU can process all JCT-VC class C testsequences (832×480 samples) in real-time. The most time-consuming decoding tasks in this configuration are motion compensation (43%/49%), entropy decoding (24%/21%) and loop filters (21%/18%).

Ultra High Definition (UHD, 3840×2160 samples) real-time decoding capabilities have been demonstrated by Bross et al. [28] using a 3.1 GHz eight-core server processor (Intel Xeon E5-2687W) and the Fraunhofer HHI HEVC software decoder in 2013. Although only achieved for lower quality levels, the general feasibility of UHD real-time decoding is shown.

Chi et al. provided a more exhaustive performance evaluation of a fully vectorized HEVC software decoder on multiple AMD, Intel and ARM processors in 2015 [31]. In addition to the existing evaluation by Bossen et al., the effect of multiple vector extensions is investigated. They demonstrate a substantial speedup due to the newer AVX2 vector extension which doubles the vector register size compared to SSE 4.1 and AVX. Further optimizations such as chroma interleaving have been applied to achieve a better utilization of the vector registers. Furthermore, UHD video material has been used for performance measurements as it is one of the emerging use cases of HEVC. The authors also used a wider spectrum of video quality levels (QP 24, 28, 32, 36). Multithreading has also been added to the evaluation. A WPP approach has been extended to multiple frames [30] to efficiently exploit TLP and get the most out of modern multi-core architectures. The evaluation shows that multithreaded real-time software decoding of 1080p videos is easily achievable for all architectures except an ARM Cortex-A9. Some of the more powerful processors can even decode 10-bit 2160p videos in real-time.

Real-time decoding capabilities have been demonstrated for many types of video coding applications since the introduction of HEVC. However, as computer architectures and video coding use cases have evolved in the last years, we provide an up-to-date evaluation. It is necessary to identify the main throughput bottlenecks so that the optimization

work leads to the largest speedup for the overall decoder. The main contributions compared to existing works are:

- An analysis of real-time decoding capabilities and performance bottlenecks that need to be addressed in future standards such as VVC.
- Multiple state-of-the-art processor architectures that represent different computing devices.
- A testset of commonly used video coding applications.
- Evaluation of the latest 512-bit vector extensions (AVX-512).

In the following, we describe the experimental setup and show performance and profiling results for different applications on a variety of processors. Furthermore, we draw main conclusions from the performance analysis.

2.2.1 Experimental Setup

One of the main goals of the performance analysis is to provide an overview of the real-time decoding capabilities of modern GPPs for typical video coding applications. Four different platforms are used for evaluation to cover a wide range of computing systems. Each of the selected processors is representative of a class of devices: smartphones, mobile computers, desktop computers and high-performance desktop computers. An overview of the selected platforms can be seen in Table 2.3. The Kirin 970 is a commonly used smartphone processor that consists of four high-performance ARM Cortex-A73 cores and four low-power ARM Cortex-A53 cores. We further employ the Intel Core i7-8550U ultrabook CPU and the Intel Core i7-7700K desktop CPU for evaluation. Although all these three CPUs have four main processor cores, their performance is very different. The ultrabook processor and especially the smartphone processor are designed to be very energy-efficient. Therefore, they operate at a lower clock frequency and have a different cache architecture. Besides the smaller vector register size, the Kirin 970 also lacks simultaneous multithreading, i.e. the ability to execute two threads on the same core at the same time while sharing functional units. The high-performance desktop processor (Intel Core i9-7960X) has 16 cores which makes it much more powerful, even with the slightly reduced clock frequency. Furthermore, the doubled vector register width in comparison to the conventional desktop and ultrabook CPU is expected to allow a better exploitation of the available DLP in video coding applications. We use an optimized HEVC decoder developed by Spin Digital Video Technologies GmbH [78]. A previous version of it has already been used in [31]. We employ WPP with one thread per core for the smartphone processor and two threads per core for all others for the evaluation.

Video coding is used in very different fields and the corresponding applications have different requirements regarding video resolution, frame rate, color format and depth, quality and bitrate. In the following, we list six typical applications that will be used for the evaluation in this work.

Table 2.3: Architectural parameters of the evaluation platforms. I$: instruction cache, D$: data cache

	Smartphone	**Ultrabook**	**Desktop**	**HP Desktop**
Processor	HiSilicon Kirin 970	Intel Core i7-8550U	Intel Core i7-7700K	Intel Core i9-7960X
Cores	4 Cortex-A73 + 4 Cortex-A53	4	4	16
Base Frequency	2.36/1.80 GHz	1.80 GHz	4.20 GHz	2.80 GHz
Turbo Frequency (all cores active)	n.a.	3.70 GHz	4.40 GHz	3.60 GHz
Simultaneous Multithreading	✗	✓	✓	✓
Vector Extension	Neon (128 bit)	AVX2 (256 bit)	AVX2 (256 bit)	AVX-512 (512 bit)
RAM	6 GB	16 GB	32 GB	64 GB
Cache (per core)	64 KiB L1 I$/D$ (A73) 32 KiB L1 I$/D$ (A53)	32 KiB L1 I$/D$ 256 KiB L2 Cache	32 KiB L1 I$/D$ 256 KiB L2 Cache	32 KiB L1 I$/D$ 1 MiB L2 Cache
Cache (shared)	2/1 MiB L2 Cache (A73/A53)	8 MiB L3 Cache	8 MiB L3 Cache	22 MiB L3 Cache

Table 2.4: Video coding applications. Separate bitrate ranges are shown for the lower and higher quality levels.

	Video Class	#Videos	Mode	QPs	Bitrate (MBit/s)
Low-delay	Common testset Class E	3	LD	22, 27, 32, 37	0.1 - 0.4 0.4 - 2.2
Screen content	Range Extensions YCbCr 444 SC	6	RA	22, 27, 32, 37	0.2 - 1.8 0.5 - 6.6
Animation	Range Extensions YCbCr 444 Animation	3	RA	22, 27, 32, 37	0.3 - 1.9 1.8 - 7.3
FHD playback	Common testset Class B	5	RA	22, 27, 32, 37	0.4 - 2.4 1.7 - 33.1
Video production	Range Extensions RGB 444	8	AI	12, 17, 22, 27	27.9 - 548.1 74.0 - 1117.1
UHD playback	EBU UHD-1	4	RA	12, 17, 22, 27	5.1 - 87.1 135.9 - 793.6

- **Low-delay** applications, e.g. video conferencing and video chatting in LD mode.
- **Screen content**, e.g. presentation slides and web browsing videos.
- **Animation** as found in animated movies.
- **FHD playback** of videos that are streamed over the internet or stored on media such as DVDs.
- **Video production** in AI mode with high quality and color depth.
- **UHD playback** of videos as in FHD playblack, but typically at higher quality levels.

An overview of the testsets is provided in Table 2.4. More details about the testsets can be found in [23] [24] [25]. We use the same modules as seen in Figure 2.3 for the profiling of the video decoder components, however, there are two exceptions. First, inverse quantization is merged with inverse transform (IQ/IT) because they are often processed together. Second, there is a category for remaining tasks that do not belong to any of the existing categories, e.g. writing reconstructed frames to memory (Others).

2.2.2 Evaluation

Table 2.5 shows the real-time decoding capabilities of all platforms for the selected testsets. We use the average decoding performance for evaluation. However, it should be noted that the complexity of different frames can vary significantly. Consequently, higher performance might be necessary for judder-free playback. All platforms deliver sufficient performance for the basic low-delay, screen content, animation and FHD applications. On the other hand, only the HP desktop can process the high-bitrate video production testset as well as the high-resolution UHD testset in real-time. In the

Table 2.5: Performance overview showing the worst-case performance of any video of the corresponding testset compared to the required real-time performance.

	Smart-phone	Ultra-book	Desktop	HP Desktop
Low-delay	5.35×	11.75×	23.93×	22.45×
Screen content	2.40×	10.47×	18.50×	30.49×
Animation	1.96×	6.11×	12.23×	23.51×
FHD playback	1.22×	2.67×	5.15×	11.76×
Video production	0.15×	0.24×	0.49×	1.30×
UHD playback	0.12×	0.22×	0.31×	1.12×

Table 2.6: Detailed performance results of selected testsets in fps.

a) Smartphone FHD (1920×1080), RA	**QP22**	**QP27**	**QP32**	**QP37**
BasketballDrive 50 fps	96.14	121.60	142.87	157.18
BQTerrace 60 fps	73.45	132.05	156.68	169.36
Cactus 50 fps	100.09	135.74	153.75	162.87
Kimono1 24 fps	107.83	126.02	140.00	157.18
ParkScene 24 fps	95.03	121.31	138.63	160.14

b) HP Desktop Video Prod. (1920×1080)*, AI	**QP12**	**QP17**	**QP22**	**QP27**
DucksAndLegs 30 fps	89.20	93.60	101.22	124.43
EBULupoCandlelight 50 fps	108.56	184.28	349.50	538.31
EBURainFruits 50 fps	114.18	175.58	254.40	362.29
Kimono1 24 fps	79.49	105.46	202.58	368.25
OldTownCross 50 fps	64.83	75.60	97.41	168.28
ParkScene 24 fps	65.90	83.65	122.02	192.28
Traffic 2560x1600, 30 fps	46.27	61.36	84.39	123.66
VenueVu 30 fps	162.82	270.85	378.11	490.16

c) HP Desktop UHD (3840×2160), RA	**QP12**	**QP17**	**QP22**	**QP27**
EBULupoCandlelight 50 fps	62.94	109.66	244.84	322.95
EBULupoConfetti 50 fps	71.28	176.07	312.37	365.96
EBURainFruits 50 fps	56.09	112.65	210.54	288.36
EBUWaterfallPan 50 fps	62.28	128.29	207.46	272.76

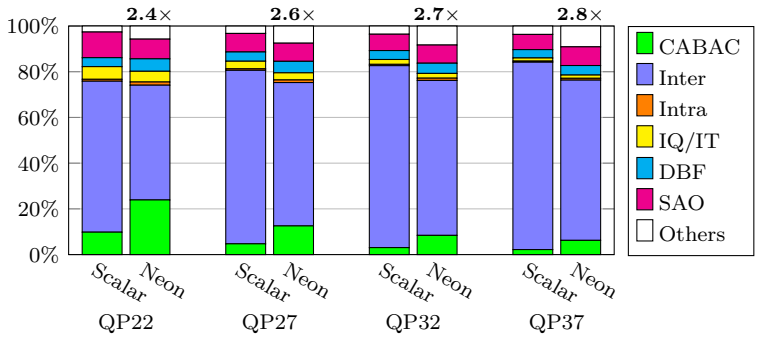


Figure 2.6: Smartphone + FHD testset profiling with Neon vectorization. Speedup over Scalar shown on top.

following, we take a more detailed look at the performance measurements for specific testsets (Table 2.6). We further aim to identify performance bottlenecks by analyzing the decoding time that is spent on specific tasks in the single-threaded HEVC decoder (Figures 2.6, 2.7, 2.8).

Playback of FHD content on smartphones is one of the most widely used applications. Therefore, the corresponding performance results are shown in Table 2.6a. There is at least a 90% performance margin over real-time decoding for all videos and quality levels except BQTerrace at QP22 (22%). While this provides sufficient performance for video decoding, it is not a fully satisfying situation as other aspects also need to be considered, e.g. video resolutions beyond FHD, other applications running on the processor, voltage and frequency scaling for improved energy efficiency. The profiling results in Figure 2.6 show that inter prediction is clearly the dominating kernel across all quality levels on the smartphone platform. Due to the limited vector size of 128 bits, the massive amount of DLP cannot be efficiently exploited. This is a huge disadvantage compared to other processors when dealing with data-parallel applications, not only for performance but also for energy efficiency [79]. The emergence of ARM's Scalable Vector Extensions (SVE, [82]) is expected to address this issue in the next years. However, the underlying memory system needs to be adjusted accordingly.

We also look at the detailed results for the most challenging applications (video production, UHD playback) to see how the HP desktop processor deals with them. The results in Table 2.6b demonstrate general real-time decoding capabilities of the HP desktop processor for the video production testset. The measured performance for all videos is at least 30% higher than the required real-time performance. The variance between different frames is also much smaller because all frames are intra-predicted. Nevertheless, the profiling results in Figure 2.7 show two critical issues. First, CABAC clearly contributes most of the decoding time, e.g. 40% to 70% for the scalar decoder. When vectorization is applied, the contribution even increases to above 80% as it is the only kernel that does not benefit from vectorization due to the lack of DLP. This directly results in the second critical aspect: poor vectorization speedups. The overall speedup is at most 1.8× due to the small vectorizable fraction of the decoder. Another

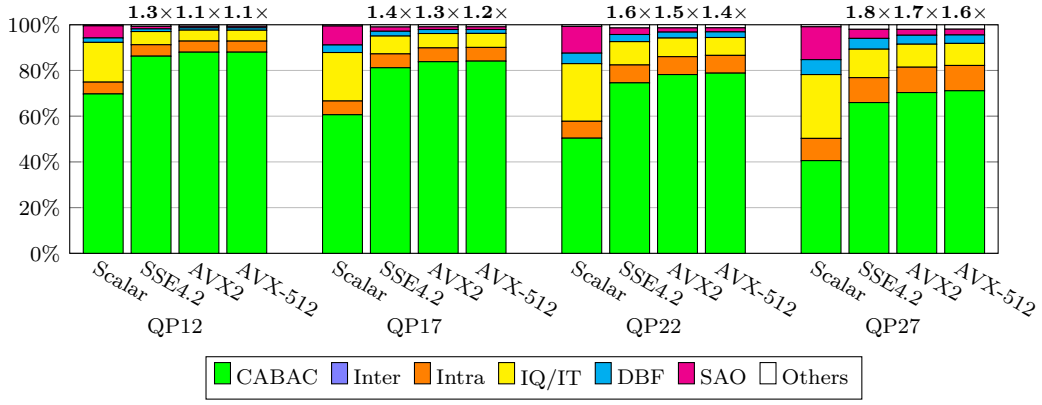


Figure 2.7: HP desktop + video production testset profiling with different vector extensions. Speedup over Scalar shown on top.

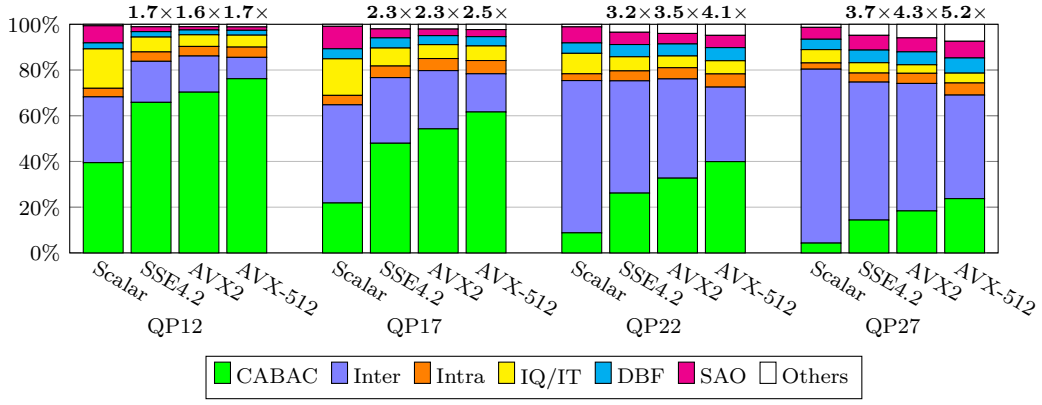


Figure 2.8: HP desktop + UHD testset profiling with different vector extensions. Speedup over Scalar shown on top.

influencing fact is that higher video quality tends to result in smaller blocks for the prediction, transform and filtering kernels. Consequently, larger vectors cannot be fully utilized, thus they do not improve performance. Even worse, most processors operate at a lower clock frequency to stay in their thermal budget when wider vectors are used. This results in smaller speedups with wider vectors as shown for all quality levels. We even observed performance drops compared to the scalar baseline for some videos when vectorization was applied.

The UHD testset pushes the HP desktop to its limits. The average decoding performance satisfies real-time requirements, but only by a small margin as can be seen in Table 2.6c. For the highest quality (QP12) the achieved performance is between 12% and 43% above the required real-time performance. Considering the varying complexity of different frames this might not be sufficient for smooth playback. On the other hand, at least 4× the real-time performance can be reached with QPs 22 and 27. Figure 2.8 shows that vectorization yields a higher overall application speedup for these quality levels (up to 5.2×). Vectorization becomes less effective for lower QPs, e.g. only a 1.7× speedup is possible with QP12. The same reasons as discussed for the video production testset can be found here, i.e. a significant CABAC decoding time fraction and lower vector utilization for higher video quality, although not to the same extent.

2.2.3 Conclusions and Recommendations

Although modern processors are very well capable of real-time decoding suitable workloads, we identified two main bottlenecks that lead to performance constraints and inefficient decoding. Entropy decoding is the only strictly sequential part of today's hybrid video decoders. Therefore, it cannot exploit the constantly improving vector extensions of modern processors and consequently contributes most of the decoding time for high-quality videos and in certain applications that require AI mode. Regarding entropy coding, there are only small modifications in HEVC's successor VVC, which will not affect this bottleneck. A fundamental redesign of the entropy coding algorithm is required or additional parallelization opportunities need to be established. This thesis aims to address the critical CABAC throughput bottleneck in high-quality video decoding with performance optimizations on different levels as well as architectural enhancements.

The evaluation has also shown that vectorization becomes less efficient with higher video quality. The constantly growing vector registers cannot be fully utilized as block sizes for different kernels tend to decrease with higher quality. VVC allows larger blocks, however, this does not affect high-quality videos which rarely use such large blocks. On the other hand, the introduction of differently shaped blocks partly addresses the problem. Microarchitectural support for two-dimensional block processing (e.g. [80]) might be much more beneficial for video coding and many other applications than pushing the vector sizes to new dimensions. This can also be emulated in software to some extent, however, gather-load and scatter-store operations might be the main limitation with current memory architectures.

More potential for improvement can be found in the field of heterogeneous computing. It has been shown that the use of processor cores with different instruction set architectures (ISA) allows substantial performance improvements and energy savings compared to homogeneous-ISA multiprocessors [81]. Modern video coding standards do not use floating-point operations and especially entropy coding can work with a basic 16-bit integer ISA and without vector extensions. This allows multiple of such cores instead of a single general-purpose core at the same chip area, thereby improving parallelization opportunities and energy efficiency.

2.3 Context-based Adaptive Binary Arithmetic Coding

CABAC was first introduced in the H.264 video coding standard [45] as an alternative to the less complex but also less efficient CAVLC. The underlying M coder, which implements the binary arithmetic coding algorithm without multiplications, is described in [47]. Arithmetic entropy coding methods have been used in video coding before, e.g. in H.263 [16] where it could optionally be used. CABAC addressed some drawbacks of earlier arithmetic coders by introducing adaptive probability models and a simplified binary arithmetic coding process. Consequently, it is the only entropy coding method in HEVC due to its superior compression performance. A CABAC implementation consists of three main parts: a general binary arithmetic coding implementation, suitable binarization schemes for specific syntax elements, and an application-specific context model design. All these aspects will be discussed in the following.

2.3.1 Binary Arithmetic Coding

Arithmetic coding is a lossless entropy coding method that allows the compression of data at minimum entropy [48]. According to Shannon [49], the minimum entropy for encoding data consisting of N symbols with probabilities p_i is

$$H = -\sum_{i=1}^{N} p_i \log p_i$$

Consequently, for achieving minimum overall entropy, a symbol must be represented by a number of bits equal to the negative logarithm of its probability: $-\log_2 p_i$. More probable symbols are coded with fewer bits which leads to a reduction of the average number of bits per symbol. A direct translation of symbols to bits works well for probabilities that are the inverse of powers of two, e.g. $-\log_2(\frac{1}{2}) = 1$ bit per symbol, $-\log_2(\frac{1}{4}) = 2$ bits per symbol, etc.. For other probabilities, an approximation of the number of bits for minimum entropy must be derived that allows a proper translation of symbols to bits. A set of symbols (A,B,C) with probabilities $p_A = 0.7$, $p_B = 0.2$ and

$p_C = 0.1$ could be coded with an average of

$$\begin{aligned} H &= -p_A \log_2 p_A - p_B \log_2 p_B - p_C \log_2 p_C \\ &= -0.7 \cdot \log_2(0.7) - 0.2 \cdot \log_2(0.2) - 0.1 \cdot \log_2(0.1) \\ &\approx (0.36 + 0.46 + 0.33) \text{ bits per symbol} \\ &\approx 1.16 \text{ bits per symbol} \end{aligned}$$

The compression with 1.16 bits per symbol can only be achieved when a non-integer number of bits can be used to represent a symbol, however, only integer numbers can be used in many entropy coding methods. This requires an approximation of the optimal code lengths which could look as follows: $A = 0_2$, $B = 10_2$, $C = 11_2$. This coding requires on average

$$\begin{aligned} H &= (p_A \cdot 1 + p_B \cdot 2 + p_C \cdot 2) \text{ bits per symbol} \\ &= (0.7 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 2) \text{ bits per symbol} \\ &= (0.7 + 0.4 + 0.2) \text{ bits per symbol} \\ &= 1.3 \text{ bits per symbol} \end{aligned}$$

This reduction in compression efficiency is one of the main disadvantages of many entropy coding methods compared to arithmetic coding, which does not suffer from this limitation and can compress symbols with minimum entropy.

An arithmetic encoder works with a range that is initialized to [0,1). When encoding a symbol, the range is divided into subranges for all potential symbols. The sizes of the subranges are proportional to their probabilities. Depending on the symbol, the corresponding subrange is chosen as the range for the next symbol. This recursive subinterval division leads to a step-wise reduction of the size of the range. At the end of the encoding process, the sequence of symbols can be represented by any value inside the final range. For best compression, an offset within the range is selected that can be represented by the smallest number of bits.

The arithmetic decoder also starts with a range [0,1). Furthermore, it uses the offset that resulted from the encoding process. For every symbol, the range is again divided into subranges according to the probabilities of the respective symbols. The decoded symbol is selected based on the subrange the offset is located in. The subrange is selected as the new range for the next symbol and the process is recursively repeated until all symbols are decoded.

A binary arithmetic coder only uses two symbols and divides the range into two subranges accordingly. Binarization is performed when more than two potential symbols are available. This way, they can also be processed by the binary arithmetic coder. An example of the binary arithmetic encoding process for the bin sequence 010_2 is shown in Figure 2.9. The specific probabilities are chosen for the purpose of demonstration. The range is commonly represented by two values, i.e. its lower boundary (low) and its size ($range$). They are initialized to $low = 0.0$ and $range = 1.0$. Encoding $bin_0 = 0$ results in the new range [0.0,0.6) which is represented by $low = 0.0$ and $range = 0.6$. The right subrange [0.3,0.6) is chosen when encoding $bin_1 = 1$. In the end, [0.3,0.39) is

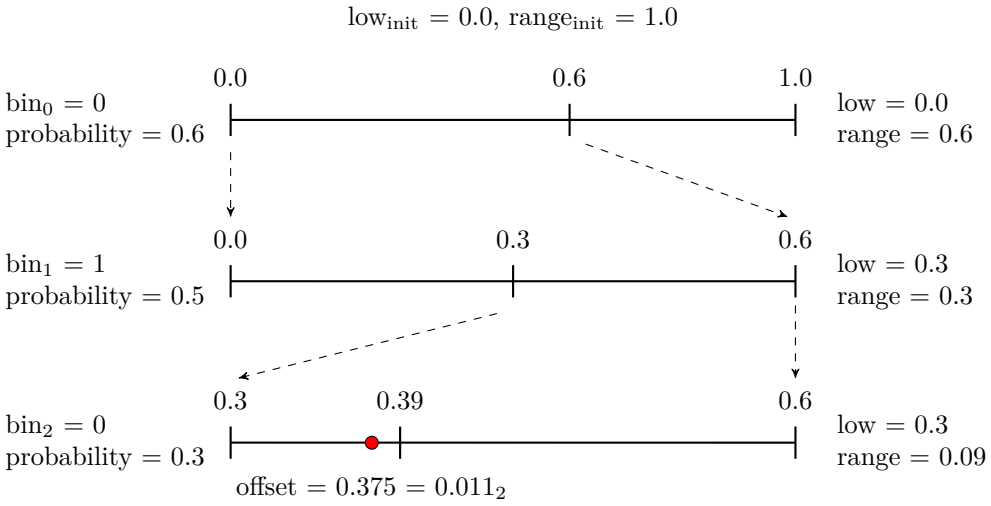


Figure 2.9: Arithmetic encoding example for the bin sequence 010_2. The offset can be any value in the range [0.30,0.39). 0.375 is chosen because it can be represented by the least number of bits.

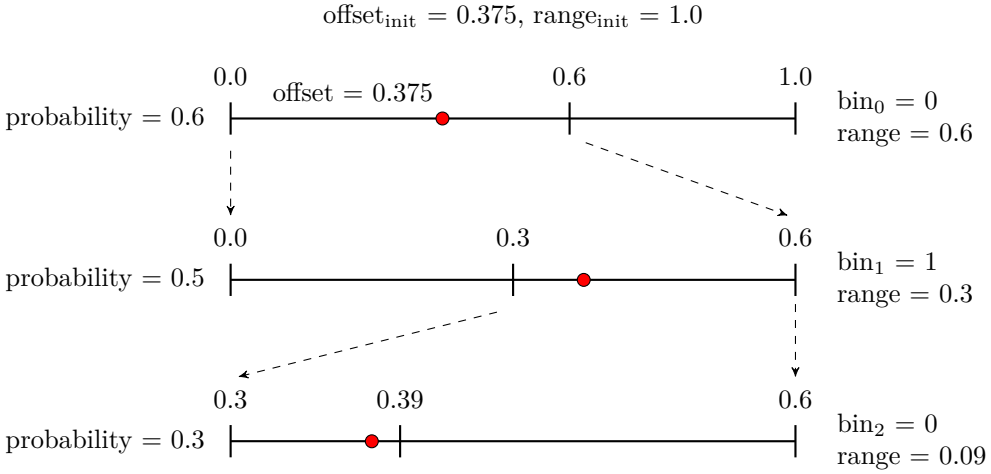


Figure 2.10: Arithmetic decoding example for the bin sequence 010_2.

the final subrange after encoding $bin_2 = 0$. Any value from this range can be selected to represent the encoded bin sequence, however, $0.375 = 0.011_2$ can be coded with only three bits (011_2) because only the fractional part needs to be stored.

The decoding process depicted in Figure 2.10 is very similar. The order of the symbols and the corresponding subranges are the same as in the encoding process. Starting with the offset 0.375 and recursively following the subranges the offset is located in restores the bin sequence 010_2.

2.3.2 Context Modeling and Adaptation

Arithmetic coding allows the compression of symbols with minimum entropy if the probabilities for the respective symbols can be accurately estimated. Consequently, probability modeling is an essential task in arithmetic coding. A statistical analysis of the probability distribution of all potential symbols can be performed and the results can be used in the encoding and decoding process. For example, looking at the English alphabet consisting of 26 characters, ten digits, and approximately a dozen special characters, one will notice that the whitespace and the letter e appear much more frequently than the letters j and z. Incorporating this probability distribution allows the compression of English texts very close to the minimum entropy. Context models estimate the probabilities for symbols in a specific context, e.g. the general probability for the letter u is significantly increased when the previous symbol was the letter q. Context modeling can greatly enhance the probability estimation but requires additional knowledge of the characteristics of an application. Context models are also used in video coding, e.g. for deriving probabilities for block partitioning syntax elements based on the partitioning of neighboring blocks. Furthermore, context models in HEVC are initialized based on the QP which determines the video quality.

Context models are particularly simple in binary arithmetic coding. Only the least probable symbol (LPS) and its probability need to be stored as the probability for the most probable symbol (MPS) can be derived by

$$p_{MPS} = 1 - p_{LPS}$$

The same simplicity can be found in the calculations for the subranges:

$$\begin{aligned} range_{LPS} &= range \cdot p_{LPS} \\ range_{MPS} &= range \cdot p_{MPS} = range \cdot (1 - p_{LPS}) \end{aligned}$$

Accurate probability estimation becomes very challenging when the characteristics of an application are unknown or dynamically changing such as in video coding. Adaptive context modeling can be used to train and improve the context models. Every time a context model is used, its probability distribution is updated based on the result of the encoded or decoded symbol. This allows the step-wise improvement of the probability estimation over time in case the application characteristics are changing or after an unsuitable context model initialization. A probability estimation is not possible in

Table 2.7: Binarization schemes in HEVC with $N_{max} = 7$.

N	Fixed-length	Unary	Truncated Unary	Exponential Golomb
0	000	0	0	1
1	001	10	10	010
2	010	110	110	011
3	011	1110	1110	00100
4	100	11110	11110	00101
5	101	111110	111110	00110
6	110	1111110	1111110	00111
7	111	11111110	1111111	0001000

some cases. To reduce the number of maintained context models, arithmetic coding is performed with equal probabilities for all symbols in this case. These so-called bypass-coded (bc) bins are also easier to encode and decode. Context-coded (cc) bins are associated with context models for improved compression efficiency.

2.3.3 Binarization

Binarization is used in binary arithmetic coding to transform N-ary symbols into one or multiple bins. Efficient binarization schemes lay the foundation for the compression rates achieved by entropy coders such as CABAC. Four different binarization schemes are used in HEVC (see Table 2.7). **Fixed-length** binarization transforms every symbol into a bin sequence of the same length. The length is determined by the maximum symbol value N_{max} such that $\lceil \log_2(N_{max} + 1) \rceil$ bins are used. Every symbol is then transformed into its binary representation. **Unary** binarization transforms a symbol N into a sequence of N 1-bins followed by a zero bin. **Truncated unary** binarization works the same way except for the largest symbol N_{max}. The trailing zero-bin can be skipped since it is known that no larger symbols exist. **Exponential Golomb** binarization consists of a prefix and a suffix. The prefix is a sequence of leading zeros followed by a one-bin. The suffix contains as many bins as there are zero-bins in the prefix. The symbol value is then derived by the number of leading zeros in the prefix (nlz) and by the suffix: $N = 2^{nlz} - 1 + suffix$.

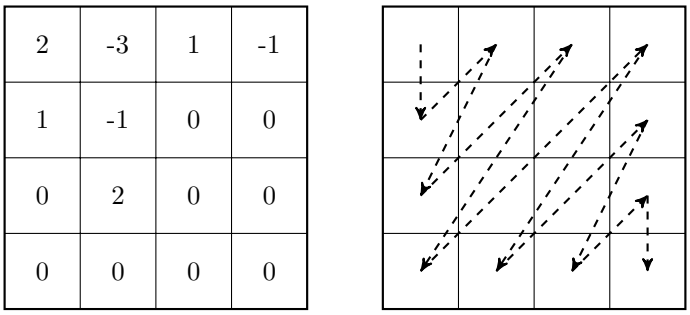
Fixed-length binarization is commonly used when the symbols are evenly distributed. On the other hand, the smallest symbol values are much more frequently found for many syntax elements. Unary, truncated unary, and exponential Golomb are more efficient binarization schemes under these conditions. Unary and truncated unary binarization is more efficient up to $N = 4$ than exponential Golomb. The latter is the preferred option for $N \geq 5$ as the length of the bin sequence is logarithmically growing compared to the linear growth of the unary and truncated unary binarization schemes.

2.3.4 HEVC CABAC

The H.264 CABAC implementation has been significantly improved in HEVC to allow higher throughput. The main improvements are described by Sze et al. [44] and a quantitative throughput comparison is provided in [46]. First of all, the overall number of bins was reduced by using optimized binarization schemes and inferring the values of some bins. A hierarchical approach in the transform unit coding also avoids redundant bins. The number of cc bins was reduced, e.g. by replacing many of them with bc bins for motion vector difference and transform coefficient level coding. The grouping of bins with the same context models reduces speculative memory accesses when attempting to decode multiple cc bins simultaneously, e.g. when coding the significance map. The same goal is targeted when reducing context model selection dependencies. The grouping of bc bins can enable additional parallelization opportunities. Their decoding is relatively simple and multiple bc bins can potentially be decoded at the same time. This has been applied to motion vector difference coding and especially for transform coefficient levels and sign bits. Parsing dependencies with other decoder modules might stall the whole entropy decoding process which is already a throughput bottleneck. To avoid this, CABAC decoding has been decoupled from all other modules, e.g. by removing the dependence on the merge candidate list generation. Finally, the reduction of memory requirements was one of the main goals as memory accesses are often part of the critical path and significantly contribute to the energy consumption of the decoder. This was achieved by reducing the number of context models from 447 in H.264 to 154. In addition to that, a 20× reduction of the line buffer size was achieved.

The implementation of the arithmetic coding process in HEVC is realized using integer arithmetic for reduced complexity. This requires two things: a suitable representation of range and offset, as well as renormalization. The state of the arithmetic coder can be represented by two unsigned integer variables. One of them is used for the size of the range. In the encoder, the other one (low) stores the lower boundary of the range (see Figure 2.9). The second variable in the decoder (offset) stores the distance of the offset relative to the lower boundary of the range. Renormalization is necessary because a 32-bit offset variable cannot contain all of potentially millions of fractional digits of the real offset. Therefore, only the most significant bits are kept in the offset variable. Furthermore, the size of the range can also become infinitely small. To address this issue, the size of the range is always kept in the interval [256,510]. Whenever a selected subrange is smaller than 256, both, the range and low/offset are repeatedly multiplied by two (left-shifted) until the range is in the allowed interval. At the same time, new fractional bits from the bitstream are inserted in the least significant positions of the offset. It is common to implement an offset buffer in the eight rightmost offset bits. This buffer contains up to eight bits more than actually needed for the decoding of the current bin, thereby allowing a more efficient refill of the buffer due to the byte-wise memory access.

HEVC uses context models for most of its syntax elements, e.g. for block partitioning of the coding and transform quadtree, for inter and intra prediction, for TBs and for the SAO filter. In the following, we will provide a detailed description of the TB syntax elements. They contribute most bins in high-quality video coding and we will refer to



Coefficients (reverse order)	**-1**	**0**	**2**	**0**	**1**	**-1**	**0**	**-3**	**1**	**2**
last_sig_coeff_x	3									
last_sig_coeff_y	0									
sig_coeff_flag		0	1	0	1	1	0	1	1	1
coeff_abs_lvl_greater1_flag	0		1		0	0		1	0	1
coeff_abs_lvl_greater2_flag			0							
coeff_abs_lvl_remaining								1		0
coeff_sign_flag	1		0		0	1		1	0	0

syntax elements

111	0	010110111	0100101	0	100	1001100

bins

Figure 2.11: TB compression example.

them frequently in this thesis. An overview of the distribution of syntax elements for different video quality levels is provided in [39].

Figure 2.11 illustrates how a 4×4 TB can be represented by the corresponding syntax elements. First of all, the coefficients are parsed in a specific scan order. This example assumes a diagonal scan order, however, horizontal or vertical orders may also be used. The position of the last non-zero coefficient in scan order is coded by the *last_sig_coeff_x* and *last_sig_coeff_y* syntax elements. In this case, it is the coefficient in the top-right position of the TB at the coordinates (3,0). Beginning from this position, all coefficients are parsed in reverse scan order and a *sig_coeff_flag* is assigned to all of them to indicate whether they are equal to zero or not. The last significant coefficient does not require this flag because it is implicitly set to 1 and does not need to be coded. This information completes the significance map. As a next step, the level of the significant coefficients needs to be determined. Therefore, a *coeff_abs_lvl_greater1_flag* is signaled for up to eight coefficients to indicate whether their absolute level is greater than one. The first coefficient with a positive *coeff_abs_lvl_greater1_flag* also gets a *coeff_abs_lvl_greater2_flag*. All coefficients whose absolute levels are not determined yet

are assigned a *coeff_abs_lvl_remaining* syntax element which contains the difference between the absolute level and the level that has been determined so far by other syntax elements. Finally, one *coeff_sign_flag* is assigned to every significant coefficient. The syntax elements are then binarized if necessary, e.g. *last_sig_coeff_x* $= 3$ is binarized as 111_2 (truncated unary, $N_{max} = 3$) and *coeff_abs_lvl_remaining* $= 1$ is binarized as 10_2 (truncated unary for small values, exponential Golomb for larger ones). Afterwards, the syntax elements are grouped to conform with the throughput improvement techniques described above, i.e. grouping of bins with the same context model and grouping of bc bins.

2.3.5 Other Entropy Coding Methods

There are other entropy coding methods besides CABAC. The most relevant in the context of this thesis will be discussed in the following.

Huffman coding [50] is one of the most well-known entropy coding methods. It generates a codeword table for all symbols based on their frequencies. As with all entropy coding methods, more frequent symbols are coded with fewer bits. Every symbol is then replaced by its corresponding codeword during compression and vice versa during decompression. The main limitations are the restriction to integer-sized codewords and the complex adaptation. Huffman coding can compress data at minimum entropy when the probabilities of the symbols are the inverse of powers of two. Only under these circumstances, Huffman coding can provide the same compression efficiency as arithmetic coding. In other cases, especially for probabilities that are greater than 50%, the restriction to integer-sized codewords constrains the compression rate. This can be partly addressed by assigning codewords to combinations of symbols, however, the exponential growth of the codeword table limits this approach to only a few symbols. When the probabilities change dynamically, the whole Huffman codeword table has to be regenerated, which is impractical in real-time applications such as video coding. This is much easier in CABAC as only the corresponding context model needs to be updated. CAVLC is based on Huffman coding and has been used in H.264 as one of two entropy coding methods besides CABAC.

Probability Interval Partitioning Entropy (PIPE, [51] [52]) coding has been developed to address the throughput limitations of CABAC. It divides the probability interval (0.0,0.5] into multiple partitions and distributes all bins among them according to their probabilities. Multiple bins per partition can be coded at the same time because the corresponding codewords are precomputed with a representative probability for every interval. Furthermore, the bins for all partitions can also be coded in parallel. This allows a highly parallel entropy coding algorithm. The main drawbacks of PIPE are the additional hardware requirements for the parallel interval processing and the reduced compression efficiency compared to arithmetic coding due to the probability approximation. Kirchhoffer et al. report an average 0.5% bitrate increase compared to arithmetic coding when combining PIPE with a set of systematic variable-to-variable length codes [52].

Arithmetic coding can also be used with N-ary symbols by dividing the range into N subranges. While this allows removing the binarization process, it is computationally more complex. It requires to store $N-1$ probabilities and to perform the same number of range partitioning operations and comparisons. Binary arithmetic coding is often preferred due to its simplicity as it requires only one probability to be stored. The encoding can be done with one subrange calculation and only one comparison is needed to determine the bin value at the decoder.

CHAPTER 3

Related Work

This chapter provides an overview of the state-of-the-art in CABAC decoding. The analysis is not limited to HEVC CABAC because the CABAC implementation in H.264 is very similar and most of the optimizations proposed for H.264 can also be applied to HEVC. On the other hand, HEVC CABAC has been designed for high-throughput processing [44] which enables new optimization approaches.

The performance of a CABAC decoder can be decomposed into two components: its throughput T_{cyc} in bins per cycle and the clock frequency f in cycles per second it operates at. The overall throughput T in bins per second is

$$T = T_{cyc} \cdot f$$

An improvement in either of the components T_{cyc} and f directly results in the same improvement in the overall throughput T. T_{cyc} can be increased by decoding multiple bins simultaneously while f benefits from architectural improvements. A strict separation of parallelism and clock frequency is not possible in most cases. Many approaches implement complex parallelization techniques that affect the critical path of the decoder and consequently lead to slightly reduced clock frequencies. This chapter is structured according to the two components that determine the throughput in CABAC decoding. First, we will analyze techniques for parallel CABAC decoding on different levels, i.e. CTU- and bin-level. Frame-level parallelization techniques are not discussed because there are no dependencies in the CABAC decoding process between different frames. Therefore, the parallelization at this level is only limited by computational and memory resources. We also analyze architectural optimizations such as pipelining, context model caches and data path optimizations. We derive research ideas based on the analysis of existing works, which will be discussed in more detail in the following chapters.

3.1 CTU-level Parallelism

There are substantial parallelization opportunities within a frame in HEVC and also in other video coding standards. Multiple high-level parallelization techniques are specified

in the HEVC standard to exploit the capabilities of modern multi- and many-core processors: Slices, Tiles and WPP. They have been described in Section 2.1 (see Figure 2.5). Slices can divide a frame into multiple independent areas and there must be at least one slice per frame. Consequently, Slices can be used for intra- and inter-frame parallelization. Tiles are restricted to only one frame which can be divided into rectangular areas. WPP processes consecutive rows of CTUs in parallel with a horizontal offset of at least two CTUs. This allows exploiting prediction opportunities between vertically neighboring blocks and consequently leads to smaller coding losses than Slices and Tiles. Furthermore, context models are forwarded to the next row after processing the second CTU which enables a faster context adaptation process than Slices and Tiles offer.

Chi et al. found that Tiles and especially Slices induce a much higher coding penalty when dividing a frame into a similar number of partitions as WPP [30]. The average coding losses are 8.8% with one slice per CTU row (17 rows), 3.9% with 6×3 tiles, and 1.4% with WPP (also 17 CTU rows) for a FHD testset [23]. Similar results are also presented for higher video resolutions. The difference is mainly due to the restricted inter-slice and inter-tile prediction opportunities, as well as overhead in the slice header. While all three techniques depend on even load balancing for good parallel scalability, the parallel efficiency of WPP also suffers from a ramp-up and -down phase in the number of active threads within a frame. When the processing of a frame is started, only one thread can be active. After it passed the second CTU, the second thread can start to process the next CTU row. In a FHD frame (30×17 CTUs), not even all threads are allowed to start processing their corresponding CTU rows when the first thread has finished its work. So, the maximum amount of parallelism is never reached. When multiple frames are processed simultaneously, the ramp-up/down problem becomes negligible as it occurs only once per video instead of once per frame. However, the simultaneous processing of multiple frames is very limited for low-delay applications and also for embedded processors due to their restricted computational and memory resources.

Multiple works exist that aim at addressing the shortcomings of WPP. Chi et al. proposed OWF to extend WPP to multiple frames [30]. OWF substantially improves the parallel scalability but puts restrictions on the vertical motion vector components. This means that it cannot be used for HEVC decoding in general, but only if motion vectors are restricted in the encoder accordingly. Inter-Frame Wavefront processing has been proposed for HEVC encoding by Chen et al. [41]. The authors exploited additional parallelization opportunities compared to OWF by taking the dependencies between consecutive uni- and bi-predicted frames into account. Zhang et al. developed a mathematical model for WPP and experimented with different optimization techniques, i.e. smaller CTU sizes, and combining WPP with slice- and frame-level parallelism [42]. However, the assumption of equal processing time for all CTUs is highly optimistic because there can be huge differences due to the varying complexity of the corresponding frame content.

We propose multiple fine-grained WPP implementations (FG-WPP, [7]) to address the shortcomings of conventional WPP and the other high-level parallelization techniques (see Table 3.1 for a comparison). We are able to substantially increase the parallel intra-frame efficiency of WPP by performing more fine-grained dependency checks as

Table 3.1: Comparison of high-level parallelization tools in HEVC.

	Intra-Frame Scalability	Inter-Frame Scalability	Coding Losses	Full HEVC Conformance
Slices	high	high	high	✓
Tiles	high	low	medium	✓
WPP	medium	medium	low	✓
OWF	medium	high	low	✗
FG-WPP	high	medium	low	✓

on a CTU level. This can be achieved with the same low coding losses as conventional WPP and also with negligible implementation overhead. The increased intra-frame scalability is especially beneficial for low-delay applications, as well as for performance- and memory-constrained architectures. Furthermore, it is orthogonal to frame-level parallelization techniques and can be combined with them. A detailed description of the FG-WPP implementations for CABAC decoding is provided in Chapter 4. Although it is limited to CABAC decoding in this thesis, the approach can also be extended to the other main modules in the HEVC decoder [5].

3.2 Bin-level Parallelism

There are multiple dependencies between consecutive bins that make their parallel decoding very challenging. First of all, the range and offset variables are required for the decoding of all bins. So they need to be updated before the next bin can be decoded. A special case occurs during the decoding of bc bins. As the renormalization of the range compensates for their division into two equally sized subranges, the range variable always keeps the same value when a bc bin is decoded. The decoding of cc bins is more complex because context models are involved. This allows the division of the range at arbitrary positions. Furthermore, the context model adaptation also requires an update before the decoding of the next bin. Although the grouping of bins with the same context models allows some optimizations regarding the context model memory access, it introduces more frequent context model dependencies between consecutive bins. Finally, the values of many bins determine the next syntax elements that need to be decoded. This means that the context model selection can only be initiated after the previous bin is decoded. The number and versatility of bin-to-bin dependencies limits bin-level parallelization approaches to very few bins in the general case.

A parallel CABAC algorithm has been proposed by Sze et al. [56]. The range is divided into four subranges to decode two bins in parallel. In general, this approach can be used for an arbitrary number of parallel bins, however, it is limited to two or three bins in practice due to the exponential complexity increase. While this allows a significant average throughput gain of 1.37× to 1.79× for different video quality levels with a 1.3% bitrate increase, the reported theoretical reduction in energy consumption is not

evaluated. It is also not evaluated how much of the throughput gain is compensated by the reduced clock frequency due to the more complex circuit. Multi-symbol cc bin decoding has also been implemented by Lin et al. [53]. The decoding of multiple cc bins in parallel requires multiple context models which can only be provided by multi-port context model memories. MPS speculation is implemented in the CABAC decoder of Kim and Park [72]. Although it is just a concatenation of two bin decoders, the update process is simplified because the second decoder speculates that the first decodes the MPS. This allows optimizations in the update processes which shortens the critical path. As a result, two bins together can be decoded faster than if they were decoded separately, however, sometimes the second bin needs to be discarded in case the speculation was wrong. Another speculative technique is proposed by Yang et al. [60]. It relies on situations when a context model update does not modify its state. In this case, the consecutive decoding of two cc bins can be simplified. Although this situation is quite rare in common use cases, it might appear very frequently for significance flags and coefficient level flags in high-quality videos when using HEVC. Liao et al. [59] improve the decoding of two cc bins even more. They merge two bin decoders and perform mathematical transformations to precompute all subranges in a highly parallel way. Speculation is also not required in this work. Another CABAC decoder was presented by Chen et al. [61]. It can process combinations of two cc bins, two bc bins, and a cc bin followed by a bc bin. Two concatenated bin decoders are used for that. Furthermore, the authors employed a prediction-based parallel processing method as well as a context memory reallocation scheme to improve the utilization of the two-bin arithmetic decoder. Zhang et al. implemented a variable-bin-rate decoder [62]. It is based on the observation that a single bit in the compressed bitstream might be used to decode multiple cc bins as long as the MPS is decoded for all of them.

The parallel decoding is much easier for bc bins since context models are not involved and the range is not updated. It is also very efficient in HEVC as it can be used more frequently because of the grouping of bc bins. Furthermore, the group size is also often known before which removes parsing dependencies. The hardware decoding of multiple bc bins per clock cycle has been implemented among others by Sze [71], Habermann et al. [1] [3] and Shi et al. [73]. In this thesis, we also present a technique for highly parallel software bc bin decoding. GPPs suffer from frequent hardly predictable branches in CABAC decoding. Replacing the code by branch-free but more complex arithmetic instructions leads to throughput improvements. The exploitation of bc bin grouping in HEVC and the implementation of a bc bin reservoir allows getting significant speedups on a wide range of video qualities and characteristics.

Plenty of research on high-performance HEVC software decoding has been performed, however, it is either focused on vectorization of all decoder modules except CABAC [31] or on high-level parallelization techniques [30]. On the other hand, only a few works for CABAC decoding on GPPs exist as most of the proposed optimizations are hardware-based. A TLP approach has been presented by Chen et al. [63] with three separate bitstream partitions. Other bitstream partitioning approaches that can be used on GPPs, although mainly developed for hardware decoders, include Syntax Element Partitioning (SEP [70] [69]) by Sze and Bin-Based Bitstream Partitioning (B3P [4][6]) by Habermann et al.. Optimization methods for Very Long Instruction Word processors

Table 3.2: Comparison of bitstream partitioning approaches and high-level parallelization tools. Tiles are used in a configuration that allows similar speedups as WPP and B3P, i.e. 3×2 tiles.

	Speedup	Hardware Cost	Coding Losses
WPP	high	high	low
Tiles	high	high	medium
SEP	medium	low	low
B3P	high	low	low

have been proposed in two US patents [66][67]. The bc bin reservoir is the first low-level CABAC optimization technique for HEVC decoding on GPPs. A detailed description and evaluation of the bc bin reservoir is provided in Chapter 7.

Many of the bin-to-bin dependencies can be removed when parallel bin decoding is applied on a slightly higher level, but still within a CTU. Bitstream partitioning approaches have been proposed that distribute specific types of syntax elements among different partitions in the compressed bitstream, e.g. syntax elements for prediction information and for luma and chroma transform coefficients. In this case, only high-level parsing dependencies need to be considered to allow the parallel decoding of all bitstream partitions. In P3-CABAC [63] Chen et al. divide all H.264 syntax elements into three groups for parallel decoding on multi-core processors. SEP is a hardware approach by Sze that divides H.264 syntax elements into five groups which are dynamically assigned to three bitstream partitions for better load balancing. SEP can be a good alternative for CTU-level parallelization approaches such as Tiles and WPP because it does not require a full replication of the decoding hardware. Only 70% more resources are needed for 3× parallelization. Furthermore, the coding losses are much smaller. The main drawback of SEP is that it performs well mainly for low video qualities when the real-time throughput requirements for CABAC decoding are relatively low. We address this shortcoming by proposing an HEVC-based bitstream partitioning approach (B3P) that provides the highest speedups for high-quality videos, i.e. when CABAC decoding is the most critical part in the overall video decoder. The implementation of eight partitions allows a higher degree of parallelization than SEP. On the other hand, even slightly fewer hardware resources are needed due to a static partitioning approach that still allows the highest speedups for high-quality videos and similar coding losses. A qualitative comparison of SEP, B3P, as well as the high-level parallelization tools WPP and Tiles is provided in Table 3.2. A more detailed description and evaluation of our bitstream partitioning approach is presented in Chapter 5.

3.3 Architectural Optimizations

Increasing the clock frequency, i.e. the number of clock cycles per second, is the only way to improve the overall throughput when the number of bins per clock cycle is at

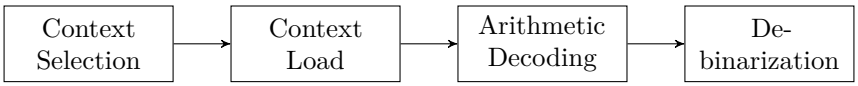


Figure 3.1: Four-stage CABAC decoder pipeline.

its limit. In the following, we will discuss existing work for multiple optimization approaches. Pipelining is probably the most well-established technique for clock frequency improvements and can be found in most CABAC decoders. We will further look at context model caches as they accelerate the context model memory access which is most often in the critical path. Other data path optimizations are also discussed.

3.3.1 Pipelining

A pipeline in the CABAC decoder divides the decoding process for a bin into multiple steps. Instead of processing all steps for a bin before decoding the next one, the decoding of the second bin can be started as soon as the first step for the previous bin has been completed. In this way, the decoding of multiple consecutive bins can be overlapped. The same throughput in bins per cycle can be reached when the pipeline is filled, however, much higher clock frequencies can be used as the critical path of the decoding pipeline is also divided. Efficient pipelining depends on an even distribution of the critical path among the pipeline stages as the achievable clock frequency is determined by the longest stage.

Most works agree on a conceptual pipeline of four stages (see Figure 3.1). The context model that is needed for the decoding of the next bin is determined in the context selection stage. Afterwards, the context model memory is accessed in the context load stage. Next, the actual binary arithmetic decoding task is performed before the final debinarization step produces the decoded syntax element. Neighboring stages are often merged because the dependencies between consecutive bins limit the pipelining approach to only a few stages. In general, the outcome of the debinarization stage might be needed as input for the context selection stage. In this case, pipeline stalls cannot be avoided and the throughput is reduced. Binary arithmetic coding allows removing some dependencies because a bin can only have two different values. This was exploited by Habermann [1] when implementing the context selection stage for both possible bin values. As soon as the bin is provided by the arithmetic decoding stage, the correct context model can be selected. This technique allows removing a lot of pipeline stalls at a slightly increased hardware cost. The same strategy can be applied at the debinarization stage, which can be merged with the arithmetic decoding stage for a shorter pipeline with the same throughput.

Decoders with three pipeline stages were presented by Yi and Park [75], Kim and Park [72] and Chang [74]. Shi et al. designed a four-stage pipeline that resolves all structural and data hazards with forwarding and redundant circuits [73]. Chen and Sze [71] proposed an even deeper pipeline than the one presented in Figure 3.1. A fifth pipeline stage is added at the beginning to compute a binary decision tree that

Table 3.3: Comparison of CABAC decoders with context model caches.

	Cache Design Space	Prefetching	Evaluation
Yi et al. [75]	1×8	none	performance
Yang et al. [54]	2×9	none	performance
Hong et al. [55]	2×?	static	performance
Habermann et al. [3]	1×4 - 64×4 1×8 - 32×8	adaptive	performance, miss rate, energy efficiency

is used for state prefetching in the remaining stages, thereby reducing pipeline stalls. Additionally, their implementation decodes up to two bc bins per cycle, resulting in a throughput of up to 1.7 Gbins/s, which represents the state-of-the-art.

3.3.2 Context Model Cache

The context model memory access is often part of the critical path. Consequently, the context load stage often determines the clock frequency in a pipelined design. The context load stage can be shortened when the context model memory is replaced by a small cache. The stage might be even removed when the cache access fits into an adjacent stage. The shorter pipeline might then result in fewer pipeline stalls. Cached designs for H.264 CABAC decoding have been proposed by Yi and Park [75], Yang and Guo [54] and Hong et al. [55], but potential performance degradations due to cache misses have not been evaluated. Prefetching was used to reduce the cache miss rate [55], but results for this optimization were also not provided. Our work evaluates both, the cache miss rate and the effectiveness of prefetching [2] [3]. Furthermore, it is the first HEVC CABAC hardware decoder with a context model cache. We perform a design space exploration of different cache sizes and propose an optimized context model memory layout for HEVC. We also designed a prefetching algorithm that uses the context models' adaptation capabilities to deliver high performance for all types of video content with different characteristics. Table 3.3 shows that our evaluation is way more comprehensive than related work and can assist in selecting the best cache architecture for high performance and energy efficiency. The full design space exploration of different cache architectures is provided in Chapter 6.

3.3.3 Data Path Optimizations

Sze et al. proposed subinterval reordering to relax dependencies in the decoding process of a single cc bin and allow partly parallel processing [57] [58]. The size of the LPS range is commonly derived by a table lookup. It is then subtracted from the range to get the size of the MPS range. A comparison between the offset and the MPS range finally yields the decoded bin. This three-step process is necessary because the MPS

range is by definition the lower part of the range while the LPS range is the upper part. Switching the order of the subintervals allows the comparison of the offset to the LPS range. As a result, the subtraction operation to calculate the MPS range does not need to be performed before the comparison to determine the bin. It can be postponed and executed in parallel with other post-comparison update operations. This optimization leads to a 14% to 22% reduction in critical path delay of the circuit.

Liao et al. designed an optimized data path for the decoding of two consecutive cc bins [59]. They realize a more efficient circuit compared to the pure concatenation of two bin decoders by applying mathematical transformations and precomputing hierarchical subranges. Another optimization in this decoder affects the context model memory design. Up to three context models are needed for the decoding of two cc bins in one clock cycle. This cannot be realized with a single SRAM. On the other hand, storing all context models in registers results in very high hardware cost. The authors distributed all context models among SRAM and registers based on the analysis which context models can be used together. This hybrid memory significantly reduces the hardware cost while maintaining the throughput of two cc bins per clock cycle.

3.4 Summary

The performance of a CABAC decoder is determined by two components, i.e. its throughput in bins per clock cycle as well as its clock frequency. The former can be improved with parallelization techniques at different levels while the latter benefits from architectural optimizations such as pipelining, context model caches, and other data path optimizations. The analysis of existing work has shown a lot of potential for further improvement. We will present different approaches from most of these categories in the following chapters which address the shortcomings of existing works.

CHAPTER 4

Wavefront Parallel Processing

WPP [35] is a high-level parallelization technique that provides better compression efficiency compared to Slices and Tiles [30] as discussed in Section 2.1.3. On the other hand, it suffers from a ramp-up and -down in the number of active parallel threads due to the delayed decoding start of consecutive CTU rows in HEVC. This is not an issue when multiple frames are decoded simultaneously, however, the decoding is most often limited to a few frames at a time on memory-constrained systems and especially for low-delay applications, such as video streaming/conferencing. A more efficient WPP implementation can increase the parallelism per frame and thereby reduce the required number of frames in flight to reach the target performance.

CABAC decoding is especially challenging for low-delay applications. First, the restricted encoding time usually leads to higher bitrates which results in more work for the CABAC decoder. Second, the computational work for motion compensation – commonly the main throughput bottleneck besides CABAC – is significantly reduced when only uniprediction is used, which is very common in low-delay video coding. Furthermore, many typical low-delay applications, such as video conferencing/chatting, have low motion in general due to a static background and little movement of persons. Consequently, the relative decoding time for CABAC is increased.

Efficient CABAC decoding requires the exploitation of TLP as with WPP to compensate for the lack of DLP and reach similar throughput as other video coding tasks. A theoretical analysis of the worst-case dependencies of the HEVC decoding tasks with respect to WPP is provided in [5]. It shows that a more efficient implementation than conventional WPP is feasible, especially for CABAC. This work aims to exploit this opportunity, thereby making the following contributions to efficient parallel video decoding.

- An improved WPP implementation for CABAC decoding with no overhead in the dependency check.
- Two fine-grained WPP implementations with dependency checks at CU and syntax element granularity. Both significantly improve parallel efficiency while introducing only a small overhead.

The work is based on the following publications:

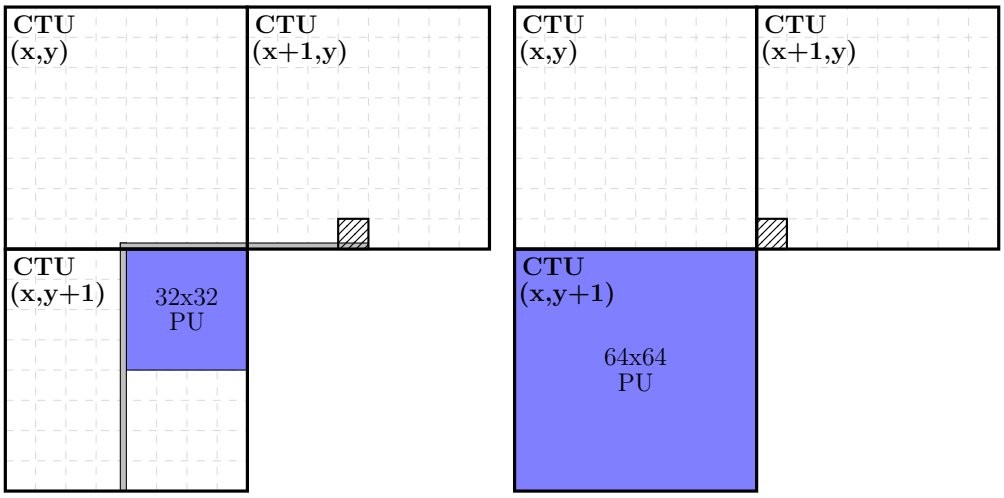
- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Improved Wavefront Parallel Processing for HEVC Decoding"*, Proceedings of the 13th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES 2017), pp. 253-256, Fiuggi, Italy, July 2017
- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Efficient Wavefront Parallel Processing for HEVC CABAC Decoding"*, Proceedings of the 28th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2020), pp. 339-343, Västerås, Sweden, March 2020

An analysis of the CABAC decoding dependencies with respect to WPP is provided in Section 4.1. Based on the analysis we present three improved implementations in Section 4.2. The performance of these implementations in terms of speedup and parallel efficiency is evaluated in Section 4.3. Finally, the work is concluded in Section 4.4.

4.1 Dependency Analysis

WPP is commonly implemented with a horizontal offset of two CTUs between consecutive rows (WPP2) to satisfy all decoding dependencies (see Figure 2.5c), including the context model initialization for CABAC decoding, intra prediction, and motion vector derivation. However, this two-CTU offset is only needed at the beginning of each CTU row for the context model initialization. Afterwards, the CTU row processing threads could be allowed to come below the two-CTU threshold. This is possible because the decoding of different CTU rows requires different amounts of time due to the varying complexity of the corresponding frame content. The worst case dependencies in terms of distance between consecutive CTU rows are between one and two CTUs for intra prediction and motion vector derivation [5] (see Figure 4.1). The distance is less than one CTU for CABAC decoding after the context model initialization which allows a more efficient WPP implementation when decoupling it from the reconstruction process. The number of active threads can be increased faster, thus leading to a larger amount of parallelism and consequently a lower delay in decoding. The following dependencies need to be considered for CABAC decoding when WPP is used.

1. The **context model initialization** at the beginning of each CTU row needs the context models after the decoding of the second CTU from the row above.
2. The context model selection for the **split_cu_flag** is based on the coding quadtree depth of the above CU.
3. The context model selection for the **cu_skip_flag** depends on the same syntax element in the above CU.
4. An **intra prediction mode candidate** from the above PU might be needed to determine the intra prediction mode for the current PU.



(a) Intra prediction **(b)** Motion vector derivation

Figure 4.1: Worst-case dependencies in WPP. PUs require information from the hatched CUs.

Dependencies 2, 3 and 4 only need to be checked if the CU above is not in the same CTU. Otherwise, the availability of the required information is ensured by the z-scan order inside a CTU.

4.2 Fine-grained WPP

Based on the analysis of the CABAC decoding dependencies we propose three methods for more efficient WPP CABAC decoding. First, a **one-CTU offset (WPP1)** is implemented. Of course, a two-CTU offset must be ensured at the beginning of each CTU row for the context model initialization. Afterwards, this distance is reduced to one CTU. This method allows for the faster progression of dependent threads and does not add any implementation overhead as the dependency check is still performed at the CTU level. Two more fine-grained WPP methods work as the previous one but allow threads to come even below the one-CTU offset. They differ in the granularity at which dependency checks are performed: at SE level or CU level. For **syntax-element-based WPP (SE-WPP)**, dependency checks are performed when a split_cu_flag or cu_skip_flag needs to be decoded or if an intra prediction mode candidate is required for one of the top CUs of a CTU. If the required information from the CTU above is not yet available, the processing thread has to be stalled until then. The fine-grained dependency check adds some implementation overhead, however, the required data is commonly already stored in the line buffers and therefore easily accessible. **CU-based WPP (CU-WPP)** aims to reduce the implementation overhead by performing more coarse-grained dependency checking. CUs are marked as available as soon as all prediction

information is decoded but before the decoding of the transform tree. There can be a significant amount of transform tree data, especially for high-quality videos. However, the information is never needed by the dependent thread. Before the decoding of a CU, this thread checks if the above neighboring CUs are available. This can be implemented with one integer counter per CTU row that is incremented in z-scan order.

4.3 Evaluation

A behavioral model of a CABAC decoder that can decode one cc bin or up to two bc bins per clock cycle (as implemented in [3]) has been integrated into the HEVC reference software [10]. It is used to simulate the behavior for WPP2, WPP1, CU-WPP and SE-WPP CABAC decoding with one thread per CTU row. The JCT-VC common test conditions (class A-F) [23] are used for evaluation. Every class consists of three to five videos. The geometric mean of the single speedups is presented in the following. The videos are encoded in AI, RA and LD modes with QPs of 22, 27, 32 and 37. It should be noted that some combinations of video classes and coding modes are not specified in the common test conditions, i.e. class A in LD mode and class E in RA mode.

The evaluation is focused on the improvement of our proposed techniques compared to common WPP2, i.e. improving the parallel efficiency of WPP inside a single frame with the same number of threads. Other related techniques improve the performance due to an increased number of threads by extending WPP to multiple frames or exploiting parallelism from bitstream partitioning. Both techniques are orthogonal to our approach and can be combined with it, hence the comparison between them does not provide meaningful insights.

4.3.1 Decoding Speedup

Figure 4.2 shows the decoding speedup of the proposed methods over conventional WPP2 for the full test videos with 64×64 CTUs. The results for single frames are highly variant and range from 1.00× (same performance) up to 2.56× because the speedup highly depends on the frame content. The optimal case is low complexity on the left side of the frame and high complexity and therefore lots of decoding work in the center or the right side. With this work distribution, threads can quickly process the first two CTUs of each row and allow the next thread to start very soon. In contrast to WPP2, the proposed methods allow dependent threads to catch up and not be stalled as soon as the above threads proceed slower when reaching more complex frame areas.

The WPP1 implementation gives solid speedups between 1.06× and 1.55× which is already a good improvement considering that there is no additional complexity in the dependency check. Both fine-grained implementations allow for even higher speedups in the range of 1.07× to 1.83×. Best results are achieved for class E (video conference

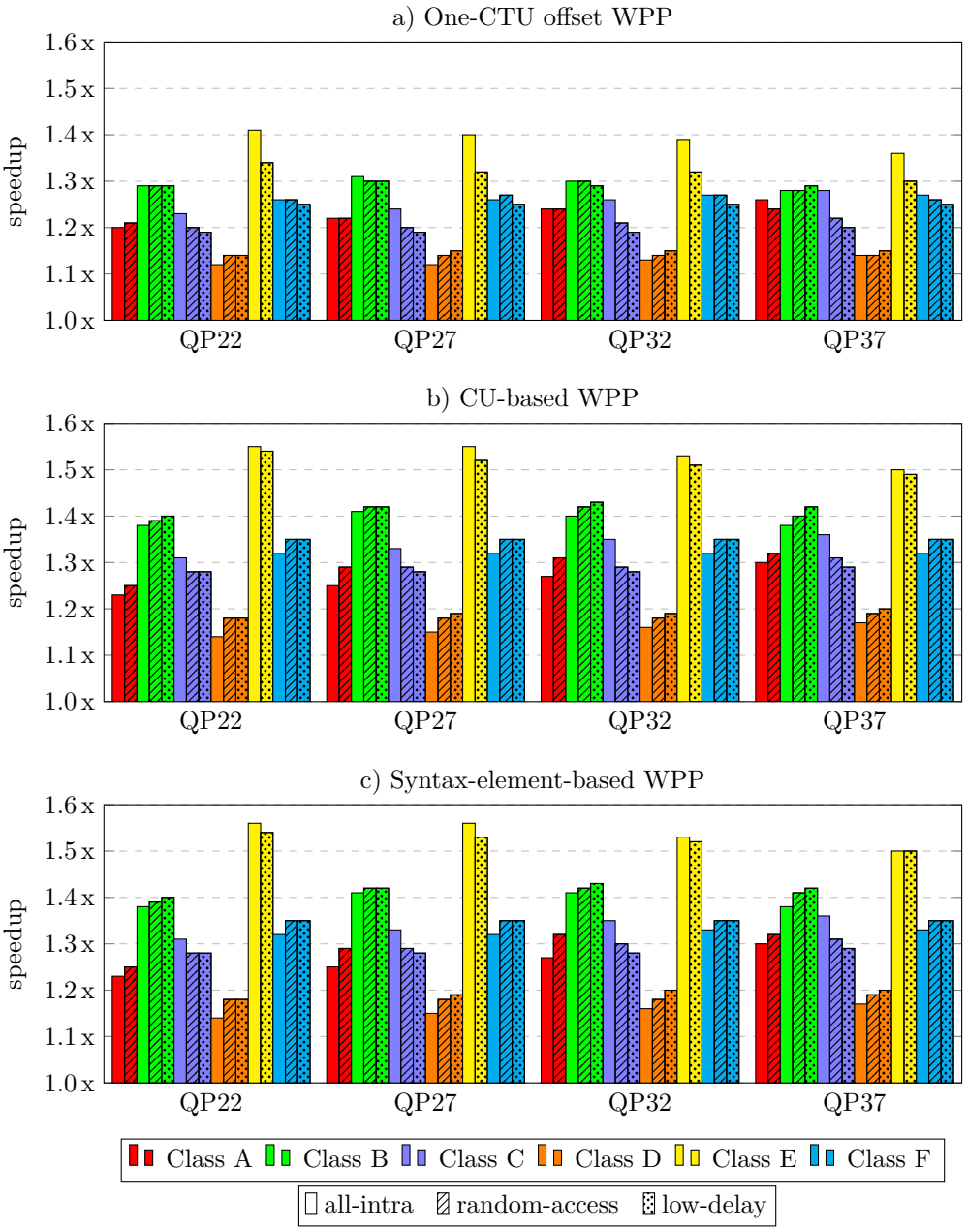


Figure 4.2: Speedup of improved WPP implementations over two-CTU offset WPP.

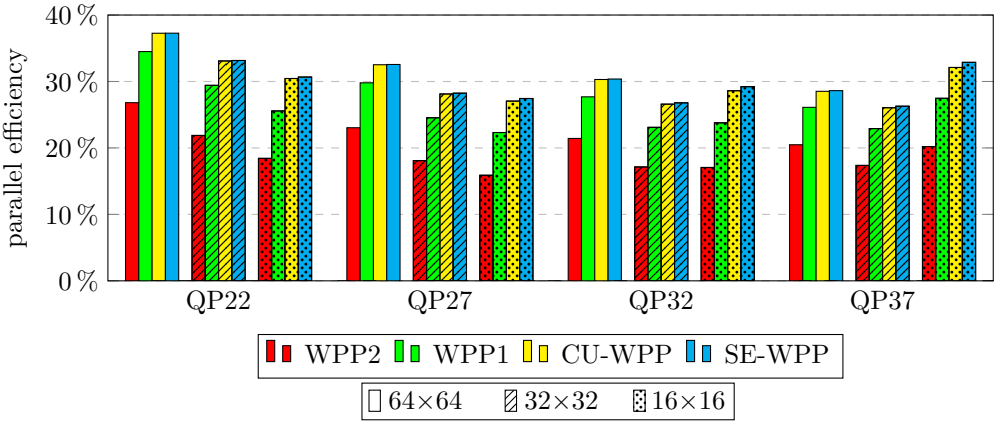


Figure 4.3: Parallel efficiency for WPP CABAC decoding of class B videos in RA mode with one thread per CTU row and different CTU sizes (64×64, 32×32, 16×16).

content). In video conferences, people are usually found in the center of the frame with a static background. This is very close to the optimal work distribution described above. High-quality video conferencing (class E in LD mode with QP 22-27) gives an average speedup of 1.52× - 1.54× for both fine-grained WPP methods. This is a very impressive result, considering that only a more fine-grained dependency check is implemented to enable more parallelism. The decoding performance for class B (FHD entertainment videos, 1.38× - 1.43× speedup) is also significantly better with fine-grained WPP.

The results for CU-WPP and SE-WPP are very similar. In fact, the difference is not even noticeable in most cases. This makes CU-WPP the superior method as it achieves almost the same speedups as SE-WPP with an implementation that is only slightly more complex than conventional WPP2.

4.3.2 Parallel Efficiency

A speedup in parallel processing can be achieved by adding more threads or by increasing the parallel efficiency of the implementation. Parallel efficiency describes how much of the maximum potential speedup can be reached with a specific number of threads. The proposed WPP implementations improve the parallel efficiency and use the same number of threads as conventional WPP2. A common way to allow more threads to work on the decoding of a frame is the reduction of the CTU size to 32×32/16×16 samples. This doubles/quadruples the number of CTU rows for WPP but also increases the bitrate, e.g. by 2.20%/12.05% for class B videos in RA mode (representative of FHD entertainment applications such as video streaming). As a result, more work has to be done by the CABAC decoder, thus reducing the parallel efficiency. Other decoder modules are also negatively affected as the smaller CTU sizes lead to decreased

vectorization opportunities due to smaller blocks.

Figure 4.3 shows the parallel efficiency of all WPP implementations in combination with different CTU sizes for class B videos in RA mode. WPP2 with 64×64 CTUs and QP 22 results in a parallel efficiency of only 26.8% on average. The use of WPP1 increases the efficiency to 34.5% and the fine-grained WPP implementations even reach 37.3%. For the same configuration, smaller CTU sizes allow a speedup as the number of parallel threads can be doubled or quadrupled, however, the parallel efficiency is decreased to 21.9%/18.4% with 32×32/16×16 CTUs. The quantitative results are slightly different for higher QPs as this affects the spatial work distribution in a frame. However, the significant parallel efficiency improvement of the proposed WPP implementations can be demonstrated for all video quality levels.

4.4 Conclusions

We have proposed three methods to improve the parallel efficiency of WPP for HEVC CABAC decoding. They differ in the granularity at which dependency checks are performed. One-CTU offset WPP already offers up to 1.55× speedup without any implementation overhead. The more fine-grained CU-based and syntax-element-based WPP implementations even allow speedups up to 1.83×. CU-based WPP is the preferred method as it can be implemented at very low cost while showing no noticeable difference in performance to syntax-element-based WPP. The proposed WPP implementations perform best for video content that is typical for low-delay applications such as video conferencing. Especially these applications benefit most from the increased parallelism within a frame because the number of simultaneously processed frames is very limited due to delay restrictions. The increased parallel efficiency is also important for memory- and performance-constrained systems that cannot process multiple frames at the same time. Both fine-grained WPP methods can be applied to other tasks of the HEVC decoder as well but the expected speedup is smaller as it is for CABAC. The proposed techniques can tolerate load imbalances better than conventional WPP and thereby reduce the impact of one of the most critical drawbacks of WPP.

WPP and its fine-grained variations exploit parallelism at the CTU level. In the next chapter, we will present a bitstream partitioning approach that allows exploiting bin-level parallelism within a CTU.

CHAPTER 5

Bin-based Bitstream Partitioning

In this chapter, we propose a modified bitstream format that enables additional thread-level parallelism for CABAC decoding: Bin-Based Bitstream Partitioning (B3P). We also present the corresponding hardware decoder to show that B3P is especially beneficial for hardware decoding in terms of performance and hardware cost. This is the main advantage compared to existing high-level parallelization approaches such as Tiles and WPP because their use requires the replication of the complete CABAC decoding hardware.

To address this issue, Sze proposed SEP [69] for H.264/AVC. Parallelism is exploited by distributing five groups of syntax elements among three different bitstream partitions so that they can be decoded simultaneously. This enables a significant decoding speedup with only minimal losses in coding efficiency. As only parts of the decoding hardware need to be replicated, there is only a 70 % increase in hardware cost. This proposal requires a modification of the bitstream format and is therefore not compliant with the H.264/AVC standard. However, the multiplication of the decoding throughput with minimal coding losses and moderate hardware requirements makes the SEP concept a promising candidate for adoption in future video compression standards. The main drawback of the proposed SEP scheme is that it performs best for low-quality videos when CABAC decoding throughput is least critical. P3-CABAC [63] is a similar proposal from Chen et al. that divides syntax elements into three groups for parallel decoding on multicore processors.

We propose an improved bitstream partitioning scheme based on HEVC CABAC. It is specifically tailored to perform best for high-quality videos when CABAC decoding throughput is the main bottleneck for the overall video decoder, as was shown in Section 2.2. In contrast to the high-level parallelization approaches that process multiple frame areas or even frames at the same time and thus require significantly more memory, our approach exploits parallelism at the syntax element and bin level which needs almost no extra buffering capacity. The parallelization at this level within a specific frame area is also beneficial for low-delay applications such as video conferencing.

This work is based on the following publications:

- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Syntax Element Partitioning for high-throughput HEVC CABAC Decoding"*, Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017), pp. 1308-1312, New Orleans, LA, USA, March 2017
- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"A Bin-Based Bitstream Partitioning Approach for Parallel CABAC Decoding in Next Generation Video Coding"*, Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2019), pp. 1053-1062, Rio de Janeiro, Brazil, May 2019

The general bitstream partitioning approach and the proposed B3P format are presented in Section 5.1. Afterwards, Section 5.2 provides a detailed overview of the B3P hardware decoder implementation. Experimental results for decoding speedup, bitstream overhead and hardware cost are discussed in Section 5.3. Finally, the work is concluded in Section 5.4.

5.1 Bitstream Format

Bitstream partitioning aims to divide a common bitstream into multiple parts that can be decoded in parallel. Figure 5.1 illustrates the effect by showing the decoding process for three groups of syntax elements. In the example, there is one for luma and one for chroma TBs, as well as a group for control information that contains all remaining syntax elements, e.g. for block partitioning, prediction modes and loop filters. In a common HEVC bitstream, all syntax elements are coded consecutively in a single partition, which makes their sequential decoding necessary (Figure 5.1a). However, if they are distributed among different partitions, parallel decoding is possible (Figure 5.1b). Luma and chroma TBs are completely independent of each other. Their decoding process can be started as soon as the corresponding control block is decoded. At the same time, the decoding of the next control block can be initiated. This allows the overlapped decoding of all three partitions, resulting in shorter decoding time. The reconstructed video is the same as with the non-partitioned bitstream because the syntax elements are not modified but only distributed differently.

The proposed B3P scheme consists of eight partitions. First, the common bitstream is divided into three parts according to the example in Figure 5.1: control, luma and chroma. Each of these partitions is further split into separate parts for cc and bc bins. The bc bins are coded without context models, which simplifies the decoding process. In fact, a bc bin corresponds to a bit and does not need to be encoded or decoded at all if it is not interleaved with cc bins in a common bitstream. This separation allows the highly parallel retrieval of bc bins as they only need to be read from memory. Unfortunately, the luma and chroma cc partitions still contain significantly more bins than others for

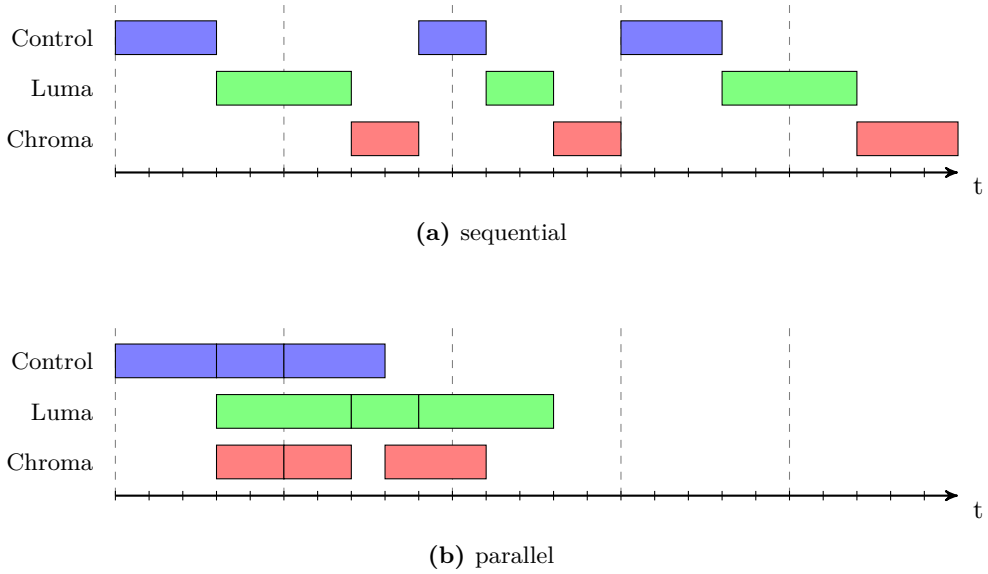


Figure 5.1: Decoding of syntax element partitions.

high-quality videos. To achieve a more balanced distribution, they are divided into two parts, one for the significance map and the other for the coefficient levels. All other bins are moved to the *Control CC/BC* partitions.

A further split into partitions for both chroma components is not gainful as they use the same context models, thus making their parallel decoding very challenging. In contrast to SEP, we use a static partitioning scheme that does not adapt to video characteristics. A dynamic scheme allows a more balanced distribution of bins among the partitions for all test sequences. However, the *Luma/Chroma Significance Map* partitions almost always contain the majority of cc bins for high-quality videos. Although the *Luma/Chroma BC* partitions contain even more bins, these bc bins can be decoded in a highly parallel way, so that the maximum speedup is still determined by the *Luma/Chroma Significance Map* partitions. As they cannot be split further, a dynamic partitioning would not lead to a higher speedup. However, the corresponding decoding hardware can be simplified for static partitions because every subdecoder only processes bins of specific syntax elements. The decoding of low-quality videos is most often dominated by the size of the *Control CC* partition and does not benefit from this static partitioning in terms of load balancing. Nevertheless, the throughput requirements for low-quality videos are also very low, so that real-time decoding is possible even without the use of B3P. An overview of the proposed distribution among bitstream partitions is provided in Table 5.1. It should be noted that some syntax elements appear in more than one partition as they consist of cc and bc bins. Also, the same syntax elements exist for luma and chroma TBs.

Table 5.1: Bitstream partitions (CC: context-coded, BC: bypass-coded)

Partition	Syntax Elements
Control CC	end_of_slice_segment_flag, end_of_subset_one_bit, sao_merge_left_flag, sao_merge_up_flag, sao_type_idx_luma, sao_type_idx_chroma, split_cu_flag, cu_transquant_bypass_flag, cu_skip_flag, pred_mode_flag, part_mode, pcm_flag, prev_intra_luma_pred_flag, intra_chroma_pred_mode, rqt_root_cbf, merge_flag, merge_idx, inter_pred_idc, ref_idx_l0, mvp_l0_flag, ref_idx_l1, mvp_l1_flag, split_transform_flag, cbf_luma, cbf_cb, cbf_cr, abs_mvd_greater0_flag, abs_mvd_greater1_flag, cu_qp_delta_abs, cu_chroma_qp_offset_flag, cu_chroma_qp_offset_idx, log2_res_scale_abs_plus1, res_scale_sign_flag, transform_skip_flag, explicit_rdpcm_flag, explicit_rdpcm_dir_flag, last_sig_coeff_x_prefix, last_sig_coeff_y_prefix, coded_sub_block_flag
Control BC	sao_type_idx_luma, sao_type_idx_chroma, sao_offset_abs, sao_offset_sign, sao_band_position, sao_eo_class_luma, sao_eo_class_chroma, part_mode, mpm_idx, rem_intra_luma_pred_mode, intra_chroma_pred_mode, merge_idx, ref_idx_l0, ref_idx_l1, abs_mvd_minus2, mvd_sign_flag, cu_qp_delta_abs, cu_qp_delta_sign_flag, last_sig_coeff_x_suffix, last_sig_coeff_y_suffix
Luma Sig Map	sig_coeff_flag
Luma Coeff Level	coeff_abs_level_greater1_flag, coeff_abs_level_greater2_flag
Luma BC	coeff_sign_flag, coeff_abs_level_remaining
Chroma Sig Map	sig_coeff_flag
Chroma Coeff Level	coeff_abs_level_greater1_flag, coeff_abs_level_greater2_flag
Chroma BC	coeff_sign_flag, coeff_abs_level_remaining

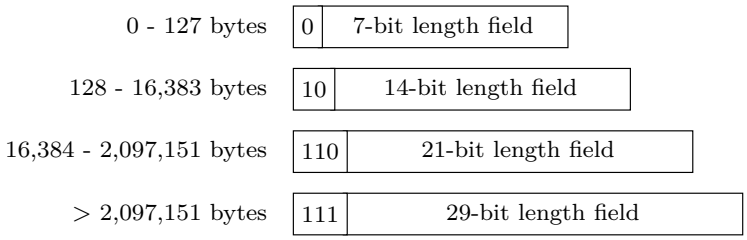


Figure 5.2: Variable-sized partition length fields.

The following features are the main improvements compared to the existing SEP scheme by Sze:

- Eight instead of three partitions to enable more parallelism
- A static partitioning scheme for reduced complexity and bitstream overhead, as well as for optimal customization of the subdecoders
- A more fine-grained partitioning based on bins instead of syntax elements
- Separate bc bin partitions for highly parallel decoding
- An optimized partitioning scheme for the best workload distribution for high-quality videos

The common bitstream format has to be modified to implement the B3P scheme. Instead of one length field for the slice bitstream, every bitstream partition needs a separate length field. To minimize the overhead, a variable-sized length field is used (see Figure 5.2). Small partitions (up to 127 bytes) will only need one byte, while only very large partitions (more than 2 MB) need four bytes. Especially low-quality videos benefit from the variable-sized length fields because the overhead constitutes a significant fraction of the overall bitstream size for them. The static partitioning scheme requires less extra bits per partition than the dynamic scheme of SEP because no information about the distribution of syntax element groups among bitstream partitions needs to be transmitted. This partly compensates for the overhead induced by the increased number of partitions.

5.2 Parallel Decoder Architecture

The B3P scheme is designed to allow a highly efficient hardware decoder implementation. First of all, eight bitstream partitions create plenty of opportunities for parallel processing. Additionally, the static partitioning scheme enables the implementation of highly specialized subdecoders. The high-level decoder architecture can be seen in Figure 5.3. It consists of a *Control Decoder* and two *Transform Block Decoders*, one for luma and

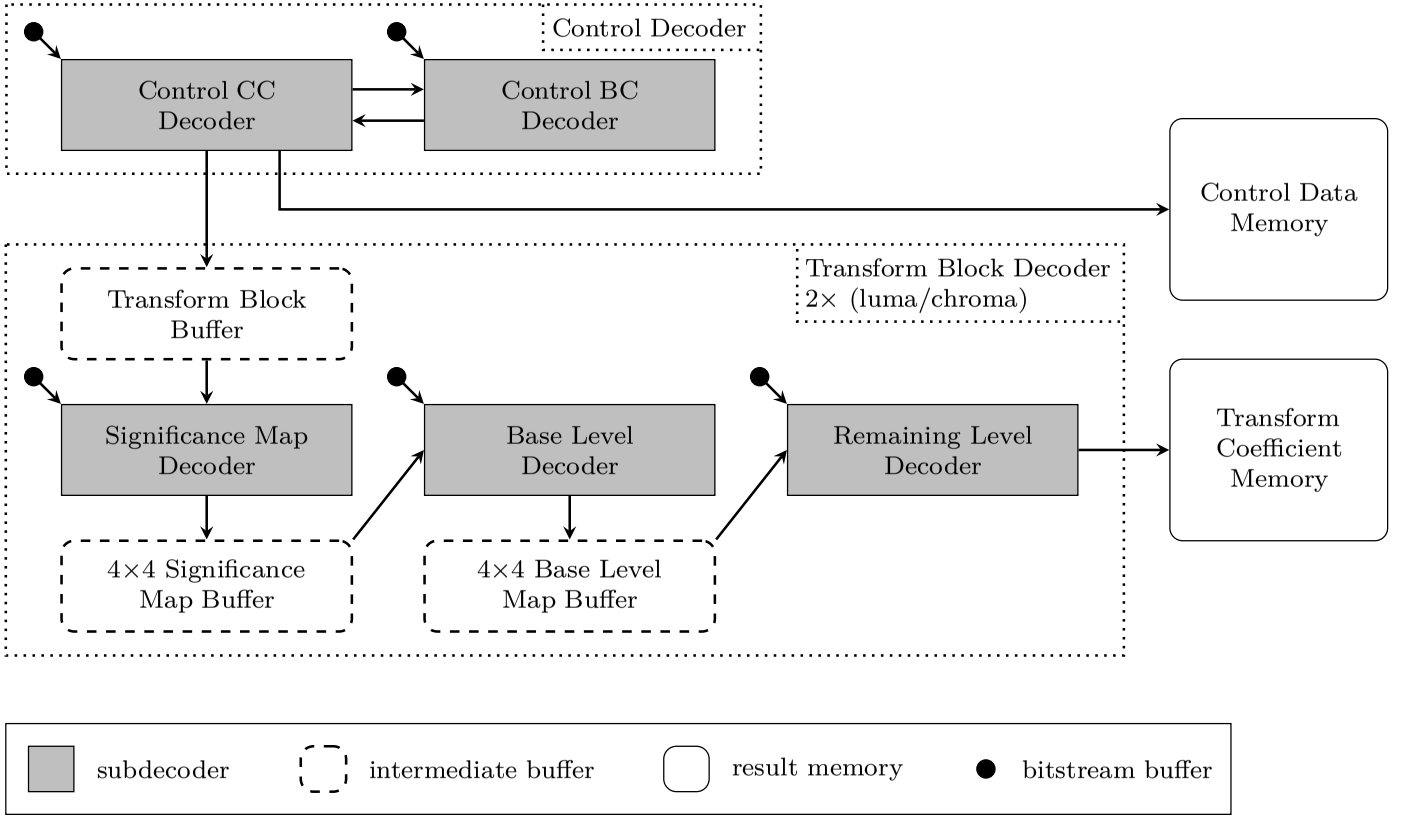


Figure 5.3: Parallel B3P decoder architecture.

one for chroma. The *Control CC/BC Decoders* process the corresponding partitions, thereby extracting information that is needed to control other decoding tasks, e.g. intra and inter prediction. It also sends job information to the *Transform Block Decoders*, such as the position and size of TBs to be decoded. The *Significance Map Decoder* reads this information from the corresponding buffer and determines the positions of all non-zero transform coefficients by decoding the *sig_coeff_flags*. This significance map is forwarded to the *Base Level Decoder* at a 4×4 block granularity to keep the size of the intermediate buffers and the latency small. The *Base Level Decoder* extracts *coeff_abs_level_greater1_flags* and *coeff_abs_level_greater2_flags* and forwards the updated 4×4 block to the *Remaining Level Decoder*. Here, the final transform coefficient levels are determined after decoding the *coeff_abs_level_remaining* syntax elements and *coeff_sign_flags*.

5.2.1 CC Bin Subdecoder

The subdecoders for cc bins, i.e. the *Control CC Decoder* as well as the *Significance Map Decoders* and *Base Level Decoders* for luma and chroma, are implemented as a two-stage pipeline (see Figure 5.4). In the context model selection stage, the required context model index is calculated based on the current state of the state machine, some settings registers, previously decoded syntax elements, and the previously decoded bin. The index is used as an address for the synchronous context model memory which separates the two pipeline stages. The size of the memory varies for all the subdecoders as different numbers of context models are needed (see Table 5.2). The table also shows that the number of states in the state machines is highly varying as it depends on the number of different syntax elements that are decoded. The actual bin decoding is performed in the binary arithmetic decoding stage, using the context model, bitstream state registers as well as data from the encoded input bitstream. The context model and bitstream state are updated and fed back to their respective memories. The decoded bin is used for debinarization in case the processed syntax element consists of multiple bins. Fully decoded syntax elements are then stored in local registers before they are finally forwarded to other B3P subdecoders or consecutive decoder components, such as intra/inter prediction, inverse transform and loop filters.

5.2.2 BC Bin Subdecoder

The architecture of the bc bin subdecoders, i.e. the *Control BC Decoder* and the *Remaining Level Decoders* for luma and chroma, are less complex than the cc bin subdecoders. Context model selection and the context model memories are not needed as the decoding is performed without context models. Furthermore, the bin decoding process is as simple as reading bits from memory because a bc bin corresponds to a bit when no cc bins are in the same bitstream partition. This allows the very efficient retrieval of multiple bc bins per clock cycle. All bc bin subdecoders can process up to 16 bc bins per clock cycle. The subdecoders need to extract fixed-length codes, sequences

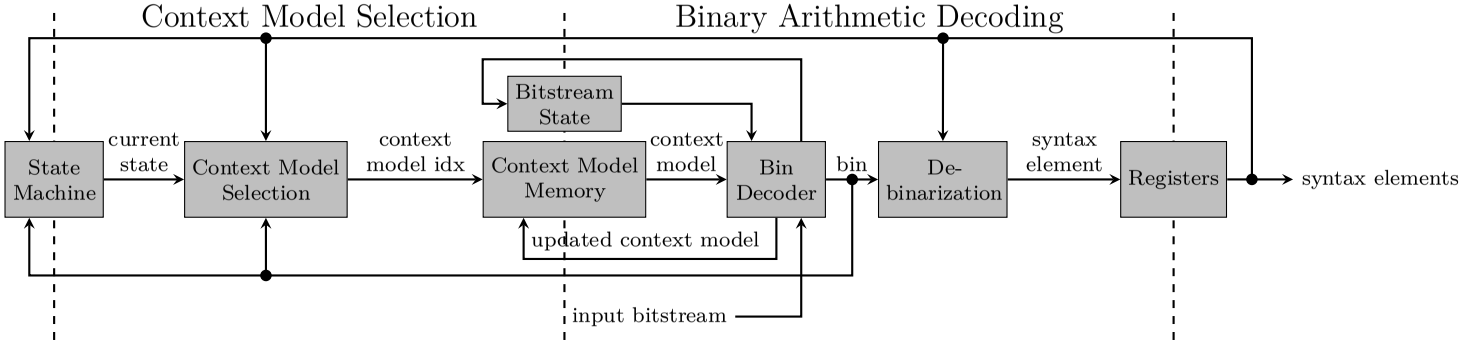


Figure 5.4: Architecture of a subdecoder for context-coded bins. The vertical dashed lines represent pipeline stage boundaries. All modules on pipeline stage boundaries are synchronous.

Table 5.2: Architectural characteristics and maximum clock frequencies for a sequential CABAC decoder and B3P subdecoders. Clock frequencies have been derived with a 32nm Synopsys standard cell library.

Decoder	# States	# Context Model Memory Entries	Clock Frequency	Speedup
Sequential	63	162	405 MHz	baseline
B3P Control CC	39	88	463 MHz	14.4%
B3P Control BC	18	0	463 MHz	14.4%
B3P Significance Map Luma	2	28	699 MHz	72.7%
B3P Significance Map Chroma	2	16	699 MHz	72.7%
B3P Base Level Luma	3	20	725 MHz	79.0%
B3P Base Level Chroma	3	10	725 MHz	79.0%
B3P Remaining Level Luma	2	0	556 MHz	37.2%
B3P Remaining Level Chroma	2	0	556 MHz	37.2%

of leading ones, as well as combinations of these two from the bitstream. While the decoding of fixed-length codes is straightforward, determining the length of a sequence of one-bits is more complex and requires a specialized circuit to be performed efficiently and with low delay. Figure 5.5 shows a circuit for determining the number of leading ones from an eight-bit input sequence. It serves to demonstrate the functionality of the circuit for any input sequence of 2^N bits and can be easily extended to the required 16 bits.

The *Remaining Level Decoders* are also implemented as a two-stage pipeline where the sequences of leading ones and fixed-length codes are extracted in the first stage while the remaining coefficient level is calculated in the second stage. The *Control BC Decoder* is not pipelined because the coefficient level calculation is not needed for the syntax elements it decodes.

5.2.3 Improvements over Sequential Decoding

Besides the parallelism that comes from the bitstream partitioning, the specialization of the subdecoders results in significant improvements in clock frequency which also contributes to the overall speedup (see Table 5.2). In comparison to a sequential decoder that also uses a two-stage pipeline, it can be seen that most of the B3P subdecoders need only a small fraction of decoder states and context model memory entries, which is the main reason for the improved clock frequencies. The *Control CC/BC Decoders* reach only a slightly higher (14.4%) clock frequency than the sequential decoder because they still contain most of the complexity, i.e. they decode most of the syntax element types. They are tightly coupled to allow efficient bidirectional data exchange. Therefore, they are operated at the same clock frequency. On the other hand, the *Significance Map Decoders* only process one syntax element type. Due to this high degree of specialization, they can operate at a 72.7% higher clock frequency. The *Base Level Decoders* process two types of syntax elements and can operate at a 79.0% higher clock frequency. Although

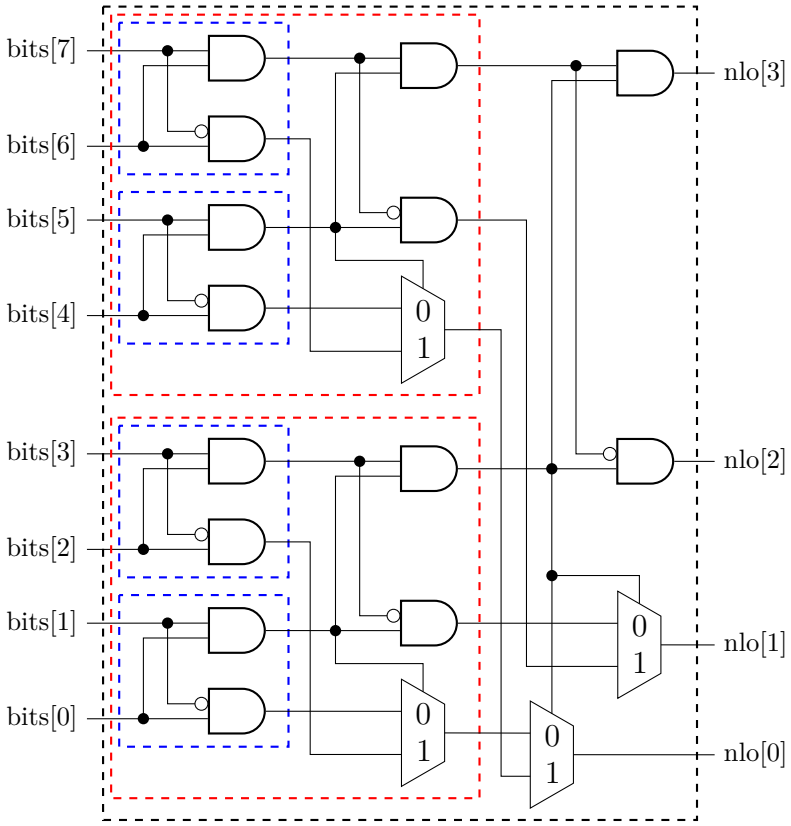


Figure 5.5: Circuit for computing the number of leading ones (nlo) from a sequence of eight bits. It is composed of smaller circuits that compute the number of leading ones for two bits (dashed blue boxes). The circuit for four bits (dashed red boxes) uses two of them, computes the two MSBs from their respective MSBs, and selects the LSBs from both of their LSBs. The same strategy is used for the eight-bit circuit and for 2^N bits in general. In this way, a hierarchical design with logarithmically growing delay can be built.

the *Remaining Level Decoders* can only operate at a 37.2% increased clock frequency, they offer much higher throughput than the other subdecoders due to the highly parallel bc bin decoding.

5.2.4 Design Challenges

The operation of eight subdecoders and the data exchange between them leads to some design challenges, load balancing and multiple clock domains being the most important ones. Effective load balancing is of utmost importance for the overall decoder performance. On a high level, the B3P decoder can be viewed as a pair of four-stage pipelines, one for luma and one for chroma. If only one of the subdecoders is not able to deliver the required throughput, all other subdecoders in the same pipeline will be stalled. Unfortunately, the load for the subdecoders varies significantly for different videos for two major reasons: First, the video quality has a huge impact on the load distribution between the *Control Decoder* and the *Transform Block Decoders*. This has been discussed in Section 5.1 and was considered in the design of the B3P scheme. The problem is addressed by optimizing the bitstream partitioning format as well as the hardware decoder for high bitrates, i.e. allowing the highest throughput for the *Transform Block Decoders*. The *Control Decoder* is the main throughput bottleneck for low bitrates, however, the required throughput is very low in this case and could even be reached by a sequential decoder. So, the CABAC decoding performance is not critical for the overall performance of the video decoder in this case. Second, in a video and even in a single frame there are huge load distribution differences due to the varying complexity of the corresponding frame content. To address this, sufficiently sized *Transform Block Buffers* are needed to avoid the stalling of *Transform Block Decoders* due to the absence of decoding tasks, as well as the stalling of the *Control Decoder* due to full buffers. The *Transform Block Buffers* for luma and chroma both contain 16 entries. Although there are potentially twice as many chroma TBs as luma TBs, the former can most often be processed faster, thus allowing the same buffer size for both. The intermediate buffers between the subdecoders that are part of the *Transform Block Decoders* are used to forward data at a 4×4 block granularity to reduce the latency. Only two entries are used for every intermediate buffer to minimize the hardware cost because the use of larger intermediate buffers did not result in significant performance improvements.

Operating all subdecoders at the same clock frequency would limit the performance of most of them drastically as the lowest frequency has to be used for all. Therefore, four different clock domains are used in the B3P hardware decoder, one for the *Control CC Decoder* and *Control BC Decoder*, one for both *Significance Map Decoders*, one for both *Base Level Decoders*, and one for both *Remaining Level Decoders*. The data exchange between different clock domains is made possible by using SRAM for all intermediate buffers which allow the operation of read and write ports at different clock frequencies. As maintaining four clock domains can add noticeable overhead to the decoder, there is also a way to use only two clock domains without affecting the overall decoder performance too much. Operating the *Significance Map Decoders* and the *Base Level Decoders* at the same clock frequency (699 MHz) reduces the maximum

performance of the latter by only 3.6%. As they need to process fewer bins per 4×4 block than the *Significance Map Decoders*, this performance reduction is not noticeable. Furthermore, the *Remaining Level Decoders* can be operated at the clock frequency of the *Control CC/BC Decoders* (463 MHz) which decreases their potential throughput by 16.7%. However, due to their highly parallel bin decoding, they still deliver sufficient throughput and rarely stall other subdecoders.

5.3 Evaluation

The HEVC reference software [10] has been modified to encode and decode bitstreams according to the proposed B3P scheme. Furthermore, a cycle-accurate architectural model of the corresponding hardware decoder has been implemented as part of the reference software to evaluate the speedup that can be achieved with the parallel decoding. A 32 nm Synopsys standard cell library (saed32lvt_ss0p95v25c) has been used for synthesis to evaluate the clock frequency improvements and hardware cost. Besides the parallel decoder, a functionally equivalent sequential version is used as the baseline for throughput and hardware cost evaluation. To cover a wide range of video sequences, the following JCT-VC test sets are used for evaluation.

- Common test conditions (class A-F) [23]
- Natural content coding conditions for HEVC range extensions (YCbCr 4:2:2, YCbCr 4:4:4, RGB 4:4:4) [24]

They are encoded in AI, RA and LD modes with QPs from 12 up to 37 (common test set only specified for QP 22 to 37). The presented speedup results are the geometric means while the bitstream overhead results are the arithmetic means of all test sequences of a specific combination of video classes and coding modes. The remaining evaluation section covers the decoding speedup, bitstream overhead and hardware cost resulting from the implementation of the proposed B3P scheme. We also provide comparisons to SEP, as well as the HEVC high-level parallelization tools WPP and Tiles. Slices are not considered in the comparison because their main purpose is the resynchronization in the event of data losses. A comparison to the presented low-level optimization approaches is not provided as they exploit parallelism on a different level. Therefore, they can be easily combined with each of the high-level approaches. In fact, the B3P decoder and the sequential baseline both implement the most common low-level parallelization techniques: pipelining and multiple bins per clock cycle.

5.3.1 Parallel Decoding Speedup

Two effects contribute to the significant speedup of the parallel B3P decoder implementation. The first is the parallel decoding of eight bitstream partitions. Second, the

specialization of the subdecoders allows a much higher clock frequency than with the sequential decoder, as discussed in Section 5.2 (see Table 5.2). A sequential hardware decoder version is used as a baseline and can either process one cc bin or up to two bc bins per clock cycle.

The most significant improvements can be reached for AI sequences (see Figure 5.6). They require the highest bitrates as they go without the effective inter-picture prediction. Smaller QPs also raise the speedups as the resulting increased bitrates lead to a more balanced distribution of bins among the different partitions. For very low bitrate sequences, the *Control CC* partition contains most bins and determines the overall decoding throughput. Furthermore, the fraction of bc bins grows with decreasing QPs. This also improves the throughput as they can be decoded in a highly parallel way.

For all high bitrate sequences from the common test set, the *Luma Significance Map* partition is the decoding bottleneck. The maximum speedup for a single sequence from this test set is 6.2×. This is a significant improvement compared to the implementation of Sze, which reached a speedup of up to 2.3× for high bitrates. The sequences from the range extensions test set allow an even better distribution of bins among the partitions due to the reduced chroma subsampling. 4:2:2 subsampling results in the best-balanced partitions, while the decoding of 4:4:4 sequences is dominated by the size of the *Chroma Significance Map* partition. The result is a maximum speedup of 8.5× for a single test sequence, which leads to the highest measured throughput of 3.94 Gbins/s. The upper speedup limit for Tiles grows linearly with the number of tiles, and for WPP with the number of CTU rows respectively. However, this limit can only be reached with perfect load balancing, which is unrealistic due to the varying complexity of the corresponding frame content.

5.3.2 Bitstream Overhead

The ability to decode multiple bitstream partitions in parallel comes at the cost of additional bitstream overhead. First, there is a variable-sized length field for every partition (1-4 bytes) to signal the starting position of the next partition. Additionally, there is an arithmetic coding overhead for each of the five cc partitions (2 bytes). Finally, byte alignment bits are added to all partitions (0-7 bits). Altogether, this adds 11-43 bytes of additional bitstream size per slice. We measured that 23-31 bytes per slice are typically needed for high-quality videos, while the overhead of SEP is approximately 19 bytes.

The Bjøntegaard delta bitrate (BDR) is the average difference in bitrate at the same PSNR (peak signal-to-noise ratio). As the video is not changed due to the modified bitstream format, the relative bitstream overhead is the same as the BDR. It highly depends on the bitrate of the video and can be significant for very low bitrates (see Figure 5.7). This means that AI videos add less relative overhead to the bitstream than RA and LD videos. Also, lower QPs result in a smaller relative overhead. Except for the very low bitrate videos in LD mode or with high QPs, the overhead is less than one percent and therefore negligible. This is especially true for the range extensions test

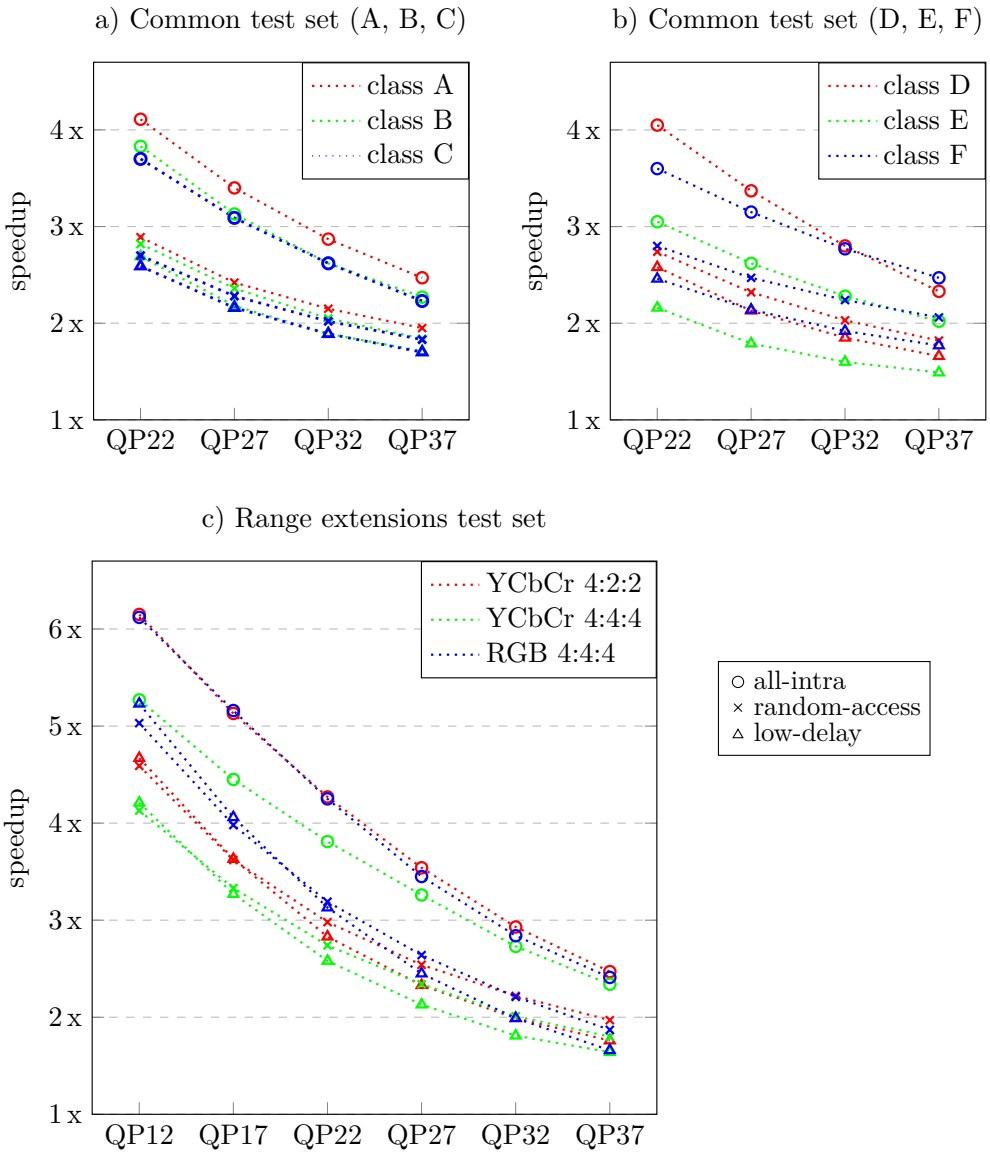


Figure 5.6: Speedup of the B3P decoder compared to the sequential baseline.

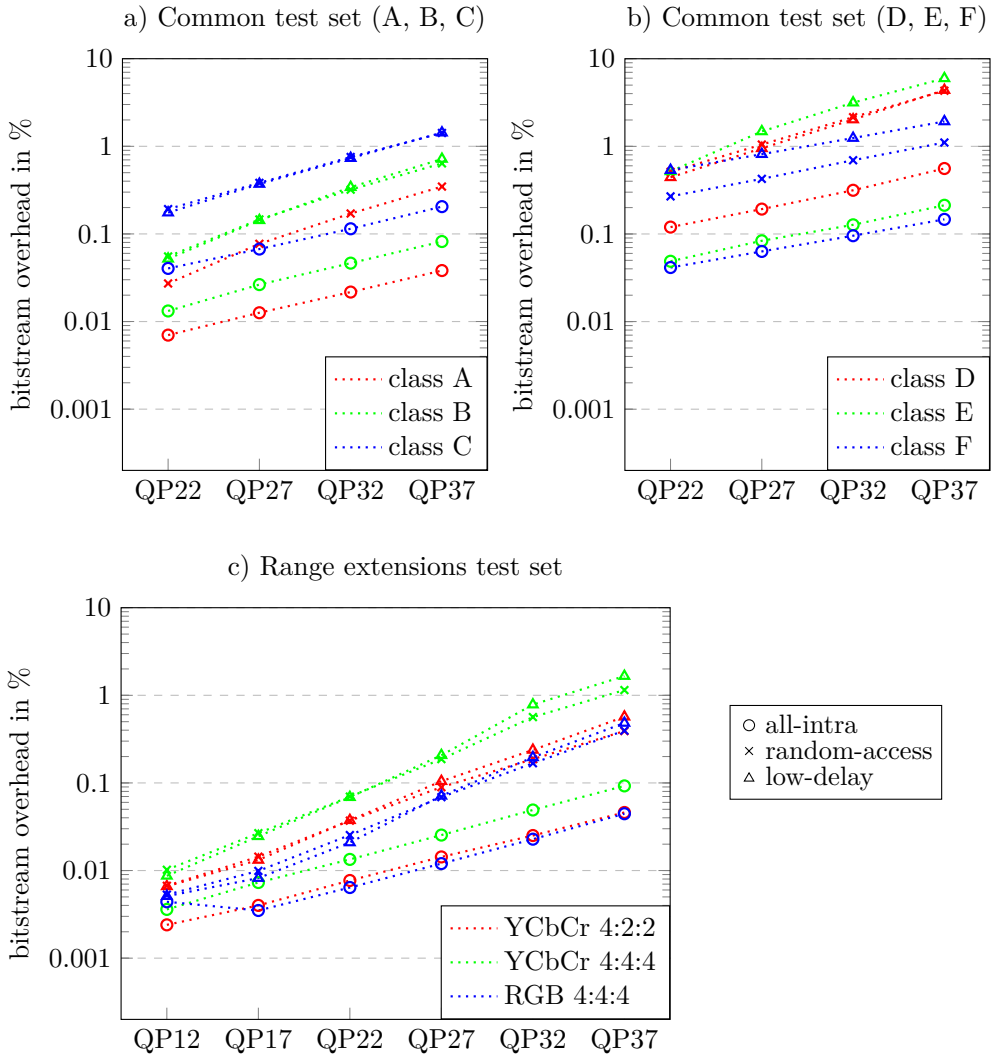


Figure 5.7: Bitstream overhead with the B3P scheme compared to a non-partitioned bitstream.

set. B3P can be disabled for videos where it results in a significant overhead as their throughput requirements are very low. Even a sequential decoder can achieve real-time decoding throughput in these cases. A single bit in the sequence parameter set or the slice header can be used to control the usage of B3P.

There is one abnormal value in the results because a single test sequence (DucksAndLegs) has 60× more overhead than the other sequences from the RGB 4:4:4 class when encoded in AI mode with QP 12. The reason is that there are many zero bytes in one of the partitions. According to the HEVC standard [9], an *emulation_prevention_3_byte* is always added after two consecutive zero bytes. This behavior depends on the video characteristics and cannot be avoided. However, the resulting overhead of the specific sequence is still only 0.024% and therefore negligible.

Figure 5.8 shows a bitstream overhead comparison between the proposed B3P scheme and SEP, as well as WPP and Tiles. We provide results for two common video coding use cases. The first represents video streaming over the internet or to a television device as well as video playback from a DVD or Blu-Ray Disc. Class B from the common test set (FHD resolution) in RA mode is a typical configuration for this use case. The second application is video editing as it can be found in video content production. Much higher video quality is required in this case. Therefore, the RGB-444 class from the range extensions test set is used in AI mode. While the results for B3P, WPP and Tiles are measured by comparing different encodings of the test videos with the HEVC reference software, the bitstream overhead for SEP is calculated based on the description of the author. She states that approximately 50 additional bits are needed for each of the three bitstream partitions. For Tiles, the videos are encoded with two rows and three columns of rectangular tiles because this allows similar speed-ups as with B3P.

B3P has a very similar bitstream overhead as SEP for the video streaming/playback configuration (see Figure 5.8a), even though it uses eight instead of three bitstream partitions. This is due to the variable-sized partition length fields that most often only use one byte for these low to medium bitrate videos. WPP and Tiles need 2× to 4× more additional bits than B3P and SEP. This is a very important advantage for B3P and SEP, especially for low-bitrate videos, where the bitstream overhead adds a substantial amount of data to the original bitstream. SEP performs better than B3P for video production applications (see Figure 5.8b). While SEP has a constant bitstream overhead, B3P suffers from the increasing size of the partition length fields. However, as the maximum measured overhead is only 0.024%, the difference is not important as it contributes only a very small part of the overall bitstream size. The results for QP12 with B3P are affected by the abnormal value for one test video as described above. The use of Tiles leads to a much higher overhead while WPP performs only slightly worse than B3P for QP22 and QP27. It is interesting to note that the use of WPP decreases the bitstream size for some videos with QP12 and QP17. Apparently, the local copying of context models in the two left CTU columns of a frame results in a better CABAC learning process than the row-wise processing with the same context models for the whole frame in some cases. This highly depends on the frame content and can only be noticed because the overhead for the use of WPP is negligible when very high bitrates are used.

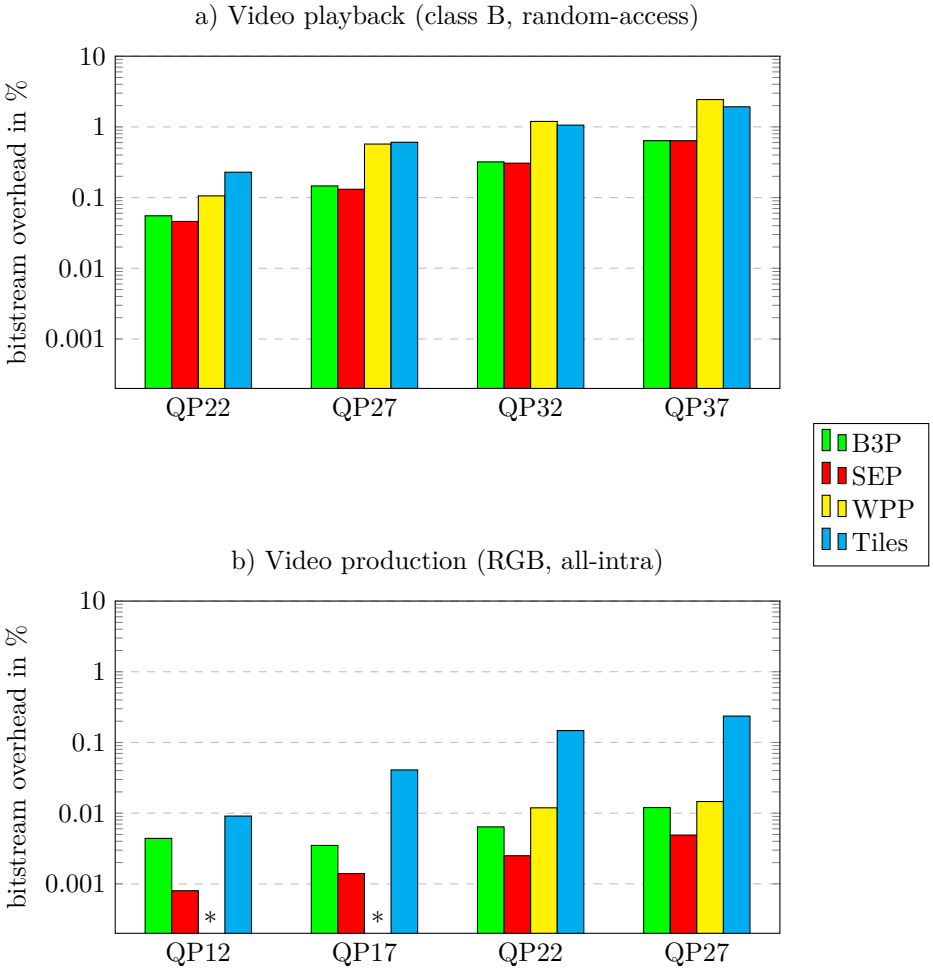


Figure 5.8: Bitstream overhead for B3P, SEP, WPP, and Tiles. (* Negative values cannot be displayed on a logarithmic axis.)

Overall, the bitstream overhead is insignificant for all parallelization approaches for very high bitrates as they can be found in video production applications. On the other hand, the overhead that is added for the implementation of the presented approaches for low to medium bitrates can contribute a substantial amount of data to the bitstream. Thanks to the well-thought-out design, B3P performs as good as SEP in this case, while it significantly outperforms WPP and Tiles. B3P can be combined with Tiles and WPP when ultra-high throughput is required. Therefore, eight partitions are needed per tile or CTU row for WPP, as well as the corresponding bitstream overhead. However, the bitstream overhead will not grow linearly with the number of tiles/CTU rows because the partitions will be smaller, thus requiring shorter partition length fields. Also, the combination of these parallelization techniques becomes necessary only for very high bitrates. In these cases, the bitstream overhead is usually less than 0.01%, so that even a duplication per tile/CTU row is barely noticeable.

5.3.3 Hardware Cost

Eight parallel subdecoders require more hardware resources than a single sequential decoder because some components need to be replicated, e.g. bin decoders and bitstream buffers. Other parts, such as control state machines and context model memories, are distributed among the subdecoders. This does not result in a significant increase in resources. Also, memories for the decoding results are not modified. The B3P decoder uses 61.9% more logic area (SEP uses 70.0% more) and 6.4% more SRAM area than the sequential decoder. The overall extra hardware area is only 9.2% because SRAM consumes more than 90% of the decoder area. The total area of the parallel decoder including SRAM is 1.56mm^2 when implemented with the 32 nm standard cell library.

B3P requires significantly less hardware than the high-level parallelization techniques WPP and Tiles. For those, the speedup scales linearly with the hardware cost in the best case, i.e. with perfect load balancing. Although an encoder can attempt even load distribution, this can rarely be achieved due to the highly varying complexity of different frames and frame areas.

5.4 Conclusions

We have presented a bin-based bitstream partitioning approach for parallel CABAC decoding in next-generation video coding using the widely adopted HEVC/H.265 standard for evaluation. Significant decoding speedups up to 8.5× are achieved by distributing bins over eight fixed bitstream partitions. The specialization of the parallel subdecoders allows their operation at a much higher clock frequency, resulting in a throughput of up to 3.94 Gbins/s. The hardware overhead is 61.9% for the decoder and only 9.2% when the bitstream and result memories are also considered. A comparison to the related SEP, as well as to the HEVC high-level parallelization tools WPP and Tiles is provided in Table 6.4. Our approach outperforms SEP significantly, especially for

Table 5.3: Comparison with SEP (some results for SEP are not available (n.a.)), as well as WPP and Tiles (implemented with six threads as this allows similar speedups as with B3P). The maximum speedup for WPP and Tiles is the theoretical upper limit with perfect load balancing. Bitstream overhead for QP 22 and 37 uses the results from class B in RA mode, while QP 12 shows the results for the RGB-444 class in AI mode.

	B3P	**SEP [69]**	**WPP (6 threads)**	**Tiles (3×2, 6 threads)**
Design	8 fixed partitions HEVC/H.265	3 dynamic partitions H.264/AVC	parallel processing of CTU rows	parallel processing of rectangular areas
Max. Speedup (QP 37/22/12)	4.0× / 6.4× / 8.5×	2.8× / 2.2× / (n.a.)	<6.0× / <6.0× / <6.0×	<6.0× / <6.0× / <6.0×
Bitstream Overhead (QP 37/22/12)	0.64% / 0.06% / <0.01%	0.64% / 0.05% / <0.01%	2.43% / 0.11% / <0.01%	1.92% / 0.23% / <0.01%
Hardware Cost (decoder only/incl. SRAM)	+61.9% / +9.2%	+70.0% / n.a.	+500% / +500%	+500% / +500%

high-quality videos, where CABAC decoding throughput is most critical for the overall decoding performance. At the same time, the bitstream overhead is comparable and the hardware cost is even slightly better. B3P needs significantly less hardware in contrast to WPP and Tiles as they require a full replication of the CABAC decoding hardware to allow parallel processing. Today's mainstream consumer applications benefit from B3P, however, its main target is ultra-high bitrate video coding as it can be found on the content production side as well as in emerging 4K/8K use cases. We strongly recommend considering the B3P approach for adoption in next-generation video coding standards because it addresses the critical CABAC decoding throughput bottleneck for high-resolution and high-quality videos.

The bitstream partitioning approach in this chapter, as well as the fine-grained WPP implementations in the previous chapter, increase the number of decoded bins per clock cycle. An architectural optimization to improve the clock frequency of a CABAC hardware decoder will be presented in the next chapter.

CHAPTER 6

Context Model Cache and Prefetching

The context model memory access is often part of the critical path in the pipeline of CABAC hardware decoders. The replacement of the memory by a smaller cache has been proposed to optimize the critical memory access and thereby improve the throughput [75] [54]. Unfortunately, the effect of potential cache misses has not been properly evaluated. Cache misses result in a performance degradation that might jeopardize the throughput improvements reached by the introduction of the cache. Hong et al. have proposed prefetching to address this issue [55], but also did not quantitatively evaluate their proposal. In this work we evaluate both, caches and prefetching, to justify if these are gainful optimizations. We perform a design space exploration of different cache architectures, thereby making the following contributions for HEVC CABAC hardware decoding:

- an optimized context model cache architecture (as a result of an extensive evaluation of different configurations),
- an efficient context model memory layout optimized for spatial locality and prefetching efficiency, as well as the corresponding adaptive prefetching algorithm,
- an evaluation of prefetching efficiency, real-time decoding capabilities and energy efficiency for different cache configurations.

This work is based on the following publications:

- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *"Optimizing HEVC CABAC Decoding with a Context Model Cache and Application-specific Prefetching"*, Proceedings of the 11th IEEE International Symposium on Multimedia (ISM 2015), pp. 429-434, Miami, FL, USA, December 2015, **Best Student Paper Award**
- **P. Habermann**, C. C. Chi, M. Alvarez-Mesa and B. Juurlink, *Application-Specific Cache and Prefetching for HEVC CABAC Decoding*, IEEE Multimedia, volume 24, issue 1, pp. 72-85, January 2017

The proposed decoder architecture is presented in Section 6.1. It contains a description of the context model cache design, the context model memory layout, as well as the adaptive prefetching algorithm. The evaluation results in terms of clock frequency, cache miss rate, and overall performance improvement are discussed in Section 6.2. We also provide results regarding the real-time decoding capabilities, hardware cost, and energy efficiency of the CABAC decoder. Finally, our work is concluded in Section 6.3.

6.1 Decoder Architecture

The proposed decoder architecture is implemented with a two-stage pipeline (see Figure 6.1). The use of more pipeline stages is avoided to keep the design complexity low. The Context Model Selection stage computes the next state of the decoder control state machine, calculates the index of the required context model and accesses the context model cache. The cache contains context model sets (CMS's). It is clocked with a phase-shifted clock signal to allow the access at the end of the pipeline stage. In the Binary Arithmetic Decoding stage one cc bin or up to two bc bins are decoded by the arithmetic decoder. 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 is performed to build syntax elements from the decoded bins.

6.1.1 Context Model Cache and Memory Layout

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 (WPP, Slices, 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 prefetching unit sends CMS requests to the memory. The CMS's are written back when they have to be replaced. In our implementation, a cache miss results in a 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. The decoder can be configured to use CMS's of four or eight context models.

Tables 6.1 and 6.2 show the optimized context model memory layout for the configuration with eight context models per CMS (CMS8). Mostly, context models for the same type of syntax element are grouped to exploit spatial locality, e.g. *last_sig_coeff_x/y_prefix* (ll. 0-4), *sig_coeff_flags* (ll. 8-10, 12, 13) and *coeff_abs_level_greater1/2_flags* (ll. 16-21). In other cases, context models for consecutively decoded syntax elements are grouped (ll. 5-7, 14). Figure 6.2 shows an example where the decoding flow can go three different ways based on the decoding of two consecutive syntax elements. This can make the prefetching of the required context models very difficult because it has to be

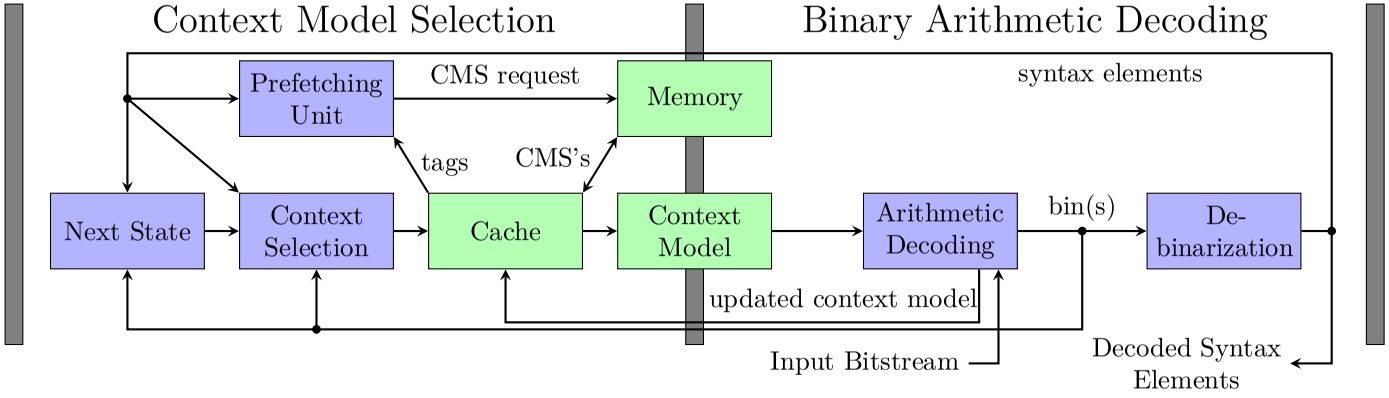


Figure 6.1: Two-stage pipeline of the proposed decoder architecture (CMS: context model set, green modules: synchronous memories, purple modules: combinational logic).

Table 6.1: Context Model Memory Layout (CMS8) - Y: luma, Cb/Cr: chroma, SAO, coding quadtree, CU and intra PU, inter PU, transform tree, last_sig_coeff_x/y_prefix, transform skip flags, significance flags (sig: sig_coeff_flag, csb: coded_sub_block_flag), coefficient level flags (absG1/2: coeff_abs_level_greater1/2_flag).

0	last sig x prefix Y 4x4 0 last sig y prefix Y 4x4 0	last sig x prefix Y 4x4 1 last sig y prefix Y 4x4 1	last sig x prefix Y 4x4 2 last sig y prefix Y 4x4 2	transform_skip_flag Y sig DC Y
1	last sig x prefix Y 8x8 0 last sig y prefix Y 8x8 0	last sig x prefix Y 8x8 1 last sig y prefix Y 8x8 1	last sig x prefix Y 8x8 2 last sig y prefix Y 8x8 2	
2	last sig x prefix Y 16x16 0 last sig y prefix Y 16x16 0	last sig x prefix Y 16x16 1 last sig y prefix Y 16x16 1	last sig x prefix Y 16x16 2 last sig y prefix Y 16x16 2	last sig x prefix Y 16x16 3 last sig y prefix Y 16x16 3
3	last sig x prefix Y 32x32 0 last sig y prefix Y 32x32 0	last sig x prefix Y 32x32 1 last sig y prefix Y 32x32 1	last sig x prefix Y 32x32 2 last sig y prefix Y 32x32 2	last sig x prefix Y 32x32 3 last sig y prefix Y 32x32 3
4	last sig x prefix Cb/Cr 0 last sig y prefix Cb/Cr 0	last sig x prefix Cb/Cr 1 last sig y prefix Cb/Cr 1	last sig x prefix Cb/Cr 2 last sig y prefix Cb/Cr 2	transform_skip_flag Cb/Cr sig DC Cb/Cr
5	cu_transquant_bypass_flag part_mode 0	pred_mode_flag part_mode 1	prev_intra_luma_pred_flag part_mode 2	intra_chroma_pred_mode part_mode 3
6	merge_flag inter_pred_idc 0	merge_idx inter_pred_idc 1	inter_pred_idc 2	inter_pred_idc 4 inter_pred_idc 3
7	abs_mvd_greater0_flag mvp_l0/1_flag	abs_mvd_greater1_flag	ref_idx_l0/1 1	ref_idx_l0/1 2
8	sig Y 8x8 diag top-left 0 sig Y 8x8 hor/ver top-left 0	sig Y 8x8 diag top-left 1 sig Y 8x8 hor/ver top-left 1	sig Y 8x8 diag top-left 2 sig Y 8x8 hor/ver top-left 2	
9	sig Y 8x8 diag other 0 sig Y 8x8 hor/ver other 0	sig Y 8x8 diag other 1 sig Y 8x8 hor/ver other 1	sig Y 8x8 diag other 2 sig Y 8x8 hor/ver other 2	csb Y 0 csb Y 1
10	sig Y 4x4 0 sig Y 4x4 4	sig Y 4x4 1 sig Y 4x4 5	sig Y 4x4 2 sig Y 4x4 6	sig Y 4x4 3 sig Y 4x4 7

Table 6.2: Context Model Memory Layout (CMS8), continued.

11	split_cu_flag 0 cu_skip_flag 0	split_cu_flag 1 cu_skip_flag 1	split_cu_flag 2 cu_skip_flag 2	sao_merge_flag sao_type_idx
12	sig Cb/Cr 4x4 0 sig Cb/Cr 4x4 4	sig Cb/Cr 4x4 1 sig Cb/Cr 4x4 5	sig Cb/Cr 4x4 2 sig Cb/Cr 4x4 6	sig Cb/Cr 4x4 3 sig Cb/Cr 4x4 7
13	sig Cb/Cr 8x8 0 sig Cb/Cr 16x16 0	sig Cb/Cr 8x8 1 sig Cb/Cr 16x16 1	sig Cb/Cr 8x8 2 sig Cb/Cr 16x16 2	csb Cb/Cr 0 csb Cb/Cr 1
14	split_transform_flag 0 cbf_luma 0	split_transform_flag 1 cbf_luma 1	split_transform_flag 2 cbf_chroma 0	rqt_root_cbf cbf_chroma 1
15	cu_qp_delta_abs 1	cu_qp_delta_abs 2		
			cbf_chroma 2	cbf_chroma 3
16	absG1 Y top-left 1.0	absG1 Y top-left 1.1	absG1 Y top-left 1.2	absG1 Y top-left 1.3 absG2 Y top-left 1
17	absG1 Y top-left 2.0 sig Y 16x16/32x32 top-left 0	absG1 Y top-left 2.1 sig Y 16x16/32x32 top-left 1	absG1 Y top-left 2.2 sig Y 16x16/32x32 top-left 2	absG1 Y top-left 2.3 absG2 Y top-left 2
18	absG1 Y other 1.0 last sig x prefix Y 32x32 4	absG1 Y other 1.1 last sig y prefix Y 32x32 4	absG1 Y other 1.2	absG1 Y other 1.3 absG2 Y other 1
19	absG1 Y other 2.0 sig Y 16x16/32x32 other 0	absG1 Y other 2.1 sig Y 16x16/32x32 other 1	absG1 Y other 2.2 sig Y 16x16/32x32 other 2	absG1 Y other 2.3 absG2 Y other 2
20	absG1 Cb/Cr 1.0	absG1 Cb/Cr 1.1	absG1 Cb/Cr 1.2	absG1 Cb/Cr 1.3 absG2 Cb/Cr 1
21	absG1 Cb/Cr 2.0	absG1 Cb/Cr 2.1	absG1 Cb/Cr 2.2	absG1 Cb/Cr 2.3 absG2 Cb/Cr 2

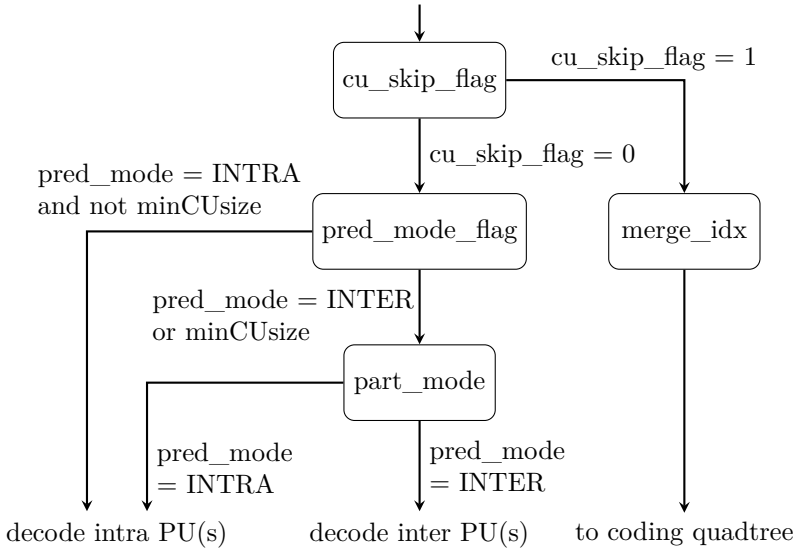


Figure 6.2: CU decoding flow extract in a B/P-Slice.

initiated two clock cycles before the context models are needed. The CMS8 configuration allows removing the dependency from the decoded *pred_mode_flag* as all potential next context models fit in a CMS (l. 5), while there is no conflict with the goal to exploit spatial locality (*part_mode* requires four different context models). The memory layout for four context models per CMS (CMS4) cannot use this technique as often as CMS8. This is one of the advantages of using eight instead of four context models per CMS. Another one is the improved capability to exploit spatial locality, especially for the *sig_coeff_flags* of 4×4 transform blocks (ll. 10, 12) and the consecutively decoded *last_sig_coeff_x/y_prefixes* (ll. 0-4), as the required corresponding context models exceed the CMS4 capacity.

Other CMS sizes are also imaginable. However, at least three *last_sig_coeff_x/y_prefix* and *sig_coeff_flag* context models as well as four *coeff_abs_level_greater1_flag* context models are potentially needed for the decoding of a 4×4 transform sub-block. As these sub-blocks contribute most of the bins in high bitrate bitstreams, the CMS4 configuration already represents the smallest reasonable CMS size. 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. This limits the expected gain of bigger CMS sizes. As other issues such as hardware cost and memory bus width need to be considered, our evaluation is limited to the CMS4 and CMS8 configurations.

6.1.2 Adaptive Prefetching

Application-specific context model prefetching can significantly reduce the cache miss rate in HEVC CABAC decoding. 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 in advance, but a cache hit is still guaranteed.

The prefetching module reads the current state of the control state machine, the video sequence parameters (e.g. slice type, minimum coding unit size), some decoded syntax elements (e.g. prediction mode, PU partition mode) and the currently decoded bin (see Figure 6.3). 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 candidate 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 might be used soon, while for smaller caches CMS's are only fetched when there is a high probability that they will be needed. In general, the strategy for two CLs is more careful to avoid the replacement of CMS's that might still be used. 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.

The design of a general prefetching strategy is very challenging if it has to work well for all kinds of video sequences. Both, encoding parameters as well as characteristics of the video content usually influence the decoding flow and need to be taken into account. While encoding parameters such as the QP can be considered, the prediction of video characteristics is not that obvious. We solved this problem by using context models for decisions in the prefetching strategy. Context models store an estimated probability that the corresponding syntax element will have a specific value. The MPS can be used to predict branches in the decoding flow with high accuracy. As context models are adaptable to video characteristics, the prefetching strategy inherits this capability. The proposed prefetching algorithm keeps track of the MPS's of the context models for the following syntax elements: *cu_skip_flag*, *pred_mode_flag*, *cbf_luma* and *last_sig_coeff_y_prefix* (only for first bin). Figure 6.2 can be used as an example to understand how this prediction works. Before entering the presented decoding flow, either the CMS containing the *pred_mode_flag* context model or the CMS containing the *merge_idx* context model is prefetched. The decision is based on the MPS of the *cu_skip_flag* context model. Basically, the CMS for the more probable branch is fetched. The same prediction is performed for the *pred_mode_flag* in the CMS4 configuration.

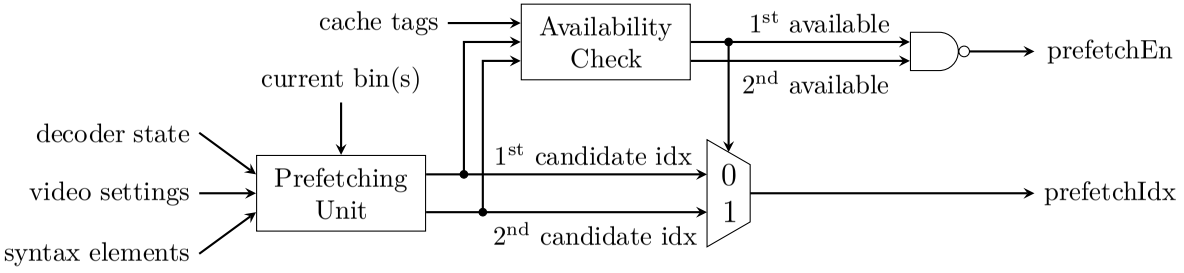


Figure 6.3: Overview of the prefetching candidate selection process.

6.2 Evaluation

A hybrid HEVC decoder has been realized as hardware-software co-design on the Xilinx Zynq-7045 SoC (system on chip) [83] to validate the functionality of the proposed CABAC hardware decoder. The CABAC decoder is implemented in the Zynq Programmable Logic, i.e. a Kintex-7 field-programmable gate array (FPGA), while the remaining decoder components are executed by the Processing System (ARM Cortex-A9 CPU). The highly optimized HEVC software decoder developed by the Embedded Systems Architecture Group at TU Berlin is used [31]. Five test sequences from the JCT-VC class B test set (FHD) serve for evaluation [23]. They are encoded in AI and RA mode with QPs of 17, 22, 27, 32 and 37. The QP of 17 is added to those specified in the JCT-VC Common Test Conditions to cover a wider range and get more meaningful results, especially for very high bitrate videos.

The remaining evaluation section is structured as follows. First, the impact of different cache configurations on the clock frequency is shown to provide an upper boundary for the overall speed-up. Then, the cache miss rate without prefetching is presented to illustrate that a cache does not improve the overall throughput without further measures. Afterwards, it is demonstrated that the miss rate can be significantly reduced when application-specific context model prefetching is used, resulting in different overall gains depending on the cache configuration. A configuration $M \times N$ represents a cache with M cache lines and N context models per cache line. All configurations from 1×4 to 64×4 and 1×8 to 32×8 will be evaluated. 64×8 is not needed as all required context models for the decoding of a CTU fit into 32 CLs due to the increased cache line capacity. Finally, we provide results regarding the real-time capabilities, resource utilization and energy efficiency of the proposed CABAC decoder.

6.2.1 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 number of cache lines and the number of context models per CMS on the maximum clock frequency of the decoder can be seen in Figure 6.4.

First, it should be noted that there is no consistent clock frequency difference between the configuration with four and eight context models per CMS (CMS4/8). The small variations can be explained with the FPGA synthesis technology that does not lead to completely predictable results. The independence of the CMS size is surprising as one would expect bigger CMS's to result in a higher delay during the selection of the required context model from the CMS. However, when the CMS size is doubled from four to eight context models, only half of the CMS's are needed to contain all context models.

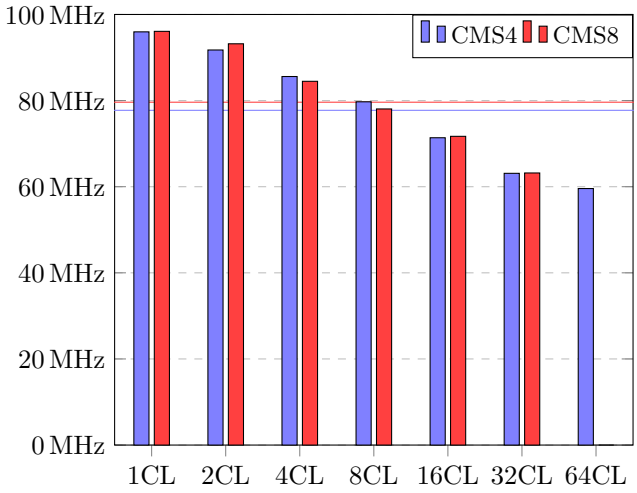


Figure 6.4: Decoder clock frequency for a different number of cache lines (CLs) and context models per CMS (CMS4/8). The horizontal lines show the clock frequencies for the corresponding non-cached designs.

This shortens the CMS address by one bit and thereby simplifies its computation. The simplification of the address generation logic compensates for the bigger selection tree and keeps the overall critical path delay constant.

On the other hand, the number of cache lines affects the achievable clock frequency significantly. For CMS4 (CMS8) it is increased by 23.4% (20.7%) for a single CL, by 18.0% (17.0%) for two CLs and by 10.1% (6.1%) for four CLs. While there is only a marginal variation for eight CLs (2.6% for CMS4, -2.0% for CMS8), 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.

While the improved clock frequency accelerates the decoding of all bins, only cc bins can result in cache misses. The fraction of cc bins in the test sequences varies from 63% to 78%. However, as up to two bc bins can be decoded in parallel, the decoding time fraction for cc bins is slightly increased (73% to 84%).

6.2.2 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. The cache miss rates for different cache sizes and video modes are presented in Figure 6.5 for CMS4 and Figure 6.6 for CMS8 configurations. The results show the arithmetic

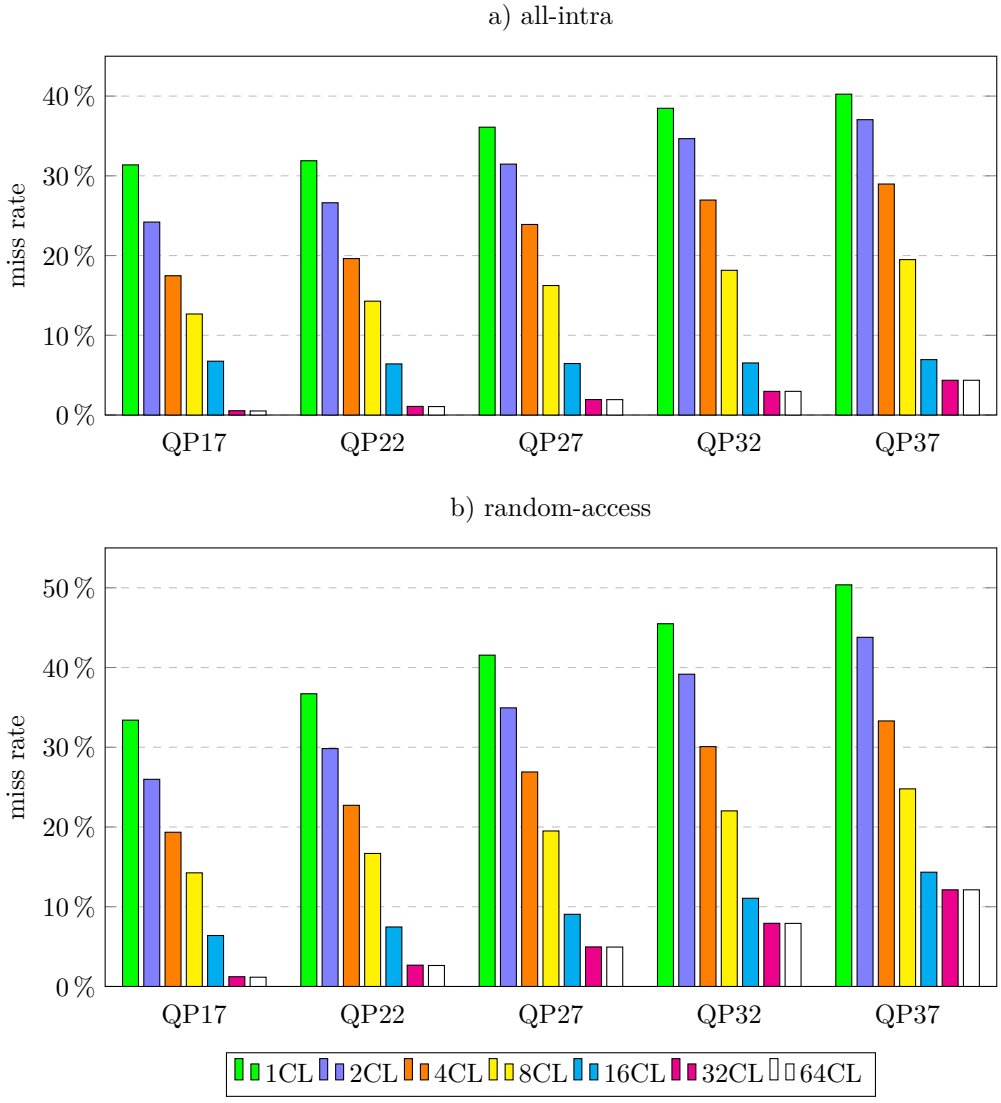


Figure 6.5: Cache miss rate without prefetching and CMS4.

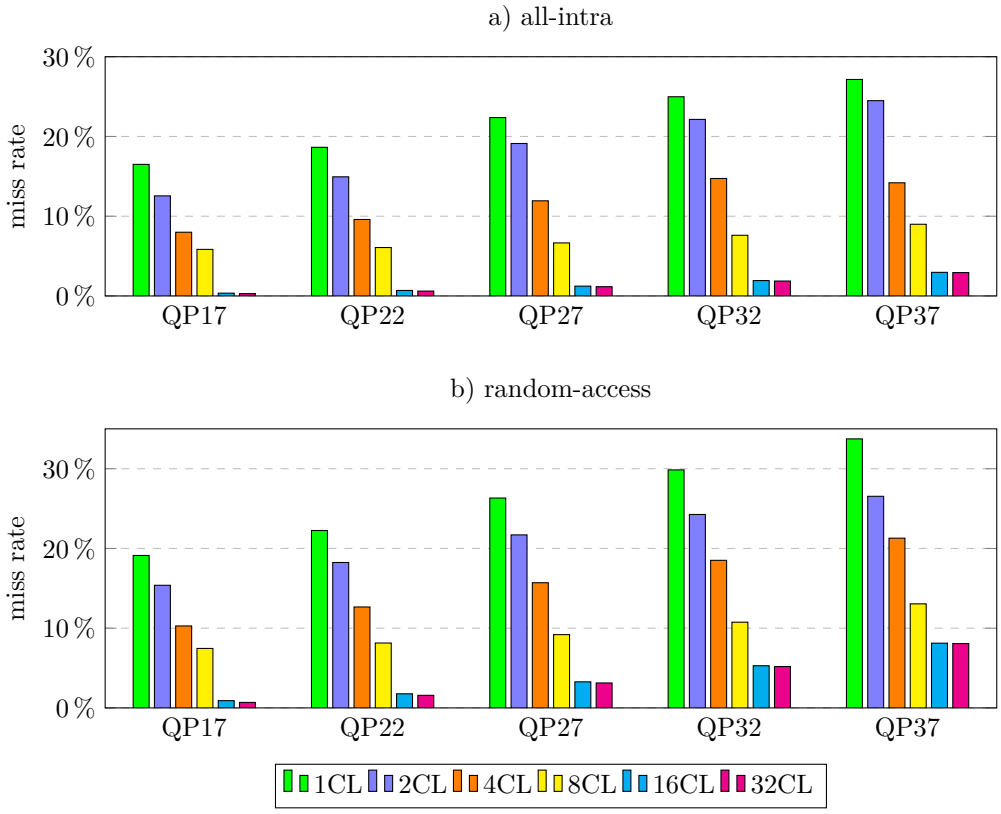


Figure 6.6: Cache miss rate without prefetching and CMS8.

mean of the five different test sequences. 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 64×4 and 32×8 configurations 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. In this case, the fraction of missing accesses decreases with the total number of cc bins, which is higher for smaller QPs. Often not all context models are used during the decoding of a CTU, especially if only a few bins are decoded as in the RA QP37 configuration. In this case, 32×4 and 16×8 are also sufficient and lead to the same results as 64×4 and 32×8.

As there are high miss rates for a few CLs and reduced clock frequencies for more than eight CLs, an overall performance improvement cannot be reached without prefetching. Even in the best case with an AI QP17 video and a 2×8 cache (73% cc bin cycle fraction, 17% higher clock frequency, 11% miss rate) the overall performance is just the same as with a non-cached decoder.

6.2.3 Performance with Prefetching

The proposed prefetching algorithm significantly reduces the cache miss rate. The miss rates with only one CL are separately presented in Table 6.3 as they reach much higher values than with more CLs. A 1×4 cache is still not acceptable as the miss rate is greater than 15% for all videos because of the restricted prefetching opportunities. The 1×8 configuration allows a satisfying overall throughput improvement of 7.0% to 13.1% (see Figure 6.9), but only for high bitrate videos (AI QP17 and QP22, RA QP17). For lowest bitrates, the decoder performance is up to 11.5% less than with the non-cached baseline. Unexpected results can be observed for the 1×4 configuration, where AI QP17 videos lead to a higher miss rate than AI QP22 videos. Usually, the miss rate decreases with a higher bitrate due to the improved exploitation of temporal and spatial locality. However, the fraction of 4×4 transform blocks also increases with higher bitrates (from 64.5% for AI QP22 to 72.8% for AI QP17 on average). Unlike bigger transform blocks they require a different context model for every bin of the luma $last\_sig\_coeff\_x/y\_prefix$ syntax elements. Furthermore, nine instead of four

Table 6.3: Cache miss rate with prefetching and 1CL.

		QP17	QP22	QP27	QP32	QP37
CMS4	**AI**	16.40%	15.85%	16.95%	17.78%	18.32%
	RA	17.31%	18.29%	19.98%	21.55%	23.24%
CMS8	**AI**	4.39%	6.09%	8.75%	11.23%	13.54%
	RA	7.73%	10.84%	14.81%	18.35%	21.90%

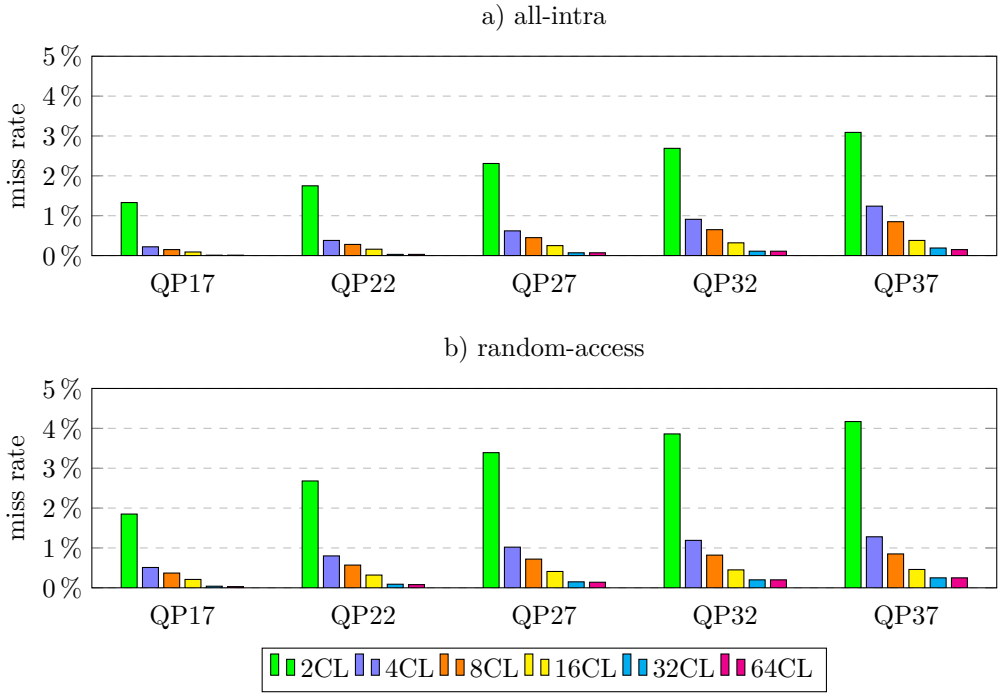


Figure 6.7: Cache miss rate with prefetching and CMS4.

context models are potentially needed for the sig_coeff_flags. These requirements combined with the limited cache size of only four context models lead to an increased miss rate.

The miss rates for more than one CL are presented in Figure 6.7 for CMS4 and Figure 6.8 for CMS8 configurations. A 2×4 cache already leads to significant improvements for all video configurations. The miss rates of 1.3% to 4.2% lead to overall throughput improvements of 15.7% to 10.4% (see Figure 6.9) due to the enhancement of the clock frequency by 18%. With a 4×4 cache and all configurations with more CLs, the miss rate is less than 1.3% for all videos. As a result, the decoder performance is not noticeably affected anymore and almost the full gain of the clock frequency improvement remains. The 4×4 cache results in a consistent throughput gain of 7.8% to 9.7%. More CLs lead to even smaller miss rates. However, a significant speed-up cannot be achieved because of the constant or decreased clock frequency.

The resulting miss rates for all configurations with eight context models per CMS are qualitatively similar to the ones with four context models per CMS but slightly lower. For a 2×8 cache, the miss rates range from 0.2% to 2.0%, leading to an overall performance improvement of 16.7% to 13.2%. More CLs reduce the miss rates to less than 0.9% and make them negligible. The 4×8 configuration is capable of increasing the throughput by 4.8% to 6.0% which is mainly limited by the clock frequency improvement of 6.1%.

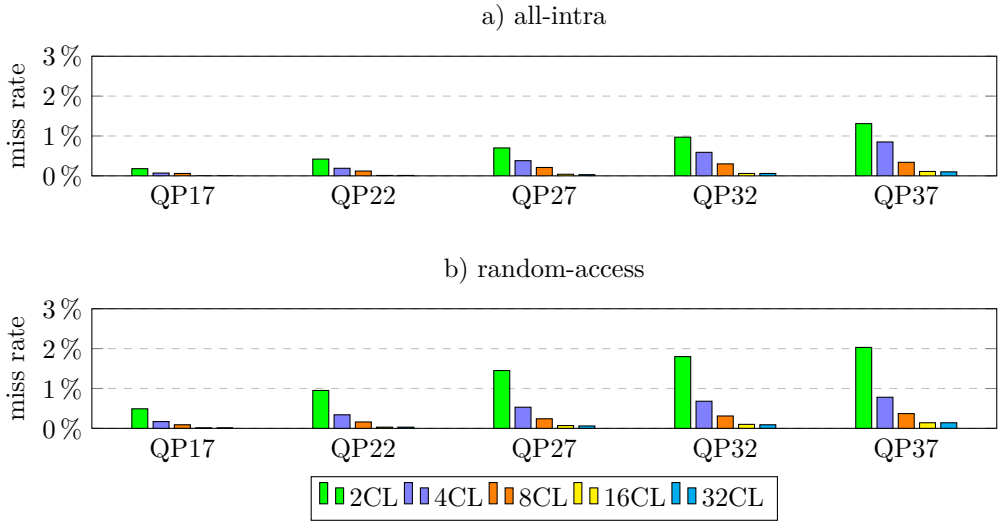


Figure 6.8: Cache miss rate with prefetching and CMS8.

More CLs reduce the decoder performance because the clock frequency falls below that of the non-cached baseline.

Figure 6.9 shows a comparison of the throughput improvement for all cache configurations with one to eight CLs. It can be observed that two CLs lead to highest improvements. Although the 2×4 cache allows for a slightly higher clock frequency improvement of 18% compared to 17% of the 2×8 cache, the latter reaches the greatest overall performance gain because the miss rates are less than half due to the doubled CMS size. For a single CL, the throughput varies heavily as the miss rate strongly depends on the video characteristics. For four and eight CLs on the other hand, the throughput improvements are very similar for all videos. Here, all miss rates are less than 1.3% which allows a throughput gain close to the clock frequency improvement.

It can be concluded that a 2×8 cache allows the highest speed-up for all test videos. Although it contains two times the number of context models as the 2×4 cache, the size is still very small (2 CLs × 8 context models × 7 bits = 112 bits). A comparison to previous cached CABAC hardware decoders is provided in Table 6.4. The comparison of our HEVC CABAC decoder to H.264 CABAC decoders can be performed because the general CABAC algorithm is the same in both standards. However, the throughput might be slightly increased in HEVC CABAC due to some optimizations in the standard that allow the decoding of more bins per clock cycle. Our configuration is very similar to the one of Yang and Guo [54] who used a 2×9 cache for an H.264 CABAC decoder. Hong et al. [55] also implemented a cache with two CLs, but information about the CL size was not provided. The proposal of Yi and Park [75] used only one CL of eight context models which might reduce the overall performance due to the mandatory stall cycles coming with every CL switch. Our decoder reaches similar throughput as the 180nm CMOS decoders but it is outperformed by the 130nm CMOS implementation

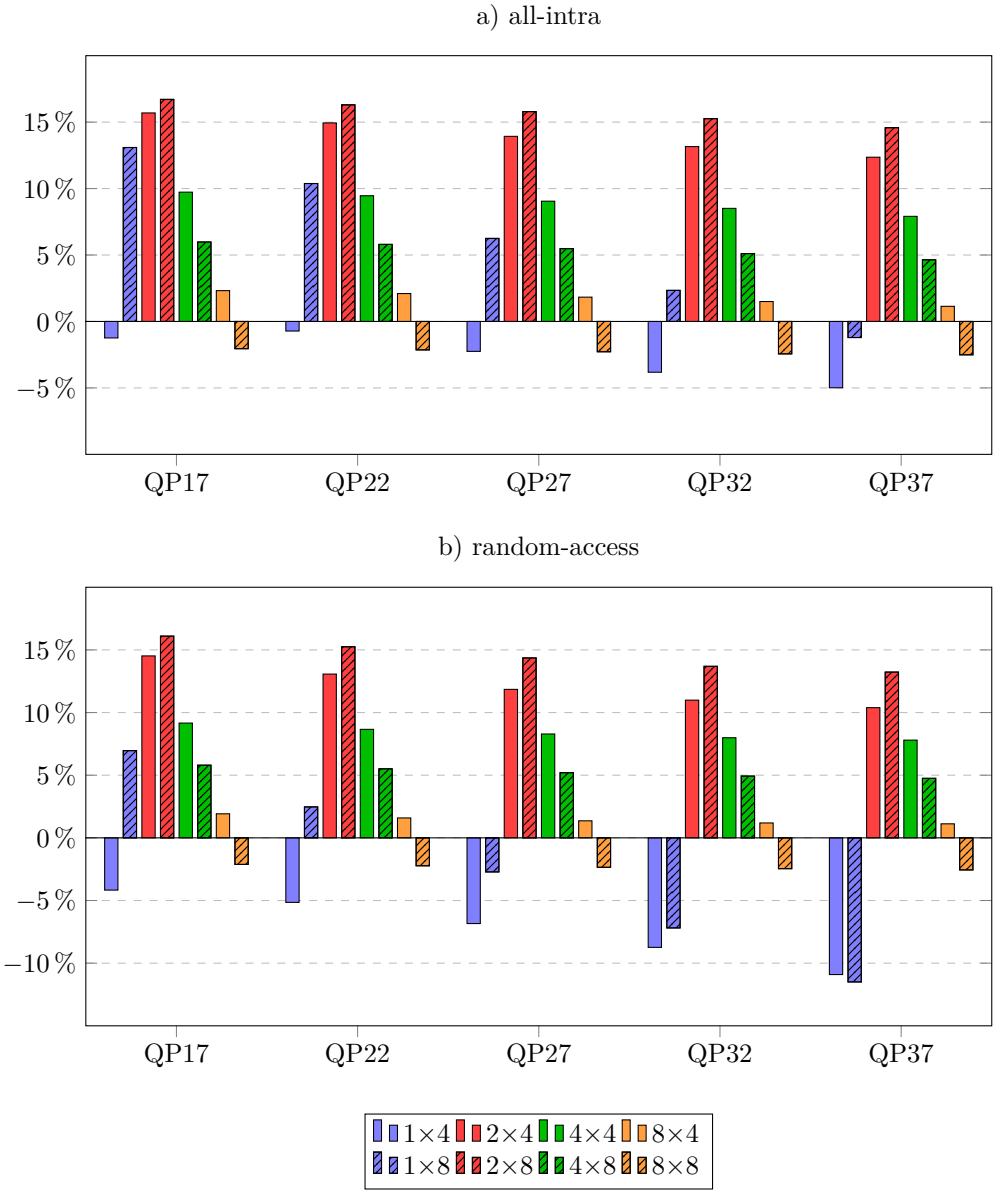


Figure 6.9: Throughput improvement with prefetching for different cache configurations (compared to non-cached decoder).

Table 6.4: Comparison to other hardware decoders.

	This work	Yi [75]	Yang [54]	Hong [55]
Standard	HEVC	H.264	H.264	H.264
Technology	28nm FPGA	180nm CMOS	180nm CMOS	130nm CMOS
Cache	2×8	1×8	2×9	2×?
Prefetching	✓	✗	✗	✓
Frequency	93 MHz	225 MHz	140 MHz	333 MHz
bins/cycle	1.13	0.24	0.86	1.08
Throughput	106 Mbins/s	54 Mbins/s	120 Mbins/s	360 Mbins/s

due to the significantly higher clock frequency. Although our decoder is implemented using an advanced 28 nm FPGA, it cannot reach the competitors' clock frequencies due to the inherent routing logic overhead and the limited customizability of the FPGA technology. A more detailed comparison of the cache architectures cannot be realized as neither the clock frequency increase nor the resulting miss rates are presented in former works.

6.2.4 Real-time Decoding

Table 6.5 shows the real-time decoding capabilities of the proposed HEVC CABAC decoder with a 2×8 cache and prefetching. Table 6.5a presents the average throughput in RA test videos with one intra frame per second. The throughput requirements are fulfilled for all videos with a QP of 22 or higher, but not for all QP17 videos. Nevertheless, a QP of 22 already allows for high perceptive video quality. As a consequence, the proposed HEVC CABAC decoder enables real-time decoding of high-quality FHD videos with a low-power platform such as the Zynq-7045.

Unfortunately, there are significant variations in the required bitrate for different frames of the same video. Especially intra frames often require much higher bitrates as they cannot remove temporally redundant information in consecutive frames using inter-picture prediction. The presentation of the throughput requirements for videos that are encoded only with intra frames (see Table 6.5b) provides an indication of the peak throughput that is needed to decode single frames in real-time. However, as the varying throughput requirements also depend on the content of a frame, the real peak throughput will be higher. The requirements are met for all videos with a QP of 27 or higher and some QP22 videos. This means that the proposed CABAC decoder cannot fulfill the real-time requirements for every frame separately. However, if the decoding latency of a few frames and the additional hardware for frame buffers are acceptable, judder-free streaming can be realized.

Table 6.5: Average throughput in Mbins/s for RA (a) and AI (b) video modes. Each cell shows the achieved throughput with a 2×8 cache and prefetching (top) as well as the required throughput for real-time decoding (bottom). The green cell color indicates that real-time throughput requirements are met.

a) RA average	QP17	QP22	QP27	QP32	QP37
BasketballDrive	97.7	95.6	92.9	90.2	87.6
(1080p, 50 fps)	88.6	22.1	7.5	3.5	1.9
BQTerrace	98.5	94.2	91.0	88.1	84.9
(1080p, 60 fps)	175.7	57.2	10.2	3.1	1.4
Cactus	97.1	94.2	93.0	90.0	86.9
(1080p, 50 fps)	101.8	23.5	6.9	3.3	1.7
Kimono	98.1	98.3	96.1	93.2	89.6
(1080p, 24 fps)	17.2	5.7	2.6	1.3	0.7
ParkScene	96.3	94.1	91.9	89.1	86.1
(1080p, 24 fps)	24.5	9.2	4.0	1.9	0.9

b) AI average	QP17	QP22	QP27	QP32	QP37
BasketballDrive	101.4	99.0	95.8	92.4	89.4
(1080p, 50 fps)	219.9	89.4	36.7	20.1	11.7
BQTerrace	105.6	102.6	99.6	95.0	91.1
(1080p, 60 fps)	331.8	224.9	95.8	49.6	28.0
Cactus	103.9	101.0	98.6	94.8	91.3
(1080p, 50 fps)	285.0	127.2	58.2	32.2	17.9
Kimono	103.1	105.1	102.6	99.2	95.7
(1080p, 24 fps)	66.0	26.1	14.4	8.3	4.8
ParkScene	105.3	102.9	99.2	95.3	91.9
(1080p, 24 fps)	117.3	62.3	34.3	18.3	9.3

Table 6.6: Resource utilization on the Xilinx Zynq-7045 SoC. Each cell shows the absolute (top) and relative (bottom) device utilization. (LUT: lookup table, BRAM: block RAM, DSP: digital signal processor)

Design	Registers	LUTs	BRAMs	DSPs
base CMS4	3,008	7,562	15	1
(area opt)	0.23%	1.15%	2.75%	0.11%
2×4 cache	3,193	8,157	15	1
(area opt)	0.24%	1.24%	2.75%	0.11%
2×4 cache	3,239	9,046	15	1
(speed opt)	0.25%	1.38%	2.75%	0.11%
base CMS8	3,061	7,628	15	1
(area opt)	0.23%	1.16%	2.75%	0.11%
2×8 cache	3,352	8,200	15	1
(area opt)	0.26%	1.25%	2.75%	0.11%
2×8 cache	3,402	9,152	15	1
(speed opt)	0.26%	1.40%	2.75%	0.11%
available	1,311,600	655,800	545	900

6.2.5 Resource Utilization

Table 6.6 compares the resource utilization of the non-cached CABAC decoder with four and eight context models per CMS with the corresponding cached designs with two 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. Three main conclusions can be drawn from the results. First, the cached designs (2×4 and 2×8) require only 6.2% and 9.5% more registers, as well as 7.9% and 7.5% more lookup tables (LUTs), compared to the non-cached designs. This is a reasonable trade-off, considering that the decoder throughput is increased by 10.4% up to 16.7%. Second, The decoder with a 2×8 cache utilizes 5.0% more registers and 0.5% more LUTs than the decoder with a 2×4 cache, while leading to 1.8 more percentage points in average throughput improvement. However, as the register utilization is only a quarter of a percent, it is not critical and the design with the higher throughput can be preferred. Finally, less than 3% of the FPGA resources are needed to implement the CABAC decoder including the processor interface. This means that more decoder components can be implemented in the FPGA as most of the resources are not used yet.

The number of block memories (BRAM) is constant for all configurations. It should be noted that only one of them is used as the context model memory for the actual decoder. The remaining BRAMs are required to store all decoded syntax elements. A single digital signal processing (DSP) unit is used in all configurations to perform the inverse quantization of decoded transform coefficients.

The comparison to related work is difficult as all previous cached CABAC hardware decoders were implemented in 130nm or 180nm CMOS technology while the proposed decoder is realized with an FPGA. Also, the proposed decoder covers HEVC while the others address the prior H.264. Another FPGA-based CABAC decoder was presented by Hahlbeck and Stabernack [68], however, we excluded it from the former comparison as it does not contain a context model cache. This particular decoder reaches 0.76 bins per cycle and a clock frequency of 180 MHz, resulting in slightly higher throughput than with our decoder. On the other hand, it uses significantly more hardware resources. Nevertheless, a detailed comparison of the resource utilization is not meaningful because the CABAC decoder has been designed to be part of a full FPGA-based HEVC decoder [43]. Therefore, it utilizes a substantial amount of additional resources that favor the throughput of the overall decoder.

6.2.6 Energy Efficiency

The power consumption of the HEVC CABAC decoder is measured with the Xilinx Vivado Power Analysis tool. A constant static power of 241 mW is consumed by all designs. The static power is only used to keep the state of the configurable FPGA logic. That is why it is not considered in the evaluation. The dynamic power is of more interest as it varies for the different designs. The best performing design with a 2×8 cache consumes 88.3 mW.

Varying cache sizes affect the power consumption in different ways. On the one hand, a bigger cache requires more hardware resources which consume more power. On the other hand, the miss rate reduction that comes with bigger caches leads to fewer memory accesses which saves power. Furthermore, it has been shown that the achievable clock frequency decreases significantly for bigger caches (see Figure 6.4), and so does the power consumption of the corresponding decoder. These opposing effects result in a power consumption that is almost constant for different cache sizes, as well as for the non-cached decoder. Considering also the average throughput for various test videos leads to the energy efficiency as shown in Figure 6.10. A peak can be observed for both configurations (CMS4 and CMS8) with 4 CLs around 1.1 Gbins/Joule. These designs are 8.5% and 12.0% more energy efficient than the non-cached baselines. Although the throughput is higher with 2 CLs, this is achieved with a higher clock frequency which degrades energy efficiency. More CLs also result in reduced energy efficiency because the throughput is significantly lower.

6.3 Conclusions

A quantitative analysis of the effects of an application-specific context model cache with prefetching in an HEVC CABAC hardware decoder has been conducted in this work. We focused on the evaluation of the miss rate when the context model memory is replaced by a smaller cache. While this replacement allows significant clock frequency improvements

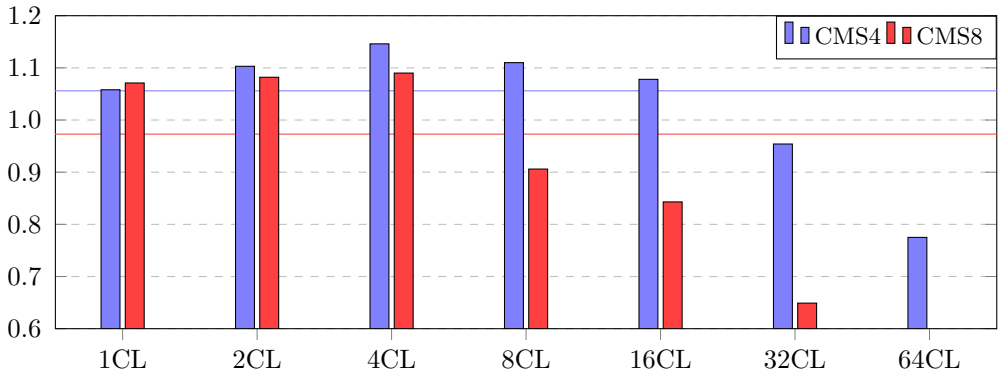


Figure 6.10: Energy efficiency in Gbins/Joule for a different number of cache lines (CLs) and context models per CMS (CMS4/8). The horizontal lines show the energy efficiency for the corresponding non-cached designs.

for small cache sizes, the resulting 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.

Results have shown that two cache lines are sufficient to reduce the miss rate to the point where it only marginally affects the overall performance and almost the full gain of the clock frequency improvement remains. The decoder configurations with two cache lines are most promising as they allow the highest speed-ups for all test videos compared to a non-cached decoder. The 2×4 cache reduces the miss rate to a fraction of 1.3% up to 4.2% which leads to a speed-up of 15.7% to 10.4% thanks to the 18% clock frequency improvement. Although the clock frequency improvement of the 2×8 cache is slightly smaller (17%), the overall throughput gain is in the range of 16.7% to 13.2% due to the miss rates of 0.2% to 2.0%. Four cache lines allow for even lower miss rates but the overall speed-up is limited due to the clock frequency improvement of only 10.1% for a 4×4 cache and 6.1% for a 4×8 cache. A single cache line results in miss rates of more than 20% and leads to significant throughput degradations for low bitrate videos.

Using the best performing configuration (a 2×8 cache), it has been shown that real-time decoding of high-quality FHD videos is possible on a low-power platform such as the Zynq-7045. This cached decoder is on average 12% more energy efficient than the non-cached decoder. 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.

The optimizations presented in the current and the preceding chapter are mainly targeting CABAC hardware decoders. In the next chapter, we will present an optimization approach for CABAC software decoding on GPPs.

CHAPTER 7

CABAC Decoding on GPPs

CABAC is the main throughput bottleneck in high-quality video decoding on GPPs in HEVC as demonstrated in Section 2.2. Although transform coefficient coding has been improved in HEVC [64][65], most of the bins in CABAC are associated with it, e.g. at least 70% for high-quality videos in a common FHD testset, and even more than 98% for single videos in extreme cases. Bc bins for transform coefficients can contribute up to 50% of the overall bins and are mainly responsible for the high bitrates for high-quality videos due to their ineffective compression. In this chapter we propose three techniques for improved bc bin decoding for transform coefficients:

- a bypass-bin reservoir for parallel bin decoding
- a division-free version of the bypass-bin reservoir
- a bitstream partitioning approach for computationless decoding

This chapter is structured as follows. The HEVC bc bin decoding implementation is described in Section 7.1. The proposed approaches for parallel bc bin decoding are presented in Section 7.2. Afterwards, Section 7.3 provides the corresponding performance evaluation before the work is concluded in Section 7.4.

7.1 HEVC BC Bin Decoding on GPPs

The foundation of the HEVC CABAC implementation using integer arithmetic has already been described in Section 2.3.4. The following pseudocode description of the CABAC bc bin decoding functionality is based on the HEVC reference software [10]. We employ C++ syntax with small variations for better readability. Irrelevant parts are removed.

The implementation of CABAC in the HEVC decoder is based on three variables that represent the bitstream state (see Listing 1). *range* and *offset* are used as described earlier (Section 2.3.4). The *bitsNeeded* variable states how many bits are needed in the

Listing 1 Bitstream state variables for HEVC CABAC decoding.

```
1  uint32_t offset;
2  uint16_t range;
3  int8_t bitsNeeded;
```

Listing 2 Common bc bin decoding function.

```
1  uint8_t decodeBcBin() {
2    offset = offset << 1;
3    if(++bitsNeeded >= 0) {
4      bitsNeeded = -8;
5      refill();
6    }
7
8    uint scaledRange = range << 7;
9    if(offset >= scaledRange) {
10     offset -= scaledRange;
11     return 1;
12   } else {
13     return 0;
14   }
15 }
```

offset buffer. The smallest possible value is -8, indicating that the buffer is full. As soon as the value becomes positive (0 or higher) the offset buffer has to be refilled.

Based on that, a function for decoding a single bc bin can be written as seen in Listing 2. Instead of dividing the range in the middle, the offset is doubled to achieve the same comparison result (l. 2). A conditional refill of the offset buffer is performed afterwards (ll. 3-6). The range is shifted to align it with the offset and ignore the offset buffer (l. 8) before the comparison to the offset is performed (l. 9). The offset is then updated in case of a one-bin (l. 10) before the bin value is returned (ll. 11 or 13).

The bin decoding function is used to implement functions for bin sequences, i.e. fixed-length and leadings-ones sequences (see Listing 3). The latter (*getNumLeadingOnes*) is realized by decoding single bc bins until a zero-bit is found. The number of preceding one-bits is returned as the result. There is also a version of this function that limits the sequence to a maximum value, however, its implementation is not discussed for brevity. The fixed-length sequence (*decodeBcBins*) is simply implemented by decoding all the required bins and concatenating them accordingly (ll. 14-15).

Although the bc bin decoding is based on very simple integer operations, its execution by modern GPPs is inefficient. There are two main reasons for this. First, the close-distance data and control dependencies limit the exploitation of instruction-level parallelism in today's superscalar out-of-order processors. Second, the high frequency of hardly predictable branches leads to frequent pipeline flushes and consequently lower

Listing 3 Functions for decoding multiple bc bins.

```
1  uint8_t getNumLeadingOnes() {
2    uint8_t leadingOnes = 0;
3    do {
4      uint8_t bin = decodeBcBin();
5      leadingOnes += bin;
6    } while(bin == 1);
7
8    return leadingOnes;
9  }
10
11 uint32_t decodeBcBins(uint numBins) {
12   uint32_t bins = 0;
13   for(int n = 0; n < numBins; n++) {
14     bins = bins << 1;
15     bins += decodeBcBin();
16   }
17
18   return bins;
19 }
```

throughput. The offset comparison (Listing 2, l. 9) is not predictable as bc bins have equal probabilities for both possible bin values by definition. However, it should be noted that there can be a slight shift in either direction due to video characteristics. A branch-free implementation of the actual bin decoding (ll. 8-14) is also possible, however, it results in sequential dependent code and more executed instructions on average.

A *cabac_bypass_alignment_enabled_flag* in the HEVC sequence parameter set can be used for alignment and consequently more efficient encoding and decoding of transform coefficient bc bins, however, it is deactivated in all except high-throughput intra profiles ([9], Annex A).

Due to these inefficiencies, CABAC decoding throughput is limited on GPPs. In the next section, we present a more efficient version that removes the majority of hardly predictable branches and decodes many bc bins in parallel.

7.2 Parallel BC Bin Decoding

The bin decoding function in Listing 2 shows a left-shift of the offset and a subtraction of the shifted range in case the offset is larger. Doing this repeatedly leads to the same result as the division *offset* / *range*, where the quotient represents the decoded bin sequence. Multiplying the decoded bins by the range and subtracting the product from the offset results in the updated offset. Considering that an integer division is a long-running operation, it is questionable whether this can improve the decoding

throughput. According to the Intel Architectures Optimization Reference Manual, a 32-bit integer division executes in 20-26 clock cycles on Skylake processors [84]. Other instructions have to wait during that time as they need the result of the division. In our experiments, we found that this parallel decoding by division resulted in a speed-up when a sequence of five or more bins was decoded. These sequences can only be frequently found in high-quality videos and rarely in others [46], making the technique in this form obsolete for general HEVC decoding. In the following, we will show how it can be implemented in a way such that it allows substantial speedups for all quality levels. We will further present a division-free implementation and a modification of the HEVC bitstream format that removes the need to decode bc bins at all.

7.2.1 Bypass-Bin Reservoir

HEVC CABAC has been designed for improved throughput [44] compared to its predecessor in H.264. Besides an overall higher fraction of bc bins and other improvements, bc bin grouping has been introduced. This mainly affects the bins for transform coefficients. While the significance flags and base levels for the coefficients use cc bins, the sign bits and remaining coefficient levels are solely composed of bc bins. Instead of interleaving cc and bc bins by grouping bins that are associated with the same coefficients, HEVC CABAC codes all significance flags and base levels in a 4x4 transform coefficient block first. Afterwards, the sign bits and remaining coefficient levels are grouped to create a large sequence of bc bins. The combination of the large group of bc bins with the division-based parallel bc decoding algorithm allows the development of our efficient bc bin decoding technique: the bypass-bin reservoir (BBR). Instead of decoding as many bc bins as needed, we decode as many as possible and store them in the BBR. The size of the offset buffer is increased to 23 bits, i.e. the maximum number that can be realized with a 32-bit unsigned integer offset, to reduce the frequency of divisions. Thereby, one division instruction can decode up to 23 bins in parallel. The bin decoding functions for fixed-length and leading-ones sequences extract the already-decoded bins from the BBR.

The BBR state is represented by three variables (see Listing 4). *initBbr* always represents the BBR after the division as this is needed to clear the BBR after decoding and put unused bits back to the offset buffer. *bbr* is updated when bins are extracted. It always contains the next bins in the leftmost bits. *numBbrBins* always contains the number of bins in the BBR. The function for filling the BBR is shown in Listing 5. The bin decoding is performed by the division (l. 2). Invalid bins are masked out in case of a non-full offset buffer ($bitsNeeded > -24$) (l. 3). Also, the number of bins in the BBR is set according to the offset buffer (l. 4). Finally, the decoded bins are shifted to the leftmost position in the BBR for fast extraction (l. 5). The BBR has to be cleared after decoding the bc bin group. This is done by masking the actually decoded bins in *initBbr*, multiplying them by *range*, and subtracting the product from *offset*.

The implementation of a decoding function for fixed-length bin sequences is straightforward. The required number of bins is taken from *bbr* and it is updated together with *numBbrBins* accordingly. The function for leading-ones sequences can make use of a

Listing 4 BBR state variables for HEVC CABAC decoding.

```
1 uint32_t initBbr;
2 uint32_t bbr;
3 int8_t numBbrBins;
```

Listing 5 Fill BBR function.

```
1 void fill_bbr() {
2   initBbr = offset / range;
3   initBbr &= ~((1 << (24 + bitsNeeded)) - 1);
4   numBbrBins = -bitsNeeded - 1;
5   bbr = initBbr << 9;
6 }
```

special instruction for counting the number of leading zeros (clz) on the inverted *bbr*. This instruction is available in many modern processors and can be used with a gcc builtin function (see Listing 6, l. 2). The *bbr* and *numBbrBins* need to be updated by one additional zero-bin that finishes the sequence of one-bins (ll. 3-4).

So far, we have ignored that there might be hundreds of consecutive bc bins but at most 23 bins available in the BBR. Therefore, a refill function for the BBR is necessary. We decided to check for the necessity to refill during the decoding of a bin sequence. If the number of bins in the BBR is not sufficient for the current bin sequence (fixed-length or leading-ones), all available bins are extracted first. Then, the BBR is refilled and the remaining bins are extracted. This has two advantages over refilling at other points in time. First, a new division operation for refilling is only executed when it is really needed. Second, the refill always works on an empty BBR which simplifies the function a lot.

The proposed BBR implementation produces mostly branch-free code. The only exception is for the refill check. If refills are needed frequently, it means that a lot of bins are decoded in parallel which is the main purpose of the BBR.

7.2.2 Division by Inverse Multiplication

The main complexity of the BBR implementation is in the fill and refill functions due to the long-running integer division. The BBR implementation works best when a fast division is available in a processor. However, the decoding process might be even decelerated in processors with slow division operations. Some processors do not even have a hardware divider, e.g. the ARM Cortex-A9. In this case, the compiler has to generate a code sequence that leads to the same result, but that most often executes in many more clock cycles. In general, a division can be replaced by a multiplication with the inverse. While this works for floating-point numbers, the inverse of any integer greater

Listing 6 Extract leading-ones from BBR.

```
1  uint8_t getNumLeadingOnes() {
2    uint8_t nlo = __builtin_clz(~bbr);
3    bbr = bbr << (nlo + 1);
4    numBbrBins -= (nlo + 1);
5    return nlo;
6  }
```

Listing 7 Division by inverse multiplication. (inv: inverse, add: add indicator, shamt: shift amount)

```
1  uint64_t tmp = (uint64_t)offset*inv;
2  initBbr = (uint32_t)(tmp >> 32);
3  initBbr += add*offset;
4  initBbr = initBbr >> shamt;
```

than one is less than one and therefore not accurately representable with integers. A common way to still implement it is to perform a 64-bit multiplication with the inverse multiplied by 2^{32}, which provides the quotient in the top 32 bits of the result (see Listing 7, ll. 1-2). Unfortunately, this is not accurate and produces incorrect results in some cases, which is acceptable in approximate algorithms but not in CABAC. The leftmost bits of the inverse are zero for larger divisors. The accuracy can be improved by shifting the inverse to the left, thereby enabling additional bits on the right for improved accuracy, and performing the inverse shift after the multiplication (l. 4). For a fully accurate division, an add indicator can be used to get the desired results. It depends on the divisor and is either zero or one (l. 3).

The inverse multiplication is commonly used by compilers when the divisor is known at compile-time. This is not the case for the *range* variable. However, as its value can only be in the interval [256:510] in HEVC CABAC, we can use a lookup table (LUT) that contains the inverse, the shift amount and the add indicator to implement the division with the algorithm described above. The LUT needs $255 \cdot (4 + 1 + 1) = 1530$ bytes in memory. It can also be compressed to 1275 bytes by using the same byte for the shift amount (at most five bits) and the add indicator (one bit). The LUT needs to be accessed only once per bc bin group for the BBR fill. The LUT entry can then be locally stored and used later for every refill because *range* is constant during that time. We refer to the division-free BBR implementation with inverse multiplication by BBR-IM for the remaining chapter.

7.2.3 Bitstream Partitioning

While BBR and BBR-IM can be used for common HEVC bitstreams, we also propose a bitstream partitioning approach with a separate bc bin partition (BBP) for the use

in future video coding standards that can accelerate CABAC decoding even more. Due to the equal probability for both values of bc bins, a bc bin corresponds to a bit in the bitstream. It is due to the interleaving with cc bins, that the bc bin encoding and decoding require computational work in the first place. When using separate bitstream partitions for cc and bc bins, the latter can simply be stored in memory when encoding and be read from memory when decoding. This computationless processing of bins is highly efficient and should be considered in future standards. After all, the large fraction of bc bins in high-quality videos allows the high potential for performance improvements when CABAC decoding is the main throughput bottleneck.

The bitstream overhead constitutes of one additional length field for the bc bin partition. A dynamic field of 1-4 bytes can be used with a leading-ones sequence of at most three bits to determine the size of the field and the remaining bits to determine the size of the partition. The size of the length field is one byte if the first bit is 0_2, two bytes if the first two bits are 10_2, three bytes for 110_2, and four bytes for 111_2. Bitstream partitioning approaches have been proposed in previous works, but in schemes with three [63][70] and eight [4][6] bitstream partitions. This proposal only makes use of the highly efficient bc/cc bin splitting while keeping the cost very low. Compared to BBR and BBR-IM, a separate bc partition improves the decoding of all bc bins, not only the ones associated with transform coefficients. Furthermore, it enables two threads to work on the decoding in parallel.

7.3 Evaluation

We evaluate the performance of the proposed BBR, BBR-IM and BBP approaches for HEVC CABAC bc bin decoding of transform coefficients. Two processors are used for evaluation: the Intel Core i9-7960X and the ARM Cortex-A9. While the former is a modern high-performance processor, the latter is much older and less powerful. The reason for including it in the evaluation is its lack of hardware division support which makes it well-suited to evaluate the division-free BBR-IM approach compared to BBR. We use the single-threaded HEVC reference software (version 16.20 [10]) for evaluation. Although it is known for low performance, e.g. due to the lack of vectorization, the bc bin transform coefficient decoding part is implemented efficiently, thus making it suitable for the evaluation of the proposed techniques. C++ *std::chrono::high_resolution_clock* is used for measurement of the time taken to process the bc bin part in transform coefficient decoding, including BBR filling, refilling and clearing.

We employ Class B of the JCT-VC common test conditions [23] and the RGB-444 class from the range extensions test conditions [24]. The former represents a widely used configuration of FHD videos in RA mode, while the latter is characterized by very high-bitrates due to higher video quality, the absence of chroma subsampling and the use of AI mode. Class B is used with QPs of 22, 27, 32 and 37. The RGB testset even uses very high-quality QPs of 12 and 17 but skips 32 and 37. The results are the geometric means of all videos from the respective testsets.

The speedups of transform coefficient bc bin decoding with the proposed techniques are shown in Figures 7.1 (Intel Core i9-7960X) and 7.2 (ARM Cortex-A9). The BBR and BBR-IM techniques provide very similar speedups on the Core i9-7960X processor in the range between 1.36× and 1.53× for the Class B testset and between 1.55× and 1.84× for the RGB-444 testset. The RGB-444 testset allows higher speedups because there are many more bc bins in these high-bitrate videos which makes the parallel decoding more effective. BBR is the preferred technique for the Core i9-7960X processor as it yields very similar performance results but does not require the 1.53 kB LUT for the data associated with the inverse multiplication, which can occupy a relevant part of the 32 kiB first-level data cache. The BBP technique avoids the fill, refill and clear operations for the BBR and consequently results in 1.90× up to 2.22× performance compared to common bc bin decoding. The maximum speedups for single videos are 2.00×, 2.04× and 2.45× for BBR, BBR-IM and BBP respectively.

The results for the Cortex-A9 processor are significantly different. In general, the performance improvements are smaller as the shorter processor pipeline in the Cortex-A9 microarchitecture (9-12 stages) induces smaller penalties for mispredicted branches than the pipeline in the Core i9-7960X's Skylake-X microarchitecture (14-19 stages). This means that the common bc bin decoding is more efficient and the opportunities for improvement are smaller. As a result, the speedup with the BBP technique is only in the range between 1.18× and 1.65×. Due to the lack of hardware division support, the BBR technique even reduces the performance down to 0.88× in some cases. For very high-quality videos, a speedup of up to 1.12× is still possible as the exploitation of parallelism for the extreme number of bc bins compensates for the inefficient division implementation. Compared to BBR, BBR-IM performs significantly better on the Cortex-A9 processor with speedups up to 1.30× and no performance degradation for low-quality videos.

The bitstream overhead for the BBP approach is negligible. It is less than 0.2% for all videos from the Class B testset and even less than 0.002% for the RGB-444 testset. Considering the substantial speedups in bc bin decoding across all quality levels and the removal of any computation for encoding and decoding, we strongly recommend considering separate cc bin and bc bin partitions in future video coding standards.

7.4 Conclusions

We have presented three techniques to improve bc bin decoding for transform coefficients in HEVC CABAC. We propose the bypass-bin reservoir (BBR) as a combination of a division-based algorithm to decode many bins in parallel and the exploitation of bc bin grouping in HEVC. A division-free version of the BBR is implemented that can achieve similar performance by replacing the division by inverse multiplication (BBR-IM). Performance improvements are demonstrated for embedded processors without hardware division support. Furthermore, BBR-IM can be beneficial for systems with less-powerful division hardware, e.g. mobile processors. Finally, we propose the use of a separate bitstream partition for bc bins (BBP), which makes their encoding and

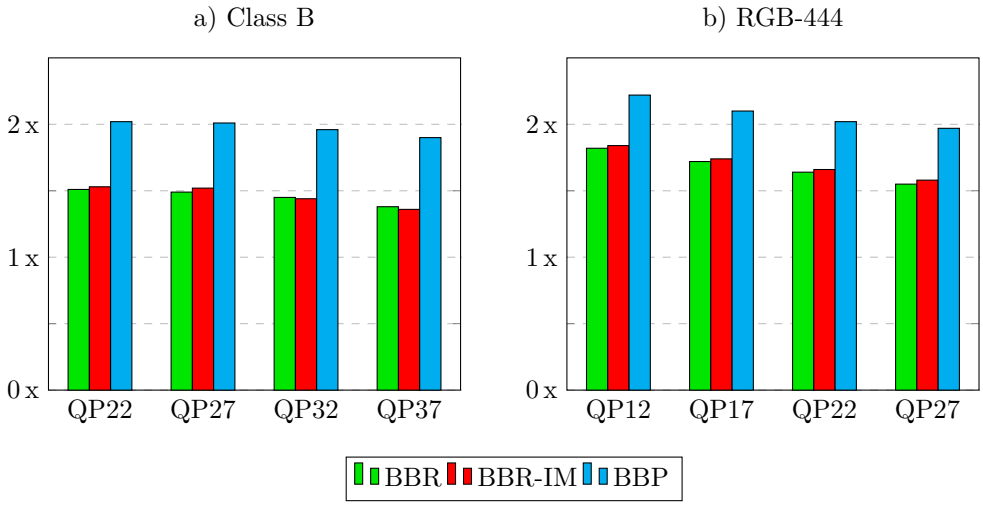


Figure 7.1: Speedup over common bc bin decoding on the Intel Core i9-7960X processor with the bypass-bin reservoir technique (BBR), its division-free implementation (BBR-IM), and the use of a separate bc bin partition (BBP).

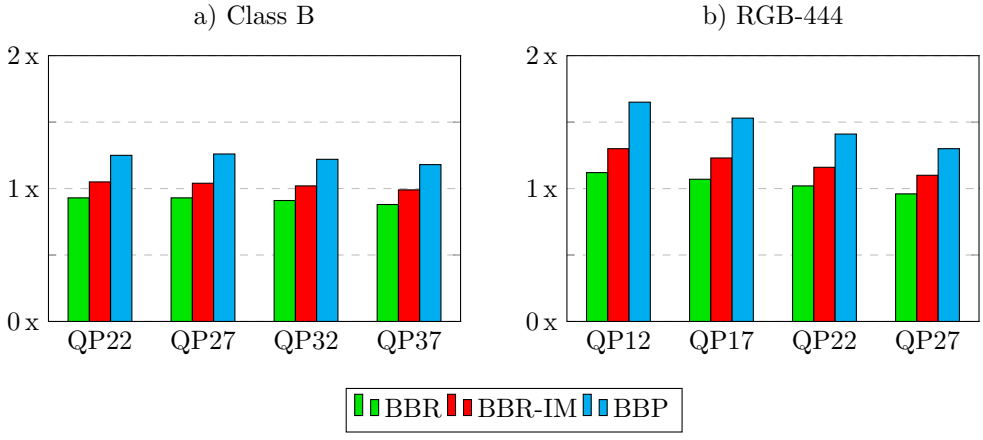


Figure 7.2: Speedup over common bc bin decoding on the ARM Cortex-A9 processor with the bypass-bin reservoir technique (BBR), its division-free implementation (BBR-IM), and the use of a separate bc bin partition (BBP).

decoding computationless and affects all bc bins. The maximum decoding speedups for transform coefficient bc bin decoding are 2.00×, 2.04× and 2.45× for BBR, BBR-IM and BBP respectively. While BBR and BBR-IM can be used with the current HEVC standard, BBP is strongly recommended for the use in future video coding standards due to its substantial performance improvements, negligible bitstream overhead, and high potential for energy savings.

CHAPTER 8

Conclusions

The ever-increasing demands for higher video resolution and quality require even greater computational performance growth. At the same time, higher compression performance is one of the most important goals of video codec development to make the best use of the available network bandwidth and device storage. Modern video compression standards represent a trade-off to reach the best compression rate while maintaining real-time capabilities. This means that the best compression algorithms cannot always be used due to performance limitations. Furthermore, energy efficiency is of utmost importance given the tremendous number of video applications used on mobile devices. Especially these devices also require optimizations in hardware area and cost. HEVCs successor VVC will require even more computational power as the compression gains are reached with the introduction of more complex coding tools.

High-quality video encoding and decoding are computationally complex processes that require highly efficient parallel implementations to achieve real-time performance. Fortunately, there is a massive amount of parallelism in video coding applications that can be exploited by today's multi- and many-core systems. Vectorization is also commonly used to improve both the performance and energy efficiency of most video coding components. The one major exception in state-of-the-art video compression standards is CABAC, which is without any doubt a challenging task for modern computer architectures. Although it can benefit from high-level parallelization, its underlying sequential algorithm and lack of DLP prevents the use of vectorization techniques and makes the development of customized hardware solutions very difficult. The profiling results in Section 2.2 clearly show that the decoding time fraction for CABAC is increasing with the vector size up to the point where it takes most of the time in many workloads on modern processors. Other optimization approaches are essential to address this major throughput bottleneck in modern video coding applications.

8.1 Summary

In this thesis, we have presented multiple approaches to improve CABAC decoding throughput (see Table 8.1). They cover parallelization on different levels and architectural optimizations.

First, we have developed three improved WPP implementations for HEVC CABAC decoding. Conventional WPP establishes a horizontal offset of at least two CTUs between consecutive CTU rows to make sure that all dependencies are satisfied. CABAC decoding only depends on the CTU above the currently processed one. As a result, decoupling CABAC decoding from the reconstruction process allows reducing the WPP offset to only one CTU once the context initialization is finished. The unbalanced load among all CTUs allows threads that process specific CTU rows to catch up with other threads once they reach more complex frame areas while conventional WPP would lead to stalls. This results in improved parallel efficiency of the WPP implementation. The relaxed WPP implementation leads to average speedups up to 1.55×. The throughput improvements highly depend on the frame content. Best results were achieved for low-delay applications such as video conferencing and video chatting, as well as for FHD entertainment applications. Two more fine-grained WPP implementations (FG-WPP) perform dependency checks at the CU and syntax element level instead of CTU level. This results in even higher speedups up to 1.83× on average. CU-level dependency checks are the preferred method because the performance is negligibly worse than with syntax element granularity while a single integer counter per CTU row that is incremented in z-scan order suffices to implement the fine-grained dependency checks. All in all, the proposed WPP implementations significantly improve the parallel efficiency within a frame at a very low cost compared to conventional WPP.

The presented B3P approach enables a substantial amount of parallelism during the decoding of a single CTU. The bins of a regular bitstream are distributed across eight static partitions for parallel decoding. This also allows the specialization of the corresponding subdecoders as they process the bins of fewer different types of syntax elements, resulting in significant clock frequency improvements and reduced additional hardware cost. The partitioning scheme was designed to reduce dependencies between partitions and to achieve optimal load balancing for high-quality videos when CABAC decoding is most critical for the overall decoding process. The B3P approach leads to speedups up to 8.5× while using only 61.9% more logic area. The bitstream overhead is negligible for high bitrates and does not exceed 0.7% for FHD entertainment applications of all quality levels. B3P outperforms high-level parallelization tools such as WPP and Tiles in terms of bitstream overhead and especially hardware cost. Compared to the related Syntax Element Partitioning we achieve much better speedups for high-quality videos at the same bitstream overhead and slightly lower hardware cost.

A context model cache architecture has been implemented to shorten the critical path of the CABAC decoder and reach higher clock frequencies. In addition, a prefetching algorithm was implemented to minimize stalls due to cache misses. The prefetching algorithm adapts to different video characteristics by using specific context models for prefetching decisions. The context model memory layout was designed to exploit spatial and temporal locality with respect to the HEVC CABAC decoding algorithm. We performed a design space exploration of different cache and cache line sizes to find the best architecture in terms of performance, hardware cost and energy efficiency. We have demonstrated that a cache with two cache lines of eight context models suffices to reduce the miss rate to at most 2% for all test videos. This leads to performance improvements of 13.2% to 16.7% thanks to the clock frequency enhancement of 17%.

Table 8.1: Overview of the presented CABAC optimization approaches. FG-WPP compared to conventional WPP. Other approaches compared to identical baseline without the technique. HW/SW suitability: ✓ (full), ❍ (limited), ✗ (none). Bitstream Overhead based on FHD testset (JCT-VC Class B). Some results are not evaluated within the scope of this thesis (n.a.).

	HW	SW	Max. Speedup	Hardware Overhead	Bitstream Overhead	HEVC Conformance
FG-WPP	✓	✓	1.83×	0.0 %	0.0 %	✓
B3P	✓	❍	8.50×	61.9 %	< 0.7 %	✗
Cache	✓	✗	1.17×	7.5 %	0.0 %	✓
BBR	❍	✓	2.04×	n.a.	0.0 %	✓
BBP	✓	✓	2.45×	n.a.	< 0.2 %	✗

Finally, we have optimized bc bin decoding on GPPs. The decoding performance of this task suffers from frequent hardly predictable branches. We proposed a Bypass-Bin Reservoir (BBR) to significantly enhance bc bin decoding throughput. This is achieved by decoding as many bc bins as possible in one step with linear but more complex arithmetic functions instead of control-intensive bin-by-bin decoding. We thereby exploit the bc bin grouping that has been introduced in HEVC for improved throughput. Furthermore, the separation of cc and bc bins into different partitions is proposed to remove the computational work for bc bin decoding at all. The BBR leads to speedups up to 2.04× while the bin partitioning approach with a separate bc bin partition (BBP) allows speedups up to 2.45×. The cost for the higher speedup with BBP is a bitstream overhead of less than 0.2%.

We have demonstrated that significant performance improvements are achievable with the presented approaches. Most of them can be combined because they are orthogonal and exploit parallelism on different levels, thus multiplying their speedups. An exception is the combination of B3P with BBR/BBP as all of them exploit bin-level parallelism. Also, the effectiveness of other approaches is affected in some cases.

Using B3P/BBP with WPP multiplies the bitstream overhead for the former as it is needed for every WPP substream. However, It should be noted that the combination of these techniques is only necessary for very high-quality video coding. As the bitstream overhead is orders of magnitude below one percent for these workloads, even a multiplication is still insignificant.

The effectiveness of a context model cache is reduced when used with the B3P approach as the original context model memory is distributed among the subdecoders. The smaller memories have less potential for clock frequency improvements when replacing them with caches. On the other hand, prefetching becomes less complex.

Overall, the thoughtful combination of the presented approaches allows reaching even higher performance goals.

8.2 Recommendations and Future Work

Video applications and computer architectures have influenced the development of each other for decades and their co-development is also essential in the future to allow further spreading and enhancement of these applications in our everyday life. In the following, we provide recommendations for video codec developers and computer architects, as well as research ideas for improved video coding in general and CABAC in particular.

Computer architectures have undergone substantial improvements in performance and energy efficiency for decades. Homogeneous Multiprocessors maintained the rate of improvement for another decade by exploiting parallelism when the pure increment in clock frequency hit power barriers. Today, the heterogeneity of processor cores and customized hardware accelerators is one of the most promising techniques for further improvements in performance and energy efficiency. The absence of floating-point arithmetic in modern video compression standards is an excellent example where GPPs and especially GPUs waste a lot of chip area that remains unused. CABAC decoding can even be implemented with a basic 16-bit integer ISA. Furthermore, the low complexity of the involved operations might lead to shorter processor pipelines, thus reducing the branch misprediction penalties for this control-intensive task. These simplifications allow multiple of such cores instead of a single GPP core at the same chip area, thereby improving high-level parallelization opportunities and energy efficiency.

Another example of heterogeneity is the emergence of FPGA-SoCs. While most video coding tasks can be quite efficiently implemented using the vector extensions of modern processors, sequential CABAC decoding can be accelerated by a tightly-coupled FPGA. FPGAs also allow optimizations regarding multi-standard video coding as many modern video compression standards are very similar in their general design principles. Reconfigurability would lead to significant savings in chip area compared to separate hardware accelerators for all required video compression standards. CABAC decoding might also be suitable for application-specific instruction set processors as the simple yet control-intensive decoding process for a bin can be merged into a single instruction, thus greatly improving this critical task.

The limited benefit from the further increasing width of vector extension registers has also been demonstrated. It might be way more useful for video applications to implement support for two-dimensional block processing. While this imposes severe challenges on the memory system, it might very well be suitable on the cache level.

Fine-grained WPP has been proven to significantly improve the parallel efficiency for CABAC decoding compared to conventional WPP with a two-CTU offset. The concept is applicable to the other major video coding components as well, although the expected benefit is smaller as it is for CABAC due to rare extreme cases that induce dependencies close to the two-CTU offset. Avoiding the use of intra prediction directions and motion vector candidates in the encoder that lead to these extreme cases might result in more efficient WPP implementations for the whole decoder with only marginal bitrate increases.

Merging blocks for concurrent processing and thereby better utilizing modern vector architectures is a very promising approach as long as the operations for these blocks are sufficiently similar. This is among others possible for motion compensation for PUs with motion vectors that differ only in the fractional part. Their memory accesses cover adjacent memory areas while the arithmetic operations are the same except for fixed coefficients that can be adjusted accordingly. Furthermore, motion vectors of adjacent PUs are often identical. The coding quad-tree structure in HEVC is highly responsible for that as it splits a block into four subblocks if only a small part of the corresponding frame area needs to be predicted differently. The more flexible coding tree structure in VVC is a good step to reduce such inefficiencies. An encoder-based approach for better vectorization opportunities is the sacrification of compression performance to a higher extent to avoid small block sizes and consequently allow higher throughput.

The sequential CABAC algorithm is for sure one of the most critical throughput bottlenecks in modern video compression standards. The situation is not expected to change in VVC as CABAC is not significantly modified. Frame- and CTU-level parallelization techniques affect all video coding tasks to a similar degree while CABAC is the only one that lacks DLP. Relaxing low-level dependencies to allow more efficient hardware solutions, and enabling more parallelization opportunities specifically for CABAC is required to compensate for the lack of DLP. The bitstream partitioning approach presented in Chapter 5 serves this purpose. A less complex yet effective approach has been evaluated in Chapter 7. The separation of cc and bc bins enables the parallel processing of both bin types and removes any computation for bc bin decoding. This partitioning approach is most effective with a high bc bin fraction, i.e. for high bitrates when CABAC decoding is most critical for the overall decoding performance. Consequently, we strongly recommend considering separate cc and bc bin partitions for future video compression standards given that the bitstream overhead is negligible.

Relaxing bin-to-bin dependencies would allow further low-level parallelization opportunities. Bins that use the same context model are often grouped to reduce the amount of context speculation. However, this introduces dependencies between the decoding processes for two consecutive bins as the context model needs to be updated before it can be reused. Using constant context models within a 4x4 subblock in TB decoding might be a reasonable trade-off. It removes dependencies and thereby allows higher throughput while not sacrificing compression efficiency too much. This technique is expected to have a significant impact on overall decoding performance as the majority of bins is associated with TB decoding in high-quality videos.

After all, further advances in computational and compression performance, energy efficiency, and hardware area and cost besides the natural evolution of computer architectures are only possible when video coding applications are designed to better match the capabilities of these architectures. They in turn need to be able to exploit the massive amount of parallelism in video coding applications at all levels to implement them most efficiently. These advances are essential to further increase the quality, further spreading, and emergence of future video applications.

List of Figures

1.1 Compression efficiency development in video coding standards 2

2.1 Partitioning of a frame into CTUs 11
2.2 Block partitioning in HEVC 11
2.2a Recursive quadtree splitting 11
2.2b PU shapes 11
2.3 Block diagram of the HEVC decoder 13
2.4 GOP structure 17
2.5 High-level parallelization tools in HEVC 18
2.5a Slices 18
2.5b Tiles 18
2.5c WPP 18
2.6 Smartphone + FHD testset profiling 24
2.7 HP desktop + video production testset profiling 25
2.8 HP desktop + UHD testset profiling 25
2.9 Arithmetic encoding example 29
2.10 Arithmetic decoding example 29
2.11 TB compression example 33

3.1 Four-stage CABAC decoder pipeline 42

4.1 Worst-case dependencies in WPP 47
4.1a Intra prediction 47
4.1b Motion vector derivation 47
4.2 Speedup of improved WPP implementations 49
4.3 Parallel efficiency for WPP CABAC decoding 50

5.1 Decoding of syntax element partitions 55
5.1a sequential 55
5.1b parallel 55
5.2 Variable-sized partition length fields 57
5.3 Parallel B3P decoder architecture 58
5.4 Cc bin decoder architecture 60
5.5 Circuit for computing the number of leading one-bits 62

5.6 Speedup of the B3P decoder 66
5.7 Bitstream overhead with the B3P scheme 67
5.8 Bitstream overhead comparison for B3P, SEP, WPP, and Tiles 69

6.1 Two-stage decoder pipeline 75
6.2 CU decoding flow extract in a B/P-Slice 78
6.3 Prefetching candidate selection process 80
6.4 Clock frequency of the cached decoder 82
6.5 Cache miss rate without prefetching and CMS4 83
6.6 Cache miss rate without prefetching and CMS8 84
6.7 Cache miss rate with prefetching and CMS4 86
6.8 Cache miss rate with prefetching and CMS8 87
6.9 Throughput improvement with prefetching 88
6.10 Energy efficiency of the cached decoder 93

7.1 Bc bin decoding speedup on the Intel Core i9-7960X processor 103
7.2 Bc bin decoding speedup on the ARM Cortex-A9 processor 103

List of Tables

2.1 Common video resolutions 10
2.2 Chroma subsampling modes 10
2.3 Architectural parameters of the evaluation platforms 21
2.4 Video coding applications 22
2.5 Performance overview 23
2.6 Detailed performance results 23
2.7 Binarization schemes in HEVC 31

3.1 Comparison of high-level parallelization tools in HEVC 39
3.2 Comparison of bitstream partitioning approaches and high-level parallelization tools 41
3.3 Comparison of CABAC decoders with context model caches 43

5.1 Bitstream partitions 56
5.2 Architectural characteristics and maximum clock frequencies for B3P subdecoders 61
5.3 Comparison of B3P, SEP, WPP and Tiles 71

6.1 Context Model Memory Layout (part 1) 76
6.2 Context Model Memory Layout (part 2) 77
6.3 Cache miss rate with prefetching and one cache line 85
6.4 Comparison to other hardware decoders. 89
6.5 Real-time decoding throughput evaluation 90
6.6 Resource utilization on the Xilinx Zynq-7045 SoC 91

8.1 Overview of the presented CABAC optimization approaches 107

List of Listings

7.1 Bitstream state variables for HEVC CABAC decoding 96
7.2 Common bc bin decoding function 96
7.3 Functions for decoding multiple bc bins 97
7.4 BBR state variables for HEVC CABAC decoding 99
7.5 Fill BBR function 99
7.6 Extract leading-ones from BBR 100
7.7 Division by inverse multiplication 100

List of Abbreviations

AI	all-intra
ALF	adaptive loop filter
AV1	AOMedia Video 1
B3P	Bin-Based Bitstream Partitioning
BBP	bypass-bin partition
BBR	bypass-bin reservoir
BBR-IM	bypass-bin reservoir with inverse multiplication
bc	bypass-coded
BDR	Bjøntegaard delta bitrate
bin	binary symbol
BRAM	block random-access memory
CABAC	context-based adaptive binary arithmetic coding
CAVLC	context-based adaptive variable length coding
Cb	chrominance deviation from gray towards blue
cc	context-coded
CL	cache line
CMS	context model set
CPU	central processing unit
Cr	chrominance deviation from gray towards red
CTU	coding tree unit
CU	coding unit
CU-WPP	coding-unit-based Wavefront Parallel Processing
DBF	deblocking filter
DLP	data-level parallelism
DSP	digital signal processor
FG-WPP	fine-grained Wavefront Parallel Processing
FHD	Full High Definition (1920×1080 samples)
FPGA	field-programmable gate array
fps	frames per second

GOP	group of pictures
GPP	general-purpose processor
GPU	graphics processing unit
HEVC	High Efficiency Video Coding
IP	Internet Protocol
IQ/IT	inverse quantization and transform
ISA	instruction set architecture
JCT-VC	Joint Collaborative Team on Video Coding
JVET	Joint Video Exploration Team
LD	low-delay
LPS	least probable symbol
LRU	least recently used
LUT	lookup table
MPS	most probable symbol
OWF	Overlapped Wavefront
PSNR	peak signal-to-noise ratio
PU	prediction unit
QP	quantization parameter
RA	random-access
SAO	sample adaptive offset
SEP	Syntax Element Partitioning
SE-WPP	syntax-element-based Wavefront Parallel Processing
SoC	system on chip
TB	transform block
TLP	thread-level parallelism
TU	transform unit
UHD	Ultra High Definition (3840×2160 samples)
VVC	Versatile Video Coding
WPP	Wavefront Parallel Processing
WPP1	Wavefront Parallel Processing with one-CTU offset
WPP2	Wavefront Parallel Processing with two-CTU offset
Y	luminance

Bibliography

Publications

[1] P. Habermann. “Design and Implementation of a High-Throughput CABAC Hardware Accelerator for the HEVC Decoder”. In: *Lecture Notes in Informatics - Seminars, Informatiktage 2014*. 2014, pp. 213–216.

[2] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “Optimizing HEVC CABAC Decoding with a Context Model Cache and Application-Specific Prefetching”. In: *Proceedings of the 2015 IEEE International Symposium on Multimedia*. Dec. 2015, pp. 429–434.

[3] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “Application-Specific Cache and Prefetching for HEVC CABAC Decoding”. In: *IEEE MultiMedia* 24.1 (Jan. 2017), pp. 72–85.

[4] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “Syntax Element Partitioning for high-throughput HEVC CABAC decoding”. In: *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing*. Mar. 2017, pp. 1308–1312.

[5] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “Improved Wavefront Parallel Processing for HEVC Decoding”. In: *Proceedings of the 13th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems*. July 2017, pp. 253–256.

[6] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “A Bin-Based Bitstream Partitioning Approach for Parallel CABAC Decoding in Next Generation Video Coding”. In: *Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium*. May 2019, pp. 1053–1062.

[7] P. Habermann, C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. “Efficient Wavefront Parallel Processing for HEVC CABAC Decoding”. In: *Proceedings of the 28th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. Mar. 2020, pp. 339–343.

Standards, Software, Testsets

[8] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand. “Overview of the High Efficiency Video Coding (HEVC) Standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1649–1668.

[9] *Recommendation ITU-T H.265 - High Efficiency Video Coding*. Nov. 2019. URL: https://www.itu.int/rec/T-REC-H.265.

[10] *HEVC Test Model*. URL: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/.

[11] D. Flynn, D. Marpe, M. Naccari, T. Nguyen, C. Rosewarne, K. Sharman, J. Sole, and J. Xu. "Overview of the Range Extensions for the HEVC Standard: Tools, Profiles, and Performance". In: *IEEE Transactions on Circuits and Systems for Video Technology* 26.1 (Jan. 2016), pp. 4–19.

[12] J. M. Boyce, Y. Ye, J. Chen, and A. K. Ramasubramonian. "Overview of SHVC: Scalable Extensions of the High Efficiency Video Coding Standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 26.1 (Jan. 2016), pp. 20–34.

[13] G. Tech, Y. Chen, K. Müller, J. Ohm, A. Vetro, and Y. Wang. "Overview of the Multiview and 3D Extensions of High Efficiency Video Coding". In: *IEEE Transactions on Circuits and Systems for Video Technology* 26.1 (Jan. 2016), pp. 35–49.

[14] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. "Overview of the H.264/AVC video coding standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (July 2003), pp. 560–576.

[15] *Recommendation ITU-T H.264 - Advanced Video Coding for Generic Audiovisual Services*. Apr. 2017. URL: https://www.itu.int/rec/T-REC-H.264.

[16] *Recommendation ITU-T H.263 - Video coding for low bitrate communication*. Jan. 2005. URL: https://www.itu.int/rec/T-REC-H.263/.

[17] B. Bross, J. Chen, and S. Liu. *Versatile Video Coding (Draft 8)*. Jan. 2020. URL: http://phenix.it-sudparis.eu/jvet/doc_end_user/current_document.php?id=9675.

[18] *VVC Test Model*. URL: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM.

[19] Y. Chen et al. "An Overview of Core Coding Tools in the AV1 Video Codec". In: *Proceedings of the 2018 Picture Coding Symposium*. June 2018, pp. 41–45.

[20] P. de Rivaz and J. Haughton. *AV1 Bitstream & Decoding Process Specification*. URL: https://aomediacodec.github.io/av1-spec/av1-spec.pdf.

[21] D. Mukherjee, J. Han, J. Bankoski, R. Bultje, A. Grange, J. Koleszar, P. Wilkins, and Y. Xu. "A Technical Overview of VP9 - The Latest Open-Source Video Codec". In: *SMPTE Motion Imaging Journal* 124.1 (Jan. 2015), pp. 44–54.

[22] A. Grange, P. de Rivaz, and J. Hunt. *VP9 Bitstream & Decoding Process Specification*. URL: https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.6-20160331-draft.pdf.

[23] F. Bossen. "Common HM test conditions and software reference configurations". In: *Joint Collaborative Team on Video Coding document JCTVC-L1100* (2013).

[24] C. Rosewarne, K. Sharman, and D. Flynn. "Common test conditions and software reference configurations for HEVC range extensions". In: *Joint Collaborative Team on Video Coding document JCTVC-P1006* (2014).

[25] *EBU UHD-1 Test Sequences*. URL: https://tech.ebu.ch/testsequences/uhd-1.

Video Coding

[26] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand. "Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1669–1684.

[27] F. Bossen, B. Bross, K. Suhring, and D. Flynn. "HEVC Complexity and Implementation Analysis". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1685–1696.

[28] B. Bross, V. George, M. Alvarez-Mesa, T. Mayer, C. C. Chi, J. Brandenburg, T. Schierl, D. Marpe, and B. Juurlink. "HEVC performance and complexity for 4K video". In: *Proceedings of the 2013 IEEE Third International Conference on Consumer Electronics Berlin*. Sept. 2013, pp. 44–47.

[29] F. Bossen. "On software complexity: decoding 720p content on a tablet". In: *Joint Collaborative Team on Video Coding document JCTVC-J0128* (2012).

[30] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl. "Parallel Scalability and Efficiency of HEVC Parallelization Approaches". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1827–1838.

[31] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl. "SIMD Acceleration for HEVC Decoding". In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.5 (May 2015), pp. 841–855.

[32] C. C. Chi, M. Alvarez-Mesa, and B. Juurlink. "Low-power high-efficiency video decoding using general-purpose processors". In: *ACM Transactions on Architecture and Code Optimization* 11.4 (2015), p. 56.

[33] K. Misra, J. Zhao, and A. Segall. "Entropy Slices for Parallel Entropy Coding". In: *Joint Collaborative Team on Video Coding document JCTVC-B111* (July 2010).

[34] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. "An Overview of Tiles in HEVC". In: *IEEE Journal of Selected Topics in Signal Processing* 7.6 (Dec. 2013), pp. 969–977.

[35] F. Henry and S. Pateux. "Wavefront Parallel Processing". In: *Joint Collaborative Team on Video Coding document JCTVC-E196* (Mar. 2011).

[36] V. Baroncini, J.-R. Ohm, and G. J. Sullivan. *Report of results from the Call for Proposals on Video Compression with Capability beyond HEVC*. Sept. 2018.

[37] D. Grois, T. Nguyen, and D. Marpe. "Coding efficiency comparison of AV1/VP9, H.265/MPEG-HEVC, and H.264/MPEG-AVC encoders". In: *Proceedings of the 2016 Picture Coding Symposium*. Dec. 2016, pp. 1–5.

[38] T. Laude, Y. G. Adhisantoso, J. Voges, M. Munderloh, and J. Ostermann. "A Comparison of JEM and AV1 with HEVC: Coding Tools, Coding Efficiency and Complexity". In: *Proceedings of the 2018 Picture Coding Symposium*. June 2018, pp. 36–40.

[39] J. Stankowski, D. Karwowski, T. Grajek, K. Wegner, J. Siast, K. Klimaszewski, O. Stankiewicz, and M. Domański. "Bitrate distribution of syntax elements in the HEVC encoded video". In: *Proceedings of the 2014 International Conference on Signals and Electronic Systems*. Sept. 2014, pp. 1–4.

[40] G. Bjøntegaard. "Calculation of average PSNR differences between RD-curves". In: *Video Coding Experts Group document VCEG-M33*. 2001, pp. 2–4.

[41] K. Chen, Y. Duan, J. Sun, and Z. Guo. "Towards efficient wavefront parallel encoding of HEVC: Parallelism analysis and improvement". In: *Proceedings of the IEEE 16th International Workshop on Multimedia Signal Processing*. Sept. 2014, pp. 1–6.

[42] Shaobo Zhang, Xiaoyun Zhang, and Zhiyong Gao. "Implementation and improvement of Wavefront Parallel Processing for HEVC encoding on many-core platform". In: *Proceedings of the 2014 IEEE International Conference on Multimedia and Expo Workshops*. July 2014, pp. 1–6.

[43] D. Engelhardt, J. Moller, J. Hahlbeck, and B. Stabernack. "FPGA implementation of a full HD real-time HEVC main profile decoder". In: *IEEE Transactions on Consumer Electronics* 60.3 (Aug. 2014), pp. 476–484.

Entropy Coding

[44] V. Sze and M. Budagavi. "High Throughput CABAC Entropy Coding in HEVC". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1778–1791.

[45] D. Marpe, H. Schwarz, and T. Wiegand. "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (July 2003), pp. 620–636.

[46] V. Sze and M. Budagavi. "A comparison of CABAC throughput for HEVC/H.265 VS. AVC/H.264". In: *Proceedings of the 2013 IEEE Workshop on Signal Processing Systems*. Oct. 2013, pp. 165–170.

[47] D. Marpe and T. Wiegand. "A highly efficient multiplication-free binary arithmetic coder and its application in video coding". In: *Proceedings of the 2003 International Conference on Image Processing*. Vol. 2. Sept. 2003, pp. 263–266.

[48] I. H. Witten, R. M. Neal, and J. G. Cleary. "Arithmetic coding for data compression". In: *Communications of the ACM* 30.6 (1987), pp. 520–540.

[49] C. E. Shannon. "A mathematical theory of communication". In: *Bell system technical journal* 27.3 (1948), pp. 379–423.

[50] D. A. Huffman. "A method for the construction of minimum-redundancy codes". In: *Proceedings of the Institute of Radio Engineers* 40.9 (1952), pp. 1098–1101.

[51] D. Marpe, H. Schwarz, and T. Wiegand. "Entropy coding in video compression using probability interval partitioning". In: *Proceedings of the 28th Picture Coding Symposium*. Dec. 2010, pp. 66–69.

[52] H. Kirchhoffer, D. Marpe, C. Bartnik, A. Henkel, M. Siekmann, J. Stegemann, H. Schwarz, and T. Wiegand. "Probability interval partitioning entropy coding using systematic variable-to-variable length codes". In: *Proceedings of the 18th IEEE International Conference on Image Processing*. Sept. 2011, pp. 333–336.

[53] P. Lin, T. Chuang, and L. Chen. "A branch selection multi-symbol high throughput CABAC decoder architecture for H.264/AVC". In: *Proceedings of the 2009 IEEE International Symposium on Circuits and Systems*. May 2009, pp. 365–368.

[54] Y. Yang and J. Guo. "High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications". In: *IEEE Transactions on Circuits and Systems for Video Technology* 19.9 (Sept. 2009), pp. 1395–1399.

[55] Yu Hong, Peilin Liu, Hang Zhang, Zongyuan You, D. Zhou, and S. Goto. "A 360Mbin/s CABAC decoder for H.264/AVC level 5.1 applications". In: *Proceedings of the 2009 International SoC Design Conference*. Nov. 2009, pp. 71–74.

[56] V. Sze, A. P. Chandrakasan, M. Budagavi, and M. Zhou. "Parallel CABAC for low power video coding". In: *Proceedings of the 15th IEEE International Conference on Image Processing*. 2008, pp. 2096–2099.

[57] V. Sze and A. P. Chandrakasan. "Joint Algorithm-Architecture Optimization of CABAC". In: *Journal of Signal Processing Systems* 69.3 (Dec. 2012), pp. 239–252.

[58] V. Sze and A. P. Chandrakasan. "Joint algorithm-architecture optimization of CABAC to increase speed and reduce area cost". In: *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing*. May 2011, pp. 1577–1580.

[59] Y. Liao, G. Li, and T. Chang. "A high throughput VLSI design with hybrid memory architecture for H.264/AVC CABAC decoder". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. May 2010, pp. 2007–2010.

[60] Y. Yang, C. Lin, H. Chang, C. Su, and J. Guo. "A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding with Look Ahead Parsing". In: *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo*. July 2006, pp. 357–360.

[61] J. Chen and Y. Lin. "A high-performance hardwired CABAC decoder for ultra-high resolution video". In: *IEEE Transactions on Consumer Electronics* 55.3 (Aug. 2009), pp. 1614–1622.

[62] P. Zhang, D. Xie, and W. Gao. "Variable-Bin-Rate CABAC Engine for H.264/AVC High Definition Real-Time Decoding". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.3 (Mar. 2009), pp. 417–426.

[63] S. Chen, S. Chen, and S. Sun. "P3-CABAC: A Nonstandard Tri-Thread Parallel Evolution of CABAC in the Manycore Era". In: *IEEE Transactions on Circuits and Systems for Video Technology* 20.6 (June 2010), pp. 920–924.

[64] J. Sole, R. Joshi, N. Nguyen, T. Ji, M. Karczewicz, G. Clare, F. Henry, and A. Duenas. "Transform Coefficient Coding in HEVC". In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1765–1777.

[65] V. Sze and M. Budagavi. "Parallelization of CABAC transform coefficient coding for HEVC". In: *Proceedings of the 2012 Picture Coding Symposium*. May 2012, pp. 509–512.

[66] J. Sankaran. *Method of CABAC significance map decoding suitable for use on VLIW data processors*. US Patent 7,813,567. Oct. 2010.

[67] J. Sankaran. *Method of CABAC coefficient magnitude and sign decoding suitable for use on VLIW data processors*. US Patent 7,885,473. Feb. 2011.

[68] J. Hahlbeck and B. Stabernack. "A 4k capable FPGA based high throughput binary arithmetic decoder for H.265/MPEG-HEVC". In: *Proceedings of the 2014 IEEE International Conference on Consumer Electronics Berlin*. Sept. 2014, pp. 388–390.

[69] V. Sze. "Parallel Algorithms and Architectures for Low Power Video Decoding". PhD thesis. Massachusetts Institute of Technology, June 2010.

[70] V. Sze and A. P. Chandrakasan. "A high throughput CABAC algorithm using syntax element partitioning". In: *Proceedings of the 16th IEEE International Conference on Image Processing*. Nov. 2009, pp. 773–776.

[71] Y. Chen and V. Sze. "A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-Tier Applications". In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.5 (May 2015), pp. 856–868.

[72] Chung-Hyo Kim and In-Cheol Park. "High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction". In: *Proceedings of the 2006 IEEE International Symposium on Circuits and Systems*. May 2006, pp. 1707–1710.

[73] B. Shi, W. Zheng, H. Lee, D. Li, and M. Zhang. "Pipelined Architecture Design of H.264/AVC CABAC Real-Time Decoding". In: *Proceedings of the 4th IEEE International Conference on Circuits and Systems for Communications*. May 2008, pp. 492–496.

[74] Yuan-Teng Chang. "A novel pipeline architecture for H.264/AVC CABAC decoder". In: *Proceedings of the 2008 IEEE Asia Pacific Conference on Circuits and Systems*. Nov. 2008, pp. 308–311.

[75] Y. Yi and I. Park. "High-Speed H.264/AVC CABAC Decoding". In: *IEEE Transactions on Circuits and Systems for Video Technology* 17.4 (Apr. 2007), pp. 490–494.

Miscellaneous

[76] *Cisco Visual Networking Index: Forecast and Trends, 2017–2022, White Paper*. Feb. 2019. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[77] G. M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the 1967 Spring Joint Computer Conference*. 1967, pp. 483–485.

[78] *Spin Digital Video Technologies GmbH*. URL: http://www.spin-digital.com.

[79] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. "Understanding Sources of Inefficiency in General-purpose Chips". In: *Proceedings of the 37th Annual International Symposium on Computer Architecture*. 2010, pp. 37–47.

[80] J. Corbal, M. Valero, and R. Espasa. "Exploiting a new level of DLP in multimedia applications". In: *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. Nov. 1999, pp. 72–79.

[81] A. Venkat and D. M. Tullsen. "Harnessing ISA Diversity: Design of a heterogeneous-ISA Chip Multiprocessor". In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture*. 2014, pp. 121–132.

[82] *ARM Architecture Reference Manual Supplement: The Scalable Vector Extensions (SVE), for ARMv8-A*. URL: https://static.docs.arm.com/ddi0584/a/DDI0584A_a_SVE_supp_armv8A.pdf.

[83] *Zynq-7000 SoC Data Sheet: Overview*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[84] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. June 2016. URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf.

[85] *Book cover background image*. URL: http://6iee.com/data/uploads/21/467553.jpg.