UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación

Contributions to the Efficient Implementation of High-Computational Demanding Video Algorithms over Heterogeneous Platforms

TESIS DOCTORAL

Anup Saha

Doble máster en Mechatronics Engineering, Universidad de Trento, Italia y Mechanical Engineering, Universidad de Lisboa, Portugal

2023

Departamento de Ingeniería Telemática y Electrónica

Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación

Contributions to the Efficient Implementation of High-Computational Demanding Video Algorithms over Heterogeneous Platforms

Tesis Doctoral

Autor: **Anup Saha**
T.I.M.E. título de doble máster en Mechatronics Engineering, Universidad de Trento, Italia y Mechanical Engineering (Major in System Engineering), Instituto Superior Técnico – Universidad de Lisboa, Portugal

Directores:

Fernando Pescador del Oso
Doctor por la Universidad Politécnica de Madrid

Miguel Chavarrías Lapastora
Doctor por la Universidad Politécnica de Madrid

2023

TÍTULO: Contributions to the Efficient Implementation of High-Computational Demanding Video Algorithms over Heterogeneous Platforms

AUTOR: Anup Saha

DIRECTORES: Fernando Pescador del Oso
Miguel Chavarrías Lapastora

El Tribunal nombrado con fecha ..... de ..... de 2023 por el Rector Mgfco. de la Universidad Politécnica de Madrid, compuesto por los doctores:

Presidente:

Vocal:

Vocal:

Vocal:

Secretario:

Suplente:

Suplente:

realizado el acto de lectura y defensa de la tesis doctoral en la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la Universidad Politécnica de Madrid, acuerda otorgar la calificación de

---

Madrid, a ..... de ..... de 2023

EL PRESIDENTE EL SECRETARIO

LOS VOCALES

I dedicate this work to my parents, my wife, and my son.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my directors Fernando Pescador and Miguel Chavarrías for the guidance, for the ideas provided during this thesis and for constantly challenging me to overcome myself. This gratitude also extends to the rest of the research group: César Sanz, Eduardo Juárez, Matías Garrido, Ángel Manuel Groba, Pedro Lobo, thank you for having welcomed me.

I would like to especially thank Miguel for his extraordinary support from day one. I would not be able to come to Madrid and it would be very difficult for me to survive in Madrid without Miguel's support.

I express my very deep gratitude to my parents for providing me with constant support and continuous encouragement, especially to my mother and father, who always encouraged me 'you can do it'. They are my source of energy.

Now, I would like to thank Jaime, Pallab, Guille G., Gonzalo, Manu, Alberto, and Guille V. for so many good times. I also would like to thank the students who have completed their final-year project and master's thesis with me, and especially Victor, for your contribution to making this possible.

I want to express my heartfelt gratitude to Jaime's family: Pepi, Juan, Marisa, Paulina, and Aurora. They are my Spanish family who made my stay in Madrid as home.

I thank my friend Adnan for the motivation and support that even with distance makes you present, and also for all my friends that were not mentioned.

During my doctorate, I have had the opportunity to carry out an international research stay at the INESC-ID in IST Lisbon, and I would like to thank Nuno Roma for his willingness and availability to supervise me during the international stay. I learned a lot about GPU processing and I was able to excel in my research. I am also grateful to Tiago Dias for all those long discussions and for everything you have taught me. I must also thank Aleksandar Ilic for allowing me to participate in the "Parallel and Heterogeneous Computing Systems" course.

Lastly, I wish to express my deep and sincere gratitude for all the support I have received from my wife. Her patience, love, and care were essential to the development of this thesis.

Abstract

Nearly everybody in the modern era frequently uses video compression technology in the form of watching TV, under-demand content, video streaming, multimedia content on social networks, self-made videos, etc. Video compression technology is essential to transmit videos through the network and store them in an efficient way. The very first of video compression technology was introduced and patented by Ray Davis Kell in 1929. It consisted on store differences of frames as the variations in the inductance of transformer coils. However, researchers from industry and academia started working on standardising video compression technology from the 19th century. But, it was necessary to wait till 1984 for the standardization of the first video coding standard, H.120, proposed by the International Telegraph and Telephone Consultative Committee (CCITT). CCITT further developed the H.261 standard by cooperating with the International Telecommunication Union (ITU) in 1988. Afterwards, all the video coding standards followed the same fundamental video coding recommendations with a hybrid coding inner structure, where transform coding is combined with intra-prediction and, later on, inter-prediction. Since then, and almost on average every six years, a new standard was released due to the increased demand for higher quality videos and with the aim of achieving significant bitrate savings for comparable subjective video quality.

During the last decades, the demand for higher quality videos (full and ultra high definition) has increased rapidly due to the wide availability of consumer electronics devices. To meet this demand, the Joint Video Experts Team (JVET) released the Versatile Video Coding (VVC) standard in July 2020. VVC provides up to 50% bitrate savings compared to the previous High Efficiency Video Coding (HEVC) standard by keeping the same video quality. However, this reduction in bitrate causes a major increase in computational complexity, especially for real-time video processing on embedded devices with limited resources. To overcome this added computational complexity challenge for obtaining real-time performance, it is highly required to exploit the available parallelism either in software and hardware.

In this scenario, this doctoral research focused on the development of a hybrid approach to accelerate the newer VVC-based decoders over heterogeneous platforms using fine-grain parallelism, coarse-grain parallelism, and hardware accelerators. Here, data-level parallelism technique Single Instruction Multiple Data (SIMD), Central Processing Unit (CPU) + Graphics Processing Unit (GPU) based implementation were integrated with

native coding default features and coarse-grain parallelism.

First, a detailed analysis was performed by profiling the state-of-the-art VVC decoder to identify the most computationally demanding blocks. Thus, coarse-grain and fine-grain profile of the decoder were carried out. Moreover, parallel processing abilities of different VVC decoder blocks were analysed to apply optimisation techniques.

Secondly, different platforms and open source video decoders were chosen and migrated to the selected platforms. Then, SIMD optimisation was efficiently implemented on the latest version of the decoders on heterogeneous platforms. Therefore, a hybrid approach was implemented on heterogeneous platforms, both SIMD optimisation and heterogeneous CPU+GPU based implementation were used in parallel with native coding default features and coarse-grain parallelism. These processes were repeated for different versions of the decoders on different platforms. This work was verified by conducting a wide set of experiments with fifteen sequences with different configurations included in the common test conditions standard set.

Lastly, a design methodology was synthesised using the experience of previous optimization. This methodology proposes some recommendations to accelerate the optimization process of video decoders with different platforms. It has been used to optimize other decoder implementation (OpenVVC) with good results and reducing the development effort.

Keywords

Versatile Video Coding (VVC), Heterogeneous Platforms, Single Instruction Multiple Data (SIMD), Graphics Processing Unit (GPU), VVdeC, OpenVVC, Design Methodology, Multicore, Parallelism, Embedded platforms.

Resumen

En la actualidad los algoritmos de compresión de vídeo son empleados en multitud de aplicaciones de la vida cotidiana como son la televisión, el video bajo demanda, el *streaming* de vídeo, los contenidos multimedia existentes en las redes sociales, etc. En este marco la tecnología empleada para comprimir la información que tienen todas estas fuentes de vídeo de alta calidad es esencial ya que es necesario transmitirlos por redes de comunicaciones de ancho de banda limitado o almacenarlos en dispositivos de capacidad también limitada.

Los primeros avances en la compresión de vídeo fueron desarrollados y patentados por Ray Davis Kell en 1929. Estos consistían en almacenar como variaciones de la inductancia de bobinas las diferencias entre imágenes de una secuencia de vídeo. Sin embargo, fue necesario esperar hasta que en 1984 el *International Telegraph and Telephone Consultative Committee* (CCITT) propone el primer estándar de codificación denominado H.120. Posteriormente, en 1988 el CCITT en colaboración con la *International Telecommunication Union* (ITU) desarrollan el estándar H.261 que ya incluye una arquitectura híbrida que sigue estando presente en los estándares implementados en las siguientes tres décadas después.

Dicha arquitectura híbrida se basa en una predicción de las imágenes (*frames*) a partir de la información existentes en otras imágenes de la secuencia de vídeo (predicción inter) así como es la misma imagen (predicción intra). Desde entonces, y aproximadamente cada seis años, se han ido publicando sucesivos estándares para soportar el continuo incremento en la demanda de videos de mayor resolución espacial y temporal, que requieran un menor número de bits pasa su transmisión o almacenamiento manteniendo o mejorando en todo momento la calidad objetiva y subjetiva. Durante la última década, la demanda de videos de cada vez mayor calidad (*full high definition* y *ultra high definition*) se ha incrementado debido a la disponibilidad de dispositivos de electrónica de consumo que demandan este tipo de contenidos. Para atender esta demanda el *Joint Video Experts Team* (JVET) publica en julio de 2020 el estándar denominado *Versatile Video Coding* (VVC) que proporciona una reducción en la tasa de bits del vídeo codificado un 50% menor que su predecesor, el estándar *High Efficient Video Coding* (HEVC) manteniendo la misma calidad de vídeo. Sin embargo, esta reducción en la tasa de bits provoca un incremento en la complejidad en los algoritmos que es especialmente relevante para dispositivos que deben trabajar en tiempo real con unos recursos limitados, como es el caso

de los empleados para el desarrollo de esta tesis doctoral.

Para abordar este incremento en la complejidad del algoritmo y lograr el rendimiento necesario, es imprescindible explotar las posibilidades de paralelización tanto a nivel harware como software. En este escenario la presente tesis doctoral propone una arquitectura híbrida para acelerar implementaciones del estándar VVC sobre plataformas heterogéneas usando técnicas de paralelización de grado fino y grueso (*fine- and course-grain parallelism*) y aceleradores hardware. En concreto se han empleado técnicas basadas en el paralelisto a nivel de datos mediante el uso de instrucciones *Single Instruction Multiple Data* (SIMD) y arquitecturas que integran procesadores de propósito general (GPP) con procesadores que son capaces de realizar un paralelismo masivo de datos, como las unidades de procesamiento gráfico (GPU).

Para lograr estos objetivos, inicialmente se ha realizado un análisis de rendimiento del decodificador para identificar los bloques que demandan una mayor carga computacional. Este análisis se ha realizado desde un enfoque de los principales bloques del algoritmo, pero posteriormente se han analizado en detalle aquellos módulos que presentaban una mayor carga computacional y que son susceptibles de ser optimizados. Posteriormente se han evaluado diferentes plataformas hardware con diferentes arquitecturas y recursos para seleccionar las más adecuadas para estos objetivos. Paralelamente se han analizado los decodificadores de código abierto existentes en el mercado para tomarlos como punto de partida a la hora de realizar la optimización de sus prestaciones y aplicar sobre ellos las metodologías de optimización que se sintetizarían a partir de la experiencia acumulada durante la optimización de algunas partes del algoritmo.

Una vez elegidas las implementaciones a optimizar y las plataformas sobre las que llevar a cabo dichas optimizaciones se han aplicado técnicas que aprovechan el paralelismo de datos a nivel de instrucción (instrucciones SIMD) así como ejecución de parte de los algoritmos en aceleradores hardware para mejorar sus prestaciones. Este proceso se ha repetido para diferentes versiones de los decodificadores, diferentes módulos del algoritmo y con diferentes secuencias del estándar codificadas con distintos parámetros.

Finalmente, toda la experiencia acumulada tras este proceso ha permitido sintetizar una metodología de diseño que ha sido aplicada a otros módulos de otras implementaciones (como OpenVVC) obteniendo buenos resultados y reduciendo el tiempo requerido para realizar la optimización.

Palabras clave

Codificación de Vídeo Versátil (VVC), Plataformas Heterogéneas, Una Instrucción Múltiples Datos (SIMD), Unidad de Procesamiento Gráfico (GPU), VVdeC, OpenVVC, Metodología de diseño, Multinúcleo, Paralelismo, Plataformas Empotradas, Plataformas Embebidas.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

1D One-dimensional. 22, 63

2D Two-dimensional. 30, 36, 63

3D three-dimensional. 30, 32–34

AI All Intra. 48–52, 56–58, 72, 74–85, 89–92, 94–97

ALF Adaptive Loop Filter. 5, 6, 16, 22–25, 28, 48–53, 55–62, 64–68, 70, 80, 89, 97, 98, 102, 106, 107, 112–114, 116, 120, 121

ALU Arithmetic Logic Unit. 42

AMT Adaptive Multiple Transform. 6, 115

AMVR Advanced Motion Vector Resolution. 19

AoM Alliance for Open Media. 14

AOV ArenaOfValor. 72

API Application Programming Interface. 33, 42, 116

AV1 AoM Video 1. 14

AVC Advance Video Coding. 13, 14, 16

AVX Advanced Vector Extensions. 30, 54, 55

BBD BasketballDrive. 48, 72, 75, 77

BDOF Bi-directional Optical Flow. 19

BQT BQTerrace. 48, 72, 75, 77

BS Boundary Strength. 20

CABAC Context Adaptive Binary Arithmetic Coding. 15, 16, 26, 32

Cb chroma blue. 25, 66

CCALF Cross-component ALF Filtering. 25, 66

CCITT International Telegraph and Telephone Consultative Committee. 13

CCT Cactus. 72

CHCF Chroma Component Filtering. 51, 52

CHF Chroma Filter. 50–52

CITSEM Software Technologies and Multimedia Systems for Sustainability. 2, 112, 115

CMF Campfire. 48, 72, 75, 77

CPLB Calculate Position and Length of the Boundaries. 50–52

CPU Central Processing Unit. 6, 7, 28, 32, 33, 35–37, 39, 40, 43, 60–71, 89–91, 94, 95, 97, 106–108, 112–116, 119–122

Cr chroma red. 25, 66

CR1 CatRobot1. 72

CRCF Cross-Component Filtering. 51, 52

CRY Copy Reconstructed YUV. 51, 52

CSP Chroma Scaling. 20

CTU Coding Tree Unit. 15, 22, 27, 28, 31, 32, 35, 61, 63, 64, 66

CU Coding Unit. 15, 18, 19, 28, 36

CUDA Compute Unified Device Architecture. 33, 34, 42, 65, 107, 108, 116

DBF Deblocking Filter. 5, 16, 20, 22, 28, 37, 48–52, 55–58, 80, 97, 113, 123

DCT Discrete Cosine Transform. 17, 35, 36, 59

DeC Derivative Classification. 51, 52

DMS Diamond-shape. 24, 25, 60–63, 66

DMVR Decoder-side Motion Vector Refinement. 19, 50–52, 59

DR2 DaylightRoad2. 48, 72, 75, 77

DSP Digital Signal Processing. 2, 4, 35, 37, 122

DST Discrete Sine Transform. 17, 35, 36, 59

ED Entropy Decoder. 16, 48, 49, 55–58, 80, 81

EGPP Embedded General Purpose Processors. 44–46, 121

EGPU Embedded Graphics Processing Unit. 44, 45

EP Inter Prediction. 5, 16, 19, 20, 28, 48–53, 55–59, 80, 121

ETSIST Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación. 117

FD Filtering Decision. 50–52

FHD Full HD. 12, 36, 37, 73, 75, 82–84, 87, 88, 91–94, 96, 100, 122

FM4 FoodMarket4. 48, 72, 75, 77

FPGA Field Programmable Gate Arrays. 2, 4, 32, 34–37, 41, 42, 100, 101, 122

FPS frame per second. 12, 35–37, 74–79, 81–84, 87–91, 97

FRP FourPeople. 72

GDEM Electronic and Microelectronic Design Group. 2, 42, 112, 113, 115, 116, 123

GPP General Purpose Processors. 2–4, 30, 42, 46, 48, 53, 54, 59, 100, 105, 106, 111, 119

GPU Graphics Processing Unit. 2–4, 6, 7, 32–37, 39–42, 44–46, 60–72, 89–92, 94–97, 100, 105–108, 111–116, 119–123

HD High Definition. 1, 12, 73, 82–84, 87, 88, 91–94, 96, 97, 101, 122

HDL Hardware Description Language. 34

HEVC High Efficiency Video Coding. 1, 2, 13–16, 18–20, 22, 31, 32, 36, 37, 53, 122, 123

HGPP High-performance General Purpose Processors. 5, 31, 42, 43, 46–48, 53, 73, 100, 113, 114, 121

HM HEVC Test Model. 26

IEC International Electrotechnical Commission. 13

IOB Input Output Block. 34

IP Intra Prediction. 16, 18, 19, 28, 48, 49, 55–59, 80, 113, 122

ISO International Organization for Standardization. 13

ITEX Inter Texture. 50–52

ITU International Telecommunication Union. 13

JNY Johnny. 72

JVET Joint Video Experts Team. 1, 13

JVT Joint Video Team. 13

KAS KristenAndSara. 72

LB Logic Block. 34

LCF Luma Component Filtering. 51, 52

LFNST Low-Frequency Non-Separable Transform. 17

LMCS Luma Mapping with Chroma Scaling. 16, 20, 28

LMP Luma Mapping. 20

LUF Luma Filter. 50–52

MP Motion Prediction. 19

MPEG Moving Picture Experts Group. 13

MPL MarketPlace. 72

MSE Mean-Square Error. 40, 41, 109

MTS Multi Transform Selection. 17, 35

MV Motion Vector. 19

MVD Motion Vector Differences. 19

Nano NVIDIA Jetson Nano development kit. 5, 45–47, 53, 59, 69, 72–74, 82–84, 86–88, 91–95, 97, 101, 114, 116

OT other. 48–52, 57, 58, 80

PR3 ParkRunning3. 48, 72, 75, 77

PSNR Peak Signal-to-Noise Ratio. 40, 41, 109

QHD Quad HD. 12

QP Quantization Parameter. 17, 21, 48, 49, 56, 72, 74–77, 81–83, 87–92, 94, 95

RA Random Access. 48–52, 56–58, 72, 75–85, 87, 89–92, 94–97

RAM Random-Access Memory. 69

RUD RitualDance. 72

Ryzen AMD Ryzen Threadripper Processor. 5, 43, 47–51, 53, 73, 74, 113

SAO Sample Adaptative Offset. 16, 22, 24, 28, 37, 48, 49, 55–60, 66, 80

SD Standard Definition. 12

SIMD Single Instruction Multiple Data. 2–7, 27–30, 37, 39, 40, 53–60, 70–72, 74–84, 86, 88–91, 94, 96, 97, 111, 112, 114, 116, 119–123

SIMDe SIMDEverywhere. 54, 59, 60

SM Streaming Multiprocessor. 34, 44, 108

SMVD Symmetric Motion Vector Differences. 19

SoC System on Chips. 1, 34

SPUB Sub-Prediction Unit Bio. 50–52

SPUM Sub-Prediction Unit MC. 50–52

SSE Streaming SIMD Extensions. 30, 54

SSIM Structural Similarity Index Measure. 40, 41, 109

SSSE Supplemental Streaming SIMD Extensions. 30

TG2 Tango2. 72

TX Inverse Quantization and Inverse Transform. 16, 17, 28, 48, 49, 55–59, 80, 122

UDP Uni-Directional Prediction. 50–52

UHD Ultra HD. 12, 36, 37, 47, 48, 66, 73, 75, 82–85, 94, 96, 100, 122

UnOP Un-optimised. 81

UnVec Un-vectorization. 81

UPM Universidad Politécnica de Madrid. 2, 112, 113, 115–117

VBC Virtual Boundaries Check. 51, 52

VCEG Video Coding Experts Group. 13

VTM VVC Test Model. 4, 5, 26, 27, 47, 48, 50, 74, 112, 113

VVC Versatile Video Coding. 1, 3–7, 13–20, 22–28, 30, 31, 35–37, 47, 48, 52, 60, 71, 77, 101, 111–113, 115, 119–123

VVdeC Versatile Video Decoder. 4–7, 26, 27, 47, 48, 53–57, 59–62, 64–68, 70, 71, 74–77, 79–83, 86, 88–90, 92–97, 106, 107, 112, 114, 116, 120, 121

WP Weighted Prediction. 50–52

WPP Wavefront Parallel Processing. 32, 36

X-series Intel core i9-10900x Processor. 43, 44, 47, 53, 55–58, 72, 73, 75–78, 80, 114

Xavier NVIDIA Jetson Xavier development kit. 5, 44–51, 53, 55–60, 69, 72–75, 77, 78, 80–97, 101, 113–116

ZMV Zero Motion Vectors. 19

Chapter 1

Introduction

This chapter presents the motivation for the doctoral study in Section 1.1, the main research objectives of the thesis is to define new design methodologies to implement efficient video decoders on heterogeneous platforms in Section 1.2, the methodology that has been applied during the development of the thesis work in Section 1.3, the list of the main contributions of the thesis in Section 1.4, and the outline of the thesis document in Section 1.5.

1.1 Motivation

The world has entered in a new phase of information and communication technology, in which video communication has become an essential part of everyday life. The demand for full High Definition (HD) videos, with a resolution of 1920×1080 pixels, and 4k ultra HD, with 3840×2160 pixels, is growing rapidly. Notably, it is estimated that video contents will represent the 82% [1] of global internet traffic by end of 2022. The transmission of high-resolution videos requires additional bandwidth and storage facilities. However, the capacities of storage facilities and communication channels are limited, particularly for a resource-constrained embedded System on Chips (SoC). In this situation, it has become necessary to develop better video coding technology for achieving a higher video compression rate.

To handle the situation, the Joint Video Experts Team (JVET) released the current state-of-the-art Versatile Video Coding (VVC) standard [2] in July 2020. VVC offers the same video quality by allowing a bitrate saving of 50% compared with the previous High Efficiency Video Coding (HEVC) standard [3]. However, this reduced bitrate results in increased computational complexity of 10× for the encoder and 2× for the decoder

with respect to HEVC [4]. Therefore, it has become challenging to achieve real-time video encoding/decoding with single core implementation due to the higher computational complexity, especially for embedded systems.

Video compression/decompression algorithms are commonly accelerated and optimised by using a variety of parallelisation techniques. In particular, multithreading and Single Instruction Multiple Data (SIMD) intrinsics-based parallelisation are the two most used parallelisation approaches in recent video coding applications [5], [6], [7]. However, these parallelisation approaches are restricted to General Purpose Processors (GPP) and cannot take straightforward advantage of hardware accelerators or Graphics Processing Unit (GPU), which are integrated into contemporary heterogeneous system-based platforms.

Nowadays, GPUs have become an essential coprocessing unit that helps GPPs to handle computationally heavy operations such as machine learning, general graphics acceleration, image processing, etc. In brief, GPUs are powerful accelerators that are specially designed to optimise throughput [8]. Although modern GPPs have multicore processing capacity for parallel processing, GPU provides better parallelisation for algorithms with high computational demands and simple repetitive operations. However, the main challenge lies in the use of GPUs in conjunction with GPP multithreading to maximise the use of all the resources of heterogeneous system-based platforms, while avoiding undesired bottlenecks in the data path of the architecture.

In a context in which both video features and processor architectures evolve rapidly, there is a clear need for methodologies that facilitate the development of video solutions on heterogeneous architectures. For almost two decades now, researchers at Electronic and Microelectronic Design Group (GDEM), founding member of the research center Software Technologies and Multimedia Systems for Sustainability (CITSEM), integrated in Universidad Politécnica de Madrid (UPM), have been working on video decoder optimization methodologies for different types of architectures such as GPUs, GPPs, Digital Signal Processing (DSP)s [9], and Field Programmable Gate Arrays (FPGA) [10]. In addition, the GDEM participated in the IVME [11], MR-UHDTV [12], and H2B2VS [13] research projects, among others, aligned with the research of architectures for video encoding/decoding towards methodologies for the implementation of video decoders with multicore processors.

This doctoral thesis has been carried out in the GDEM research group to address the state-of-the-art video decoding algorithm using collectively the parallelism techniques mentioned above over state-of-the-art heterogeneous platforms.

1.2 Research objectives

The main aim of this doctoral thesis is to define new design methodologies to implement efficient video decoders on heterogeneous platforms. This research work identifies the key elements of the VVC decoders that are computationally intensive. Therefore, redesign the algorithm for parallel processing of videos by exploiting the multicore features of the GPP, SIMD optimisation, and GPU accelerator simultaneously.

In this way, the side objectives of this thesis are defined:

- To carry out profiling and performance analysis of the video decoders to identify computationally intensive blocks of VVC decoders.
- To analyse the parallel processing abilities of the VVC decoder blocks by tracing data and process dependencies.
- To establish a design methodology to accelerate the applications of video decoding over heterogeneous platforms.
- To parallelise the video decoding algorithm of a VVC based decoder with the highest computational demands and parallel processing capability using the multicore feature of the GPP, SIMD optimisation and GPU accelerators.
- To conduct an analysis of energy consumption and memory usage over embedded heterogeneous platforms. These are two of the most important factors for resource-constrained embedded platforms.
- To compare the proposed solutions by means of the decoding performance, energy consumption, and memory usage of different VVC decoders on different heterogeneous platforms.

1.3 Work methodology

The methodology that has been applied during the development of the thesis work is divided into three main stages in which the following activities have been carried out:

1) Study of the state-of-the-art on the implementation of video processing algorithms over heterogeneous platforms. A detailed study has been carried out about other contributions published in the scientific literature related to video coding and contemporary heterogeneous platforms mainly focusing on GPU-based technologies. This study has not

been limited to video coding applications, nor to systems based on multicore processors, and has been generalised to other types of application (image depth estimation or point cloud generations, among others) and using other platforms (FPGA, DSP, etc.) (see Section 3.3) to have a more general vision regarding the implementation of algorithms based on heterogeneous systems. In addition, a thorough analysis has been performed on different heterogeneous platforms available in the market for video and image processing applications. Here, GPU-based heterogeneous platforms were chosen as it has been found more appropriate for this doctoral thesis (see Section 4.2).

2) Implementation of VVC video decoder solutions over heterogeneous processing systems. The following has been done: firstly, the reference software has been used to have a reference against which the other implementations could be compared. The VVC Test Model (VTM) [1] is the reference software [14] for VVC which was under development during the first half part of this doctoral work[2]. Later, Versatile Video Decoder (VVdeC) [15] software solution was selected and a detailed analysis of its building blocks was carried out. VVdeC is an open source and the most performance-efficient VVC software decoder developed on top of VTM. The source code of these solutions have been modified as follows: 1) The compatibility of the selected video decoders with the selected heterogeneous platforms has been ensured (see Section 4.5.1). 2) The decoding performance of the selected video decoders have been improved by using SIMD vectorised operations (see Section 4.5). 3) The decoding speed has been accelerated using the load-sharing approach between GPP and GPU (see Section 4.6). Finally, a methodology has been developed to calculate the memory usage (see Section 4.7.1) and energy consumption (see Section 4.8) of the VVC decoders.

3) Synthesis of the methodology. The experience gained implementing VVC video decoders over two heterogeneous platforms has made the implementation generalise for different heterogeneous platforms. Lastly, methodological conclusions have been drawn that have made it possible to define a design methodology; it eases the development of these applications and helps make the implementation more flexible for different VVC decoders.

These three phases have been developed as follows: the main effort of the first phase was made at the beginning of the thesis, although it was necessary to continue to update the information collected with the new contributions that have continued to be published throughout the development of the thesis. The second and third phases have run in

[1]VTM is a reference software of VVC (Rec. ITU-T H.266 | ISO/IEC 23090-3). It is an open source project which has both encoding and decoding functionality.

[2]The choice of VTM as the first software solution is also justified as this doctoral work began in parallel with the development of the new standard.

parallel and have fed each other for much of the development of the thesis.

1.4 Main contributions

Following is a list of the main contributions of the thesis, which can be indicated with the objectives of the doctoral work:

The VTM8.0 based VVC decoder was profiled in detail on the HGPP-based Ryzen and resource-constrained embedded Xavier platform. In addition, fine-grain profiling was carried out for the Inter Prediction (EP), Deblocking Filter (DBF) and Adaptive Loop Filter (ALF) blocks. This result of the coarse and the fine-grain profile was the fundamental need to target the VVC modules for acceleration. The following publication presents these results:

- A. Saha, M. Chavarrías, F. Pescador, Á.M. Groba, K. Chassaigne, P.L. Cebrián, "Complexity Analysis of a Versatile Video Coding Decoder over Embedded Systems and General Purpose Processors," Sensors 2021, 21, 3320. https://doi.org/10.3390/s21103320. [16].

Different video decoders (VVdeC and OpenVVC) were accelerated using SIMD optimisation based on Neon for ARM-based embedded platforms Xavier and Nano. It includes Neon-based SIMD optimisation technique for VVdeC and OpenVVC decoders. In addition, memory usage and energy consumption analysis was performed for the VVdeC and OpenVVC decoders. Lastly, a comparison study was carried out between the VVdeC and OpenVVC decoders by means of decoding performance, energy consumption, and memory usage. This work result in two contributions; the first one relates to the Neon-based SIMD optimisation of the VVdeC decoder and the second one relates Neon-based SIMD optimisation of OpenVVC (currently under review process).

- A. Saha, M. Chavarrías, V. Aranda, M. J. Garrido and F. Pescador, "Implementation of a Real-time Versatile Video Coding Decoder based on VVdeC over an Embedded Multi-core Platform," in IEEE Transactions on Consumer Electronics, 2022, doi: 10.1109/TCE.2022.3202512. [17].

- A. Saha, W. Hamidouche, M. Chavarrías, G. Gautier, F. Pescador, and I. Farhat, "Performance Analysis of Optimized Versatile Video Coding Software Decoders on Embedded Platforms" [Online]. Available:`https://arxiv.org/abs/2206.15311`. (currently under review process) [18].

Finally, a methodology was developed to accelerate the VVdeC decoder using a hybrid approach based on Central Processing Unit (CPU)+GPU. Here, VVdeC ALF was accelerated using GPU and the rest of the decoder modules were accelerated using SIMD optimisation in CPU. In addition, energy consumption analysis was conducted, where almost the same energy per frame was achieved with higher performance. This final synthesis work is published in the following contribution (currently under review process).

- A. Saha, N. Roma, M. Chavarrías, T. Dias, F. Pescador and V. Aranda, "GPU-based Parallelisation of a Versatile Video Coding Adaptive Loop Filter in Resource-Constrained Heterogeneous Embedded Platform,", DOI: 10.21203/rs.3.rs-2381512/v1. (currently under review process) [19].

Partial VVC features were accelerated on top of OpenHEVC [20] over heterogeneous platforms using. Some new Adaptive Multiple Transform (AMT) features of VVC in that time were studied and accelerated using GPU and CPU-based parallelisation. The results of these works were presented at the following international conferences.

- R. Medina, A. Saha, M. Floriano, M. Chavarrías and F. Pescador, "Porting Adaptive Multiple Transforms of a Versatile Video Coding decoder using OpenMP," 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), 2019, pp. 138-139, doi: 10.1109/ICCE-Berlin47944.2019.8966176. [21].

- M. F. Vázquez, A. Saha, R. M. Morillas, M. C. Lapastora and F. P. d. Oso, "Work-in-Progress: Porting new Versatile Video Coding transforms to a heterogeneous GPU-based technology," 2019 International Conference on Compliers, Architectures and Synthesis for Embedded Systems (CASES), 2019, pp. 1-2. [22].

1.5 Thesis outline

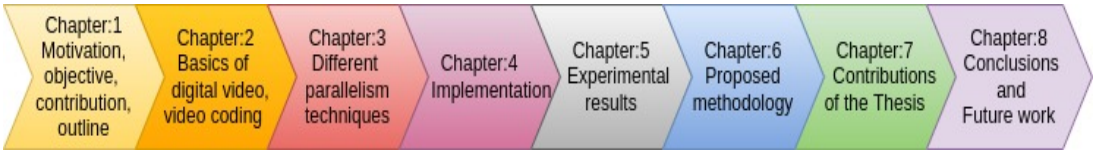


Figure 1.1. Thesis outline.

This dissertation contains eight chapters which topics are outlined (see Figure 1.1) as follows:

- **Chapter 1:** presents the motivation of the Thesis topic, outlines the research objectives, describes the methodology followed during the development of the work, and lists the main contributions of the thesis in terms of scientific publications.

- **Chapter 2:** introduces the basics of digital video, a brief historical review of video coding standards, and a brief overview of the Versatile Video Coding standard encoder and decoder. In addition, a brief overview of different VVC decoders is presented.

- **Chapter 3:** provides a brief overview of different parallelism techniques used to optimise video codecs, including fine-grain and coarse-grain parallelism. Moreover, a brief overview of different hardware accelerators for video coding tools is provided. Lastly, a brief discussion of the state-of-the-art work on different hardwares for video algorithms is presented.

- **Chapter 4:** provides a detailed description of the implementation process, including the working methodology, the use of memory, the consumption of energy, and the summary of the proposed methodology. Furthermore, a brief description of the selection of platforms, the selection of the video algorithm, the profiling and evaluation of the video algorithm, the open source decoder optimisation for ARM-based platforms, algorithm redesigning for CPU+GPU based implementation, and generalisation of the proposed implementation are included in the working methodology.

- **Chapter 5:** presents a detailed description of the obtained experimental results including performance analysis, memory usage, and energy consumption. In addition, the test bench description, the setup of the platform, and a discussion of the experimental results are included. The results contain the native GCC auto vectorizer based performance analysis, the performance results without/with SIMD optimisation, the speedup analysis based on SIMD optimisation, the CPU+GPU implementation, and memory usage, and energy consumption of the VVdeC decoder. Furthermore, the results of the OpenVVC decoder for the speedup analysis based on SIMD optimisation, the analysis of memory usage, and the analysis of energy consumption are also presented. Lastly, comparison analyses between VVdeC and OpenVVC are provided for performance, memory usage, and energy consumption.

- **Chapter 6:** provides a methodology that should be followed for future problems to address the increased complexity and performance requirements of state-of-the-art video decoding software.

- **Chapter 7:** discusses the results and contributions of the Thesis. Furthermore, a brief description is provided for research publications, research projects, international collaborations, direction of final degree projects, and scholarships obtained during the development of the doctoral Thesis.

- **Chapter 8:** highlights the main conclusions of this doctoral Thesis and offers some suggestions for future research.

Chapter 2

Background

This chapter contains background information on digital video and common terminologies in video coding (see Section 2.1). In addition, a historical overview of video coding standards is presented in Section 2.2. Therefore, a brief overview of the VVC encoder and decoder is given in Sections 2.3 and 2.4, respectively. Lastly, different open source VVC complaint decoders are presented briefly in Section 2.5.

2.1 Digital video concepts

Digital video revolutionised information and telecommunication technology by representing and storing moving visual images in digital binary format. In 1986, the first commercial uncompressed video was recorded in digital form [23]. On the contrary, in analog video, moving visual images are typically captured in the form of an analog signal stored in a film. Several sequential analog images are put together in analog videos. In digital video, images are divided into several horizontal lines or blocks inside the frame. Further, horizontal lines are divided into several dots named pixels. Pixels are represented by intensity and colour in digital video. Colour spaces and some other topics widely used in digital video are briefly discussed below for better understanding the digital video as this research is based on digital video.

2.1.1 Colour spaces

Images are mainly two types: 1) black and white or grayscale, and 2) colour. Grayscale images are represented by one numeric value for each pixel. These numeric values are the brightness of the image. On the other hand, pixels in colour images are represented by

numeric values indicated by the colour space. Colour space is a mathematical diagram that contains a variety of colours produced by combining basic colours. Commonly used colour spaces are RGB and YUV.

2.1.2 RGB colour space

RGB colour space is based on chromaticity consisting of red, green, and blue components. These three colours are combined to represent colours that range from fully white to fully black (see Figure 2.1). The basic RGB colour space contains a total of 24 bits, with 8 bits each for red, green, and blue colours. It is called RGB24 and it supports 16.7 million colour combinations. [1]

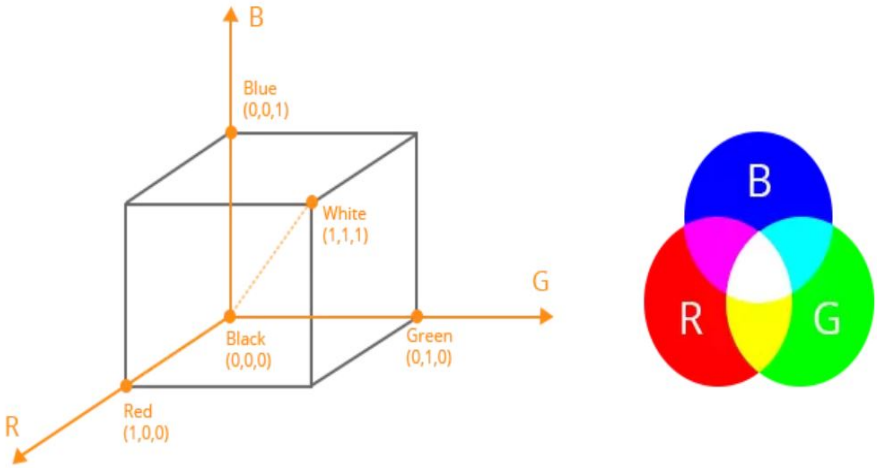


Figure 2.1. RGB color space (source:[24]).

2.1.3 YCbCr colour space

YCbCr colour spaces are designed by considering the human perception. It contains one luminance component Y and two chrominance components Cb (blue) and Cr (red). The luminance component controls the brightness, and the chrominance component controls the colour information of the digital images and videos. This colour space is widely used for compressing digital image and video. Human eyes are more sensitive to brightness than to colour information. Video compression using the YCbCr colour space benefited the human visual system by allowing independently compress of luminance and colour components. Therefore, different formations of the YCbCr colour space are used to reduce the storing and communication channel requirement for video processing and

[1] There are other formats of RGB colour space but the common one is presented.

transportation by removing less important information from the video samples. Figure 2.2[2] presents 4:4:4, 4:2:2 and 4:2:0 video sample where the luminance components are kept unchanged for all formats. In Figure 2.2a, every 4 luma Y pixel has 4 Cb and 4 Cr pixels. In Figure 2.2b, every 4 luma Y pixels has 2 Cb and 2 Cr pixels. For the first and second row two chroma pixels are taken, each in the dotted box. In Figure 2.2c, every 4 luma Y pixels has 1 Cb and 1 Cr pixel. The first two chroma pixels are taken in the dotted box.

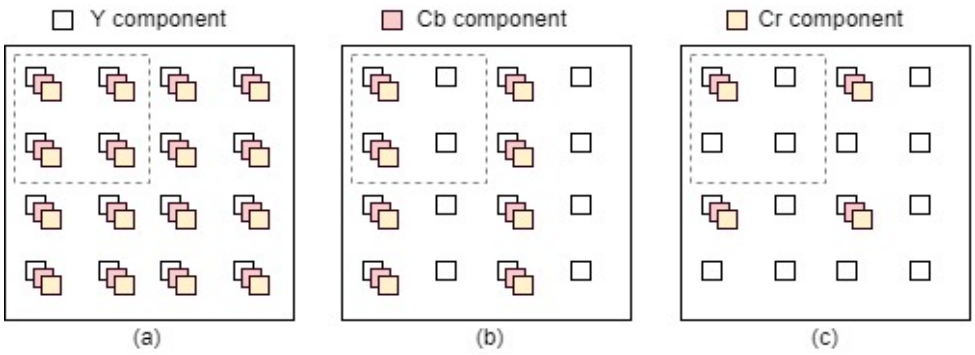


Figure 2.2. YCbCr color space.

2.1.4 RGB – YCbCr colour conversion

Conversion between RGB colour space to YCbCr colour is an essential operation in the image and video processing application. The reason is that the YCbCr colour space is widely used for image and video compression to reduce the storing space requirements, while most capturing devices store images and video in the RGB colour space. YCbCr colour space can be obtained from the RGB colour space using equation 2.1. Here, the luminance component Y is calculated by adding the multiplication of the weight factors K_r, K_g, and K_b with the red (R), green (G), and blue (B) components, respectively. Furthermore, chrominance red Cr and blue Cb are calculated using equations 2.2 and 2.3, respectively.

$$Y = K_r R + K_g G + K_b B \tag{2.1}$$

$$C_r = R - Y \tag{2.2}$$

$$C_b = B - Y \tag{2.3}$$

[2]There are other formats of YCbCr colour Space. However, the most widely used formats are presented.

2.1.5 Frame rate

Frame rate [25] is the frequency measured encoding and decoding images in frames per second. The frame rate is also used to the video camera, other capturing devices, and display devices. For instance, 60 frame per second (FPS) are maintained for television broadcast [26]. That means that 60 consecutive images are displayed in a second.

2.1.6 Video resolution

Video Resolution [27] can be expressed as an organisation of the number of pixels by width and height in a frame of the video (see 2.4). Commonly used video resolutions are Standard Definition (SD) with 640×480 pixels and 640×360 pixels, HD with 1280×720 pixels, Full HD (FHD) with 1920×1080 pixels, Quad HD (QHD) with 2560×1440 pixels, 2K with 2048×1080 pixels, 4K Ultra HD (UHD) with 3840×2160 pixels, and 8K Full UHD with 7680×4320 pixels.

$$Video\_Resolution = Pixel\_width \ * \ Pixel\_height \tag{2.4}$$

With the aim to not overextend this dissertation, in this section only the general and main basic concepts about digital video are presented. For a more detailed summary of this topic, the following reference is recommended [28].

2.2 Video coding standards

Video codecs provide a technological solution for storing and transmitting digital video. They reduce the size of raw video files by using different techniques which may introduce or not losses and or remove redundant information. The basic video codec contains the encoder and decoder, as shown in Figure 2.3. Video coding standards provide a set of specific coding tools for optimising coding efficiency targeting bit-rate savings.

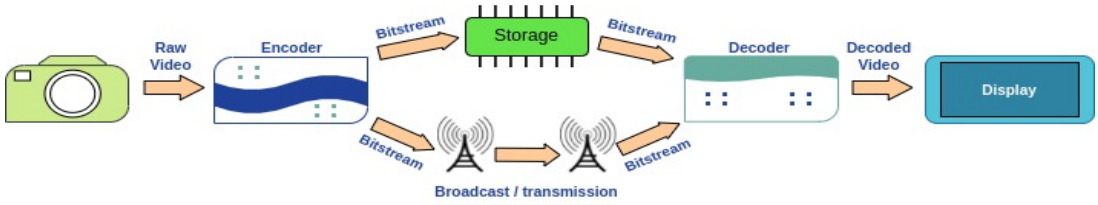


Figure 2.3. Typical video coding/decoding bitstream flow.

2.2.1 Brief historical review of video coding standards

First video coding standard H.120 [29] was introduced in 1984 by the International Telegraph and Telephone Consultative Committee (CCITT). Therefore, CCITT have collaborated with International Telecommunication Union (ITU) and created the ITU-T group where Video Coding Experts Group (VCEG) of ITU-T manages the video coding standard. In 1988, the VCEG group within ITU-T released the H.261 [30] standard. H.261 included the classical hybrid video coding scheme, transform coding, spatial-prediction, and temporal-prediction. The brief historical evolution of video compression standards is presented in Figure 2.4.

In 1993, the Moving Picture Experts Group (MPEG)-1 [31] video coding standard was introduced by MPEG. MPEG group was a result of the collaboration of two standard organisations: International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). Therefore, the VCEG group within ITU-T and ISO/IEC jointly released H.262/MPEG-2 [32] video coding standard in 1995. Furthermore, the Joint Video Team (JVT) was created by the VCEG and MPEG groups. In 2003, the JVT introduced the widely used video standard H.264/MPEG-4 also called Advance Video Coding (AVC) [33]. The continuation of this collaboration was successful, and they released the H.265/HEVC [3] standard in 2013. The Joint Video Experts Team (JVET) was then formed between VCEG (Q6/16) and ISO/IEC JTC1 SC29's MPEG in 2017 who standardised the latest H.266/VVC video coding standard in June 2020.

On the other hand, the first industrial video coding standard TrueMotion S was released by On2 Technologies in 1995. Therefore, TrueMotion RT and TrueMotion 2 [34] were released in 1996 and 1997, respectively, by the On2 technologies company. TrueMotion coding standards support processing of 2 dimensional images from 3 dimensional models. All industrial video standards are presented on the right side of the Figure 2.4. Further, On2 technologies introduced the royalty-free VP3 [35] video standard that concentrated on natural scenes images in 2000. On2 Technologies continued the release of the VP4, VP5, VP6, and VP7 video standard until 2008, before it was acquired by Google in

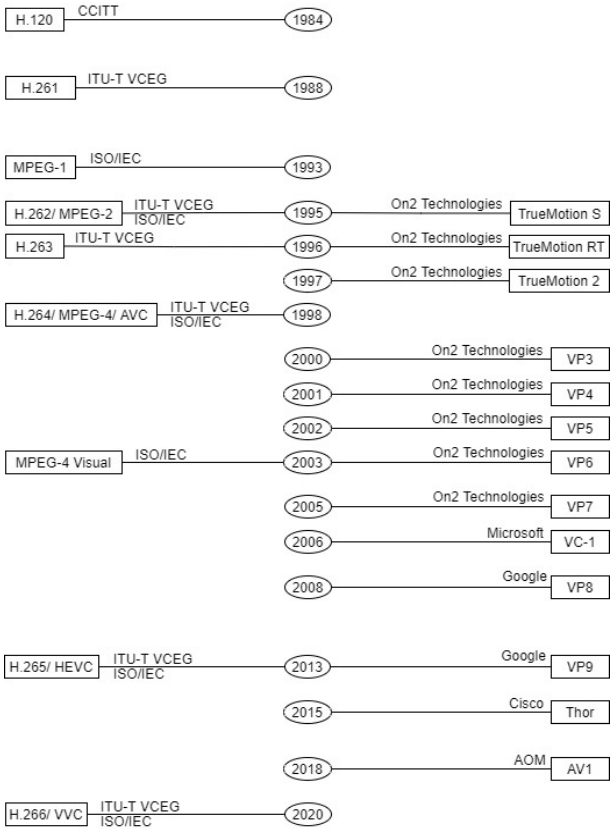


Figure 2.4. Chronological review of video compression standards.

2008. After that, Google released the open source video standard VP8 in 2008. VP8 offered features similar to H.264/AVC. Therefore, Google released the VP9 video standard in 2013. Like VP8, VP9 provided similar features of HEVC. VP9 is also used in Youtube videos. Besides, Microsoft developed the VC-1 video standard in 2006 and Cisco developed the Thor [36] video standard in 2015. In addition, Alliance for Open Media (AoM) was formed by Google, Netflix, Facebook, Amazon, Microsoft, Apple, ARM, Nvidia, Intel Corporation, Cisco, IBM, and Mozilla to develop the royalty-free AoM Video 1 (AV1) [37] video standard in 2018. AV1 is the successor of VP9 and originally focused on internet video transmissions.

In this document, special emphasis will be placed on the contextualisation of the two most current developments of the H.26x line, reaching H.266, on which the research of this Thesis has been based. In this order, first comes the H.265 or High Efficiency Video Coding standard, which introduces improvements, generally achieving a 50% higher compression rate than its predecessor, AVC/H.264. And secondly, the VVC/H.266 standard is presented as a solution to the growth forecasts of the digital video sector today.

Structurally, all coding standards since the pioneer H.261 in 1990 are based on the same

general principle known as the hybrid video coding scheme [38]. The term hybrid refers to the combination of two tools with the aim of reducing redundancy in information: the first is prediction, and the second is transformation with quantification of the prediction residue. The general scheme of VVC standards is shown in Figure 2.5, which is based on the hybrid video coding scheme.

2.3 Overview of Versatile Video Coding standard encoder

VVC has the similar basic structure as like HEVC standard. The simplified blocks diagram of VVC coder is displayed in Figure 2.5. The input of VVC encoder is the raw video signal and the output is the encoded bitstream. In addition, VVC encoder contains 1) picture partitioning, 2) Intra-Picture Prediction, 3) Inter-Picture Prediction, 4) transform/quantization, 5) in-loop filters, and 6) entropy coding.

In video encoding process of VVC standard, input video is divided into square shaped Coding Tree Unit (CTU)s. VVC supports up to 128×128 pixels CTU size, which is double than HEVC supported maximum partition size 64×64 pixels. CTUs are further divided into coding units (Coding Unit (CU)s) of different sizes. VVC supports 128×128 maximum CU size with square and non-square shaped, whereas HEVC supports only square shaped CU with maximum size 64×64. In VVC, the intra-picture prediction, inter-picture prediction, and transform/quantization are performed at CU level. Here, the intra-picture prediction predicts next block of data using the current block in the same frame. The inter-picture prediction predicts next frames from the current frame. Therefore, the integer transform is applied to the prediction residual and then quantization is applied. Then in-loop filters correct the block artifacts generated by picture partitioning and quantization. Lastly, compressed bitstream is produced by Entropy coding using Context Adaptive Binary Arithmetic Coding (CABAC) [39].

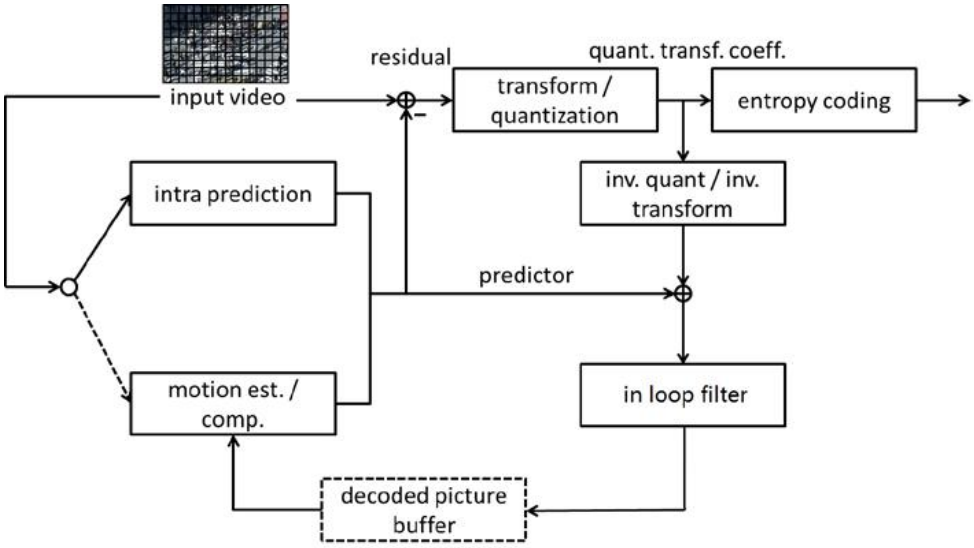


Figure 2.5. Simplified blocks diagram of a VVC encoder (source:[40]).

2.4 Overview of Versatile Video Coding standard decoder

Same as the AVC and HEVC standard, the hybrid coding scheme is the basis of the VVC standard, where transform, intra prediction and inter prediction coding are exploited. The simplified block diagram of a VVC decoder is shown in Figure 2.6, where the encoded bitstream is the input and the decoded video is the output of the processing chain. At the beginning of the decoding process, the bitstream is decoded by Entropy Decoder (ED), where CABAC is exploited to produce all essential information for video decompression. CABAC uses coded residual data, intra, and inter prediction to provide block partitioning information. Therefore, Inverse Quantization and Inverse Transform (TX) are used to recreate the coded residual data. Then, the prediction pixels of Intra Prediction (IP) or Inter Prediction (EP) are combined with these reconstructed residual data. Therefore, these cumulative predicted and residual data are filtered to obtain decoded video using four in-loop filters: Luma Mapping with Chroma Scaling (LMCS), Deblocking Filter (DBF), Sample Adaptative Offset (SAO), and Adaptive Loop Filter (ALF).

2.4.1 Entropy decoder

VVC adopted similar Entropy Decoder (ED) with a more powerful CABAC engine than HEVC standard. Probability is linearly expressed here using the current condition

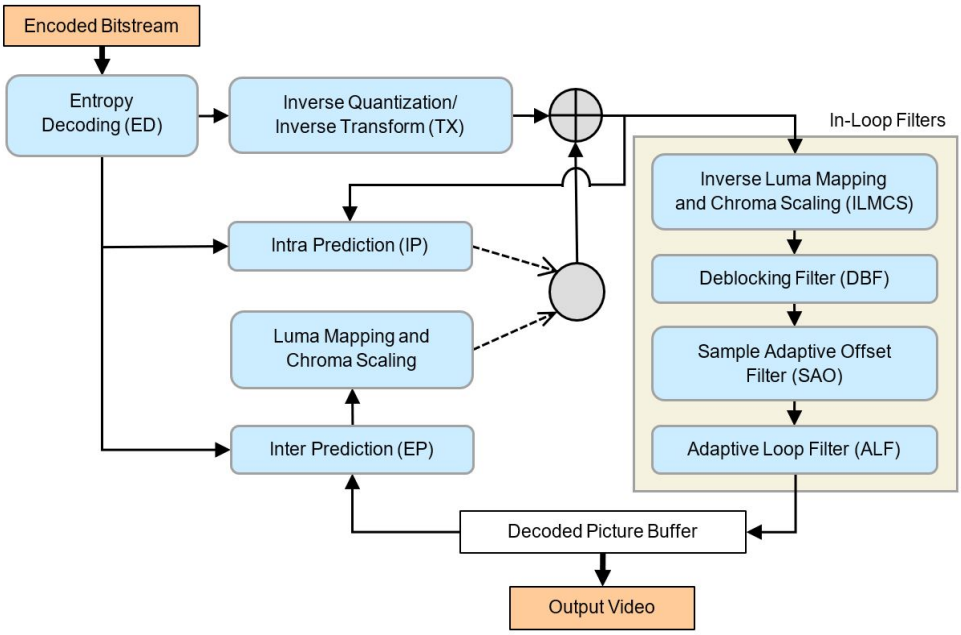


Figure 2.6. Simplified blocks diagram of a VVC decoder.

index. In order to improve accuracy, a modernised multihypothesis probability estimation method was used. In addition, the computed look-up table was discarded. A Quantization Parameter (QP)-dependent initialisation paradigm was also introduced in the VVC CABAC. Here, 6-bit precision was achieved on total initialisation values. Furthermore, 1×16, 2×8, 8×2, 2×4, 4×2, and 16×1 coefficients group sizes were added for the transformation block size to optimise the coefficient coding [40].

2.4.2 Inverse quantization and inverse transform

Inverse Quantization and Inverse Transform (TX) are implemented to extract the spatial domain coefficients from the frequency domain. Multi Transform Selection (MTS) [41], a novel tool introduced in VVC, is used to encode the residual inter- and intra-coding blocks. Furthermore, the transformations of Discrete Cosine Transform (DCT)-II, DCT-VIII, and Discrete Sine Transform (DST)-VII are supported by MTS. Here, the permitted transformations of rectangular blocks with height and width are $\leq$ 32 for DCT-VIII and DST-VII, and $\leq$ 64 for DCT-II. The high frequency coefficients are zeroed when the allowed height and width is in the highest point (e.g., 64×64 for DCT-II) to minimise computational complexity. Afterwards, for additional signal decorrelation, Low-Frequency Non-Separable Transform (LFNST) [42] is applied to the low frequency

transform coefficients, where these coefficients derive from directional intra prediction.

2.4.3 Intra Prediction

Intra Prediction (IP) in VVC supports sixty-five directional intra predictions (see Figure 2.7), one planar and one DC modes. Compared to the previous HEVC standard, VVC added thirty-two new directional intra predictions modes. Moreover, in VVC IP, several regular angular modes were removed and wide-angle intra modes were added. The precision of the prediction was increased by extending the list of candidates for the most probable modes to six. Furthermore, VVC included the following IP coding tools: 1) The matrix IP tool used the left and above lines of the reconstructed neighbouring components. These two blocks were taken as input vectors. Therefore, linear interpolation was performed in the vertical and horizontal directions after some pre-processing. 2) Intra sub-partitions tool was used for IP the block by processing sub-partitions gradually. Here, the luma coding block was divided into two sub-partitions vertically or four sub-partitions horizontally. 3) Multiple reference line tool was adapted in which the neighbouring lines 1 and 3 of the prediction block were referenced for angular prediction (see Figure 2.8) [40]. 4) The cross-component linear model tool [43] was introduced to predict the chroma components from the luma components in the same CU. 5) The mode-dependent intra-smoothing tool applied a four-tap intra filter for improving the prediction.

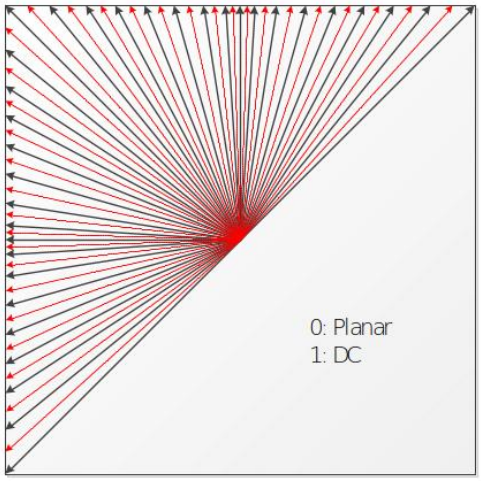


Figure 2.7. Representation of the 67 intra prediction modes (source:[40]).

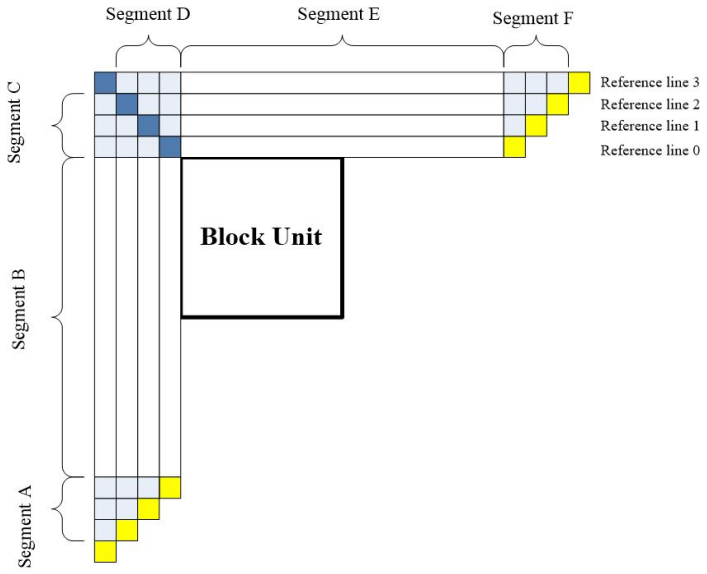


Figure 2.8. Example of four reference lines neighboring to a prediction block (source:[40]).

2.4.4 Inter prediction

In VVC Inter Prediction (EP), the merge mode was adapted along Motion Vector Differences (MVD) which improves the representation of Advanced Motion Vector Resolution (AMVR) [44] and the motion parameters collected from adjacent CUs. The merge mode in VVC is equivalent to the previous HEVC standard, which accepts candidates from temporal, spatial, and Zero Motion Vectors (ZMV). VVC EP performs Motion Prediction (MP) at the sub-CU level, which improves the prediction precision compared to the HEVC standard. Furthermore, motion-compensated prediction was obtained by applying 8-tap filters to the luma components, while 4-tap filters were applied to the chroma components to interpolate [45]. In addition, VVC includes these coding tools: (1) Symmetric Motion Vector Differences (SMVD) derived the MVD of the reference picture list 1 from the list 0 at the decoder. (2) Bi-directional Optical Flow (BDOF) [46] was applied at the pixel level, where it may be applied on top of the extended bi-prediction mode. (3) Combined EP and IP enhanced the intra mode in the inter-pictures. It combined the decided intra mode with an extra merge indexed prediction [47]. (4) Decoder-side Motion Vector Refinement (DMVR) [48] (see Figure 2.9) was applied around the initial motion vectors of the list of reference pictures L0 and L1 to improve the accuracy of the merge mode Motion Vector (MV)s.

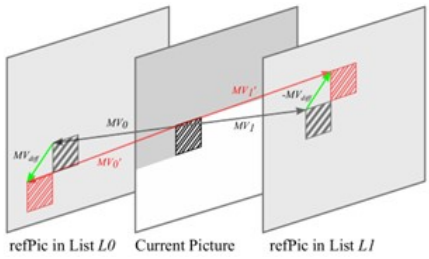


Figure 2.9. Decoding side motion vector refinement (source:[40]).

2.4.5 Luma mapping with chroma scaling

Luma Mapping with Chroma Scaling (LMCS) [49] is one of the novel tools presented by VVC. After EP, LMCS is applied in the decoding process of VVC. It was made up of Luma Mapping (LMP) and Chroma Scaling (CSP), where the predicted luma samples were processed by LMP and the chroma residues were processed by CSP. In principle, LMP maximises the utilisation of the available range of luma code values for a certain bit depth. For instance, for a 10-bit video with narrow range, the permitted luma code range is between 64 and 940, while the values of the 0 to 63 and 941 to 1023 luma code might be permitted to be used only in the coding process. In addition, LMP efficiently reallocates the luma code values in the coding domain. Furthermore, CSP is responsible for correcting the chroma residual samples with the error caused by the interaction between the luma and the chroma signals corresponding to the luma [50].

2.4.6 Deblocking filter

VVC used a similar but advanced Deblocking Filter (DBF) [51] compared to HEVC standard. The DBF in VVC is performed on the boundary aligned on 8×8 sample blocks, where the vertical edges are first filtered and then the horizontal edges are filtered. DBF was designed to improve subjective quality by reducing visible discontinuities at the block boundaries. The functionalities of DBF are briefly described as follows:

2.4.6.1 Boundary strength

Boundary Strength (BS) determined the type of components that will be filtered. There are three possible values of BS in VVC DBF: 0, 1 and 2. If the BS value is zero: DBF is not applied, if the BS value is one: only the luma components are filtered, and the BS value is two: the luma and chroma components are filtered. Furthermore, BS value is obtained from the coding mode and parameter of the boundary of the transformation

unit and the prediction unit. Figure 2.10 shows the boundary segment blocks P and Q of the four samples for lines 0 to 3.

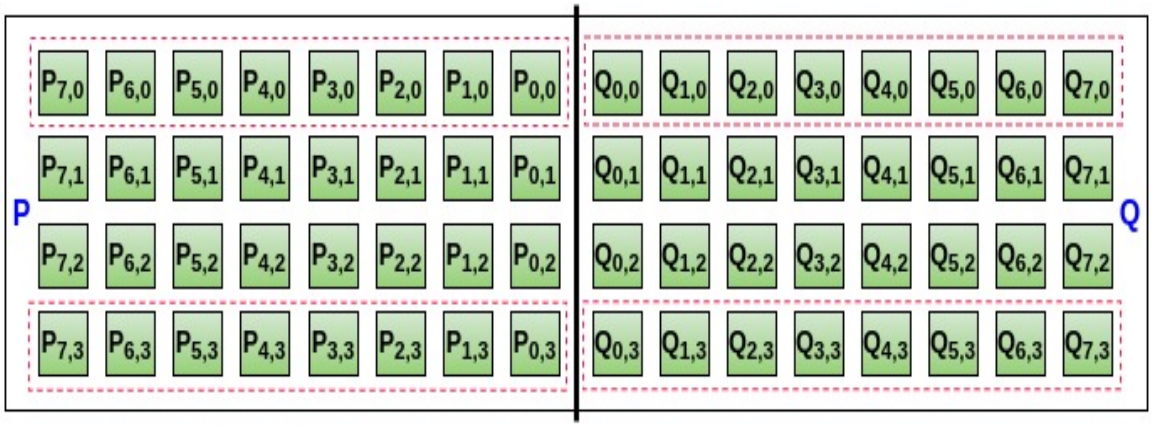


Figure 2.10. Four samples boundary segment formed by block P and block Q. Deblocking decisions are based on line 0 and line 3.

2.4.6.2 The computation of filter parameters

Two filter parameters β and t_c are responsible for the threshold at the time of filtering. These parameters are computed using QP of the boundary of both sides. The calculations are shown in equations 2.5, 2.6, 2.7 and 2.8 [52].

$$QP_L = (QP_P + QP_Q + 1) \gg 1 \tag{2.5}$$

$$Q = Clip3(0, 51, QP_L + (slice\_beta\_offset\_div2 \ll 1)) \tag{2.6}$$

$$\beta = \beta' * (1 \ll BitDepthY - 8) \tag{2.7}$$

$$t_c = t'_c * (1 \ll BitDepthY - 8) \tag{2.8}$$

Here, QP_L is the average QP value from the boundary of both sides. The QP_L limit is used to get Q parameter by clipping. Then, β' and t'_c values along with the bitdepth are used to calculate β and t_c.

2.4.6.3 Filtering decision

The Equation 2.9 is used to make the decision on basic filter. Here, first part is dedicated to P block pixels for 0th to 3rd row. In addition, the second part is focused on Q block pixels for 0th to 3rd row. Afterwards, the filter is applied depending on the decision of the type of filter.

$$\begin{aligned} |P_{2,0} - 2P_{1,0} + P_{0,0}| + |P_{2,3} - 2P_{1,3} + P_{0,3}| + \\ |Q_{2,0} - 2Q_{1,0} + Q_{0,0}| + |Q_{2,3} - 2Q_{1,3} + Q_{0,3}| < \beta \end{aligned} \tag{2.9}$$

2.4.7 Sample adaptative offset

Sample Adaptative Offset (SAO) [53] is applied after DBF in decoding process of VVC. SAO in VVC is exactly the same as HEVC [50]. SAO is used to minimise the distortion of the mean sample in a region. Here, for every category, an offset is selected using a classifier taken from several categories to filter the pixel [54]. There are two types of offset used in the SAO filtering process: 1) edge offset and 2) band offset.

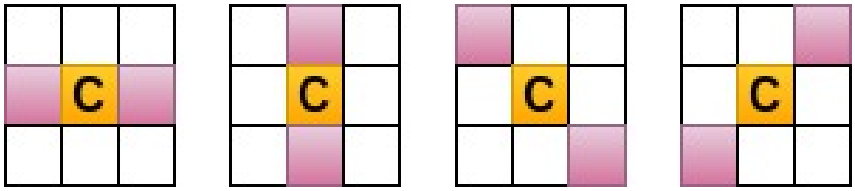


Figure 2.11. Edge offset 1D classification for 3 pixels pattern in degree: 0 (left), 90, 135, and 45 (right).

In edge offset, adjacent pixels are used to classify the type of edge offset. One-dimensional (1D) classification for the 3 pixel pattern of edge offset (0, 90, 135 and 45 degrees) is shown in Figure 2.11. Moreover, the SAO band offset divides all the pixels in thirty two equal bands as presented in Figure 2.12. As seen in Figure 2.12, the sample values for four subsequent bands are adjusted by applying an offset [55]. The same offset is applied to each CTU, each luma, and each chorma components.

2.4.8 Adaptive loop filter

Adaptive Loop Filter (ALF) is the last decoder block that is introduced in VVC. Although ALF is part of the VVC standard, it was not included in HEVC. ALF was de-

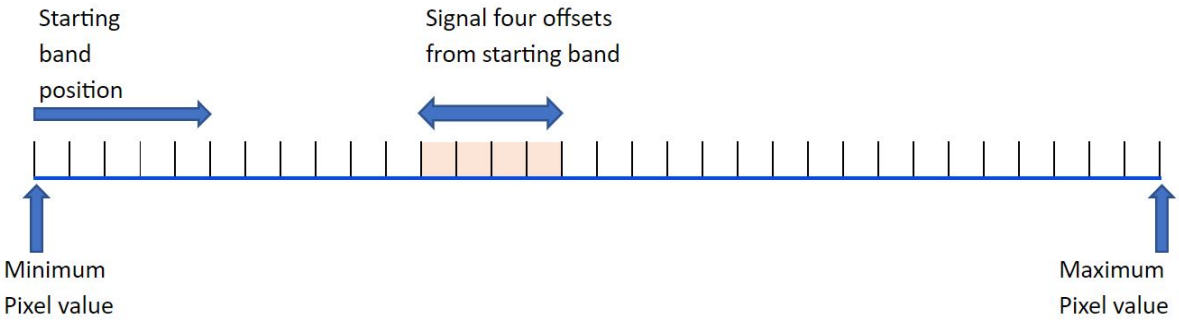


Figure 2.12. Edge offset 1D classification for 3 pixels pattern in degree: 0 (left), 90, 135, and 45 (right).

signed based on Wiener filters [56] to minimise the mean square error of the reconstructed samples with respect to the original ones. The general working flow diagram of VVC ALF is presented in Figure 2.13. VVC ALF filtering process are classified mainly in following parts that are explained later:

- luma components classification.
- Luma and chroma component filtering.
- Cross-component filtering.

2.4.8.1 Luma components classification

ALF uses the classification process at the level of sub-block, where the classification process is applied only on luma component [50]. Here, 25 classes are used to categorise each 4×4 block. Moreover, the directionality and a numerical value that depicts the activity of each sample inside the block provide the foundation for this categorisation. Equations 2.10, 2.11, 2.12 and 2.13 are used to calculate vertical (G_v), horizontal (G_h), and two diagonal gradients (G_{d0}, G_{d1}) for the reconstructed sample, Y [57]. Gradient values are used for classification of each 4×4 luma block.

$$G_v(i,j) = |2Y(i,j) - Y(i-1,j) - Y(i+1,j)| \tag{2.10}$$

$$G_h(i,j) = |2Y(i,j) - Y(i,j-1) - Y(i,j+1)| \tag{2.11}$$

$$G_{d0}(i,j) = |2Y(i,j) - Y(i-1,j-1) - Y(i+1,j+1)| \tag{2.12}$$

$$G_{d1}(i,j) = |2Y(i,j) - Y(i-1,j+1) - Y(i+1,j-1)| \tag{2.13}$$

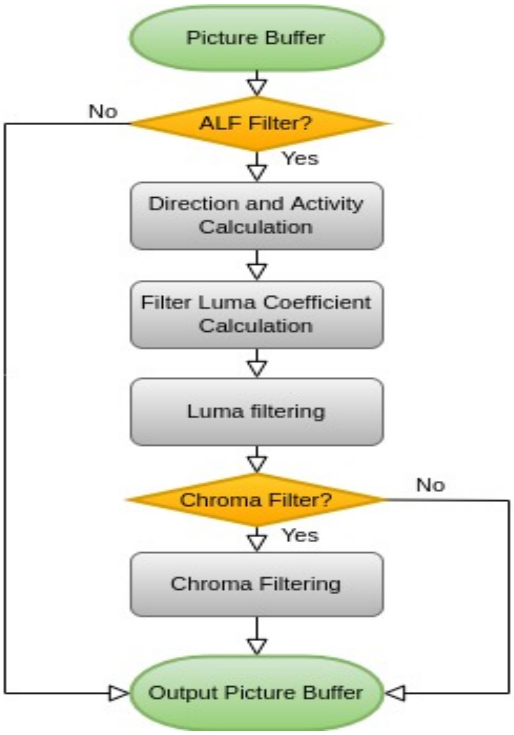


Figure 2.13. ALF filter working flow diagram.

2.4.8.2 Luma and chroma component filtering

ALF is applied to the output samples of SAO after completing the luma component classification process. In VVC ALF, the supported Diamond-shape (DMS) filters are 5×5 for the chroma component and 7×7 for the luma component, as shown in Figure 2.14. Each square denotes the luma and chroma components. In addition, a coefficient value is presented by c_i. Equation 2.14 [58] is used to filter the center square of the DMS filter.

$$\begin{aligned}\tilde{\mathrm{Y}}(x,y) = Y(x,y) + (\sum_{i=0}^{N-2} c_i(Y(x+x_i, y+y_i) - Y(x,y)) \\ + \sum_{i=0}^{N-2} c_i(Y(x-x_i, y-y_i) - Y(x,y)) + 64) \gg 7\end{aligned} \tag{2.14}$$

Where as $\tilde{\mathrm{Y}}$(x,y) is the value of the filtered component at the coordinate (x,y). c_i corresponded value of the component are $Y(x+x_i,y+y_i)$ and $Y(x-x_i,y-y_i)$. The number of coefficients is N, where N = 13 and 7 for 7×7 and 5×5 DMS filter, respectively.

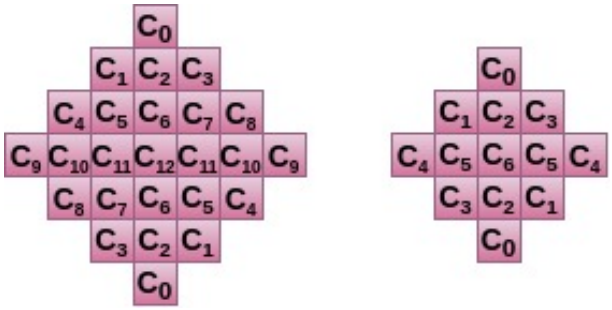


Figure 2.14. ALF DMS filters: left 7×7, right 5×5.

2.4.8.3 Cross-component filtering

The Luma components are utilised in the Cross-component ALF Filtering (CCALF) process to enhance chroma components. As shown in Figure 2.15, CCALF chroma blue (Cb) components are added with ALF Cb components and CCALF chroma red (Cr) components are added with ALF Cr components to obtain the output. For this reason CCALF received its name cross-component filtering. Furthermore, CCALF adopted the DMS filter similar to the ALF luma and chroma filter with asymmetric configuration. VVC CCALF included the 3×4 DMS filter, while initially the 5×6 DMS filter was proposed [50]. The amount of multiply and accumulate operations and the amount of coefficients needed for the implementation of CCALF was decreased due to the fact that CCALF adopted the DMS filter.

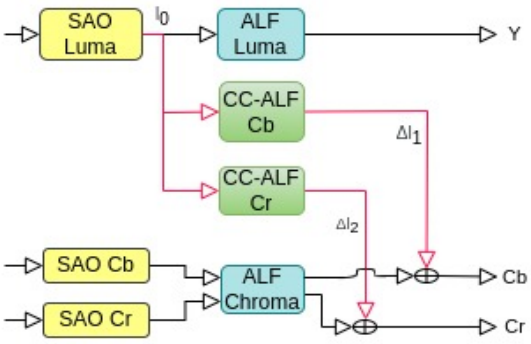


Figure 2.15. Diagram of the CCALF architecture.

2.5 Open source VVC decoders

After the standardisation in 2020, VVC reference software was available only for test and comparison. Then some optimised VVC implementation was developed by some

companies and communities. In this doctoral study, the target was to optimised open source VVC decoder using heterogeneous platforms.

At this time, a small number of open source software decoders are available that comply with the VVC standard. The decoders are the following:

- VVC test model (VTM) [14].
- Versatile Video Decoder (VVdeC) [15].
- OpenVVC [63].
- O266dec [66].

2.5.1 VVC test model

Every video standard comes with reference software solutions that includes the fundamental features of the standard. However, reference software provides very basic performance in terms of speed, but offers a base to the scientific community, industries, and research groups for developing more sophisticated and optimised solutions. VVC Test Model (VTM) [14] (see Figure 2.16 for release date of different VTM version) is the reference software solution for the VVC standard. VTM was develop on top of the HEVC Test Model (HM) [59]. Some of the block-level functionalities such as parsing CABAC and searching motion of HM were included to VTM. However, most of them were redesigned for VVC. In addition, VTM is written in the C++ programming language version 11 using a modern standard library. The code and data structure of VTM are not optimised for different platforms. Particularly for embedded platforms, which is the target of this study.

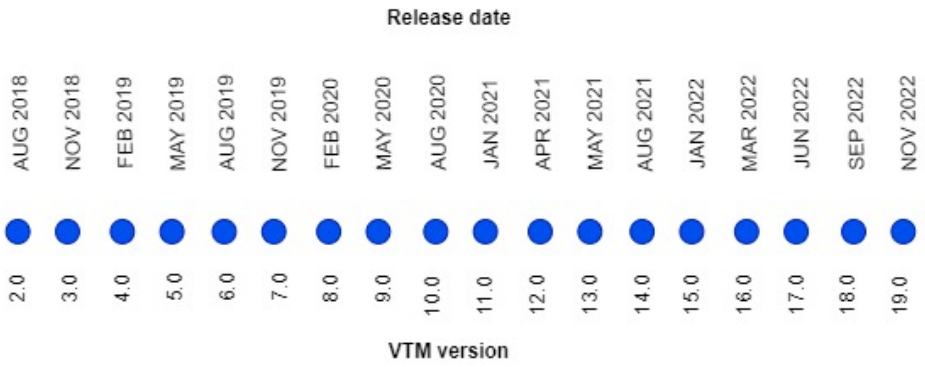


Figure 2.16. Release date of different VTM version.

2.5.2 Versatile video decoder

Versatile Video Decoder (VVdeC) is an open source and optimised VVdeC (see Figure 2.17 for release date of different VVdeC version) decoder released by the Fraunhofer Heinrich Hertz Institute in October 2020 [15]. It has been written based on VTM reference software using the C++ programming language. VVdeC is compliant with VVC Main 10 profile and is compatible with FFmpeg [60] and GPAC [61]. In addition, VVdeC is capable of decoding all the VVC encoded bitstreams [62].

VVdeC uses SIMD and multithreading parallelisation to provide optimised decoding performance. The decoding process in VVdeC begins with the simultaneous parsing of multiple frames. Then, a reconstruction process is initiated for the parsed frames, and tasks are divided based on CTUs and CTU lines. In this process, each CTU receives a stage for achieving synchronisation among the tasks. It facilitates parallel performing of tasks after dependencies are fulfilled. Here, each CTU is given to a task worker, and the thread-pool scans available tasks for the task worker. Finally, the decoded video is exported after completing the filtering of all CTUs. VVdeC has obtained as far as 90% [5] decoding run-time reduction compared to VTM.

Figure 2.17. Release date of different VVdeC version.

2.5.3 OpenVVC

OpenVVC [63] (see Figure 2.18 for release date of different OpenVVC version) is a VVC software decoder mainly developed at INSA Rennes (France) that is compliant to main profile of the VVC. It is a royalty-free open source project designed and optimised for commonly used operating systems and hardware systems. It is written completely in C programming language. In addition, it is integrated with several widely used video

players including GPAC, FFplay, and VLC [64]. Furthermore, the data level optimisation technique based on SIMD instructions is integrated into OpenVVC for x86 and ARM processors. OpenVVC also supports multiple CPU cores-based frames and tiles parallelisation, and it uses very low memory while obtaining a high decoding speed.

In the beginning of the decoding process of OpenVVC, the parameters of the picture and sequence are parsed by main thread. Therefore, main thread provides the necessary instructions and data to the worker threads to perform motion compensation. Here, reconstruction tasks are carried out at the level of CU whereas TX, LMCS, EP, and IP are the reconstructed tasks. After the reconstruction process is finished, all threads are available to process the pixel for DBF filtering at the CTU level. This prompt assignment of threads reduces the memory requirements to store quantization parameter map and CU dimension for DBF filtering [65]. At the end of the decoding process, SAO and ALF filters are performed in an orderly fashion at the level of CTU line.

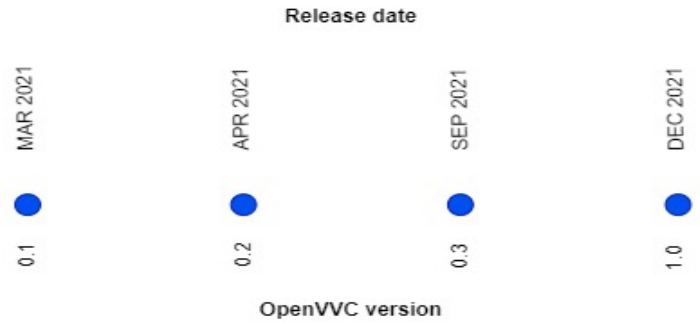


Figure 2.18. Release date of different OpenVVC version.

2.5.4 O266dec

Tencent Media Lab has released a real-time VVC decoder named O266dec, which is widely available in [66]. It integrates multiple level parallelisation strategies: picture level, task level, CTU level, and sub-CTU level. Moreover, O266dec is written in the C++ programming language version 11. It uses the C++ class, function template [67], and thread libraries to design the decoder. It offers portability to multiple operating systems with multi-threading and SIMD parallelization-based optimization [68]. However, no updates have been released since May 2021.

Chapter 3

State-of-the-art on the implementation of video decoders

This chapter is organised as follows: Section 3.1 presents different parallelism techniques used for optimising video codecs, which includes fine-grain and coarse-grain parallelism. Section 3.2 shows different hardware accelerators used for accelerating video processing algorithm. Section 3.3 provides a discussion of related work on different hardware platforms published on scientific literature.

3.1 Parallelism in video codecs

This section presents the most relevant parallelism techniques used in video codecs. First, fine-grain parallelism technique is discussed in Section 3.1.1. Therefore, different coarse-grain parallelism techniques are outlined in Section 3.1.2.

3.1.1 Fine-grain parallelism: Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is a data-level parallel technique in which multiple data are loaded into a single register (up to 512 bits) to perform arithmetic operations [18]. For instance, two 64-bit, four 32-bit, eight 16-bit, or sixteen 8-bit data can be loaded into one 128-bit SIMD register. Due to this characteristic, concurrent data processing can provide a theoretical acceleration of up to $\times 16$ for 8-bit data. SIMD offers higher processing performance by improving processor throughput, which overcomes the fact that the processor clock speed is rigid and limited. Therefore, SIMD is an interesting tool for video encoding/decoding that involves intensive arithmetic operations.

An example of multiply operation between two vectors (each having a group of operands) by SIMD is presented in Figure 3.1. Here, four values of "A" and "B" are multiplied concurrently, and outputs are stored independently in "C".

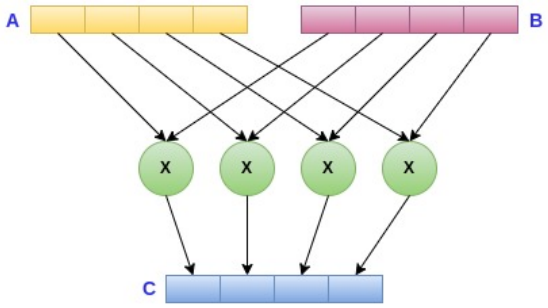


Figure 3.1. SIMD operation example where four multiplication operations are performed simultaneously.

Nowadays, SIMD units with up to 512 bits are available in most of the GPPs. x86-based processor supports Streaming SIMD Extensions (SSE), SSE2, SSE3, Supplemental Streaming SIMD Extensions (SSSE)3, SSE4.1, SSE4.2, Advanced Vector Extensions (AVX), AVX2, and AVX-512 [69] instruction sets which implement SIMD on x86-based processor. Moreover, ARM-based architecture supports Neon-based SIMD instruction sets [70].

Neon is the advanced vector computing SIMD extension for ARM processors. Most of the ARM processors including Cortex A8 and A9 included Neon technology. It was developed aiming for enhancing user experiences by speeding up audio and video processing, gaming, and Two-dimensional (2D) and three-dimensional (3D) graphics [70]. ARM provided a standard set of Neno instructions that perform operations that are difficult to write in C/C++ [71]. Neon instructions can perform the same operation on multiple elements concurrently in Neon registers (up to 128 bits). It supports multi-data types including integer and floating-point. Neon can be included in C program by simply adding arm_neon.h header file [72].

3.1.2 Coarse-grain parallelism

This section describes some widely used coarse-grain parallelism techniques in different video coding standards including the VVC standard.

3.1.2.1 Frame-level parallelism

In frame-level parallelism, several frames are processed at the same time. Here, the dependencies of the motion compensation should be fulfilled. The length of the motion vector is the deterministic factor for the parallelism level in frame-level parallelism. This is the biggest drawback, which negatively affects the parallelism of video sequences with large motion sequences [73]. However, due to the absence of motion correction requirements, sequences with all intra configuration benefit the most from frame-level parallelism. Furthermore, for parallel decoding, each thread requires extra picture buffers and local buffers storage in frame-level parallelism. Therefore, it requires more memory than sequential decoding. This is an important limitation for embedded platforms but it is widely used in HGPP-based platforms.

3.1.2.2 Tile parallelism

Tile parallelism was introduced in HEVC standard [74]. VVC standard supports tile partitioning of grid shaped only [75] as its preceding HEVC standard. Moreover, CTUs are the basic component of a rectangular tile inside a frame. Figure 3.2 shows an example of 2×2 tile partitioning of a frame, where A to D represents four tiles. The boundaries split the tiles, which makes the tile processing independent. Moreover, it removes the dependencies from the prediction but decrease the performance due to a reduction of the efficiency of the encoder. Therefore, parallel processing of a frame during decoding is facilitated using multiple threads. On the other hand, pixels from both sides of the tile boundary need to be reconstructed before processing the in-loop filters.

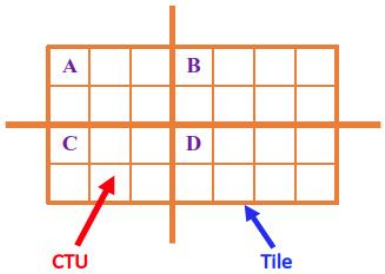


Figure 3.2. Illustration of tile partitioning in VVC decoder, where one frame is divided into four tiles.

3.1.2.3 Wavefront Parallel Processing

Wavefront Parallel Processing (WPP) also was introduced for the first time in the previous HEVC standard [76]. WPP divides frame into multiple CTU rows that can be processed at the same time. However, the dependencies remain unchanged at the row boundaries. The information of left, top-left, top and top-right CTUs is needed to process CTU. Therefore, two CTUs of the previous row needed to be processed in order to process each row in parallel by different threads. Figure 3.3 shows an example of WPP and the dependency involved in WPP between CTUs. However, for picture subdivision during the initialisation of the entropy engine at the beginning of each CTU line, the above-right coding dependency is eliminated. As a result, the CABAC context is re-initialised at the start of each CTU row and depends on the information from the first CTU of the upper row. In these cases, the encoder must encode the sequence taking each technique into account.

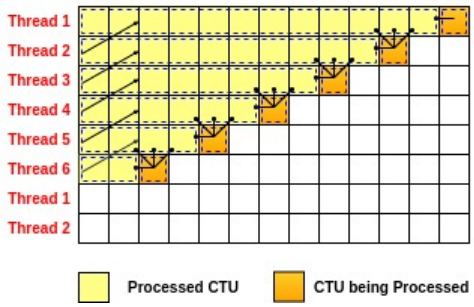


Figure 3.3. Example of Wavefront Parallel Processing (WPP) and an example of the dependency involved in WPP between CTUs using 6 threads in parallel.

3.2 Hardware accelerators for video coding tools

There are several hardware accelerator-based platforms available in the market. However, the most commonly used platforms for video processing algorithms are based on GPU and FPGAs. This section provides a brief description of both technologies. This section is only an overview of the technology, but more details can be found in [].

3.2.1 Graphics Processing Unit

GPUs are the coprocessing units of CPUs that are suitable for parallel processing. Although the initial purpose of GPU was to accelerate 3D graphics rendering, today GPU is used to accelerate various applications because GPU has become versatile and

easy programmable. GPUs are skilled at managing applications with high degree of data parallelism and high computational requirements. On the other hand, CPU is a specialist in the handling of sequential applications. The main comparison between CPU and GPU is that CPU contains a small number of processing units called cores with a higher clock frequency, while GPU comes with hundreds or thousands of cores with a lower clock frequency (see Table for some example of CPU and GPU). The high-level architecture of CPU and GPU is presented in Figure 3.4. Hence, CPU is highly dependent on data caching and control flow, while GPU is highly dependent on processing units. GPU gives priority to throughput over latency. However, GPUs work as slave devices of CPUs. Currently, NVIDIA is the market leader in GPUs manufacturing, where NVIDIA GPUs are programmed using the Compute Unified Device Architecture (CUDA) [79]. For example Sancho et al. [77] used NVIDIA GPU to generate real-time 3D models of HyperSpectral images by exploiting Hyperspectral Depth Estimation Tool for Medical Applications. In addition, Punithakumar et al. [78] proposed a GPU-based implementation to accelerate deformable image registration with improved efficiency.

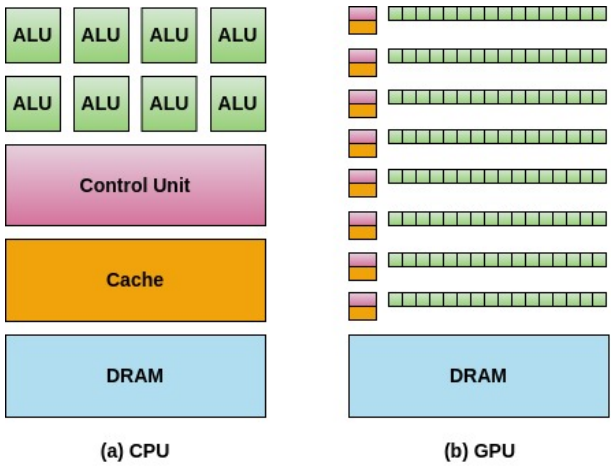


Figure 3.4. Illustration of the architecture of a generic CPU and GPU in high-level.

3.2.1.1 Compute Unified Device Architecture

Compute Unified Device Architecture (CUDA) is the programming Application Programming Interface (API) that is used to program NVIDIA GPU for CPU. The program that is executed on GPU is called the CUDA kernel. The parallel portion of the kernel is processed by the CUDA threads (see Figure 3.5: left-top) in parallel. The threads are grouped into one CUDA thread block (see Figure 3.5: left-middle), where the threads in a CUDA thread block are organised in multidimensional ways. CUDA thread blocks

are independent and one CUDA thread block can be processed by only one Streaming Multiprocessor (SM). Parallel processing of multiple CUDA thread blocks by one SM (see Figure 3.5: right-middle) is allowed. However, parallel processing of one CUDA thread block by multiple SMs are not allowed. Furthermore, multiple CUDA thread blocks are grouped into the CUDA kernel grid (see Figure 3.5: left-bottom), where the CUDA thread blocks in a CUDA kernel grid are organised in multidimensional ways. The execution of the CUDA kernel on GPU is illustrated in Figure 3.5. Here, default CUDA definition are used for 3D threads and blocks, where *threadIdx* was used for threads 3D indexing and *blockIdx* was used for blocks 3D indexing. Moreover, CUDA built-in definition *blockDim* variable provides access to the dimension of the thread block to the kernel. However, the architecture of CUDA only supports up to 1024 threads/block.

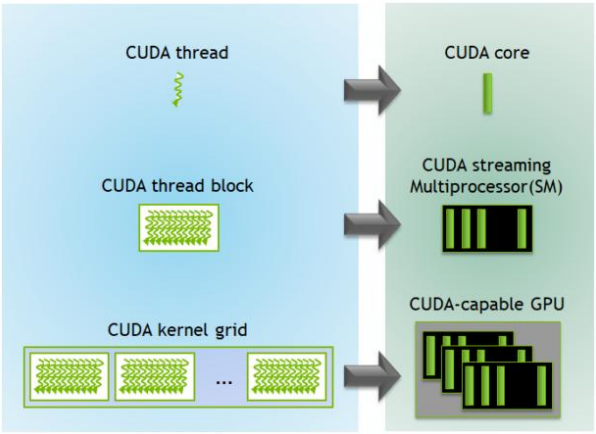


Figure 3.5. Illustration of a kernel execution on GPU (source:[80]).

3.2.2 Field Programmable Gate Arrays

FPGAs are integrated circuits consisting of programmable Logic Block (LB)s. FPGAs are highly configurable in nature, where Hardware Description Language (HDL) [81] is used to configure FPGAs. The programmer can employ sophisticated logic operations thanks to the HDL commands used to design gates and connections. Logical, arithmetic, and digital logic are supported by FPGAs. An example of FPGAs architecture is shown in Figure 3.6. Here, the internal routing SoC among the LBs is established by electrical wiring. Customisation of FPGAs can be performed by switching on and off the internal routing and programming the LBs. Therefore, FPGAs are appropriate for devices and systems that need repeated modifications, as they can be reconfigured even after deployment. Furthermore, FPGAs contain programmable Input Output Block (IOB)s for connecting to external circuits.

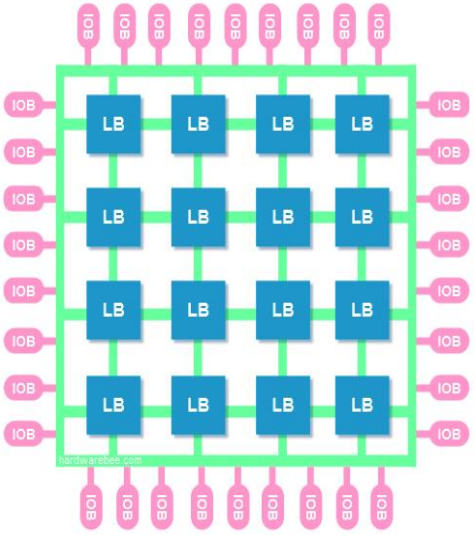


Figure 3.6. An example of the generic architecture of an FPGA (source:[82]).

3.3 Related work on different hardware

Several architectures were exploited to accelerate different video coding algorithms. However, GPU and FPGAs are the most used hardware in video coding algorithms. The authors in [83] exploited FPGAs to reduce the computational cost of DST-VII and DCT-VIII of VVC standard. They proposed hardware implementation for VVC forward and inverse MTS where the approximation method was adopted. The proposed solution achieved 386 FPS for 2K video and 96 FPS for 4K video. Garrido et al. used FPGAs to calculate MTS of VVC standard, MTS enables DCT-II, DCT-VIII and DST-VII transforms of size between 4×4 and 64×64 [10]. They proposed VVC MTS implementation based on a deeply pipelined high-performance architecture which was protyped on a system on a programmable chip. This implementation obtained up to 64 FPS for 4K video sequences with 4×4 transform sizes.

In [84], OpenCL-based design was proposed to accelerate VVC transformation and quantization in the intra-frame coding using FPGAs. Here, the authors addressed the trade-off between the usage of resources and the speed of processing as huge FPGA resources were required for acceleration. This proposed implementation showed an average 3.22 times speedup compared to CPU implementation. In addition, 83.3 FPS was achieved for 64×64 CTU size.

VVC intra prediction was accelerated using two DSP blocks and two adders of FPGA in [85]. Here, thirty DSP data paths used in a way that 2 DSP blocks and 2 adders belonged to one DSP data paths. This implementation efficiently exploited FPGA for processing

4×4, 8×8, 16×16, and 32×32 prediction unit sizes of angular intra prediction modes. 34 FPS was achieved in worst case for FHD videos using the proposed implementation.

In [86]-[87], FPGAs was used to accelerate HEVC intra prediction. All these study presented implementation for angular prediction modes of prediction unit size between 4×4 and 32×32, except [87] only implemented 4×4 DC and angular prediction modes. The proposed implementation was capable of using an average 27% FPGA resources. The authors in [88]-[89] presented the implementation of inverse DCT/DST using FPGAs. Here, real-time decoding was achieved with 54 FPS for FHD videos using high-level synthesis tool in [88]. In [89], the high-level synthesis tool was also used to implement the 2D inverse DCT/DST. Here, a 4-point and an 8/16/32-point inverse DCT/DST was included for small transform blocks and other transform blocks, respectively. This implementation obtained real-time decoding with up to 68 FPS using FPGA.

It can be seen in the literature that FPGAs were used primarily to accelerate transformation, quantization, and intra prediction blocks of different video algorithms. On the other hand, it was found in the literature that GPU was used to accelerate different decoder blocks. Han et al. [90] accelerated decoding speed using a GPU-based VVC motion compensation scheme. GPU resources were efficiently used by managing data dependency and thread organisation for different situations of CU repartition. This implementation demonstrated a speedup of 16 times in which motion compensation was completed in only 5 ms for 4K UHD videos. In [91], CTU-level parallel motion search scheme was implemented using GPU. The proposed method obtained an average of 70% time-savings over CPU only implementation. Igarashi et al. in [92] presented a parallel processing technique of HEVC transformation and quantization block by merging irregular load or store of transform unit data in different memory addresses with transformation and quantization. Here, a speedup of $\times 10.8$ with real-time performance for 4K video was obtained by the proposed implementation.

GPU also used in [93] to process one luma block with 64×64 size and two chroma blocks with 32×32 size in parallel to accelerate the HEVC inverse transformation and quantization block. This experiment obtained a speedup of 33 times for inverse transformation and quantization block which reduced the decoding time of 40.7% than the conventional implementation.

Radicke et al. [94] accelerated HEVC intra prediction blocks of encoder side by using GPU. The proposed method extended the encoder of single-threaded with the WPP mechanism. The experimental results of this methods achieved 64.5% and 94.8% time reductions for the single-threaded and single-threaded with WPP system, respectively. In [95], a pipeline structure was proposed to accelerate HEVC in-loop filter of the encoder

side, where DBF and SAO were processed on GPU. The proposed implementation obtained a time reduction of an average 47% compared to CPU only implementation. The authors in [96]-[97], used GPUs to accelerate decoder side in-loop filter. The pattern redesigning is proposed for HEVC DBF and SAO to efficiently use GPU architecture in [96]. Here, real-time decoding was achieved with approximately 50 FPS for 4K UHD videos. Souza et al. in [97] proposed a parallel algorithm for HEVC DBF, where the following three situations were evaluated using 1) CPU cores, 2) GPU only and load balance implementation, and 3) CPU cores and GPU. This experiment demonstrated that CPU+GPU implementation performed best among all the approaches that obtained a speedup of up to 35.8 times compared to the conventional method for 4K UHD videos.

Although GPU and FPGA are the most suitable for video processing algorithms, DSP was used also for different video processing algorithms in scientific research. The authors in [98]-[101] used DSP architectures to accelerate the HEVC standard. In [98], a video decoding system was proposed for the HEVC standard which is composed of two reconfigurable processors, a bitstream process unit, and a memory process unit. The proposed implementation achieved 40% performance improvements with real-time decoding for FHD videos on DSP. In addition, three different HEVC implementations on DSP were developed and compared in [99]. Here, the performance of OpenHEVC was improved by 10% by combining RVC-CAL descriptions and native C-code. Zhang et al. [100] accelerated HEVC encoder using SIMD on DSP. The experimental presented a speedup of up to 87.32 and 6.56 for the C-based encoder and O3 optimisation enabled encoder, respectively. Then, a DSP-based implementation of HEVC was proposed in [101] where Orcc and OpenMP were used together. The proposed implementation achieved 70% speedup using two DSP cores compared to one DSP core. Moreover, they claimed that the same implementation can be used for N cores.

3.3.1 Conclusion

There are not too many VVC implementations published in the scientific literature during the development of the doctoral Thesis. Therefore, various implementations of previous video coding standard was studied on different hardware accelerators, where the optimisation of full VVC decoder using software and hardware-based hybrid approach was not found. Moreover, only one GPU-based implementation of VVC standard was published. Nevertheless, after reviewing the scientific literature, it is found that FPGAs were used primarily to accelerate transformation, quantization, and intra prediction blocks of different video algorithms. In addition, GPU was used for accelerating most of the decoder blocks.

Chapter 4

Implementation of video decoders over heterogeneous platforms

This chapter presents the details of the work carried out implementing different video decoders over heterogeneous platforms (see Section 4.1). It is worth noting that the set of steps followed on the implementation process have, at the end, served to build up the proposed design methodology that it is the main contribution of this Thesis work. In details, the working methodology includes the selection of platforms (see Section 4.2), video algorithm selection (see Section 4.3), profiling (see Section 4.4), open source decoder optimisation for ARM-based platforms (see Section 4.5), CPU+GPU based hardware accelerator (see Section 4.6), and generalisation of the proposed implementation. Lastly, both memory usage and energy consumption are evaluated in Sections 4.7.1 and 4.8, respectively.

4.1 Working methodology

The general view of the working methodology is presented in Figure 4.1. Firstly, the platform (see Section 4.2) and the video algorithm (see Section 4.3) were selected. Therefore, the profile (see Section 4.4) of the selected video algorithms was performed on the selected platforms to identify the most time-consuming blocks for applying optimisation techniques. Then, a hybrid approach was implemented to accelerate the video algorithm in which the fine-grain SIMD optimisation (see Section 4.5), and CPU+GPU based hardware accelerator (see Section 4.6) were used along with the default coarse-gain optimisation in the development of the hybrid approach. After all implementations, the validation (see 4.1.1) was performed to determine the accuracy of the optimisation

process. Furthermore, the generalisation of the proposed implementation is provided.

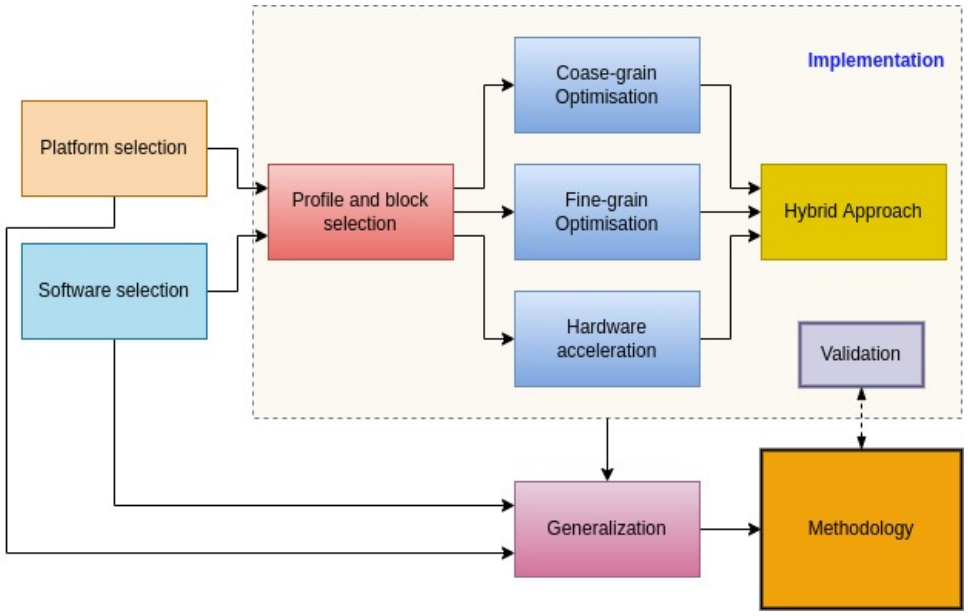


Figure 4.1. General view of applied working methodology.

4.1.1 Validation

Validation of the optimised decoded video is essential to determine the accuracy of the optimisation process. In this doctoral Thesis, the Vooya [122] application was used to analyse decoded videos. Vooya is a raw video player and viewer that supports .yuv format. It offers raw video format specification options as shown in Figure 4.2. Moreover, it has a feature to compare two videos with the value of Peak Signal-to-Noise Ratio (PSNR), Mean-Square Error (MSE), and Structural Similarity Index Measure (SSIM) metrics. It also provides a mouse highlighter that shows PSNR, MSE, and SSIM values for two different videos pixel by pixel, which also helps to identify the differences visually (see Figure 4.3). Furthermore, if two videos are the same, the values of PSNR, MSE and SSIM are infinite, zero, and one, respectively. In this Thesis, PSNR, MSE and SSIM values were checked using Vooya after every optimisation (e.g. SIMD optimisation, CPU+GPU implementation). Finally, it was made sure that the values PSNR, MSE and SSIM were infinite, zero, and one, respectively, in all cases. This means that the original decoded video was exactly the same as the decoded video with the modified decoders.

In addition, the MD5 [123] message-digest algorithm hash function was used in this Thesis for comparing two videos. MD5 generates a 128-bit hash value for each unique file. The hash value is changed even if the video is slightly changed. Therefore, this method can be used to know only if two videos are exactly the same or not.

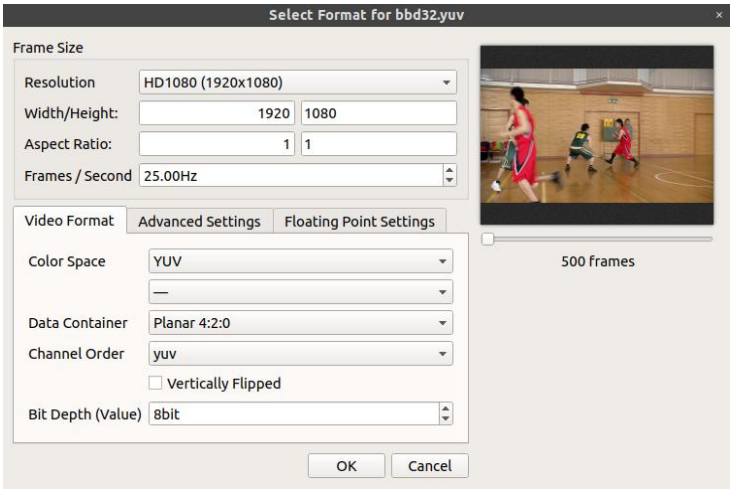


Figure 4.2. Options to specify a raw video format, with instant preview.

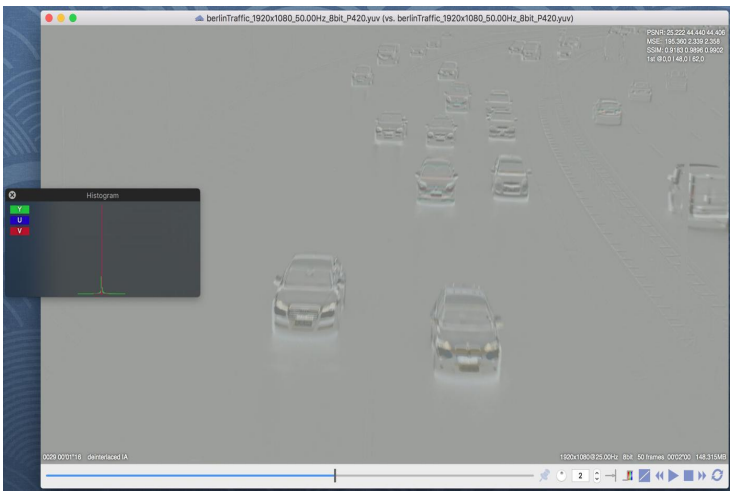


Figure 4.3. Examine the difference between two files with PSNR, MSE and SSIM.

In this Thesis, Vooya and MD5 were used to compare the videos after each implementation with the reference videos. In all cases, PSNR value was infinite, MSE value was zero, SSIM value was one and the MD5 was exactly the same.

4.2 Platform selection

There are different heterogeneous platforms available in the market for video and image processing applications. Among all, the heterogeneous platforms based on GPU and FPGA are the most suitable for the video decoding application. Generally speaking, FPGAs usually require longer design and implementation time, but they offer very good performance for highly parallel task processing, where as GPU is good at executing sim-

ple repetitive computing operations at high speeds [102]. Both platforms have several advantages. However, in this doctoral thesis GPP+GPU-based heterogeneous platforms were chosen for the following reasons.

- GPU contains numerous parallel processors (Arithmetic Logic Unit (ALU)s) that provides fast and efficient computing.
- GPU is easy to program using programming API, like CUDA.
- GPU is widely available and cheaper than FPGA.
- GPU has a large user community.
- GPU is widely used in different consumer electronics devices [103].
- Video algorithms have modules with repetitive operations (e.g. filters).
- GDEM group has long experience working with GPU and several other research lines using GPU.
- GPUs are very extended as coprocessors in consumer electronics.

Nowadays, NVIDIA is the market leader for GPU-based embedded heterogeneous platforms and has a large user community. Therefore, in this study, heterogeneous platforms based on GPU manufactured by NVIDIA were chosen. Furthermore, two HGPP were used as reference because video codecs are originally developed aimed at general purpose platforms. The test platforms used in this study are presented in Section 4.2.1.

4.2.1 Test platforms

This section presents the different platforms that have been selected to carry out the implementations and tests of the research work during the doctoral thesis. This work focuses on studying two types of platforms: 1) HGPP, and 2) embedded heterogeneous platforms.

4.2.1.1 High-performance general purpose processor

The video algorithms are initially developed for HGPP-based platforms. In this study, two different HGPP-based platforms have been used: 1) Intel core i9-10900x HGPP processor, and 2) AMD Ryzen Threadripper HGPP processor. These platforms were state-of-the-art in HGPP when the doctoral study started.

AMD Ryzen threadripper HGPP processor: AMD Ryzen threadripper (from now on, Ryzen) processor (see Figure 4.4) has an AMD Zen microarchitecture [104] based on the CPU complex. Each CPU complex contains four cores and Ryzen has four CPU complexes. Here, each core is capable of executing two threads with simultaneous multiprocessing power. Therefore, the Ryzen processor can provide a maximum of 32 concurrent threads with a clock frequency of up to 3.7 GHz.

Figure 4.4. AMD Ryzen threadripper (source:[105]).

Furthermore, Ryzen has a L1 cache memory of 96 KB and a L2 cache memory of 512 KB for each core. Moreover, 8 MB of L3 cache memory is shared by each four cores. The L3 cache is accessible to all cores of the four CPU complexes. Figure 4.5 shows a diagram of the architecture of the AMD Ryzen Threadripper processor.

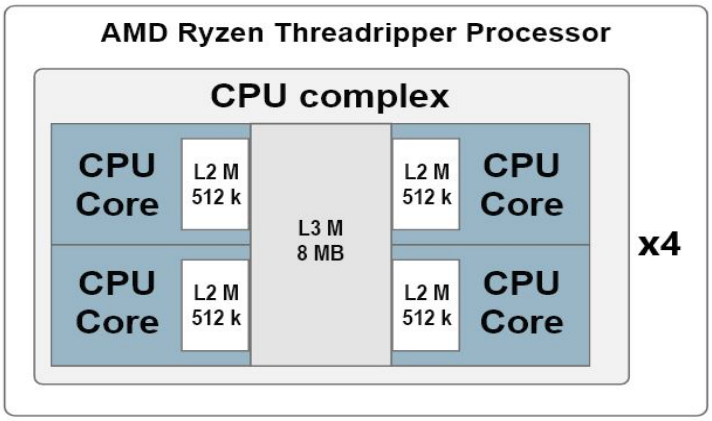


Figure 4.5. A diagram of the architecture of AMD Ryzen threadripper processor.

Intel core i9-10900x HGPP processor: Intel core i9-10900x (from now on, X-series) processor (see Figure 4.6) has a Cascade Lake architecture that offers twenty operating threads, where each ten physical cores can launch two threads concurrently using simultaneous multithreading technology. Each core can operate between a base clock frequency of 3.7 GHz and a turbo clock frequency of 4.7 GHz.

Figure 4.6. Intel X-series processor (source:[106]).

A diagram of the architecture of the Intel X-series processor is shown in Figure 4.7. Each core has a L1 cache memory of 64 KB and a L2 cache memory of 1 MB. Additionally, the ten cores share 19.25 MB of L3 cache memory. The X-series processor offers a bus speed of 8 GT/s [107].

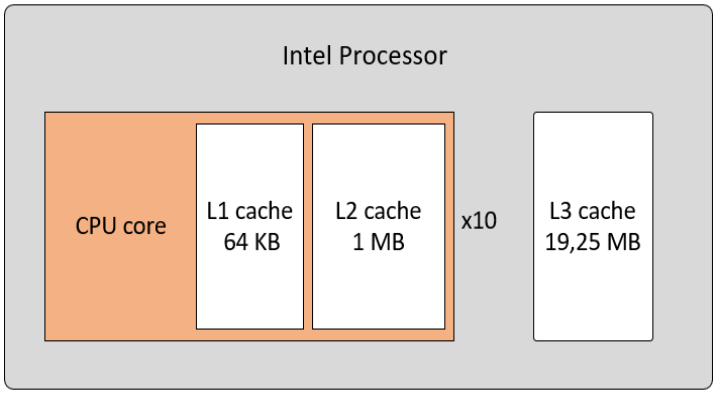


Figure 4.7. A diagram of the architecture of Intel X-series processor.

4.2.1.2 Embedded heterogeneous platforms

This doctoral study focuses on the implementation of a video decoder on heterogeneous platforms, particularly low-cost and resource-constrained embedded platforms. In this research work, NVIDIA Jetson Xavier development kit [108] and Nano development kit [109] have been used. Both platforms were integrated with Embedded General Purpose Processor (EGPP) and Embedded Graphics Processing Unit (EGPU).

NVIDIA Jetson Xavier development kit: NVIDIA Jetson Xavier development kit (from now on, Xavier) (see Figure 4.8) consists of an ARM EGPP with eight cores and a Volta architecture based EGPU complex. Here, each EGPP core has 1.19 GHz minimum and 2.26 GHz maximum clock speed. In addition, Xavier has 4 MB of L3 cache memory shared by eight EGPP cores and 8 MB of L2 cache memory shared by two cores each 2 MB. The EGPU complex in Xavier contains 512 embedded GPU cores (64 cores in each SM) with a clock frequency of 1.38 GHz maximum. EGPU complex has 512 MB of L2

cache memory. Lastly, Xavier includes random access memory of 32 GB 256-Bit with 137 GB/s speed. A diagram of the architecture of the NVIDIA Jetson Xavier development kit is shown in Figure 4.9.

Figure 4.8. NVIDIA Jetson Xavier development kit (source:[110]).

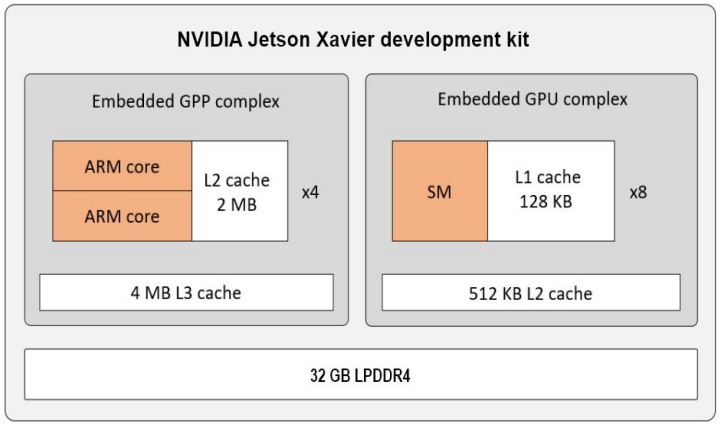


Figure 4.9. A diagram of the architecture of NVIDIA Jetson Xavier development kit.

NVIDIA Jetson Nano development kit NVIDIA Jetson Nano development kit (from now on, Nano) (see Figure 4.10) contains an ARM EGPP with four cores and a Maxwell architecture based EGPU complex with 128 embedded GPU cores. Each EGPP cores and each Maxwell EGPU core run with a maximum clock frequency of 1.48 GHz and 0.92 GHz, respectively. Moreover, EGPP in Nano has L2 cache memory of 2 MB. Lastly, Nano includes random access memory of 4 MB 64 bit with 25.6 GB/s speed. A diagram of the architecture of the NVIDIA Jetson Nano development kit is shown in Figure 4.11.

Figure 4.10. NVIDIA Jetson Nano development kit (source:[110]).

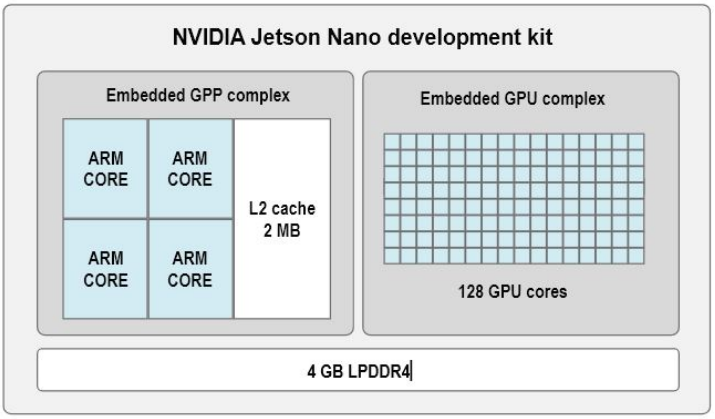


Figure 4.11. A diagram of the architecture of NVIDIA Jetson Nano development kit.

4.2.2 Summary

This section presents a summary of all the platforms selected in this thesis. Table 4.1 compare the features of two HGPPs and two heterogeneous embedded platforms used in this thesis. HGPPs come with a high-performance processor that provides high number of cores with a high clock frequency, larger cache memory, and a large number PCI Express Lanes compared to heterogeneous embedded platforms. On the other hand, heterogeneous embedded platforms are integrated with embedded GPP and embedded GPU. These EGPP have lower clock speed and lower numbers of cores than HGPP, but consumed up to 36 times less power. Furthermore, Xavier has almost 2-3 times higher processing units (GPP and GPU) than Nano. However, Nano requires 2-3 times less power to operate and less physical space for installation, and it cost approximately 8 times less than Xavier.

Table 4.1. Specifications of all platforms used in this thesis.

Platform	Ryzen	X-series	Xavier	Nano
Processor	AMD Threadripper 1950X	Intel i9-10900X	ARMv8.2	ARM A57
Max freq.	3.7 GHz	4.7 GHz	2.26 GHz	1.48 GHz
N. of cores	16 (32 threads)	10 (20 threads)	8	4
Processor tech.	14 nm	14 nm	12 nm	16 nm
L2 cache	8 MB	1 MB	8 MB	2 MB
L3 cache	32 MB	19.25 MB	4 MB	N/A
GPU: No of core	N/A	N/A	512	128
GPU: Max Freq.	N/A	N/A	1.38 GHz	0.92 GHz
RAM speed	N/A	N/A	137 GB/s	25.6 GB/S
Cost	€510	€640	€1000	€130
Power	180 w	165 w	10 w/30 w	5 w/10 w
PCI Express Lanes	1 x64 (PCIe Gen3)	1 x48 (PCIe Gen3)	1 x8 (PCIe Gen4)	1 x4 (PCIe Gen2)
Platform volume	N/A	N/A	100 x 87 x 65	69.6 x 45 x 26

4.3 Selection of video algorithm

Several implementations of VVC decoders are considered in the doctoral thesis. Firstly, VTM (see Figure 2.16 for release date of different VTM version) version 8.0 was selected as it is the VVC reference and was the only software solution available when the Thesis started. This helps to identify the key part of VVC standard for parallelising the process on heterogeneous platforms. Therefore, the VVdeC decoder (available from October 2020, see Figure 2.17 for release date of different VVdeC version) was considered to accelerate over different heterogeneous embedded platforms with limited resources. It was the first open source and optimised VVC decoder that offered real-time decoding up to UHD quality videos over HGPP. Finally, the OpenVVC decoder was considered for this study as it is a lightweight open source and optimised VVC decoder. The OpenVVC decoder (available from March 2021, see Figure 2.18 for release date of different OpenVVC version) was considered as interesting candidate to be accelerated over different heterogeneous embedded platforms with limited resources. This also helps to validate the proposed methodology on two different software solutions, which increases the robustness of the proposed methodology. Lastly, the selection of OpenVVC also provides an opportunity to compare with the VVdeC decoder.

4.4 Profiling video algorithm and decoder block selection for acceleration

Profiling provides information about the computational load of different blocks of the video processing algorithm. It is an essential process for future optimisations to

be successful. It helps to identify the most computation demanding blocks. Therefore, those blocks can be targeted for future optimisations. In this study, VTM and VVdeC decoders were profiled on the selected platforms using one core. Although VVdeC supports multithreading-based decoding, one core was used to profile for fair comparison as VTM does not support multithreading-based decoding process.

4.4.1 Profiling of VTM v8.0

This section presents coarse and fine-grain profiling of the VTM v8.0 decoder over Ryzen and Xavier platforms. This decoder was the only available when this Thesis started. The purpose of this study is to obtain the most computationally heavy modules of the VVC decoder as a first approach. It is worth considering that VTM v8.0 is not optimised at all. This information is essential because computationally heavy blocks are considered for optimisation to accelerate the VVC decoder. To do so, several timestamps were placed before and after each call of the following modules of the VTM v8.0 decoder: ED, TX, IP, EP, DBF, SAO, ALF, and other (OT). OT was calculated by subtracting the time consumed by all decoder modules from the total decoding time. In addition, an open source Valgrind [111] tool suite profiler named Callgrind [112] was used to profile the VTM v8.0 decoder running on the HGPP processor. However, Callgrind can only be used on HGPP processor as Xavier has limited run-time storage, which does not allow profiling some UHD sequences. In addition, the profiling results of both profilers were similar.

FoodMarket4 (FM4), Campfire (CMF), DaylightRoad2 (DR2), ParkRunning3 (PR3), BasketballDrive (BBD) and BQTerrace (BQT) sequences of the common JVET test condition [126] from Table 5.2 (Set A) were used to profile the VTM v8.0 decoder, all with QP values of 22, 27, 32 and 37. Figure 4.12 shows the average time consumption for different blocks of the VTM v8.0 decoder (in %) over Ryzen for All Intra (AI) (left) and Random Access (RA) (right) sequences with QP 22-37. It can be seen that the in-loop filters that include DBF, SAO, and ALF are the most time consuming decoder block. It consumed on average around 40% for AI and RA sequences. Furthermore, EP consumed on average 33% of the decoder time for RA sequences. ED and IP consumed on average 20% and 23%, respectively, for AI sequences. Lastly, TX and OT consumed on average 5%-9% of total decoding time.

Figure 4.13 presents the average time consumption for different blocks of the VTM v8.0 decoder (in %) over ARM-based GPP of Xavier for AI (left) and RA (right) sequences with QP 22-37. Similarly to the profiling on Ryzen, in-loop fiters were the most time consuming decoder module, which is at least 43% of the total decoding time for AI and RA sequences.

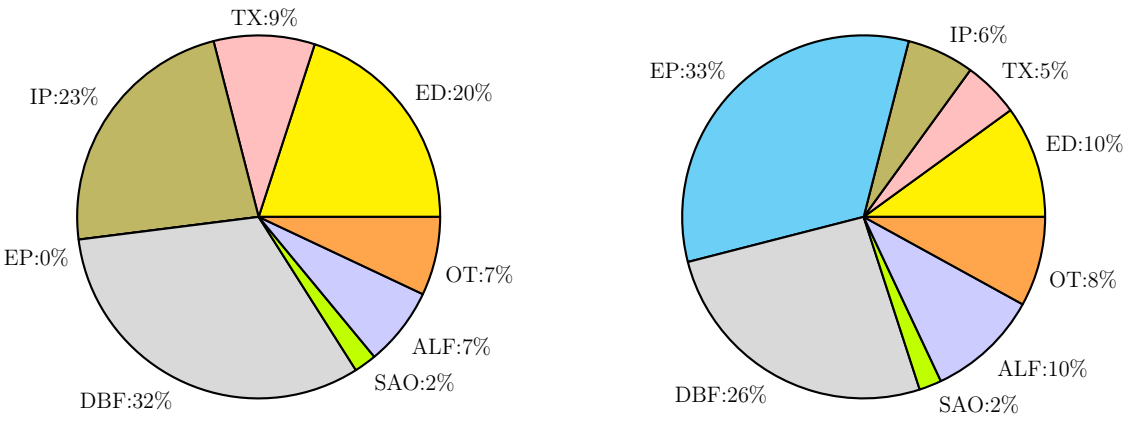


Figure 4.12. Average time distribution for different blocks of the VTM V8.0 decoder (in %) over the HGPP (Ryzen) for AI (left) and RA (right) sequences.

Furthermore, IP and ED represent 21% and 18% decoding times, respectively, for the AI sequences. For RA sequences, EP consumed 35% of the decoding time. Lastly, the rest of the blocks consumed decoding time ranging between 3% and 19% individually.

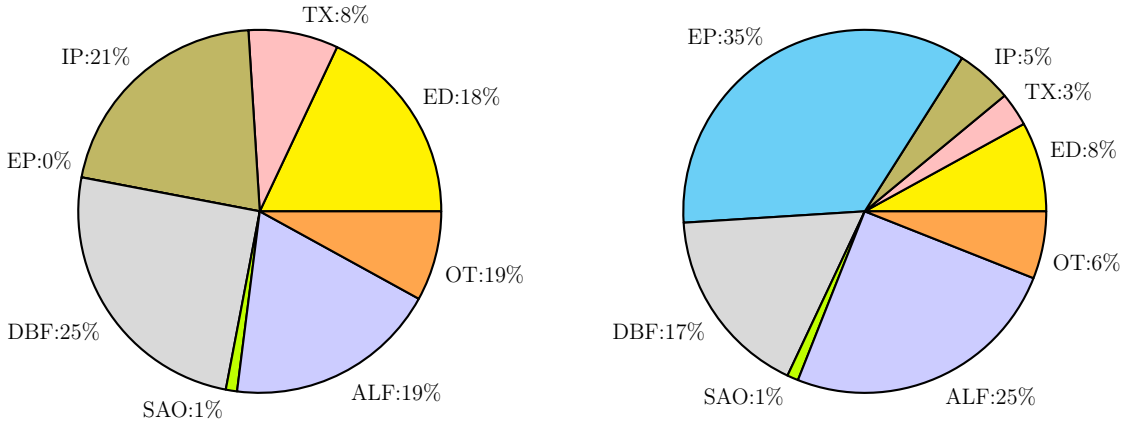


Figure 4.13. Average time distribution for different blocks of the VVC decoder (in %) over Xavier for AI (left) and RA (right) sequences.

Although the profile in % shows the computational load for the individual platform, it cannot present the performance comparison between Ryzen and Xavier. In this situation, a correlation study of the profiling was performed between Ryzen and Xavier to determine the impact of hardware resources on the performance of decoder modules. Figure 4.14 shows the average processing times (in secs) for each decoding block and the ratio between Ryzen and Xavier. Here, ED, TX, IP and OT consumed approximately $\times 2$ more time over Xavier compared to Ryzen for both AI and RA sequences. However, the time consumption by SAO is the same, and ALF roughly $\times 7$ more time over Xavier compared to Ryzen for all sequences and QPs. ALF processing over Ryzen took advantage of larger cache memory as ALF involves repetitive arithmetic operation. On the other hand, Xavier has very

smaller cache memory than Ryzen as a result ALF consumed less time on Ryzen. Lastly, for RA sequences, EP consumed ×3.5 higher on Xavier. The reason for that EP has several filtering operations (e.g. 8-tap interpolation filter) that took advantage of large cache memory.

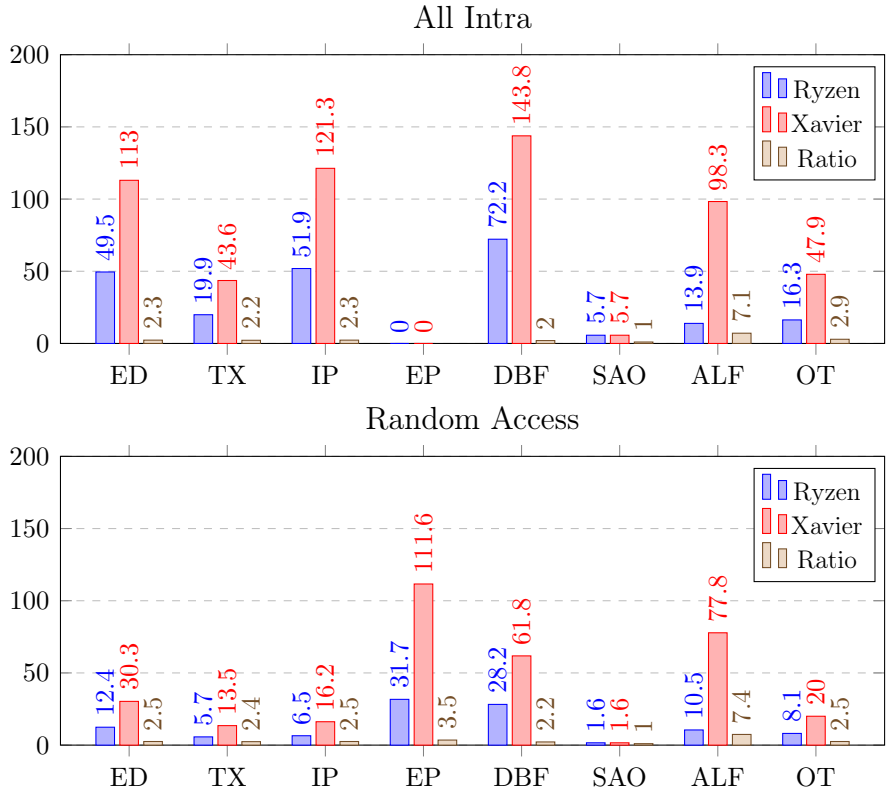
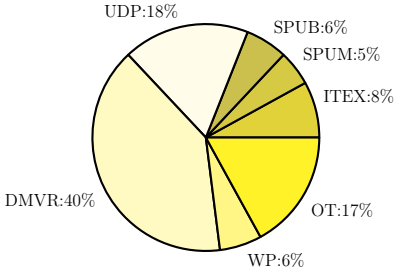


Figure 4.14. Average processing times (in secs) for each decoding block and the ratio between Ryzen and Xavier.

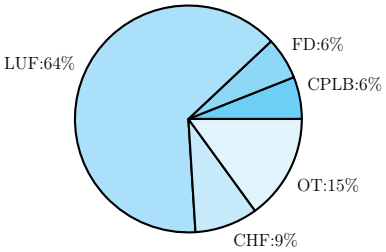
The coarse-grain profile for both platforms shows that EP, DBF and ALF are the most time-consuming decoder modules of the VTM V8.0 decoder. Therefore, a fine-grain profile of EP, DBF, and ALF was performed. The profile of these modules on Ryzen is presented in Figure 4.15. Figure 4.15(a) shows the average time consumption of different parts of EP for RA sequences. Here, the time consumption of Inter Texture (ITEX), Sub-Prediction Unit MC (SPUM), Sub-Prediction Unit Bio (SPUB), Uni-Directional Prediction (UDP), DMVR, Weighted Prediction (WP) and OT represent 8%, 5%, 6%, 18%, 40%, 6% and 17% of the total EP time, respectively.

In Figure 4.15(b) and Figure 4.15(c), the profile of DBF is presented for AI and RA sequences. Luma Filter (LUF) consumed the most DBF time, which represents more than 60% for all sequences. Calculate Position and Length of the Boundaries (CPLB), Filtering Decision (FD), Chroma Filter (CHF) consumed between 6% and 15% for all sequences.

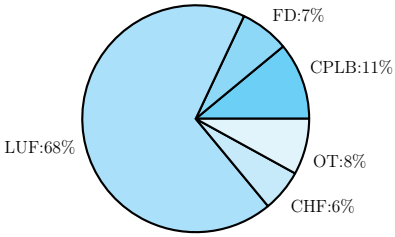
The profile of ALF is shown in Figure 4.15(d) for AI and in Figure 4.15(e) for RA sequences. Derivative Classification (DeC), Luma Component Filtering (LCF) and Chroma Component Filtering (CHCF) consumed approximately 70% of ALF time for AI and RA sequences. However, Cross-Component Filtering (CRCF) consumed double time for AI sequences compared to RA sequences. The ALF time consumption by Copy Reconstructed YUV (CRY), Virtual Boundaries Check (VBC), and OT ranges from 1% to 13%.



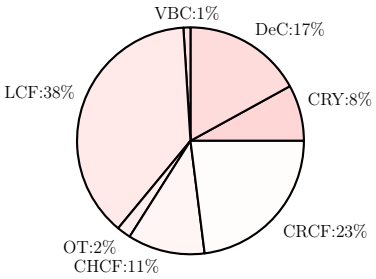
(a) EP for RA sequences.



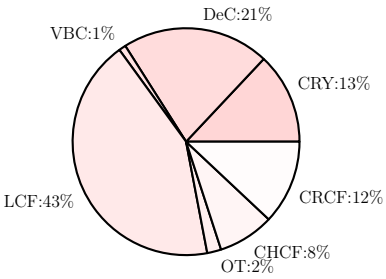
(b) DBF for AI sequences.



(c) DBF for RA sequences.



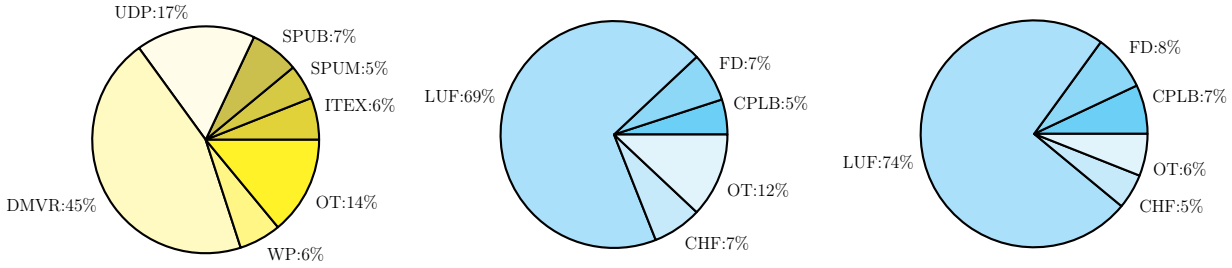
(d) ALF for AI sequences.



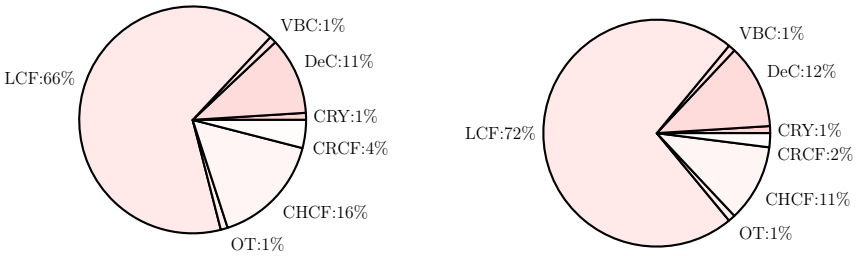
(e) ALF for RA sequences.

Figure 4.15. Average processing times (in %) for the EP, DBF and ALF block profiled over HGPP (Ryzen).

For Xavier, the profiles of EP, DBF, and ALF are shown in Figure 4.16. It can be seen in Figure 4.16(a) that the profile of EP is similar to the profile over Ryzen except DMVR consumed 5% more time. The rest of the parts were slightly differed to accommodate 5%. The scenario is the same for the profile of DBF for both AI (see Figure 4.16(b)) and RA (see Figure 4.16(c)) sequences. The time consumption of LUF over Xavier is 5% and 6% higher than over Ryzen for AI and RA sequences, respectively. The rest of the parts of DBF were slightly varied to adjust the increase percentages of LUF. Furthermore, the profile of ALF is demonstrated in Figure 4.16(d) for AI and in Figure 4.16(c) for RA sequences. LCF consumed 66% and 72% of ALF time for AI and RA sequences, respectively. Furthermore, the comparison of average processing times (in %) for the EP, DBF and ALF block on Ryzen and Xavier is presented in 4.2.



(a) EP for RA sequences. (b) DBF for AI sequences. (c) DBF for RA sequences.



(d) ALF for AI sequences. (e) ALF for RA sequences.

Figure 4.16. Average processing times (in %) for the EP, DBF and ALF block profiled over Xavier.

Table 4.2. Comparison of average processing times (in %) for the EP, DBF and ALF block on Ryzen and Xavier.

AI Sequences											
EP	Ryzen	Xavier	Diff.	DBF	Ryzen	Xavier	Diff.	ALF	Ryzen	Xavier	Diff.
				CPLB	6	5	1	CHCF	11	16	-5
				FD	6	7	-1	CRCF	23	4	19
				LUF	64	69	-5	CRY	8	1	7
				CHF	9	7	2	DeC	17	11	6
				OT	15	12	3	VBC	1	1	0
								LCF	38	66	-28
								OT	2	1	1
RA Sequences											
ITEX	8	6	2	CPLB	11	7	4	CHCF	8	11	-3
SPUM	5	5	0	FD	7	8	-1	CRCF	12	2	10
SPUB	6	7	-1	LUF	68	74	-6	CRY	13	1	12
UDP	18	17	1	CHF	6	5	1	DeC	21	12	9
DMVR	40	45	-5	OT	8	6	2	VBC	1	1	0
WP	6	6	0					LCF	43	72	-29
OT	17	14	3					OT	2	1	1

4.4.2 Generalization

It can be summarised from the profile that EP, DBF, and ALF were the most computationally heavy VVC decoder blocks regardless of the architecture. Moreover, all the

decoder blocks except EP and ALF consumed approximately ×2 more decoding time over Xavier compared to Ryzen as Ryzen has GPP with ×1.63 higher clock speed, more cores and larger cache memory than Xavier. In addition, EP and ALF consumed about ×3.5 and ×7 more time over Xavier compared to Ryzen, respectively. The reason for that is these decoder blocks contains filtering operation with repetitive arithmetic operations which took the best advantage of the larger cache memory of Ryzen. The repetitive arithmetic operations of these blocks made them the most potential candidate for future optimisation. Furthermore, the methodology used in the study for the profiling using the Callgrind profiler and timestamp which can also be used for other software solutions (e.g. HEVC or future and new standards). However, timestamps are more suitable for resource-constraint embedded platforms as it requires very low operating memory and it is easy to target any part of the software. Moreover, it is easy to implement timestamps also for other programs written in C/C++. Lastly, Callgring provides GUI and accurate profile for Linux, Android, and Mac OS operating systems with the cost of higher operating memory requirements.

4.5 Open source decoder optimisation for ARM-based platforms

For open source decoder optimisation, the VVdeC v0.2 decoder (released in December 2020) was selected to evaluate on the HGPP (X-series) and Xavier platforms. This section focuses on three parts: 1) the configuration of the VVdeC v0.2 decoder for ARM-based platforms, since the VVdeC v0.2 decoder did not support ARM-based platforms. 2) Optimising the VVdeC v0.2 decoder for ARM-based platforms as most of the consumer electronics platforms are based on ARM architecture and is also the targeted architecture of the thesis, and 3) the profile of the VVdeC v0.2 decoder with and without SIMD, which shows the impact of the optimisation on different decoding modules. Furthermore, the same methodology was applied to accelerate the OpenVVC v1.0 decoder using Neon-based SIMD optimisation on Xavier and Nano platforms (see section 4.5.4).

4.5.1 Configuration of the VVdeC v0.2 decoder for ARM-based platforms

This section presents a brief description of the migration process of the VVdeC v0.2 decoder to ARM-based platforms. This step is a prerequisite for the study to accelerate

the VVdeC v0.2 decoder using ARM-based platforms, as they were originally designed for x86 GPP. However, the adaptation of the standard code for special environments such as ARM-based platforms is challenging and experience and good knowledge about working on different video algorithms are required. Moreover, the compilation configurations of different video algorithms are very different from each other. To properly configure the ARM-based architecture, several files were modified, added, or eliminated. The main "CMakeLists.txt" file and "CMakeLists.txt" from CommonLib[1], DecoderLib[2] and vvdec source folder were modified. At the beginning of the migration process, all the dependencies of external libraries that do not exist in an ARM-based architecture were removed or adapted. Therefore, optimisations based on x86 GPP were adapted or deactivated, including the disabling of SIMD extensions for x86 processors: SSE and AVX. All these changes were made into the CMakeList.txt files given by VVdeC, which simplifies the compilation process. Then, for avoiding the compilation errors, the flags indicating the target architecture of the processor were deactivated. Finally, the sse2neon.h [113] library was included to convert some SSE instructions such as *_mm_prefetch* due to the fact that they were activated even after disabling SIMD extensions from the CMakeList.txt files.

4.5.2 Fine-grain optimising the VVdeC v0.2 decoder for ARM-based platforms

The VVdeC v0.2 decoder was optimised using ARM Neon-based [70] SIMD intrinsics. The optimisation process begins by defining the decoder modules to be processed using ARM Neon. Therefore, all functions of different decoder modules affected by SIMD are considered and targeted using macros. Then, x86 GPP-based SSE and AVX SIMD intrinsics were substituted using Neon intrinsics. This step helps the compiler to recognise the SIMD register and intrinsics in the ARM-based architecture. However, Neon intrinsic sets are not as powerful as the x86 GPP-based SIMD intrinsics. For example, some x86 GPP-based SIMD intrinsics required two or more Neon intrinsics to perform the same operation (e.g._mm512_fmaddsub_pd, _mm512_fmaddsub_round_pd, etc.). In addition, ARM-based architecture contains a maximum 128-bit SIMD register, while x86 GPP-based architecture contains a 512-bit SIMD register. For this reason, it is difficult to adapt AVX-512 instructions using Neon intrinsics. To handle these difficulties, the SIMDEverywhere (SIMDe) [114] library was adapted to convert the x86 GPP-based SIMD intrinsics to Neon intrinsics. SIMDe supports SSE, SSE2, SSE3, SSSE3, SSE4.1, AVX,

[1]CommonLib contains files used in encoder and decoder side.
[2]DecoderLib contains files used in decoder side only.

AVX2, AVX-512 conversion to Neon intrinsics.

Table 4.3. List of intrinsics extensions used by modules (source:[17]).

Module	Intrinsics used[1]	Utility
All[2]	loadu_si128	loads a value into a register
	add_epi16	adds two registers
	shuffle_epi8	shuffles a value based on a mask
	sub_epi16	subtracts two values
	set1_epi32	set a register to all ones
	prefetch	allocates cache memory
	unpacklo_epi16	interleaves two lower half integers
	unpackhi_epi16	*ibid* high half
	min_epi16	minimum values to a register
	max_epi16	maximum values to a register
	srai_epi32	shift right a number of bits
	packs_epi32	converts types
	storeu_si128/64	stores a register into memory
	setzero_si128	sets all bits to zero
ALF	madd_epi16	multiply&add horizontally the results
	blend_epi16	blenders two values into a masked reg.
DBF	set_epi64x	set a register with a specified value
	srli_epi32	variation of srl_epi32 (shift packed 32-bit integers)
SAO EP	sign_epi16	negates an integer if a given value is also negative
EP	slli_epi16	left shifting
	cvtsi128_si32	copies lower 32-bit of a value
	cvtepi16_epi32	extends sign into a 32-bit value
	bsrli_si128	right shifting

[1] All functions begin by _mm_ [2] Also including ED, IP and TX

The most used SIMD intrinsics in VVdeC are listed in Table 4.3. The list was arranged according to the decoder modules where the intrinsics are classified into decoder modules. Here, the intrinsics used in all decoder modules are included into the 'All' section. In addition, the main functions of the VVdeC v0.2 decoder that were optimised with SIMD are presented in Table 4.4. Here, the percentage of workload for each decoder block was obtained from the profile. In addition, the workload of different decoder modules before SIMD optimisation for AI and RA video sequences is shown in Table 4.4: columns 2 and 3, respectively. It can be seen from Table 4.4 that approximately 40% of ALF and EP were suitable for SIMD optimisation. However, the functions contain repetitive arithmetic, and matrix computation achieved the greatest benefit of SIMD optimisation. The profile of the VVdeC v0.2 decoder with and without SIMD over X-series and Xavier is presented in

the following section.

Table 4.4. Main functions optimised with SIMD implementation in each block for VVdeC v0.2 decoder on Xavier (source:[17]).

Module	Workload AI	Workload RA	Affected functions
ALF	44.5%	38%	cross-component filter chroma component filter luma component filtering Derivative classification
DBF	5.5%	2.5%	luma filter
SAO	2%	1%	offset block function
EP	0%	44.5%	bidirectional optical flow pred. refinement w/ optical flow interpolation filters sub prediction unit Bio weighted prediction uni-directional prediction
IP	16%	3%	prediction angle luma prediction angle chroma planar prediction prediction sample filter
TX	11.5%	2.5%	inverse transformations clipping operations
ED	12%	2.5%	CABAC engine

4.5.3 Profile of the VVdeC v0.2 decoder with and without SIMD

This section presents coarse-grain profiling of the VVdeC v0.2 decoder with and without SIMD on X-series and Xavier. The purpose of this section is to measure the impact of SIMD optimisation on different decoder blocks. The VVdeC v0.2 decoder was profiled in this section using the same profiler and test sequences as in Section 4.4.1.

Figure 4.17 shows the average time distribution for different blocks of VVdeC v0.2 (in %) without SIMD over HGPP (X-series) for AI (left) and RA (right) sequences. The average was taken for QP 22, 27, 32, and 37. ALF consumed most of the decoding time with 54% shares for AI sequences. For RA sequences, ALF consumed 43% and EP consumed 44% of the total decoding time. Furthermore, the average time distribution for different blocks of VVdeC v0.2 (in %) with SIMD over HGPP (X-series) for AI (left) and RA (right) sequences is presented in Figure 4.18. Here, for AI sequences, the percentages of average time consumption by all decoder blocks increased compared to implementation without SIMD except ALF consumed an average 43% less time. The scenario is similar

for RA sequences, where ALF consumes on average 26% less time. Although EP benefited from SIMD optimisation, it consumed the most decoding time. This is due to the fact that the profile is presented in %. The speedup result of SIMD optimisation (in sec.) is presented in Section 5.3.3.1, which explains this phenomenon.

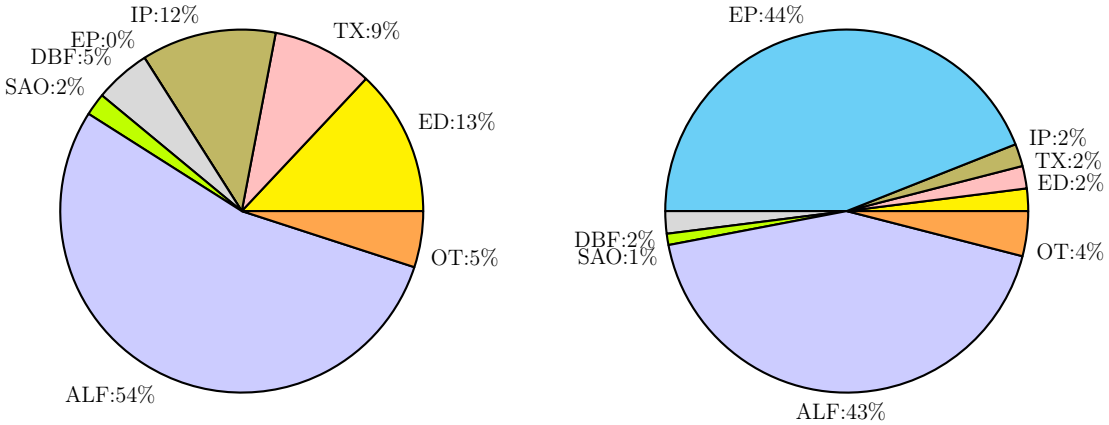


Figure 4.17. Average time distribution for different blocks of the VVdeC v0.2 (in %) without SIMD over the HGPP (X-series) for AI (left) and RA (right) sequences.

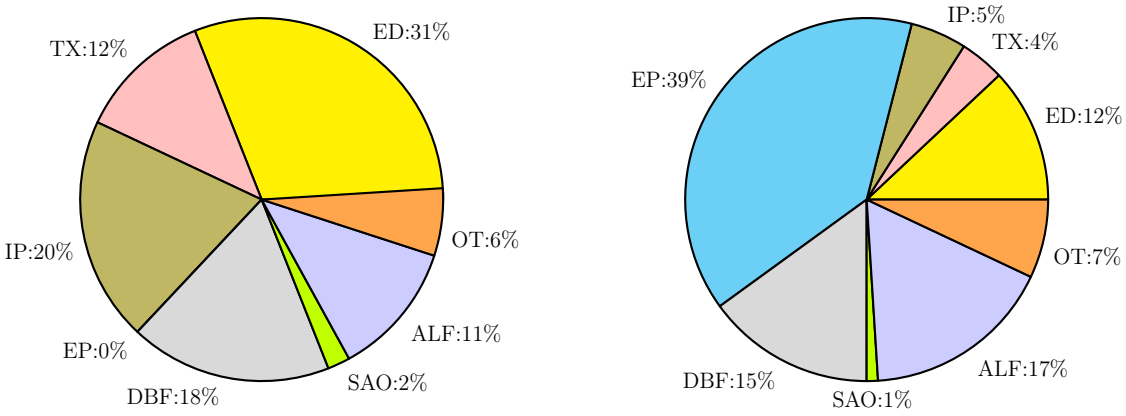


Figure 4.18. Average time distribution for different blocks of the VVdeC v0.2 (in %) with SIMD over the HGPP (X-series) for AI (left) and RA (right) sequences.

The profile result over X-series shows that the impact of SIMD optimisation. Here, EP and ALF benefited from SIMD optimisation (see Section 5.3.3.1). However, IP and DBF were not optimised enough by SIMD. Because, approximately up to 16% of IP and up to 5.5% of DBF modules were suitable for SIMD optimisation. This information is useful to targer the block for future optimisation.

The average time distribution for different blocks of VVdeC v0.2 (in %) over Xavier for AI (left) and RA (right) sequences without and with SIMD is presented in Figures 4.19 and 4.20, respectively. The profile of the VVdeC v0.2 decoder on Xavier shows a pattern

similar to the profile on X-series with and without SIMD. Here, for the implementation with SIMD, the average time consumption of ALF was reduced by 23% for AI and 13% for RA sequences compared to the implementation without SIMD. Moreover, ED, IP and DBF took the least benefit from SIMD optimisation. As a result, the percentages of average time consumption by those blocks increased a lot.

It can be seen that ED, IP, and DBF could not take enough advantage of SIMD optimisation on Xavier. Similarly to the impact of SIMD optimisation on X-series, the blocks EP and ALF were optimised the most by SIMD on Xavier (see Section 5.3.3.1). However, EP and ALF consumed the most decoding time even after SIMD optimisation. Therefore, those blocks were also interesting candidates for future optimisation to further accelerate the decoder.

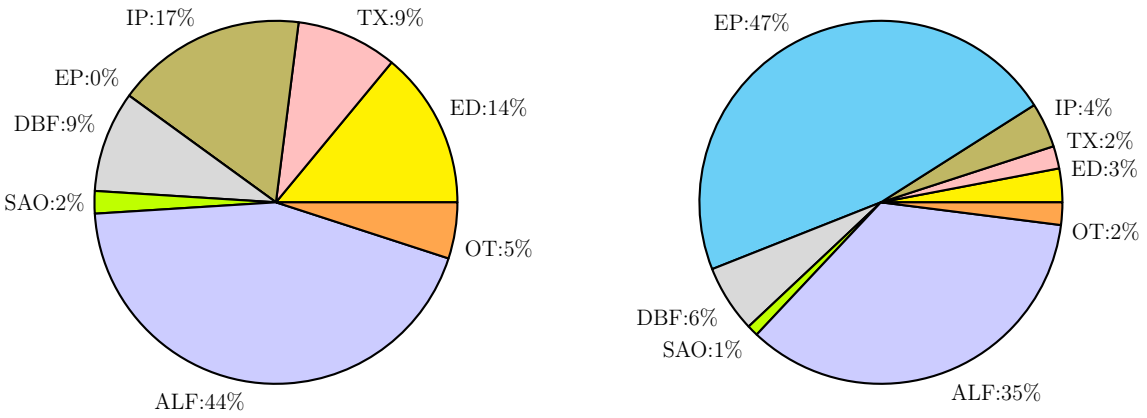


Figure 4.19. Average time distribution for different blocks of the VVdeC v0.2 (in %) without SIMD over Xavier for AI (left) and RA (right) sequences.

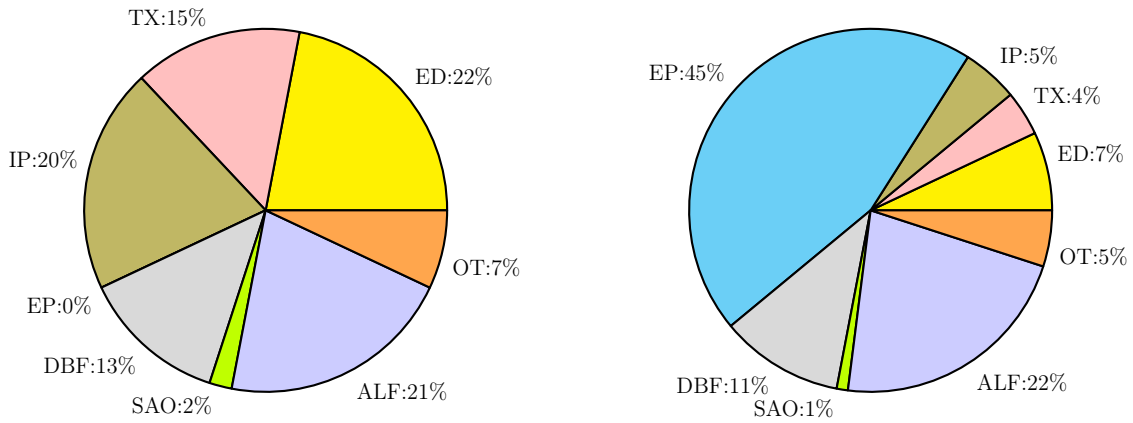


Figure 4.20. Average time distribution for different blocks of the VVdeC v0.2 (in %) with SIMD over Xavier for AI (left) and RA (right) sequences.

4.5.4 Optimising the OpenVVC v1.0 decoder for ARM-based platforms

Unlike VVdeC v0.2 decoder optimisation, no additional configuration is required to optimise the OpenVVC v1.0 decoder, as it supports ARM-based platforms. In this doctoral study, the OpenVVC v1.0 decoder was optimised using Neon-based SIMD intrinsics for ARM-based platforms. The optimisation technique was similar to the VVdeC decoder optimisation presented in Section 4.5.2, where the x86 GPP-based SIMD intrinsics used in OpenVVC were converted to Neon intrinsics by adapting the SIMDe library. This validated that the proposed methodology is applicable for different video algorithms.

The main modules of the OpenVVC v1.0 decoder that were optimised with SIMD are presented in Table 4.5. SIMD intrinsics were used to optimise different modules of TX: DCT-II and VIII, DST-VII, inter-component transform, and low-frequency non-separable transform. These modules contain numerous matrix operations, which was efficiently handled by Neon instrinsics: vmul, vadd, vand, veor, etc [115]. Furthermore, EP block contains several arithmetic and clipping operations which were tackled by vadd, vsub, vmin, and vmax Neon intrinsics. In addition, EP took advantage of vld and vst instrincs to load and store data in larger SIMD registers. Luma 8-tap filters, chroma 4-tap filters, bi-directional optical flow, DMVR, and prediction refinement with optical flow are the modules of EP that greatly benefited by the SIMD optimisation. The most beneficial modules of IP were DC, planar, cross-component linear model, and matrix-based intra prediction. The Neon intrinsics for the store masks, clip, and value offset were used to handle the prediction of pixels inside the picture of IP block. Then, the edge and band filter of SAO were optimised using the intrinsics vceq, vadd, and vsub, since it involves several arithmetical operations. Furthermore, shuffle intrinsic was used to store ALF parameters concurrently. Lastly, ALF utilised the entire 128-bit SIMD register with the help of load and store intrinsics. The results obtained by SIMD optimisation of OpenVVC v1.0 over Xavier and Nano is presented in Section 5.5.

4.5.5 Generalization

In this doctoral thesis, Neon-based SIMD optimisation was applied to accelerate VVdeC and OpenVVC decoders for ARM-based architectures. Since Neon is the standard SIMD suite for ARM-based architectures the proposed solution would be easily adaptable to other or newer platforms. As far as the integration of SIMD instructions into the source code itself is concerned, more manual work might be necessary. However, it is rare that the first open source versions of decoders do not have partial accelerations of their

Table 4.5. Main functions optimised with SIMD in OpenVVC (source:[18]).

VVC Block	Module
TX	DST-VII, DCT-II, DCT-VIII Inter-component transform Low-frequency non-separable transform
EP	Luma 8-tap filters Chroma 4-tap filters Bi-directional optical flow Decoder side motion vector refinement Prediction refinement with optical flow
IP	DC, Planar Cross-component linear model Matrix-based intra prediction module
SAO, ALF filters	Edge and band filter of SAO ALF 7×7 DMS filters for the luma component ALF 5×5 DMS filters for the chroma component Block classification of ALF

functions using SIMD for x86-type architectures. So, in the same way done in this work, it is a matter of adapting these optimisations from one architecture to another. To do this, libraries such as SIMDe can be firstly sought, leaving the manual work reduced to a more detailed optimisation and, above all, guided by the computational load study. Both aspects have been analysed and further developed in this work, validating the proposal on two different software solutions and for two different embedded platforms.

4.6 Algorithm redesign for parallelising the VVdeC decoder using CPU+GPU

The profile presented in Section 4.5.3 shows that ALF block in the VVdeC decoder is one of the most time consuming blocks (in all cases consumed more than 20% of the decoding time) on Xavier, even after SIMD optimisation was applied. Moreover, the ALF block took the most advantage of SIMD optimisation on Xavier. Therefore, VVdeC ALF was considered for further optimisation. This doctoral study proposed a hybrid approach to parallelise VVdeC ALF using CPU+GPU. However, the GPU architecture is different from the CPU architecture, and the VVdeC decoder was written for CPU only. Therefore, a redesign of the algorithm was needed to take advantage of the GPU architecture for

parallelising VVdeC ALF module.

4.6.1 Algorithm redesign

In VVdeC ALF, pixels within CTU are scanned for filtering operation following zigzag raster scan [116] pattern for each 4×4 sub-block as each 4×4 sub-block is classified for one type of filter out of 25 filters (see Section 2.4.8.1). Figure 4.21 demonstrated an example of pixels scanning pattern of VVdeC ALF for 128×128 CTU. It can be seen in the figure that pixel scanning starts from the left top (yellow color) 4×4 sub-block. It also followed zigzag raster scan pattern inside the sub-block (e.g. 0 to 3 then 128 to 131). Therefore, scanning of the next right side (green color) sub-block is started when the scanning of the first sub-block is completed. This process continued until the scanning of the CTU is completed. To execute this scanning pattern, four nested *for* loops are placed one inside another in the ALF 7×7 and 5×5 DMS filters. Two inner loops are used to access the 4×4 pixel block, and two outer loops are used to access the CTU following zigzag raster scan pattern with a four-pixel stride.

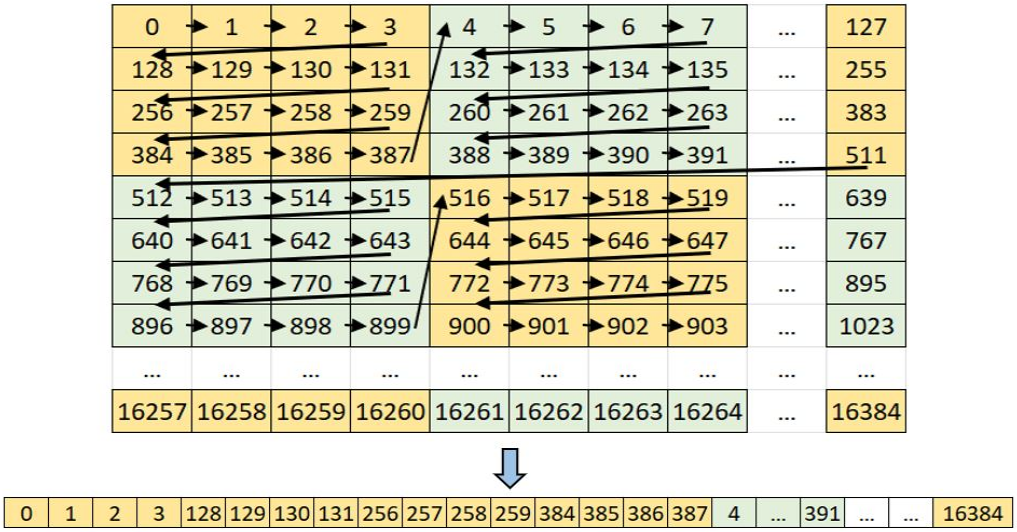


Figure 4.21. Rearranging of data access pattern.

To take advantage of parallel processing capabilities of the GPU, four nested *for* loops of VVdeC ALF were substituted with one *for* loop to access all CTU pixels. In Figure 4.21 (bottom), the proposed access pattern is shown, which made it simpler for data to be transferred from CPU to GPU. Here, data was arranged sub-block by sub-block following zigzag raster scan pattern. The reason for that GPU threads can get all the required data in the adjacent memory location. Moreover, the GPU-based implementation filters the pixels in the same way as the CPU-based implementation, as the number of GPU threads

was replaced by the size of the *for* loop. Only data pointer addresses were changed, which is presented in Section 4.6.1.1. The implementation maximised the use of GPU for VVdeC ALF filtering by setting threads per block equal to 128. Furthermore, the number of blocks was calculated using the total number of pixels divided by threads per block (=128).

4.6.1.1 Data ordering

Data ordering was one of the most important tasks for GPU-based implementation of VVdeC ALF as data transferring between CPU and GPU may imply important bottlenecks. In this study, three different data ordering approaches were implemented and evaluated for GPU-based implementation of VVdeC ALF.

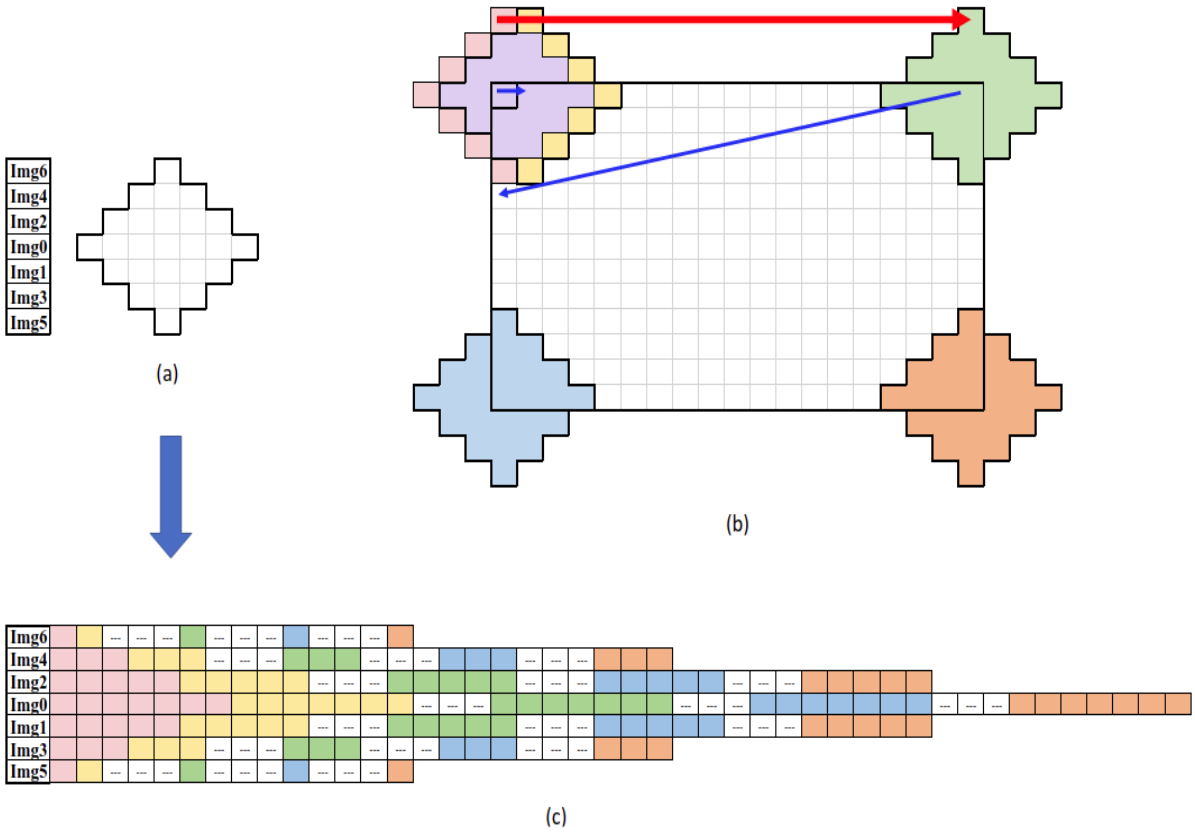


Figure 4.22. a) 7×7 DMS, b) sliding DMS filter over CTU, c) data ordering pattern in the first approach.

In all cases, ALF 7×7 DMS filter (see Figure 4.22a) (see Section 2.4.8.2) needs up to three pixels in four directions: top, bottom, left, and right. In Figure 4.22a and c, ImgX represents the rows of the DMS filter. In addition, the 5×5 DMS filter is exactly like the 7×7 DMS filter with the exception that it needs up to two pixels in four directions. The

scanning process of the DMS filter is shown in Figure 4.22b. It is exactly the same scanning zigzag pattern shown in Figure 4.21, Where DMS filter scans sub-blocks horizontally as the red arrow shows and then follows the blue arrow. In addition, inside the sub-blocks it follows the same raster scan based zigzag pattern shown in Figure 4.21. This pattern continues until the scanning of CTU is finished.

- In the first approach of data ordering, all sliding pixels of each row of the DMS filter were stored in one array, as shown in Figure 4.22c. In this case, the same pixel information was copied multiple times for the purpose of filtering neighbouring pixels.
- To avoid multiple copies of pixel information, the second approach was adapted. Here, all pixel information of each ImgX (X="0 to 6") row was copied for each CTU row. This approach prevented duplicate copy of pixel information, where horizontal neighbouring pixels share the same pixels data. However, the vertical neighbouring pixels used different pixel data. Here, the same pixel information was duplicated vertically. Although this approach copies fewer data than approach one, data are copied multiple times.
- Finally, a third approach was applied to prevent duplication of pixel data. Here, the CTU pixel information was copied between CPU and GPU along with three pixels in the top, bottom, left, and right directions, as illustrated in Figure 4.23a. In each corner, a total of six additional (3 blue, 2 pink, and 1 purple) pixels surrounding the CTU were copied because they are also required for filtering the pixels along the CTU's boundary. Then, 2D pixel data of CTU were converted to a 1D array, as illustrated in Figure 4.23b. Although these extra pixels were copied from CPU to GPU, they were not used for the 7×7 and 5×5 DMS filters. Copying these extra pixels simplifies the data addressing process in GPU.

The benefits of using this last approach are the following.

- There is only one copy of each pixel (the minimum possible).
- All pixel information is stored in a single memory vector.
- It has very low implementation complexity.

The drawbacks of using the approach are the following.

- A thread cannot have coalescent access to the information of pixels if the pixels belong to a different row as there is a stride of 6 + CTU width.

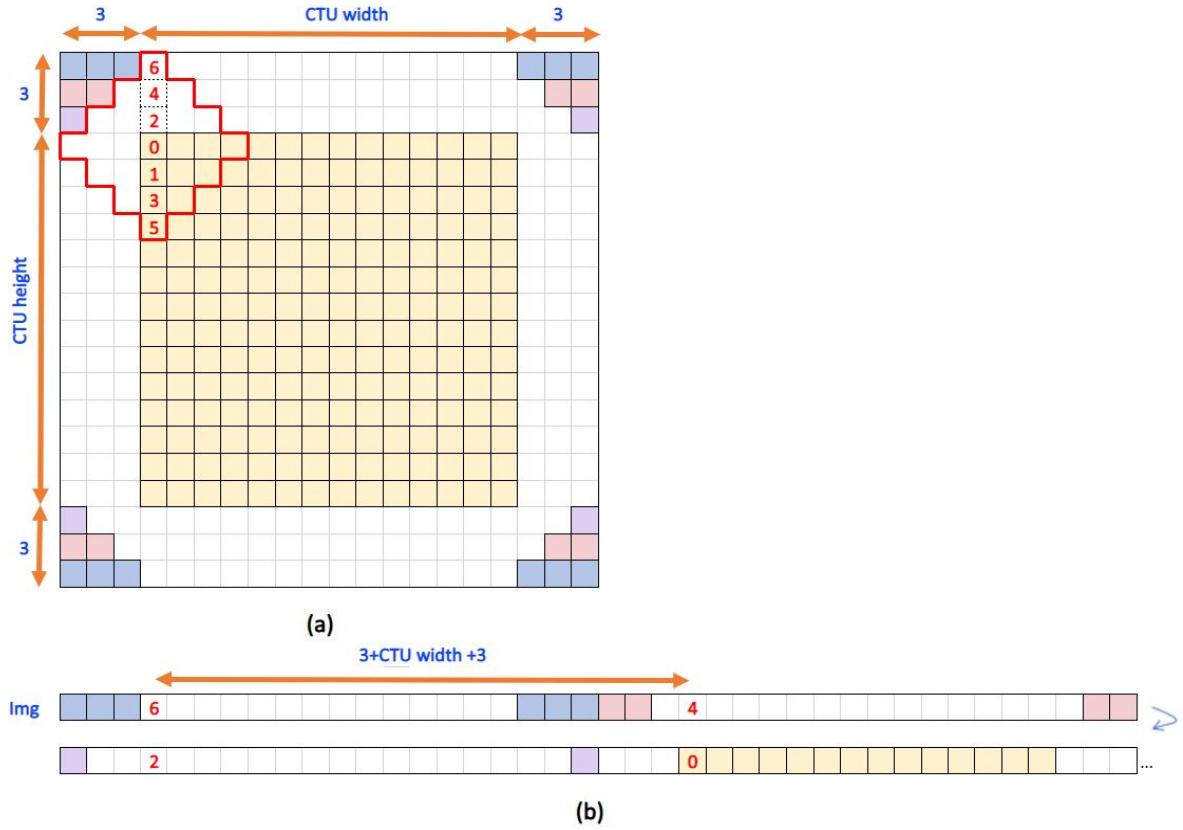


Figure 4.23. Data ordering pattern in the final approach.

A comparison among three data ordering approaches is presented in Table 4.6 for 128×128 and 64×64 CTU sizes. Here, data transfer was calculated in bytes. Furthermore, approach 3 and approach 2 obtained more than 95% and 70% reductions in data copy time between CPU and GPU compared to approach 1 for both CTU sizes, respectively.

Table 4.6. Reduction of data copied on the basis of different data ordering approaches.

	Transferred data (in bytes)			Reduction over approach 1 (in %)	
CTU size	Approach 1	Approach 2	Approach 3	Approach 2	Approach 3
128×128	409600	116992	17956	71.4%	95.6%
64×64	102400	29824	4900	70.8%	95.2%

4.6.2 Parallelise the VVdeC ALF filtering in GPU

Parallel processing of VVdeC ALF filtering using GPU involves several factors: memory allocation, data transfer, kernel distribution and task schedule. Detailed descriptions of these factors are presented in the following sections.

4.6.2.1 Memory allocation

Memory allocation is one of the important factors for the parallel processing of VVdeC ALF filtering using GPU. The reason is that GPU does not have access to the CPU's memory, even though it processes the task with the instruction received from CPU. Therefore, data transfer between CPU and GPU causes the main bottleneck for CPU+GPU-based implementation [117] [118]. In this thesis, different memory allocation techniques of CUDA API were studied to optimise the performance.

CUDA offers unified memory allocation function named *cudaMallocManaged*, which allows both CPU and GPU to access data from a single memory address. It is simpler to use, as it replaces the data allocation and copy function with only one function. However, two copies are needed under the hood to perform the memory allocation by *cudaMallocManaged* [119]: 1) unified memory to pinned memory [120] and 2) pinned memory to device. On the other hand, *cudaMallocHost* allocates page-locked memory to the host that the device can access. It offers higher bandwidth for read and write operations by device than that of pageable memory. Moreover, it comes with the constraint that performance decreases when a large amount of data is allocated. However, the allocation with *cudaMallocHost* for CPU+GPU-based VVdeC ALF implementation provides better performance than *cudaMallocManaged* as a little amount of data per filter (about 100 KB) is required. Therefore, *cudaMallocHost* was used in the final implementation of this study.

4.6.2.2 Data transfer

As mentioned in previous Section 4.6.2.1, *cudaMallocHost* was used for data transfer between CPU and GPU, which allows the use of the *memcpy* function for coping the data. The *memcpy* function consumed less copy time and faster data transfer than the *cudaMemcpy* function for CPU+GPU-based VVdeC ALF implementation. Furthermore, to parallelise data copy and CPU execution, *cudaMemcpyAsync* was examined. However, performance decreased due to the fact that *cudaMemcpyAsync* consumed approximately 18 μs setup overhead. In contrast, when *memcpy* was used synchronously, the copy time for all variables of a filter was hardly around 10 μs. Therefore, it can be concluded that to take advantage of *cudaMemcpyAsync* a large amount of data ($\geq 100MB$) is needed. Subsequently, *CudaMemPrefetchAsync* was examined, which is better than *cudaMemcpyAsync* as it spend 14.5 μs less setup overhead. But the performance was still better with *memcpy* than *CudaMemPrefetchAsync*, as *CudaMemPrefetchAsync* spends 3.5 μs when it executed.

Finally, *memcpy* was included in the final version of CPU+GPU-based VVdeC ALF implementation to copy data synchronously when the data were allocated by *cudaMallocHost* to obtain the highest performance.

4.6.2.3 Kernel distribution

ALF filtering tasks are executed in GPU by the GPU kernels. Therefore, it is important to study different kernel distributions to maximise process parallelisation so that optimum filtering performance can be obtained.

All the SAO reconstructed pixels are delivered to GPU from CPU before starting the ALF filtering in GPU as these data are needed to begin the ALF filtering. Then, all pixels within one CTU are filtered using appropriate filters between 7×7 and 5×5 DMS filters. The concurrent processing of ALF CTUs is allowed because ALF CTUs do not depend on each other.

In the first approach, one kernel was designed to execute the 7×7 DMS filter on the luma component, and if needed, the 5×5 DMS filter on the chroma Cb and Cr components of each CTU. This approach reduce the required time for kernel launching up to $3\times$ if luma and chroma filters are applied. It has a great impact on performance, as the kernel consumes some initialisation time before performing any execution in GPU. For example, kernel initialisation consumed 180μs when 8 CPU threads simultaneously call a kernel, while kernel execution consumed 7μs for processing the 7×7 DMS filter in GPU. In this approach, CPU implementation of ALF filtering performed better than the implementation of GPU. Therefore, the following approach was implemented, where one kernel was designed to perform all filters in a frame. In the beginning, all necessary data was loaded from CPU to GPU using a combination of *cudaMallocHost* and *memcpy*. Then, the exact same filtering operations in CPU were performed by the GPU kernel in parallel. Therefore, the frame is marked as reconstructed and the GPU thread is available to attend the next task. In addition, the Cb and Cr components of CCALF were loaded into GPU and added to the Cb and Cr components of the 5×5 DMS filter. Subsequently, the clipping operation was performed on the results obtained in the previous step, and the results were sent to CPU after synchronisation. Figure 4.24 shows the diagram of the hybrid approach based on CPU + GPU. This hybrid approach obtained better performance than the implementation of CPU only because the parallel processing of multiple CTUs were performed by one GPU kernel. For example, with this approach, the UHD sequences with 3840×2160 pixels can be decoded by performing 500 7×7 and 1000 5×5 DMS filters in one kernel.

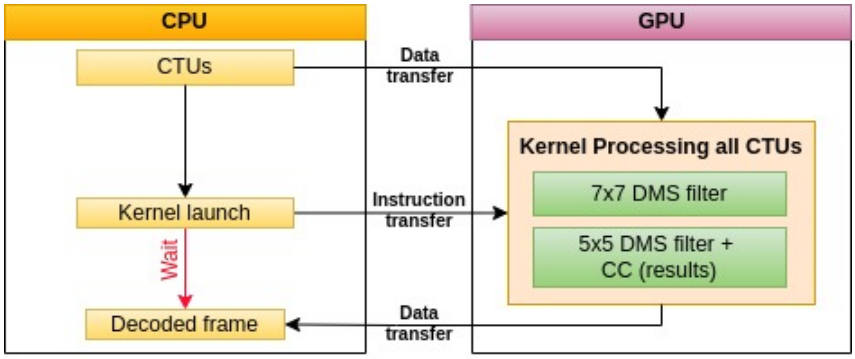


Figure 4.24. Diagram of hybrid approach using CPU and GPU.

4.6.2.4 Task schedule

There are two factors in the CPU+GPU-based hybrid approach that need to be efficiently managed to maximise the performance of the decoder: 1) GPU task scheduling and 2) transferring data between CPU+GPU. In this study, these factors were managed using the double buffer technique.

Initially, the CPU thread had to wait after the launch of the GPU kernel and until copying back the filter data from GPU to CPU. This is not a real CPU+GPU parallelisation, where the rest of the CPU threads cannot execute operations in parallel. The double buffer technique was used to achieve efficient CPU+GPU parallelisation of VVdeC ALF. Figure 4.25 presents the diagram of the GPU task scheduling with the example of two CPU threads, where two memory buffers were created using *cudaMallocHost*: 1) memory buffer 1 and 2) memory buffer 2. Here, both memory buffers were used by the CPU threads to copy the essential data (e.g. pixel data, filter coefficients, CTU size) to GPU for filtering (see dashed black arrow in Figure 4.25). The memory buffer is then released when the GPU kernel is launched (see green arrow for thread 1 and red arrow for thread 2 in Figure 4.25). In this way, the memory buffer became available for other CPU threads to continue copying the data. Finally, the image was set as reconstructed after the GPU kernel completed its operations and returned the results to CPU (see blue arrow in Figure 4.25). This strategy was replicated for all CPU threads when the GPU kernel was launched. Furthermore, the double buffer technique is capable of automatically managing the storage capacity if more data is needed to be stored by the filter.

4.6.3 Generalization

In this doctoral thesis, the VVdeC decoder was optimised using platforms with CPU+GPU, where ALF block was processed in GPU and the rest of the decoder blocks were processed

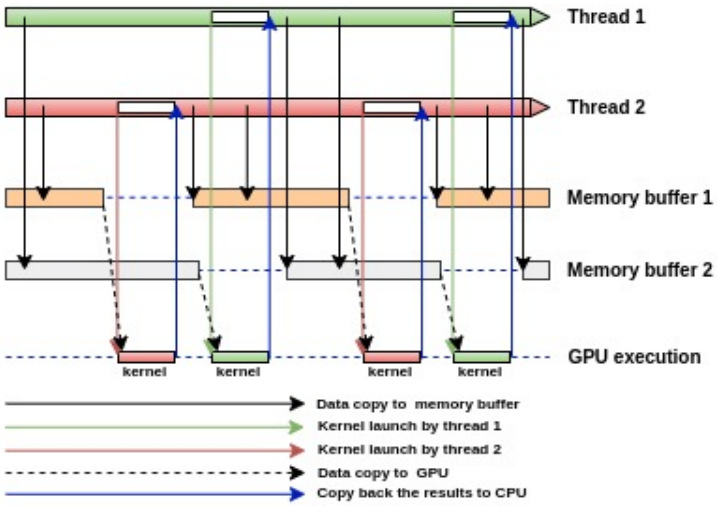


Figure 4.25. Diagram of the GPU task scheduling.

in CPU. Here, the algorithm redesign approach is proposed to take advantage of the parallel processing capabilities of GPU to accelerate VVdeC ALF. This approach is also applicable to other algorithms that use raster scan for accelerating using GPU. In addition, this approach will help other algorithms that exploited GPU to reduce memory usage by lowering data transfer between CPU and GPU. Furthermore, the proposed methodology will be suitable for other types of GPU as here the number of data transfers was optimised by improving the data access pattern, which makes the implementation straightforward for other GPUs.

4.7 Memory management

Memory usage and memory transfer are the very essential aspects of video processing over resource-constrained heterogeneous platforms. Section 4.6.1 discusses 3 approaches to reduce the memory transfer requirement that helps to obtain efficient performance using CPU and GPU. In addition, memory usage information is also needed to manage available resources.

4.7.1 Memory usage

For video processing operations, memory usage is one of the important factors for heterogeneous embedded platforms with limited resources. In this thesis, the maximum memory usage was measured using the GNU/Linux command */usr/bin/time -f %M* for

Xavier and Nano. This command provides maximum memory usage (in kilo bytes) of Random-Access Memory (RAM) during video processing operations [124]. The same methodology can be applied to other platforms configured with GNU/Linux for measuring the maximum memory usage. In this thesis, the maximum memory usage was only calculated for different video decoders, because the focus of the thesis was mainly based on optimising different video decoders on different heterogeneous platforms to extract a work methodology. Yet, the information of the maximum memory usage by video decoder is very essential to assess the portability of different platforms with limited resources. The results of memory usage are presented in Section 5.10.

4.8 Energy consumption

Energy consumption is also an important factor for heterogeneous embedded platforms with limited resources. In this study, the energy consumption analysis of video decoding was performed on Xavier and Nano platforms, since both heterogeneous embedded platforms have limited resources. Both platforms are integrated with a built-in power monitor [125] that provides power consumption readings in milliwatts (mW) for CPU and GPU. For the energy consumption analysis of video decoding, the average energy consumption (in Joule) was measured from the average power consumption during video decoding multiplied by the total decoding time as shown in Equation 4.1.

$$Energy\_Consumption = Power\_Consumption * Decoding\_Time \quad (4.1)$$

The same methodology can be applied to calculate energy consumption on other platforms if the hardware provides the monitor. Moreover, other latest platforms from different manufacturers also contain built-in power monitor. In this thesis, energy consumption was measured but not optimised. The reason for that this study mainly focused on optimising different video decoders on different heterogeneous platforms.

However, at the time of writing this doctoral thesis, this data related to energy consumption is being heavily used to construct machine learning models which may adjust and improve the energy consumption. This point is introduced as future work in Section 8.2.

4.9 Summary of the proposed methodology

The platform and the video algorithm were selected in the beginning of the implementation of video decoders over heterogeneous platforms, The coarse and fine-grain profile of the selected video algorithms was carried out on the selected platforms to identify the most time-consuming blocks. Then, Neon-based SIMD optimisation was implemented for VVdeC and OpenVVC decoders. Furthermore, the hybrid approach was implemented to accelerate the VVdeC decoder where the fine-grain SIMD optimisation and CPU+GPU based hardware accelerator were used along with the default coarse-gain optimisation in the development of the hybrid approach. In CPU+GPU, algorithm redesigning was proposed to accelerate VVdeC ALF. Lastly, the methodology to measure memory usage and energy consumption were presented.

The obtained experimental results using the methodology proposed in this Chapter are presented in Chapter 5. In addition, Chapter 6 synthesises and collected these proposals into the design methodology.

Chapter 5

Experimental Results

In the previous chapter, the implementation and optimisation of different VVC decoders on different platforms was presented in detail. This chapter will show the experimental results obtained in this thesis work, including performance analysis, memory usage, and energy consumption. Here, the test bench description and platform setup are included in Section 5.1. The native GCC auto vectorizer based performance analysis is presented in Section 5.2. The performance results without/with SIMD optimisation and GPU optimisation of VVdeC decoder is demonstrated in Section 5.3, 5.4 and 5.8. During the doctoral study, different versions (v0.2, 1.0, 1.1, 1.2, 1.3) of the VVdeC decoder were accelerated using SIMD optimisation and GPU (see Section 5.8). However, in this chapter, only the results of the VVdeC v0.2 and the latest v1.3 (see Section 5.4) version are presented to avoid repeating the same results. Similarly, only the results with SIMD optimisation of OpenVVC v1.0 are included, but not the results for v0.2 and 0.3 to avoid repetition (see Section 5.5). Therefore, the results of the performance analysis is compared in Section 5.7 for the VVdeC v1.3 and OpenVVC v1.0 decoder. Then, the results of the comparative analysis on memory usage and energy consumption are presented in Sections 5.10 and 5.11.1 for the VVdeC v1.3 and OpenVVC v1.0 decoder, respectively. Furthermore, a study of energy consumption is included in Section 5.11.2 to compare CPU and CPU+GPU implementations with SIMD optimisation. Lastly, a discussion of the experimental results is provided in Section 5.12. In Table 5.1, a summary of the organisation of the sections is provided. Here, the columns contain the following information of the section: decoder, sequence set (see Section 5.1.1), platform (see Section 5.1.2) and optimisation used.

Table 5.1. Summary and organisation of sections in Chapter 5.

Section	Decoder	Sequences	Platforms	SIMD	Tile	GPU
5.3	VVdeC v0.2	Set A	X-series, Xavier	Yes	—	—
5.4	VVdeC v1.3	Set A	Xavier, Nano	Yes	—	—
5.5	OpenVVC v1.0	Set B	Xavier, Nano	Yes	Yes	—
5.6	OpenVVC v1.0	Set B	Xavier, Nano	Yes	Yes	—
5.7	OpenVVC v1.0, VVdeC v1.3	Set B	Xavier, Nano	Yes	Yes	—
5.8	VVdeC v1.3	Set A	Xavier	Yes	—	Yes
5.9	VVdeC v1.3	Set A	Xavier, Nano	Yes	—	Yes
5.10	OpenVVC v1.0, VVdeC v1.3	Set B	Xavier, Nanoo	Yes	Yes	—
5.11.1	OpenVVC v1.0, VVdeC v1.3	Set B	Xavier, Nano	Yes	Yes	—
5.11.2	VVdeC v1.3	Set A	Xavier, Nano	Yes	—	Yes

5.1 Test bench description and platform setup

This section presents the description of the test bench used in the doctoral thesis and the platform setup needed to run the video decoders on targeted heterogeneous embedded platforms.

5.1.1 Test bench description

A total of fifteen video sequences were used in this doctoral thesis. These sequences are from the common JVET test sequences [126]. Three sequences of Class A1 with resolution 3840×2160: Tango2 (TG2), FoodMarket4 (FM4), and Campfire (CMF); three sequences of Class A2 with resolution 3840×2160: CatRobot1 (CR1), DaylightRoad2 (DR2), and ParkRunning3 (PR3); and six sequences of Class B with resolution 1920×1080: MarketPlace (MPL), RitualDance (RUD), Cactus (CCT), BasketballDrive (BBD), BQTerrace (BQT), and ArenaOfValor (AOV); three sequences of Class E with resolution 3840×2160: FourPeople (FRP), Johnny (JNY), and KristenAndSara (KAS). All test sequences were encoded with QP 22, 27, 32 and 37 for the AI and RA configurations. The features of these sequences are presented in Table 5.2.

In this thesis, Set A sequences were mainly used for most of the experiments as all the decoders support frame partitioning. A second set of sequences, Set B, was also used. This set was encoded using a 8 titles with 4×2 configuration and has been used to test OpenVVC decoder as it supports tile parallelism. In Set B sequences only QP 27 and 37 (see Table 5.3) have been used. Lastly, the bit depth was set to 10 for all the sequences of both sets.

Table 5.2. Features of the test video sequences Set A.

Class	Sequence	Resolution	No. Frames	Bitdepth
A1 (UHD)	Tango2 (TG2)	3840×2160	294	10
	FoodMarket4 (FM4)	3840×2160	300	10
	Campfire (CMF)	3840×2160	300	10
A2 (UHD)	CatRobot1 (CR1)	3840×2160	300	10
	DaylightRoad2 (DR2)	3840×2160	300	10
	ParkRunning3 (PR3)	3840×2160	300	10
B (FHD)	MarketPlace (MPL)	1920×1080	600	10
	RitualDance (RUD)	1920×1080	600	10
	Cactus (CCT)	1920×1080	500	10
	BasketballDrive (BBD)	1920×1080	500	10
	BQTerrace (BQT)	1920×1080	600	10
	ArenaOfValor (AOV)	1920×1080	600	10
E (HD)	FourPeople (FRP)	1280×720	600	10
	Johnny (JNY)	1280×720	600	10
	KristenAndSara (KAS)	1280×720	600	10

Table 5.3. Features of the test video sequences Set B.

Class	Sequence	Resolution	No. Frames	Bitdepth
A1 (UHD)	Tango2 (TG2)	3840×2160	294	10
	FoodMarket4 (FM4)	3840×2160	300	10
	Campfire (CMF)	3840×2160	300	10
A2 (UHD)	CatRobot1 (CR1)	3840×2160	300	10
	DaylightRoad2 (DR2)	3840×2160	300	10
	ParkRunning3 (PR3)	3840×2160	300	10
B (FHD)	MarketPlace (MPL)	1920×1080	300	10
	RitualDance (RUD)	1920×1080	300	10
	Cactus (CCT)	1920×1080	300	10
	BasketballDrive (BBD)	1920×1080	300	10
	BQTerrace (BQT)	1920×1080	300	10
	ArenaOfValor (AOV)	1920×1080	300	10
E (HD)	FourPeople (FRP)	1280×720	300	10
	Johnny (JNY)	1280×720	300	10
	KristenAndSara (KAS)	1280×720	300	10

5.1.2 Platform setup

Two HGPPs: Ryzen and X-series (see Section 4.2.1.1); and two heterogeneous embedded platforms: Xavier and Nano (see Section 4.2.1.2) were configured with the Ubuntu

18.04 operating system, GCC version 7.5, and CMake [127] version 3.16.5. In addition, the compiler optimisation -O3 was activated for all platforms to achieve the maximum performance in terms of speed. The detailed analysis of the GCC auto vectorizer [128] is presented in Section 5.2. Furthermore, two embedded heterogeneous platforms: Xavier and Nano were configured with CUDA (see Section 3.2.1.1) version 10.2. All experiments were performed using the maximum clock rate of the CPUs and the GPUs.

It is worth mentioning that the Ryzen platform was mainly used only in the early stages of the development of the thesis. Specifically, it was used in the evaluation of the first versions of the VTM software, analysing the computational load of the main blocks integrating the decoder. The results of these analyses were published in [16] (See Section 7.3). For this reason, and in order not to overextend this dissertation, no further details of this study are included here.

5.2 Performance analysis of the native GCC auto vectorizer

This section presents the performance analysis of the VVdeC decoder over Xavier without SIMD using the different combinations of the native GCC auto vectorization[1] options. The purpose of this study was to evaluate the performance of different auto vectorization options on the targeted heterogeneous platforms and to choose the best option for all future experiments. Here, the average FPS obtained for all sequences shown in Section 5.3.1 with QP 27 and QP 32 for different combinations of native GCC auto vectorization options "-O1", "-O2", "-O3", "-Os" (optimise for size), "-Ofast" (disregard strict standards compliance with "-O3"), and "-ftree-vectorize" (performs vectorization on trees) is provided in Table 5.4. It can be seen that "-Os" was the worst of all options. On the contrary, "-O3" was the best option for all configurations, although the combination of "-O3" and "-ftree-vectorize" was slightly better (0.1 FPS) for AI sequences with QP 27. Moreover, the performance using "-Ofast", and also the combination of "-Ofast", and "-ftree-vectorize" options was similar to that using "-O3" option for some configurations. Therefore, the resulting fastest option "-O3" was selected to use in all experiments over all platforms.

[1]GCC compiler automatically looks for loops and transforms loops to vector operations if auto vectorization flags are enabled [129]. An example of vectorization operation is given in Section 3.1.1.

Table 5.4. Performance (in FPS) obtained by VVdeC decoder v0.2 over Xavier for different combination of the native GCC auto vectorization options using 8 cores without SIMD.

Average FPS	AI Sequences		RA Sequences	
	QP 27	QP 32	QP 27	QP 32
O1	15.0	17.7	16.6	19.0
O2	15.5	17.4	16.8	19.3
O3	15.9	18.7	17.8	20.2
Os	13.2	15.6	15.8	18.1
Ofast	15.9	18.7	17.3	19.8
O1-ftree	15.6	18.3	16.6	19.0
O2-ftree	15.7	18.4	17.1	19.6
O3-ftree	16.0	18.7	17.5	20.0
Os-ftree	13.2	15.5	15.8	18.1
Ofast-ftree	15.9	18.6	17.3	19.7

5.3 Performance and speedup analysis of VVdeC v0.2 decoder

This section contains performance analysis of VVdeC v0.2[2] decoder (released on Dec. 2020) with and without SIMD, speedup analysis for SIMD, and comparison analysis of speedup for vectorization and SIMD.

5.3.1 Preliminary analysis of VVdeC v0.2

This section presents the global performance (in FPS) of VVdeC v0.2 for different numbers of cores available on X-series and Xavier without SIMD[3]. Here, CMF, DR2, FM4, PR3, BBD and BQT test sequences of Set A (see Table 5.2) were used for all experiments. Table 5.5 shows the performance (in FPS) obtained for VVdeC v0.2 decoder over X-series without SIMD optimised for 1 to 15, 18 and 20 threads. Here, it can be seen that real-time decoding was obtained for some UHD sequences with 20 threads, and for all FHD sequences with 10 threads, except BQT QP22. In the following section of this document only average results are presented due to improve the readability of the document and not to overextend this dissertation.

[2]The reason for choosing VVdeC v0.2 for this experiment is that it was the available version of VVdeC on the time of experiment (Dec. 2020).

[3]VVdeC v0.2 does not support the ARM-based platform and is not SIMD optimised for the ARM-based platform. Therefore, this preliminary analysis provides the reference performance for speedup analysis.

The presented results were averaged for all sequences and for four QPs (22, 27, 32, and 37). Figure 5.1 shows the average performance in FPS and the speedup obtained from the decoding of VVdeC v0.2 using 1 to 20 threads over X-series for AI (left) and RA (right) sequences without SIMD. For AI sequences, the average between 5.1 and 43.5 FPS was obtained using between 1 and 20 threads without SIMD. In addition, a 8.5 speedup was achieved using 20 threads. Furthermore, 50.2 FPS with a 9.7 speedup was obtained using 20 threads for RA sequences without SIMD.

Table 5.5. Performance (in FPS) obtained for VVdeC v0.2 decoder over X-series for different threads without SIMD optimisations.

	Seq.	QP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	18	20
All Intra	CMF	22	2.3	3.1	4.5	6.0	7.5	8.9	10.4	11.7	13.1	14.3	14.6	14.9	15.2	15.5	15.7	16.5	16.9
		27	2.6	4.1	6.1	7.9	9.6	11.4	13.3	14.9	16.8	18.4	18.8	19.1	19.4	19.7	20.0	21.0	21.5
		32	3.0	5.4	8.0	10.3	12.5	14.9	17.3	19.4	21.8	24.1	24.5	24.9	25.3	25.6	26.0	26.9	27.6
		37	3.0	5.9	8.7	11.5	14.3	17.0	19.8	22.2	25.0	27.6	28.0	28.5	28.9	29.2	29.6	30.4	31.4
	DR2	22	2.0	2.5	3.7	4.9	6.1	7.3	8.4	9.5	10.7	11.7	11.9	12.1	12.3	12.5	12.8	13.5	13.9
		27	2.7	4.6	6.7	8.7	10.5	12.5	14.5	16.3	18.3	20.2	20.6	20.9	21.3	21.6	21.9	22.8	23.3
		32	2.9	5.2	7.6	9.9	12.0	14.2	16.6	18.6	20.9	23.1	23.4	23.9	24.2	24.5	24.9	25.9	26.5
		37	3.1	5.8	8.4	11.1	13.9	16.5	19.2	21.5	24.1	26.7	27.2	27.6	28.0	28.5	28.8	29.9	30.6
	FM4	22	2.6	4.3	6.3	8.3	10.3	12.3	14.3	16.0	18.0	19.8	20.2	20.5	20.8	21.1	21.4	22.3	22.9
		27	2.9	5.1	7.5	9.8	11.8	14.1	16.4	18.3	20.5	22.7	23.1	23.5	23.9	24.2	24.4	25.5	26.1
		32	3.1	5.6	8.2	10.7	12.9	15.4	17.9	20	22.5	24.9	25.3	25.7	26.1	26.3	26.6	27.8	28.4
		37	3.1	6.0	8.8	11.6	14.4	17.1	20.0	22.4	25.2	27.8	28.3	28.7	29.1	29.5	29.8	30.8	31.6
	PR3	22	2.1	2.9	4.2	5.6	7.0	8.3	9.7	10.9	12.2	13.3	13.6	13.8	14.1	14.3	14.5	15.3	15.6
		27	2.3	3.6	5.3	6.8	8.2	9.8	11.4	12.7	14.3	15.7	16.1	16.4	16.7	16.9	17.2	18.0	18.4
		32	2.5	4.2	6.1	7.9	9.6	11.4	13.3	15.0	16.7	18.5	18.8	19.1	19.5	19.8	20.0	20.9	21.5
		37	2.8	4.8	7.0	9.2	11.5	13.7	15.9	17.9	20.1	22.2	22.5	22.9	23.2	23.6	23.9	24.9	25.5
	BBD	22	8.4	11.8	17.3	22.6	28.3	33.9	39.5	45.0	50.6	55.3	56.3	57.2	58.1	59.0	59.9	62.7	63.4
		27	10.1	16.5	24.1	31.0	37.5	44.9	52.3	59.6	66.9	73.5	74.5	75.5	76.4	77.5	78.5	81.5	82.8
		32	10.7	19.3	28.2	36.5	44.1	52.8	61.4	70.0	78.3	86.4	87.6	88.8	89.7	90.7	92	94.9	96.0
		37	11.0	21.8	31.7	41.8	52.0	62.2	72.4	82.6	92.4	102.0	103.0	104.0	106.0	107.0	108.0	111.0	112.0
	BQT	22	8.5	9.7	14.2	18.7	23.3	27.9	32.4	37.0	41.4	45.0	45.9	46.8	47.7	48.5	49.5	51.9	51.2
		27	9.3	13.9	20.2	26.0	31.7	37.9	44.1	50.3	56.4	61.7	62.7	63.8	64.8	65.8	66.7	69.5	70.7
		32	10.3	16.8	24.7	31.8	38.4	46.0	53.5	61.0	68.4	75.2	76.2	77.1	78.4	79.5	80.5	83.7	85.0
		37	11.2	19.1	27.9	36.8	45.7	54.7	63.6	72.6	81.3	89.4	90.8	92.0	93.2	94.4	95.6	99.0	100.0
Random Access	CMF	22	2.5	4.2	6.1	8.1	10.1	12.1	14.0	15.9	17.8	19.6	19.8	20.1	20.3	20.7	21.0	21.9	22.4
		27	3.1	5.8	8.7	11.4	13.9	16.6	19.3	21.9	24.4	26.9	27.1	27.5	27.8	28.2	28.4	29.7	30.3
		32	3.6	6.9	10.1	13.2	16.0	19.1	22.2	25.2	28.2	31.3	31.5	31.9	32.3	32.6	33.1	34.5	35.3
		37	3.8	7.2	10.5	14.0	17.4	20.8	24.2	27.5	30.8	34.1	34.6	34.6	35.5	35.6	36.4	37.3	38.7
	DR2	22	2.1	4.0	5.9	7.8	9.7	11.7	13.5	15.4	17.3	19.0	19.3	19.5	19.7	19.9	20.1	20.8	21.2
		27	2.4	4.5	6.7	8.8	10.6	12.6	14.7	16.7	18.8	20.7	21.0	21.2	21.4	21.6	21.9	22.6	23.2
		32	2.6	5.1	7.5	9.8	11.7	14.1	16.4	18.6	20.9	23.1	23.4	23.6	23.9	24.2	24.5	25.3	25.9
		37	3.0	5.9	8.6	11.4	14.1	16.9	19.7	22.4	25.1	27.8	28.0	28.3	28.8	29.2	29.5	30.4	31.1
	FM4	22	2.3	4.3	6.4	8.5	10.5	12.6	14.7	16.7	18.7	20.7	20.9	21.1	21.4	21.7	21.9	22.5	23.0
		27	2.5	4.9	7.2	9.4	11.3	13.6	15.8	18.0	20.2	22.3	22.5	22.8	23.1	23.3	23.6	24.4	24.9
		32	2.8	5.5	8.1	10.5	12.6	15.1	17.6	20.0	22.5	24.9	25.1	25.4	25.8	26.0	26.4	27.3	27.9
		37	3.0	5.9	8.6	11.4	14.2	17.0	19.7	22.4	25.2	27.8	28.1	28.5	28.8	29.2	29.5	30.5	31.1
	PR3	22	1.8	3.1	4.6	6.0	7.5	9.0	10.5	11.9	13.4	14.7	14.9	15.1	15.2	15.4	15.7	16.2	16.6
		27	1.9	3.6	5.3	6.9	8.3	9.9	11.5	13.1	14.7	16.2	16.4	16.6	16.8	17.0	17.2	17.8	18.2
		32	2.0	3.9	5.7	7.4	9.0	10.7	12.5	14.2	16.0	17.7	17.9	18.1	18.3	18.5	18.7	19.3	19.8
		37	2.1	4.0	5.9	7.9	9.8	11.7	13.7	15.6	17.5	19.4	19.6	19.8	20.0	20.2	20.5	21.2	21.7
	BBD	22	8.6	15.7	23.0	30.4	37.8	45.1	52.4	59.5	66.6	73.6	74.2	74.7	75.7	76.3	77.2	80.3	82.0
		27	9.8	18.3	27.1	35.1	42.3	50.4	58.5	66.5	74.3	81.9	82.7	82.9	83.9	84.9	85.8	89.9	92.7
		32	10.8	20.6	30.3	39.3	47.4	56.5	65.6	74.3	83.4	92.2	92.7	93.8	94.6	95.2	96.7	101.0	103.0
		37	11.0	21.1	30.9	41.0	51.1	60.8	70.5	80.1	89.6	99.2	100.0	101.0	102.0	103.0	104.0	106.0	111.0
	BQT	22	8.2	14.3	20.9	27.7	34.5	41.2	47.9	54.3	60.8	67.0	67.7	68.4	69.6	70.5	71.4	74.1	76.0
		27	10.3	19.3	28.4	36.8	44.4	52.9	61.4	69.6	77.8	85.5	86.6	87.6	88.7	89.8	90.8	94.0	96.4
		32	11.6	22.3	32.7	42.4	51.1	61.0	70.8	80.3	89.8	99.3	100.0	101.0	102.0	103.0	105.0	108.0	110.0
		37	12.2	23.5	34.3	45.4	56.5	67.3	78.0	88.5	99.0	109.0	110.0	111.0	113.0	114.0	115.0	119.0	122.0

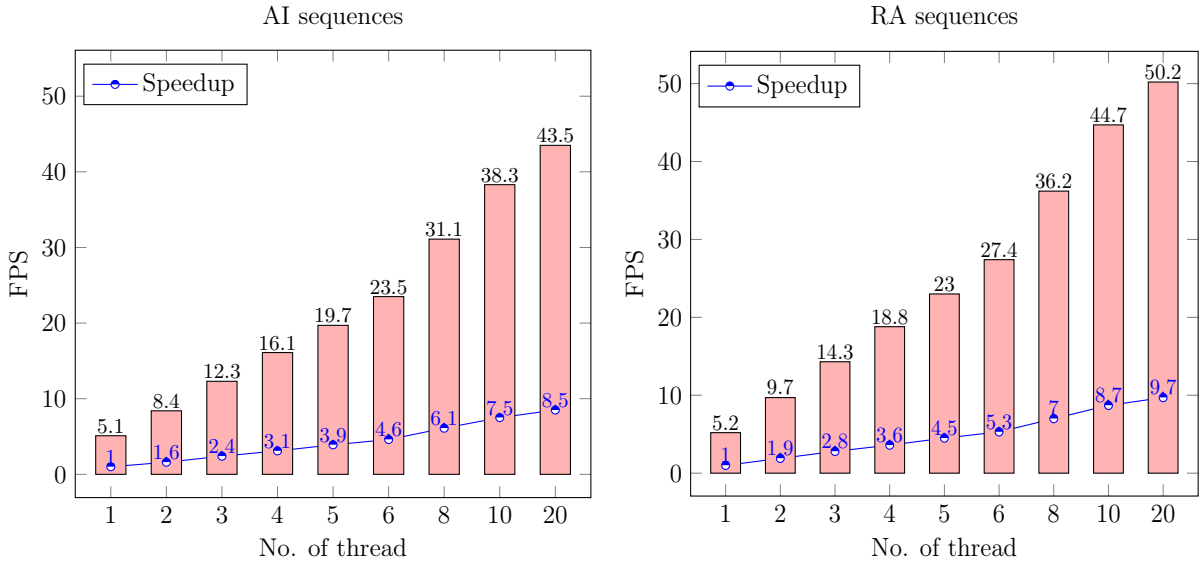


Figure 5.1. Average performance (in FPS) and speedup obtained for VVdeC v0.2 decoding for different thread numbers without SIMD optimisation over X-series for AI (left) and RA (right) sequences.

Figure 5.2 shows the average performance in FPS and the speedup obtained from the decoding of VVdeC v0.2 using 1 to 8 threads over Xavier for AI (left) and RA (right) sequences without SIMD. For AI sequences, the average between 2.5 and 16.4 FPS was obtained using between 1 and 8 threads without SIMD. In addition, speedup a 6.6 was achieved using 8 threads. Moreover, a 18.7 FPS with a speedup of 6.7 was obtained using 8 threads for RA sequences without SIMD. The algorithm has a high degree of parallelism because with 8 cores VVC obtained an speedup from 6.6 to 6.7 on Xavier. Moreover, the difference between Xavier and X-series in speedup is very small.

5.3.2 Performance analysis of VVdeC v0.2 with SIMD activated

This section presents the results obtained for the performance analysis of VVdeC v0.2 with SIMD optimisation explained in Section 4.5.2. Similarly to Section 5.3.1, the test sequences FM4, CMF, DR2, PR3, BBD and BQT of Set A and four QP (22, 27, 32 and 37) were used to average performance in FPS. The average FPS and the speedup obtained for VVdeC v0.2 over X-series using 1-20 cores for AI and RA with SIMD are presented in Figure 5.3. Here, 12.1 to 90.5 FPS was obtained using 1 to 20 threads for the AI sequences, where a 7.5 speedup was obtained using 20 threads. For RA sequences, 155.3 FPS with 8.2 speedup was obtained using 20 threads. Moreover, Figure 5.4 presents the average performance in FPS and the speedup achieved for VVdeC v0.2 decoding for different

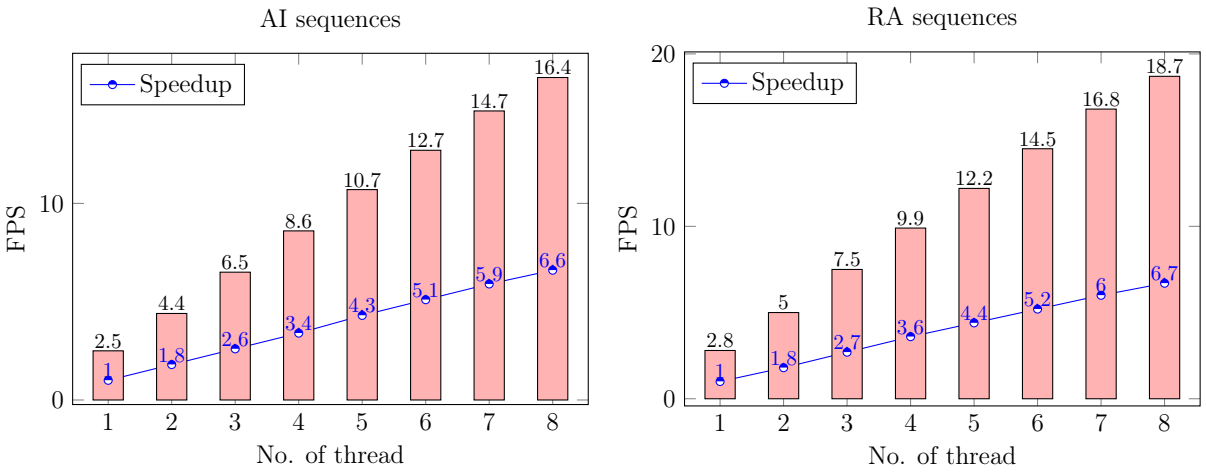


Figure 5.2. Average performance (in FPS) and speedup obtained for VVdeC v0.2 decoding for different thread numbers without SIMD optimisation over Xavier for AI (left) and RA (right) sequences.

thread numbers without SIMD over Xavier for AI (left) and RA (right) sequences. For AI sequences, the average FPS achieved from 5.1 to 27.4 using 1 to 8 threads with SIMD optimisation. a 5.4 speedup was gained using 8 threads compared with one thread over Xavier. Moreover, 38.8 FPS was obtained with a 6.5 speedup using 8 threads for RA sequences.

5.3.3 Speedup analysis

The first part of this section focuses on speedup analysis for SIMD optimisation over X-series and Xavier platforms. The second part focuses on the comparison analysis of speedup for vectorization and SIMD.

5.3.3.1 Speedup analysis with SIMD optimisations

This subsection asses the speedup achieved by different decoder blocks using SIMD optimisation on X-series and Xavier platforms. The speedup was calculated using the Equation 5.1:

$$SU_{fps} = FPS_{simd}/FPS_{wosimd} \tag{5.1}$$

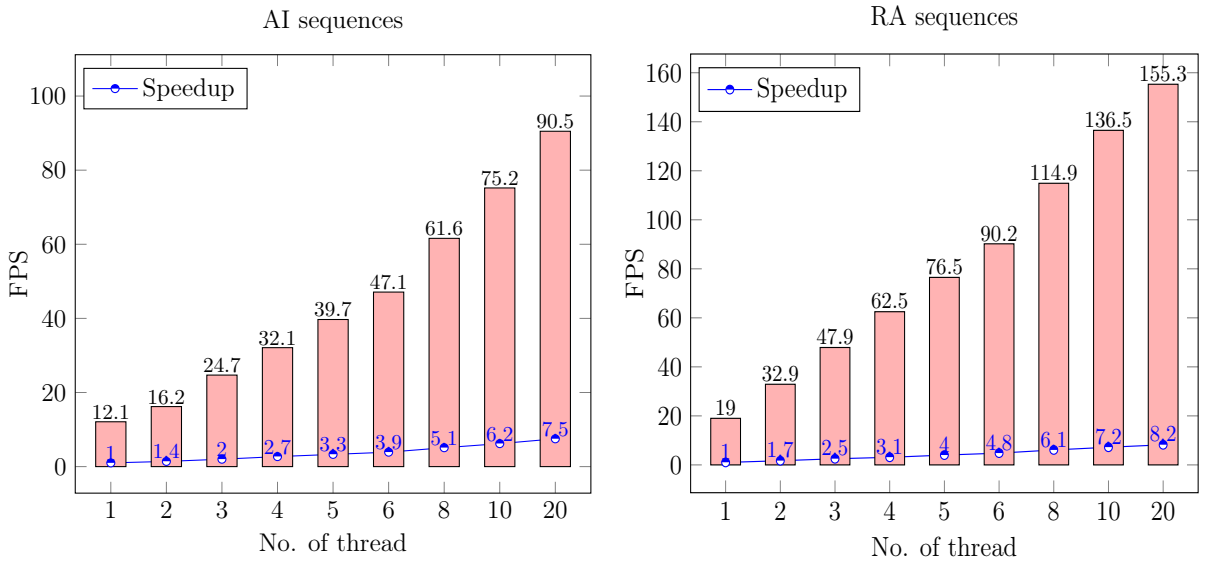


Figure 5.3. Average performance (in FPS) and speedup obtained for VVdeC v0.2 decoding for different thread numbers with SIMD optimisation over X-series for AI (left) and RA (right) sequences.

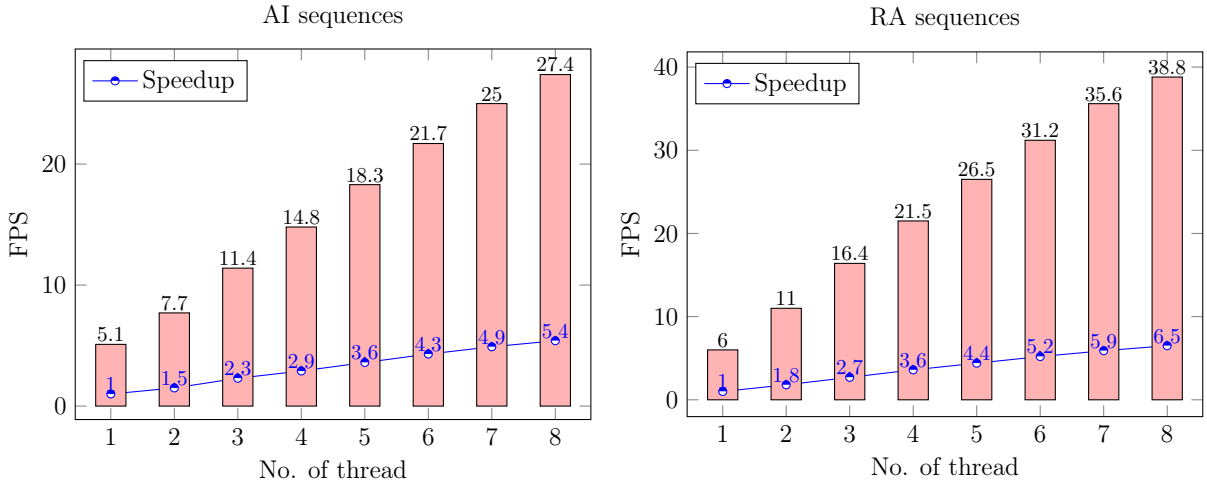


Figure 5.4. Average performance (in FPS) and speedup obtained for VVdeC v0.2 decoding for different thread numbers with SIMD optimisation over Xavier for AI (left) and RA (right) sequences.

Where, SU_{fps} represents speedup for FPS, FPS_{simd} is FPS obtained with SIMD and FPS_{wosimd} is FPS obtained without SIMD.

Figure 5.5 shows the average speedup obtained for different blocks of the VVdeC v0.2

decoder by using SIMD over X-series (left) and Xavier (right) for AI and RA sequences using 8 cores. ALF block of VVdeC v0.2 decoder is the most benefited from SIMD optimisation, where roughly average ×8 speedup over X-series and ×3.5 speedup over Xavier were obtained for AI and RA sequences. For RA sequences, EP block gained average ×3.64 and ×2.45 speedup over X-series and Xavier, respectively. Furthermore, over X-series, the TX, IP, DBF, and SAO blocks obtained around average ×1.6, ×1.3, ×1.2, and ×2.1 speedup, respectively for AI and RA sequences. However, ED and OT blocks gained a little average speedup for all sequences over X-series.

Moreover, the scenario is similar over Xavier, the IP block obtained average ×1.3, the DBF block obtained average ×1.1, and SAO block obtained average ×1.6 for all sequences. But, the ED, TX and OT blocks achieved a little average speedup ranging between ×1.01 and ×1.1 for all sequences over X-series. In addition, average ×2 and ×3.3 speedup is obtained of total decoding over X-series for AI and RA sequences, respectively. On Xavier, the average speedup obtained of total decoding was ×1.6 for AI and ×2.1 for RA.

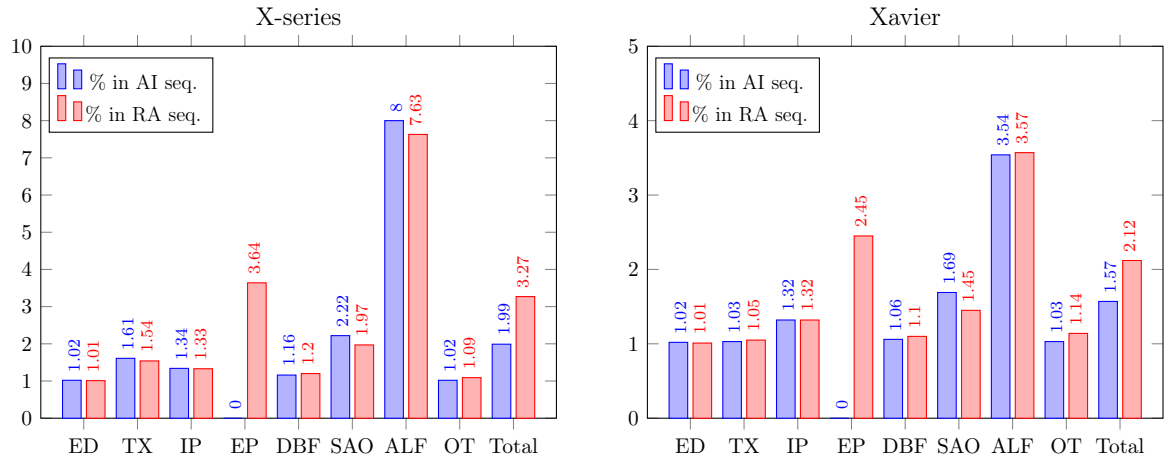


Figure 5.5. Average speedup for different blocks of the VVdeC v0.2 decoder by using SIMD extensions over X-series (left) and Xavier (right).

The results of the speedup analysis of the VVdeC v0.2 decoder show that the similar speedup was obtained of different decoder blocks for two types of platforms. However, most benefited blocks (e.g. ALF or EP) obtained better speedup over X-series than Xavier, which impact on the overall decoding time. This is reasonable since X-series has higher number of cores with a higher clock speed, larger cache memory, different SIMD register size, and a large number of PCI Express Lanes with higher memory speed than

Xavier. Moreover, the ED block did not benefit from the SIMD optimisation, as this decoder block is serial in nature.

5.3.3.2 Comparison analysis of speedup for vectorization and SIMD

The performance and speedup comparison on Xavier among Un-optimised (UnOP)[4] + Un-vectorization (UnVec)[5], UnOP, and optimised by SIMD for eight cores are presented in this section. Figure 5.6 displays the FPS and speedup comparison: UnOP+UnVec vs. UnOP vs. optimised VVdeC v0.2 implementations over Xavier for AI (left) and RA (right) sequences. Here, UnOP implementation gained ×1.2 speedup and the SIMD optimisation implementation gained ×2.0 with respect to UnOP+UnVec implementation for AI sequences with all QPs (22-37). For RA sequences, the average ×1.1 and ×2.4 speedup obtained by UnOP and the SIMD optimisation implementation, respectively.

This analysis shows that auto-vectorization[6] achieved a small improvement (10-20%). The proposed SIMD optimisation is needed to improve performance on heterogeneous platforms with limited resources.

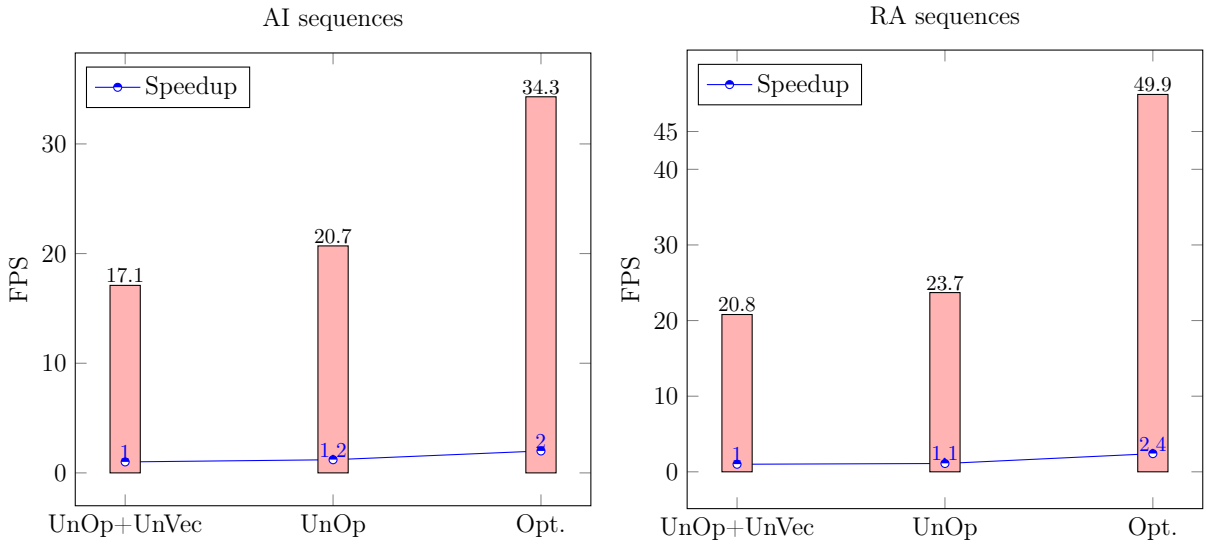


Figure 5.6. FPS and speedup comparison: un-optimised and Un-vectorization vs. un-optimised vs. optimised VVdeC v0.2 implementations over Xavier for AI (left) and RA (right) sequences.

[4]Un-optimised version is without SIMD optimization.
[5]Un-vectorization version is without auto vectorization.
[6]Auto vectorizer looks for loops and, if possible, executes them automatically using the vector registers and instructions of the targeted device.

5.4 Performance and speedup analysis of VVdeC v1.3 decoder for embedded platforms

As it was introduced in Section 5.3, a new version of VVdeC was released in Dec. 2021 and it supports better tile parallelism, and uses 3 times less memory than previous version, which is essential to compare with OpenVVC. This section presents the performance (in FPS) with and without SIMD optimisation and speedup obtained with SIMD optimisation for VVdeC v1.3 decoder using maximum number of cores over Xavier and Nano platforms. All test sequences of Set A (see Table 5.2) were used for all experiments. The presented results were averaged for all sequences and for four QPs (22, 27, 32, and 37).

Figure 5.7 illustrates the average FPS obtained with or without SIMD and speedup for different video quality of the VVdeC v1.3 decoder by using SIMD optimisation over Xavier for AI (left) and RA (right) sequences. For AI sequences, the average $\times 1.6$ speedup was achieved, where the average 39.9 and 60 FPS were obtained with and without SIMD optimisation, respectively. For RA sequences, real-time decoding was achieved for all HD and FHD, and some of the UHD video sequences with SIMD optimisation. Moreover, for RA sequences, the average FPS 75.5 and 144.1 was achieved with and without SIMD optimisation with an speedup of $\times 2.1$.

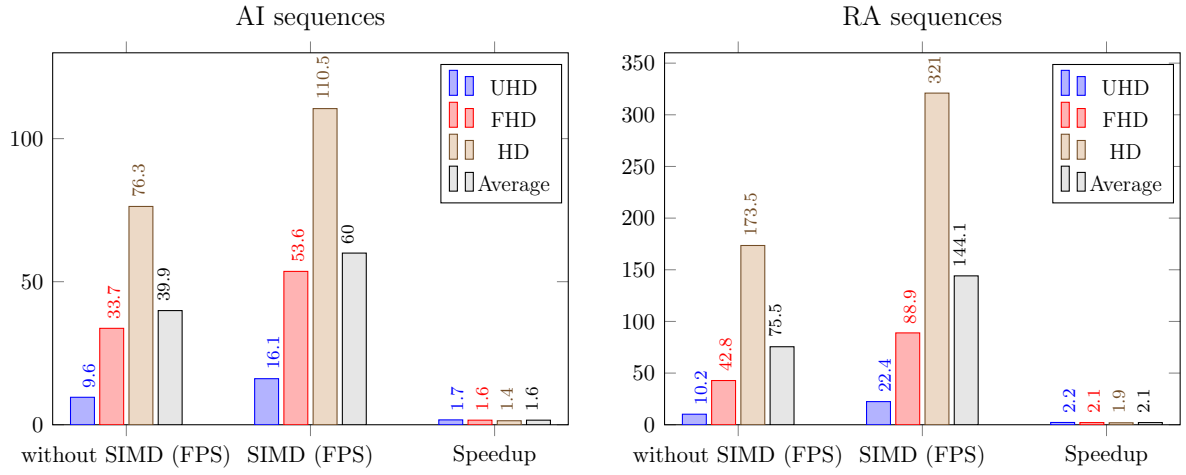


Figure 5.7. Average FPS obtained with/without SIMD and speedup for different sequences of VVdeC decoder v1.3 over Xavier for AI (left) and RA (right) sequences.

This result shows how SIMD optimisation of the VVdeC v1.3 decoder affects different qualities of videos on the Xavier platform. Here, all FHD sequences obtained real-time

decoding and HD reached up to approximately ×7 more FPS than real-time decoding using SIMD optimisation. Moreover, up to about ×2.2 in performance (FPS) was obtained for the UHD sequence, where some of them achieved real-time decoding. This information are useful for future research to focus more on acceleration of UHD videos.

Figure 5.8 shows the average FPS obtained with/without SIMD and speedup for video of the VVdeC v1.3 decoder by using SIMD optimisation over Nano for AI (left) and RA (right) sequences. The average speedup of ×1.5 and ×1.8 was achieved for both AI and RA sequences, respectively. The average of 9.3 FPS for AI and 19.2 FPS for RA sequences was obtained without SIMD optimisation. In addition, 14.4 FPS for AI sequences and 33.6 FPS for RA sequences was obtained with the SIMD optimisation. Only HD video sequences achieved real-time decoding for RA sequences with SIMD optimisation.

Over the Nano platform, the SIMD optimisation of the VVdeC v1.3 decoder obtained real-time decoding only for HD videos. FHD and UHD videos were very far from real-time decoding. Therefore, it can be concluded that this kind of platform is not recommended if real-time decoding of FHD and UHD videos is required. This result is expected because the hardware resources are very limited but the speedup is similar to the one obtained with Xavier.

5.5 Performance and speedup analysis of OpenVVC v1.0 decoder

This section presents the performance (in FPS) without and with SIMD optimisation (see Section 4.5.4) and speedup obtained with SIMD optimisation for OpenVVC v1.0 decoder (released on Dec 2021) over Xavier and Nano. The purpose of the experiment is to assess the performance of the OpenVVC decoder for SIMD optimisation on embedded platforms. It serves to validate the methodology proposed in this Thesis because the same methodology was applied for VVdeC. Furthermore, in this experiment, all test sequences of Set B (see Table 5.3) were used as OpenVVC supports tile parallelisation (see Section 5.6 for different frame-tile configurations of OpenVVC).

Figure 5.9 shows the average FPS obtained with and without SIMD and speedup for different video quality of the OpenVVC decoder v1.0 by using SIMD optimisation over Xavier for AI (left) and RA (right) sequences. The presented results were averaged for all sequences and for two QPs[7] (27 and 37). For AI sequences, average ×1.8 speedup

[7]two QPs are included in this experiment for reducing number of experiments and accelerating the results obtaining process.

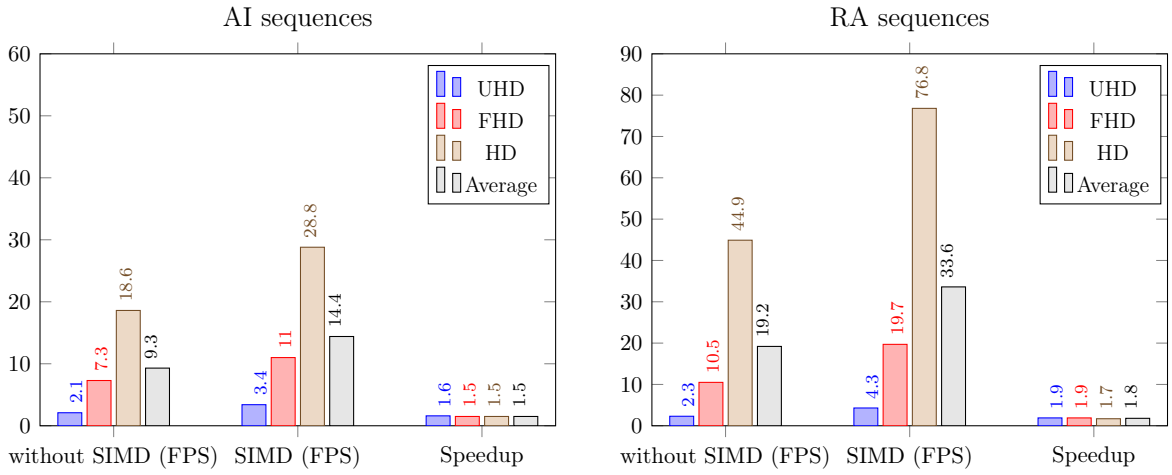


Figure 5.8. Average FPS obtained with/without SIMD and speedup for different video quality of VVdeC decoder v1.3 by using SIMD optimisations over Nano for AI (left) and RA (right) sequences.

was achieved, where the average of 76.3 FPS and 136.3 FPS were obtained with and without SIMD optimisation, respectively. In addition, real-time decoding was achieved for all HD and FHD, and some of UHD video sequences with SIMD optimisation. The scenario was similar with RA sequences. An average 97 FPS and 171.9 FPS were obtained with and without SIMD optimisation, where the achieved speedup was $\times 1.9$. It can be observed from the results that FPS obtained on Xavier for HD videos was more than $\times 4$ than real-time decoding. As a result, FHD and HD video sequences were included in the experiments presented in the following sections.

In Figure 5.10, the average FPS obtained with/without SIMD and speedup for different video quality of the OpenVVC v1.0 decoder by using SIMD optimisation over Nano for AI (left) and RA (right) sequences are presented. Here, the average speedup of $\times 1.7$ was achieved for both AI and RA sequences. The average of 18.9 and 23.4 FPS was obtained without SIMD optimisation for AI and RA sequences, respectively. Furthermore, 30.9 FPS was obtained for AI and 37.7 FPS was obtained for RA sequences with SIMD optimisation. It can be concluded from this study that only HD video sequences achieved real-time decoding with SIMD optimisation for all sequences. Therefore, HD video sequences were included in the experiments presented in the following sections.

Similar to results presented in the previous Section 5.4, the SIMD optimisation of the OpenVVC v1.0 decoder obtained real-time decoding only for HD videos on the Nano platform and for FHD and HD on the Xavier platform. Moreover, most of the UHD

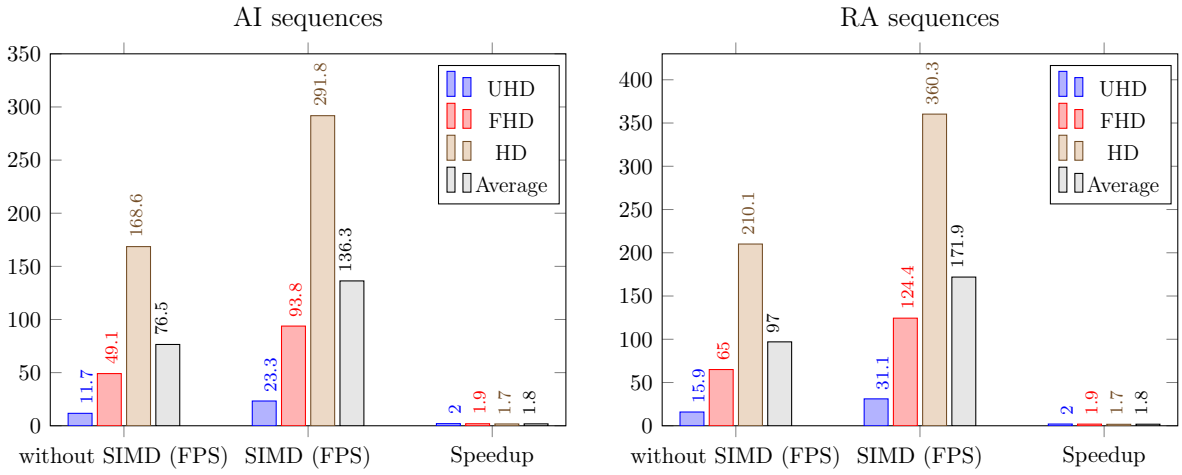


Figure 5.9. Average FPS obtained with/without SIMD and speedup for different video quality of OpenVVC decoder v1.0 by using SIMD optimisations over Xavier for AI (left) and RA (right) sequences.

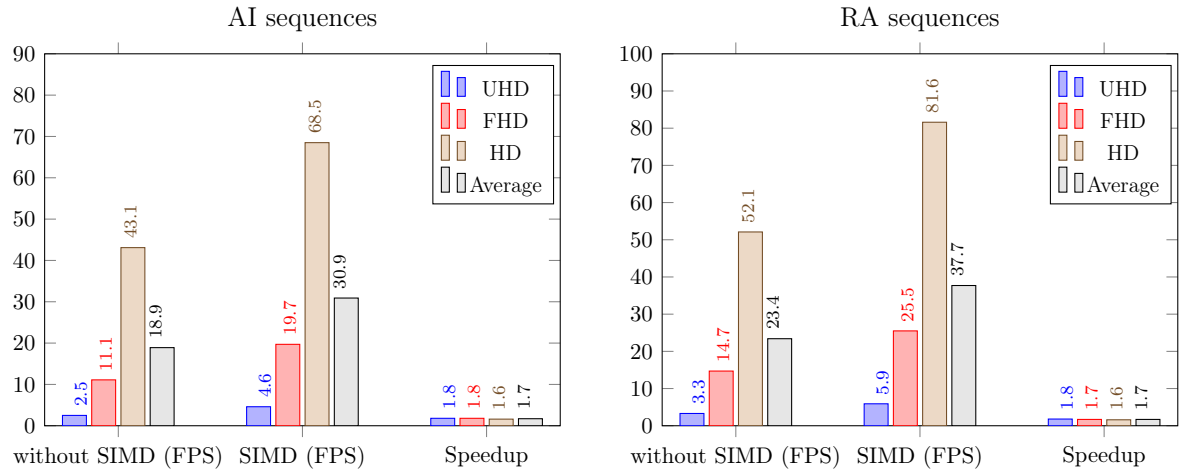


Figure 5.10. Average FPS obtained with/without SIMD and speedup for different video quality of OpenVVC decoder v1.0 by using SIMD optimisations over Nano for AI (left) and RA (right) sequences.

videos are close to real-time over Xavier platform. Therefore, it is suggested to consider the OpenVVC v1.0 decoder for further optimisation to reach real-time decoding for UHD videos.

The presented results justified the fact that the proposed methodology reduced the decoding time for the OpenVVC decoder as like VVdeC decoder, demonstrating its adaptability and generality. The working time required for the SIMD optimisation process was reduced for OpenVVC since the same experience and the defined methodology was used.

5.6 Decoding performance analysis of OpenVVC v1.0 decoder for different frame-tile configurations

The influence of tiles in the decoder performance and the selection of the best configuration is analysed in this section. In this study, the OpenVVC v1.0 decoder performance analysis was conducted for different frame-tile configurations over Xavier and Nano. Here, five and four combinations of frame-tile configurations were used to analyse the decoding performance of OpenVVC v1.0 decoder considering Xavier has 8 cores and Nano has 4 cores, respectively. This analysis was done to determine the best frame-tile configuration[8]. The configurations for Xavier are the following, where f denotes the number of frames and t denotes the number of titles processed in parallel.

- 8-frame and 0-tile per frame in parallel (f8/t0) (only frame parallelism without tile parallelism).
- 1-frame and 8-tile per frame in parallel (f1/t8).
- 2-frame and 4-tile per frame in parallel (f2/t4).
- 4-frame and 2-tile per frame in parallel (f4/t2).
- 2-frame and 8-tile per frame in parallel (f2/t8).

The four configurations for Nano are the following:

- 4-frame and 0-tile per frame in parallel (f4/t0) (only frame parallelism without tiles parallelism).
- 1-frame and 4-tile per frame in parallel (f1/t4).
- 2-frame and 2-tile per frame in parallel (f2/t2).
- 2-frame and 4-tile per frame in parallel (f2/t4).

[8] An example of 4 tiles per frame is given in Section 3.1.2.2.

Figure 5.11 illustrates the average decoding performance (in FPS) of the OpenVVC decoder v1.0 for different frame-tile configurations with QPs 27 and 37 RA sequences on Xavier (left) and Nano (right). It can be seen that, as expected, the configuration with only the frame configuration (f8/t0 for Xavier and f4/t0 for Nano) performed worse than any frame-tile configuration. For the analysis over Xavier, f4/t2 was the best performing and f1/t8 was the worst performing configuration for video with all qualities and resolutions. The FPS obtained for the f4/t2 combination was average ×1.4 for FHD and ×1.3 for HD compared to the f1/t8 combination. Moreover, f2/t2 was the best performing configuration and f1/t4 was the worst performing configuration for HD video with all QPs over Nano. The combination f2/t2 was achieved in average ×1.1 fps compared to the combination f1/t4 for HD sequences.

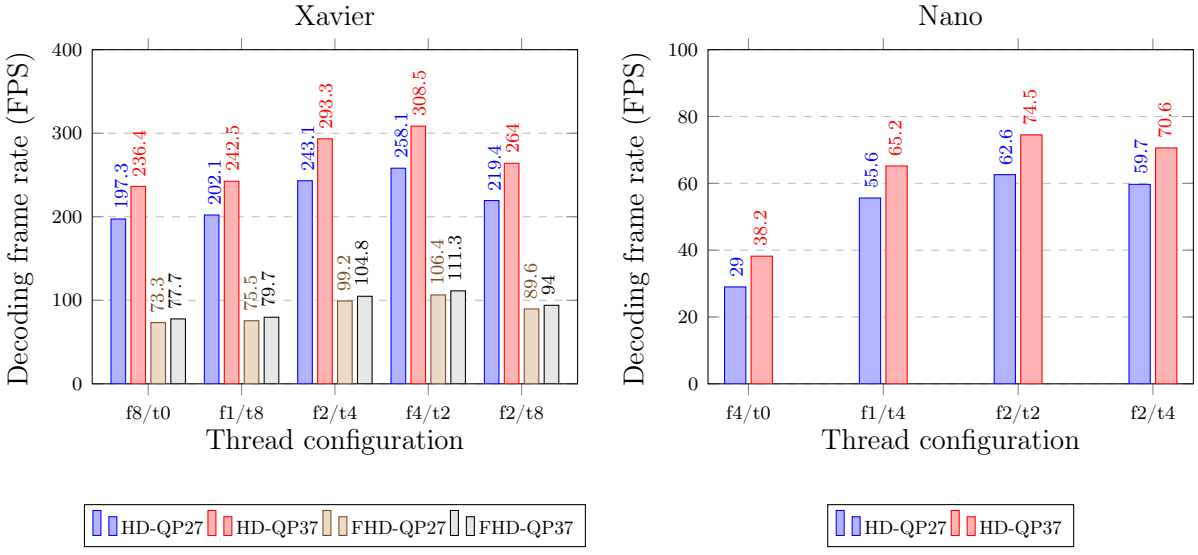


Figure 5.11. Average decoding performance (FPS) of the OpenVVC v1.0 decoder for different frame-tile configurations with QPs 27 and 37 RA sequences on Xavier (left) and Nano (right).

The results show the performance of the OpenVVC v1.0 decoder for different frame-tile configurations on two platforms. Here, the configuration with two tiles per frame (f4/t2 for Xavier and f2/t2 for Nano) achieved the highest performance for both platforms. This information is useful in choosing the number of frames and tiles depending on the available cores of the platform to maximise performance.

5.7 Performance comparison between VVdeC v1.3 and OpenVVC v1.0 decoder

This section presents the performance (in FPS) comparison between VVdeC v1.3 and OpenVVC v1.0. The purpose of the study is to compare the performance between VVdeC and OpenVVC. All FHD and HD test sequences of Set B with QPs 27 and 37 were used in this section for a fair comparison. This comparison was performed using the default (best) frame-tile configuration for theVVdeC v1.3 and OpenVVC v1.0 decoder. Figure 5.12 shows average decoding performance (in FPS) of OpenVVC and VVdeC for QPs (27 and 37), number of cores over Xavier (left) and Nano (right). It can be seen that in all cases VVdeC obtained similar but a little higher FPS over Xavier and Nano. VVdeC achieved up to average ×1.11 more FPS compared to OpenVVC on Xavier using 1 to 8 threads. However, the saturation point was reached by VVdeC using 7 cores for HD video sequences on Xavier. Furthermore, the maximum average ×1.16 more FPS was obtained by VVdeC compared to OpenVVC on Nano using 1 to 4 threads.

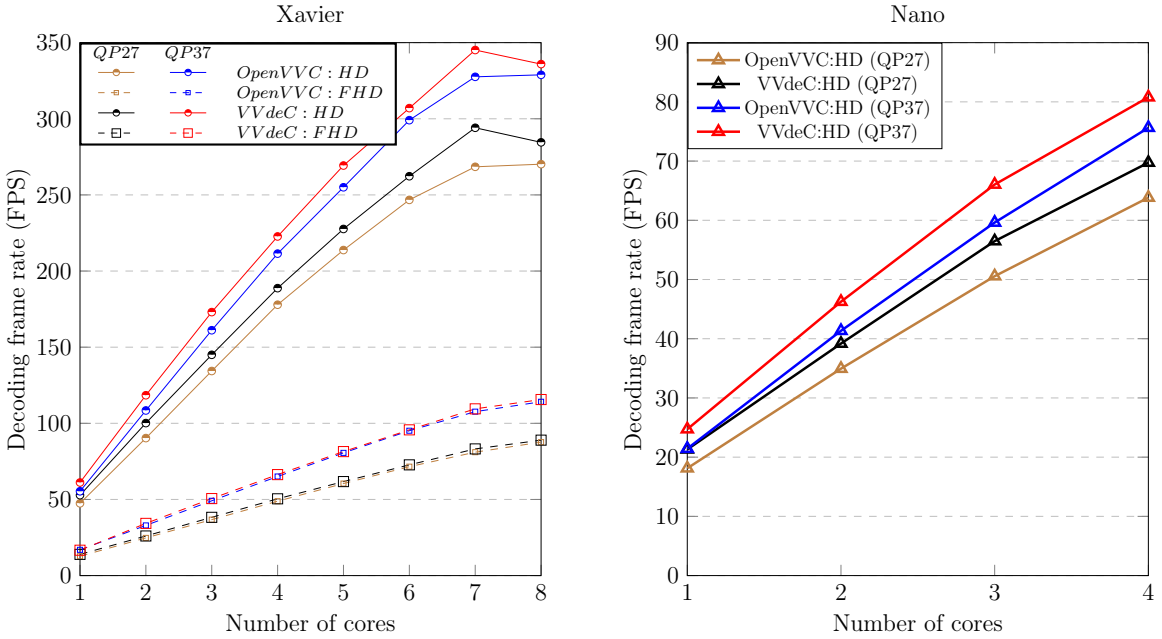


Figure 5.12. Average decoding performance (in FPS) of OpenVVC and VVdeC for QPs (27 and 37), number of cores over Xavier (left) and Nano (right).

This result presents the performance achieved by the VVdeC v1.3 and OpenVVC v1.0 decoder using the SIMD optimisation on two platforms. Here, it can be seen that the performance of the VVdeC v1.3 decoder decreased for the eighth core and the trend in the performance of the OpenVVC v1.0 decoder was flattening after the seventh core. This means that platforms with more cores with the same feature will barely help to improve

the performance. The proposed methodology took almost maximum advantage of the SIMD optimisation for the VVdeC v1.3 and OpenVVC v1.0 decoder on both platform.

5.8 Experimental results of the CPU+GPU implementation of VVdeC v1.3 decoder

This section presents the speedup and performance (in FPS) obtained with CPU+GPU based implementation (see Section 4.6) of VVdeC v1.3 ALF block over Xavier platform. All test sequences of Set A (see Table 5.2) were used for all experiments. Here, the average time distribution for different blocks of the VVdeC v1.3 decoder (in seconds) using CPU and CPU+GPU of Xavier with SIMD activated for AI (left) and RA (right) sequences is shown in Figure 5.13. The presented results were averaged for all sequences and for four QPs (22, 27, 32, and 37). It can be seen that all decoder blocks consumed similar time for both CPU and CPU+GPU implementation, except the ALF block which consumed roughly 50% less time using CPU+GPU implementation compared to CPU implementation for all sequences. Moreover, the total decoding time of CPU+GPU is 11% and 12.6% less for AI and RA sequences, respectively. As the ALF block was processed on GPU, ALF and the total decoding time consumed less time for CPU+GPU implementation.

Table 5.6 presents the average speedup obtained for ALF and the total decoding time using CPU+GPU over CPU with SIMD activated for four QPs (22, 27, 32 and 37). The ALF was accelerated on average $\times$2.02 and $\times$2.01 for AI and RA sequences, respectively. As a result, an average increase of $\times$1.13 in the total decoder times was achieved for all sequences.

This improvement may seem slightly limited. However, it is worth remembering that 1) it is considering the overall improvement of the whole decoder by migrating only one of its main blocks, 2) it is including in the processing time the transfer operations between memories, and 3) the performance comparison is being made against an already optimised software. The following section elaborates more on the details of these results.

Table 5.6. Average speedup obtained for ALF and total decoding time (TOT) using CPU+GPU over using only CPU with SIMD activated for four QPs (22-37).

	AI Sequences					RA Sequences				
QP	22	27	32	37	Avg.	22	27	32	37	Avg.
ALF	1.91	1.93	1.95	2.28	2.02	2.02	2.09	2.01	1.93	2.01
TOT	1.08	1.12	1.15	1.18	1.13	1.13	1.14	1.13	1.11	1.13

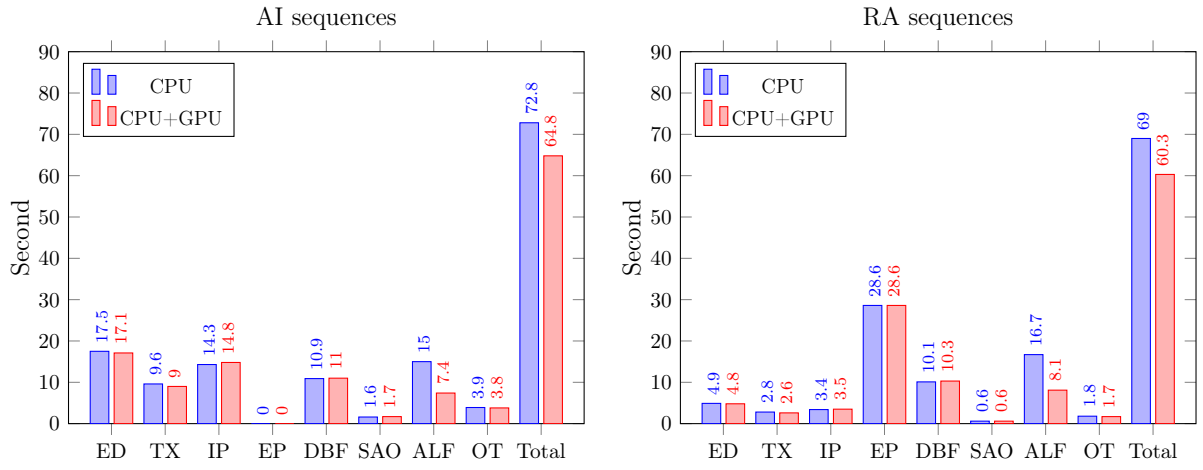


Figure 5.13. Average time distribution for different blocks of the VVdeC v1.3 decoder (in sec.) using CPU and CPU+GPU of Xavier with SIMD activated for AI (left) and RA (right) sequences.

5.9 Comparison performance of VVdeC v1.3 decoder for different implementations

This section presents the performance (in FPS) comparison among CPU-only, CPU+SIMD and CPU+GPU+SIMD implementation of the VVdeC v1.3 decoder for four QPs (22, 27, 32 and 37). The purpose of this experiment is to present a performance comparison of fine-grain optimisation using SIMD and CPU+GPU based hardware accelerator along with the reference implementation without SIMD.

Figure 5.14 illustrates the average FPS obtained for the proposed implementation on 1) CPU without SIMD (dotted line), 2) CPU with SIMD (dashed line) and 3) CPU+GPU (solid line) with SIMD activated for different thread numbers with QP 22 to 37 of the sequences AI (left) and RA (right) on Xavier. Here, the obtained FPS was averaged for the sequences with QP 22, 27, 32 and 37 of Set A. The average speedup of $\times 1.7$ (maximum/minimum: $\times 2.0/\times 1.4$) and $\times 2.2$ (maximum/minimum: $\times 2.5/\times 1.7$) was obtained using CPU+GPU+SIMD over CPU-only implementation on Xavier with 8 cores and QP 22, 27, 32 and 37 for the sequences AI and RA, respectively. In addition, the average speedup of $\times 1.1$ was obtained using CPU+GPU+SIMD over CPU+SIMD implementation on Xavier with 8 cores for all sequences of all video qualities.

Figure 5.15 presents the average FPS obtained for the proposed implementation on

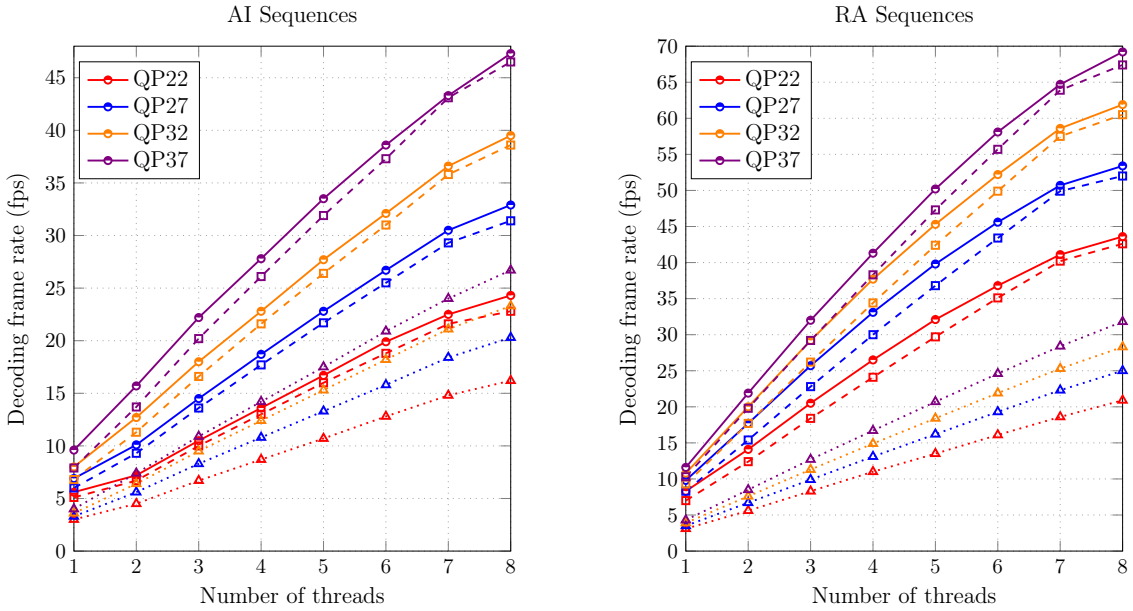


Figure 5.14. Average FPS obtained for the proposed implementation on 1) CPU-only without SIMD (dotted line), 2) CPU with SIMD (dashed line) and 3) CPU+GPU (solid line) with SIMD activated for different thread numbers with QP 22 to 37 of AI (left) and RA (right) sequences over Xavier.

1) CPU-only without SIMD (dotted line), 2) CPU with SIMD (dashed line), and 3) CPU+GPU (solid line) with SIMD activated for different thread numbers with QP 22 to 37 of AI (left) and RA (right) sequences on Nano. Here, the obtained FPS was averaged for all the HD and FHD sequences of Set A. The average speedup of $\times 0.6$ was obtained for AI and $\times 0.7$ for RA using CPU+GPU+SIMD implementation over CPU+SIMD in Nano with 1 core for all the sequences of all video qualities. Moreover, the average speedup of $\times 0.3$ and $\times 0.5$ was obtained using CPU+GPU+SIMD over CPU+SIMD implementation on Nano with 4 cores and QP 22 to 37 for the sequences AI and RA, respectively. It can be seen from the result that the performance was slower for CPU+GPU+SIMD implementation, where performance were dropped further with the increment of number of cores. The main reason for the performance drop are data transfers, which caused a bottleneck in the decoding process. Furthermore, the decoder algorithm and execution were studied in detail and found that GPU threads waited the majority of the time for the data transferring from CPU to GPU. This behaviour is due to the fact that Nano has a memory bandwidth of 7 GB/s and Xavier has a memory bandwidth of 170 GB/s. This very limited memory bandwidth of Nano caused the obstacle in the transfer of data between CPU and GPU. In addition, the request for data transfer between CPU and GPU increased tremendously with the number of CPU cores increased. As a result, most of the time average estimated around two CPU cores waited for data transferring between CPU and GPU. Although the proposed methodology could not benefit on the Nano platform,

it could be applied to other GPUs. In conclusion, the platform was the main limiting factor for the performance loss on Nano.

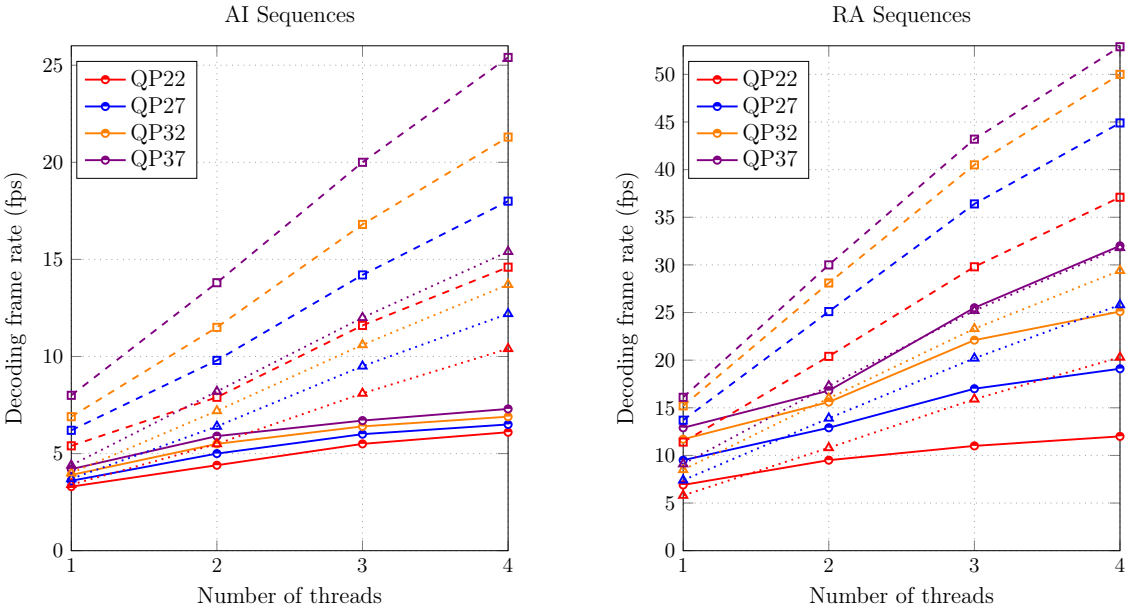


Figure 5.15. Average FPS obtained for the proposed implementation on 1) CPU-only without SIMD (dotted line), 2) CPU with SIMD (dashed line) and 3) CPU+GPU (solid line) with SIMD activated for different thread numbers with QP 22 to 37 of AI (left) and RA (right) sequences over Nano.

5.10 Comparison study of memory usage for VVdeC v1.3 and OpenVVC v1.0 decoder

The memory usage of a decoder is a very important factor because nowadays various video processing applications are performed on limited resource mobile devices and embedded platforms. Moreover, the demand for video processing applications on limited resource mobile devices and embedded platforms is growing rapidly. This section presents the comparison of the maximum memory usage (in MB) (see Section 4.7.1) between VVdeC v1.3 and OpenVVC v1.0. All FHD and HD test sequences of Set B with QPs (27, 37) were used in this section for a fair comparison. This comparison was made using the default (best) frame-tile configuration of the VVdeC v1.3 decoder and different frame-tile configurations (see Section 5.6) of the OpenVVC v1.0 decoder. Figure 5.16 shows the average maximum memory (in MB) used for QPs 27 and 37 sequences over Xavier (left) and Nano (right).

On Xavier, the configuration f2/t8 consumed the highest memory and f1/t8 the lowest

memory among all the OpenVVC v1.0 decoder configuration for all video qualities. It justifies that memory consumption increases according to the number of concurrent frames decoding and then concurrent titles decoding in a frame. Moreover, the VVdeC v1.3 decoder used ×2.7 and ×2.1 additional memory than the OpenVVC with f2/t8 configuration for the FHD and HD sequences, respectively. Furthermore, the highest memory usage configuration of OpenVVC was for configuration f2 / t4 and the lowest memory usage OpenVVC configuration was f1/t4 for all HD sequences on Nano. On the other hand, the average ×2.4 memory was consumed by VVdeC compared to the OpenVVC f2/t4 configuration.

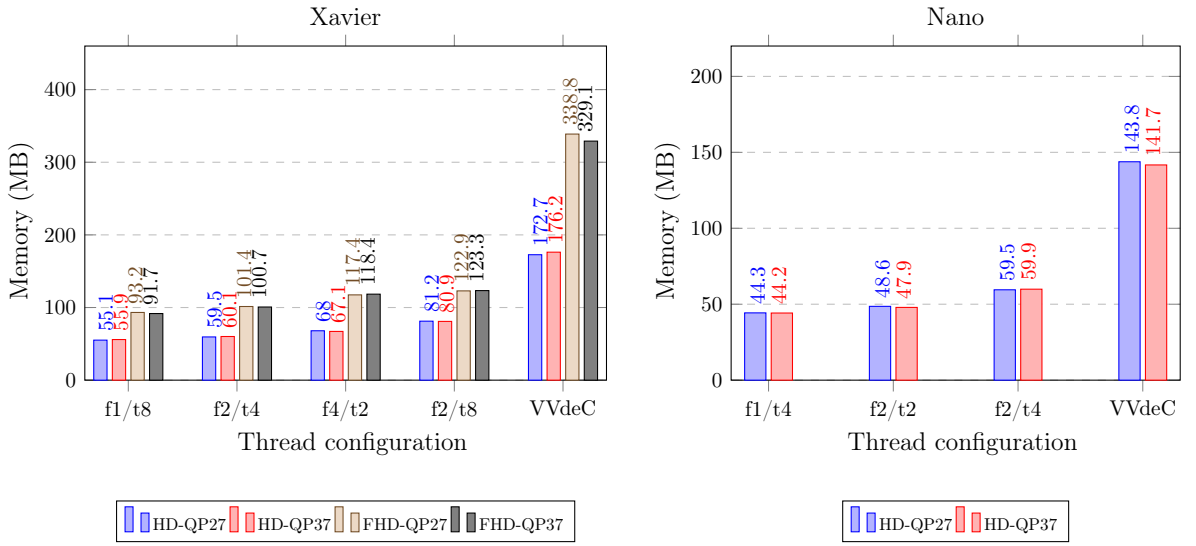


Figure 5.16. Average maximum memory (in MB) used for different thread configurations of OpenVVC and VVdeC with QPs 27 and 37 sequences over Xavier (left) and Nano (right).

This result presents the maximum memory usage of VVdeC v1.3 and the different frame-tile configurations of the OpenVVC v1.0 decoder. Here, it can be seen that the VVdeC v1.3 decoder consumed a higher amount of memory than OpenVVC. Moreover, this result shows how the maximum memory usage varies for different frame-tile configurations. These information will be useful in choosing the appropriate decoder and the configuration for designing low resources consumer electronics devices.

5.11 Energy consumption analysis

This section contains the energy consumption analysis of different decoders on different platforms and different types of architecture. The section is divided into two parts: 1) comparison study of energy consumption for the VVdeC v1.3 and OpenVVC v1.0 decoder on Xavier and Nano, and 2) comparison study of energy consumption for CPU and CPU+GPU implementation of VVdeC v1.3 decoder on Xavier.

5.11.1 Comparison study of energy consumption for VVdeC v1.3 and OpenVVC v1.0 decoder

This section presents the comparison of the energy consumption (in J) (see Section 4.8) between VVdeC v1.3 and OpenVVC v1.0. All FHD and HD test sequences of Set B with QPs (27, 37) were used in this section for a fair comparison. This comparison was made using the default (best) frame-tile configuration of the VVdeC v1.3 decoder and different frame-tile configurations (see Section 5.6) of the OpenVVC v1.0 decoder. The average energy (in J) consumed for QP 27 and 37 sequences over Xavier (left) and Nano (right) is presented in Figure 5.17. Here, the OpenVVC configuration f1/t8 consumed the most energy and f4/t2 consumed the least energy among all OpenVVC configurations for all sequences on Xavier. Compared to the OpenVVC f4/t2 configuration, VVdeC consumed $\times 1.04$ and $\times 1.17$ more energy for FHD and HD sequences, respectively. However, the scenario was the opposite; OpenVVC consumed on average $\times 1.2$ more energy than VVdeC for all sequences over Nano. In addition, all OpenVVC configurations consumed similar energy over Nano.

5.11.2 Comparison study of energy consumption for CPU and CPU+GPU implementation of VVdeC v1.3 decoder

This section presents the comparison of the energy consumption per frame (in J/frame) (see Section 4.8) between CPU and CPU+GPU implementation of VVdeC v1.3 with SIMD activated. All UHD and FHD test sequences of Set A with QPs (22, 27, 32 and 37) were used for the comparison study. Figure 5.18 illustrates the average energy consumed (in J/frame) of FHD and UHD over CPU and CPU+GPU for AI (left) and RA (right) configurations with QP 22, 27, 32 and 37, and SIMD activated. It can be seen that the energy consumption by CPU+GPU implementation was a bit higher than CPU implementation for both AI and RA sequences. The CPU+GPU implementation consumed

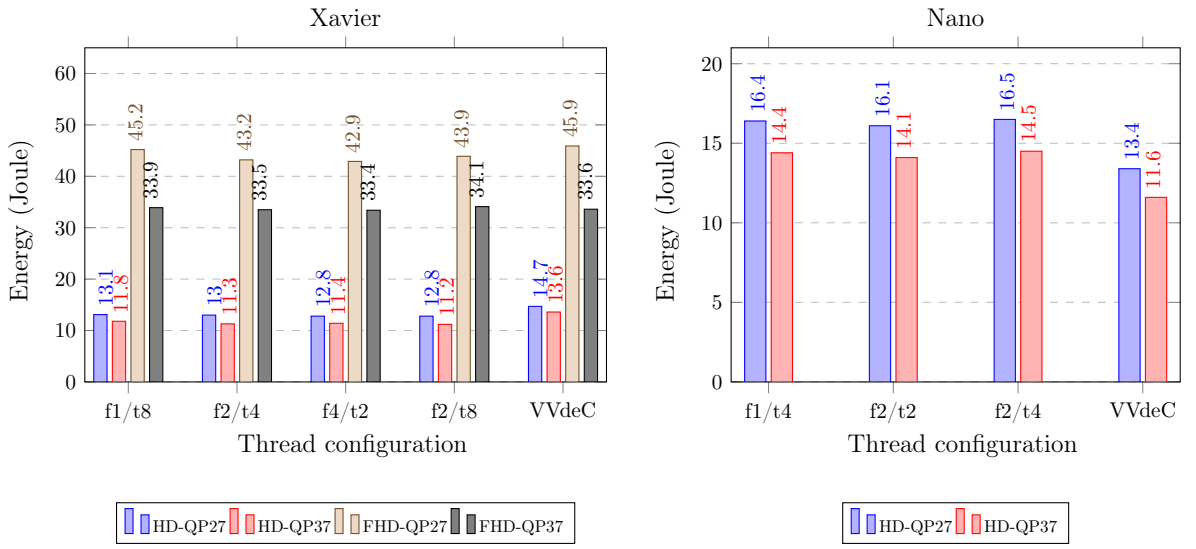


Figure 5.17. Average energy (in J) consumed for different thread configurations of OpenVVC and VVdeC with QP 27 and 37 sequences over Xavier (left) and Nano (right).

4% more energy for AI sequences and 3% more energy for RA sequences compared to the CPU implementation. In addition, low quality videos with lower QPs consumed less energy than higher quality videos with higher QPs as it has higher computational load. For AI sequences, the sequences with QP22 consumed roughly average $\times 2$ energy than the sequences with QP37. Moreover, the sequences with QP22 consumed approximately average $\times 1.5$ energy than the sequences with QP37 for RA sequences.

This result of the energy consumption analysis shows that similar energy was required to decode videos using VVdeC and OpenVVC decoder over two different platforms. It validates that the energy consumption results are reasonable. Moreover, this information will be useful in choosing the appropriate energy source while designing consumer electronics devices. Furthermore, the energy consumption by CPU+GPU implementation shows that similar energy consumption per frame compared to only CPU implementation can be obtained by the proposed methodology, where as GPUs normally consume very high energy.

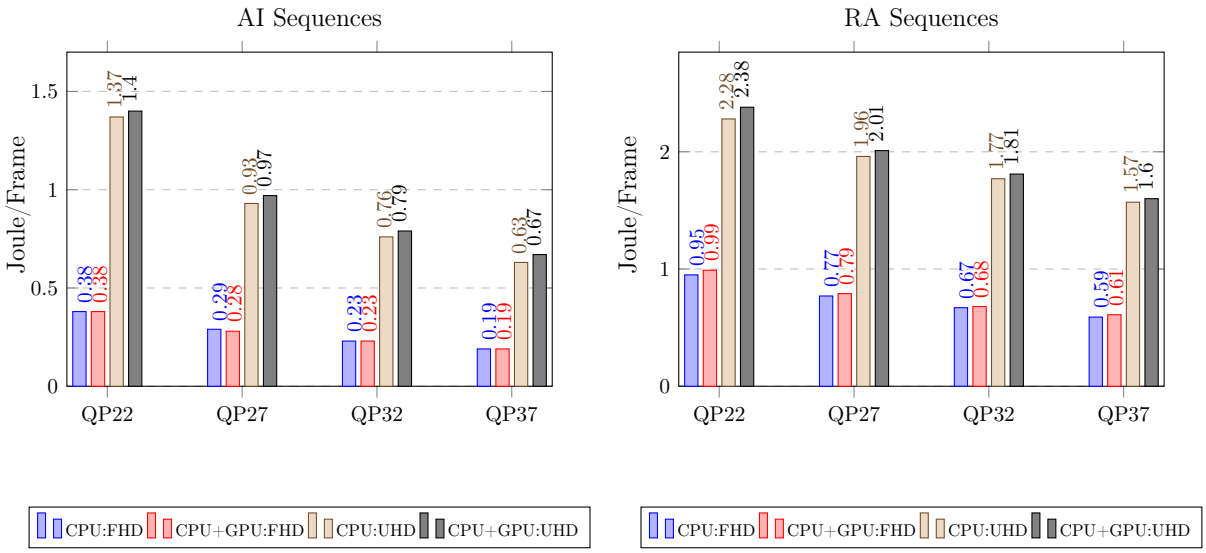


Figure 5.18. Average energy consumed (in J per frame) of FHD and UHD over CPU and CPU+GPU for AI (left) and RA (right) configurations with QP 22-37 and SIMD activated.

5.12 Summary and discussion

The experimental results of the doctoral thesis show that the VVdeC decoder was overall accelerated on average ×1.7 for AI sequences and ×2.2 for RA sequences on a resource-constrained embedded platform Xavier using SIMD optimisation and GPU. Real-time decoding of all HD, FHD, and some UHD sequences was achieved using SIMD optimisation and GPU over the embedded heterogeneous platform Xavier. The experimental results are briefly summarised as follows: First, the results of the VVdeC decoder without SIMD optimisation demonstrated that none of the sequence achieved real-time decoding in Xavier, which led to the optimisation of the VVdeC decoder. Then, different GCC auto vectorization options were analysed and found that "-O3" is the best performing option. Therefore, VVdeC v0.2 decoder was accelerated by average ×1.6 for AI sequences and ×2.1 for RA sequences using SIMD optimisation over most of the decoding blocks on Xavier, although the most efforts were invested on EP and filtering blocks (mainly DBF and SAO). The same acceleration was obtained for the VVdeC v1.3 decoder using SIMD optimisation on Xavier. These speedups were considered significant as were obtained for the whole decoding process against an already optimised decoder which achieved real-time decoding up to FHD sequences and some UHD sequences over resource-constrained embedded platform Xavier.

Furthermore, the VVdeC v1.3 decoder obtained average ×1.5 and ×1.8 speedup on Nano for AI and RA sequences, respectively. In addition, OpenVVC was accelerated using SIMD optimisation by average ×1.9 on Xavier and ×1.7 on Nano for video sequences with RA configurations. Then, an analysis was performed to compare the performance (in FPS) between the VVdeC v1.3 decoder and the OpenVVC v1.0 decoder, where the VVdeC v1.3 decoder obtained similar but little higher FPS than the OpenVVC v1.0 decoder. Moreover, the decoding results of VVdeC and OpenVVC over Nano were real-time for all HD ready sequences, which are substantial for a very low-cost and highly resource-constrained embedded platform.

Subsequently, the VVdeC v1.3 decoder was further optimised using GPU, where the average speedup of ×1.1 was obtained in Xavier for all video configurations. Here, VVdeC ALF obtained average ×2 speedup against SIMD optimisation, which was already accelerated the average ×3.55 against without SIMD implementation. Therefore, ALF was accelerated average ×7.1 compared to the without SIMD implementation which caused an average up to ×2.2 speedup for total decoding time. Moreover, DBF could be considered as future work to be accelerated using GPU because it did not benefit much from SIMD optimisation (DBF consumed an average 12% of decoding time). GPU-based DBF implementation might cause speedup ×1.1-×1.2 for total decoding time. However, VVdeC v1.3 decoder did not get benefit from the optimisation based on GPU over Nano due to the fact that Nano has very little bandwidth (×24 lower than Xavier). Therefore, a study was carried out to compare the memory usage between VVdeC v1.3 and OpenVVC v1.0 decoder and found that the VVdeC v1.3 decoder consumed on average ×2.4 maximum memory than the OpenVVC v1.0 decoder on Xavier and Nano.

Moreover, a study was performed to compare the energy consumption between the VVdeC v1.3 and OpenVVC v1.0 decoder, where the VVdeC v1.3 decoder consumed average ×1.1 more energy and ×1.2 less energy than the OpenVVC v1.0 decoder over Xavier and Nano, respectively. Finally, a comparative analysis of energy consumption was performed between CPU and CPU+GPU implementation of the VVdeC v1.3 decoder. Here, the energy consumption by CPU+GPU implementation was up to 4% higher than CPU implementation for both Xavier and Nano. It can be seen from the result that obtained performance compensated the energy consumption/frame as GPU consumed much higher energy than CPU.

As a general conclusion, and from a platform- and software-agnostic point of view, the following can be stated: it might be noted that the largest share of the performance improvements obtained in all platforms and experiments comes from the optimisation of the source code through the use of vectorised SIMD instructions. The improvements

resulting from the migration to the GPU are clearly in second place. However, as outlined above, the increase obtained from this second line of work is the result of moving only ALF filtering. Furthermore, the result obtained has been compared to a highly optimised version and includes all the penalties for memory transfers in this type of architecture. Furthermore, it has been shown that this solution has a negligible impact in terms of energy consumption.

Chapter 6

Proposed methodology

In Chapter this chapter, the working methodology followed on the implementation of video decoder over heterogeneous platforms is presented. Later, in Chapter 5, the experimental results obtained using the working methodology are shown. This chapter gathers that work and proposes a design methodology based on experimental results and the experience obtained during the development of this Thesis.

This thesis address the increased complexity and performance requirements of state-of-the-art video decoding software and the complexity of the embedded heterogeneous platforms, which provide the following proposals that should be followed for future optimizations.

6.1 Methodology specification

To address the increased complexity and performance requirements of state-of-the-art video decoding software, the proposed methodology shall fulfil the following requirements:

- Selection of the target platform.
- Selection of the reference software.
- Analysis of the computational performance of the algorithm and the most computational intensive modules.
- Evaluation and adoption to the most suitable techniques for parallel processing.
- Algorithm acceleration on the heterogeneous platform using the accelerator available.

- Validation.

Here, the platform and software selection can be done in parallel at the beginning. Therefore, the decoder blocks of the selected software must be analysed and selected over the targeted platform before starting the optimisation. Then, different parallelism techniques must be explored and selected. Finally, a validation has to be performed to verify whether the impact of the modification in the algorithm affects the quality.

6.1.1 Selection of the target platform

Platform selection is an important and difficult task for any application. However, for video coding applications, it is much more challenging as video coding demands heavy computational capabilities while many target applications will be mobile devices with important constraints. Even so, video codecs are used in various areas. Therefore, and at least, the following factors must be considered for choosing the platform:

- Required video resolution (e.g. SD, FHD, UHD, etc.) and quality.
- Power consumption.
- Memory speed, size and management.
- Cost and availability of the platform.
- Physical space of the platform.

As mentioned above, one of the most important requirements for platform selection is the video quality requirement. This methodology supports the well-known fact that HGPP is the most suitable for decoding high-resolution and quality video. Therefore, a HGPP-based platform must be considered for the application demands very high-resolution video (e.g. UHD 4K, 8K). Because HGPP comes with high number of cores, high clock speed, larger memory and large cache, which can handle higher computational load demands of high-quality videos. On the other hand, it will consume higher power. However, embedded heterogeneous platforms are suggested in the case where the application demands mobile platforms and/or less physical space and/or low power and resources. More and more, the embedded heterogeneous platforms come with integrated GPU or FPGA accelerators. In addition, GPP in embedded heterogeneous platforms, like high performance ARM-based ones, have lower clock and less speed than HGPP. Even so, after a guided optimisation process, they are able to support real-time FHD videos

comfortably, as shown in this Thesis. Moreover, very low cost platforms like Nano are suggested for applications requiring video quality up to HD ready. Nano cost almost 8 times less and consumes 3 times less physical space than Xavier. Therefore, the platform selection process is the trade-off between computational power, cost, and space. On the other hand, FPGA-based platform could be considered for this kind of applications requiring reconfigurable hardware accelerator. The literature review presented in Section 3.3 shows that FPGA-based platforms are suitable for inverse transformation and quantization, and intra prediction block of H26x decoders. Therefore, it must be considered while choosing the platform but the development time with this platform is probably higher due to the experience needed to optimise modules using HDL languages.

6.1.2 Selection of the reference software

H.26x series, VPx series or AV1 are the well-known video standard families with a wide range of available solutions (implementations) in the market. VPx series and AV1 can be considered for applications that primarily focus on internet video transmission. Then, if the application focuses on consumer electronics, but not only, the video coding software based on H.26x standard series should be considered. As this doctoral study is mainly focused on consumer electronics, it was certain to choose H.26x family. In addition, it is important to take into account the large number of companies, members of industry, universities, centres and research groups directly or indirectly involved in the different groups participating in the standardisation process. This guarantees the impact of the standard, its use and future development.

When undertaking work such as that presented in this Thesis, there is a need to choose a software or reference solution over which to carry out most of the development. Once a standard has been stabilised, it is relatively easy to find a plethora of available solutions in the state of the art, both open source (the case of this thesis) and proprietary. However, if this work takes place 1) either during the process of discussing and building a new standard (in the case of VVC between 2018 and 2020), or 2) in the period just after the standardisation, this would be a problem.

In the first case, it is usual to find partial developments of some of the potential new features of the standard. Bearing in mind that until official standardisation some proposals could fall out of the final release. In this period, the reference software is very changeable and it is difficult to make improvements to the codec as a whole. In the second case, there is usually a fast-paced development. Optimisations and enhancements rush in, and new versions are released in a short space of time. During this period it is very interesting to

follow the standardisation process and to profile the different versions in order to know the most important challenges to solve when the proposal is finally standardised. Important efforts in optimisation can not be very efficient because the changes are continuous in this phase of the coded development.

Based on the experience accumulated during the development of this Thesis, and taking into account the above mentioned points, the following aspects are highlighted to be taken into account when looking for a software as a reference for the implementation and optimisation process of video codecs for consumer electronics. Here, technical aspects, such as the paradigm or the desired programming language (based on data flow definitions, HDL, C/C++, etc.) are left aside. The aspects are as follows:

- Widely used: It is important to choose software that is used and supported by a wide community. It continuously ensures that the reference is 'alive' and will receive updates of the software and support. Thus, making easier further optimisation works.
- Open source: It is important to have an open source solution for scientific research as it allows collaboration among different research groups. In addition, no cost involved to use the software.
- Performance: It is beneficial to choose software that comes with added optimisation. It helps to increase added performance for the future optimisation. However, it might increase adaptation difficulties.
- Compatibility: It is advantageous if the software comes with integrated support for different types of architecture, i.e. SIMD optimisations across X86 architectures or ARM-based processors.

6.1.3 Analysis of the computational performance of the algorithm

The proposed methodology is based on the analysis of the computational load of the different blocks integrating the decoder. This work has proven to be essential for decision-making and the effective targeting of working efforts.

Initially, it might be thought that this analysis is only necessary when considering the optimisation or migration to a hardware accelerator of one -or several- blocks; as stated in this work with ALF filtering. However, this type of analysis has also been positive entailing the efficiently managing of data transfers between memories or redesigning some of the algorithms for their better adaptation to the hardware architecture of an accelerator.

These are the clear advantages of having a detailed analysis of the computational load of the target software.

On the other hand, the following drawbacks should be taken into account: 1) the dependence of the results on software versions. That is, each time a substantial change is included in a new version of the decoder, the process would have to be repeated in order to adjust to the new computational load distribution. 2) Hardware dependency, in case different platforms are to be worked with, an analysis should be performed for each of them, and 3) profiling may result in a high time-consuming task.

Therefore, in the first approach, a coarse-grain profile is suggested to have a whole point of view of the computational profile of the decoder block. Then, a fine-grain profile may be conducted for those computationally intense decoder blocks with higher parallel capabilities.

Fortunately, there are numerous profilers available on the market. GNU Profiler (gprof), Intel VTune Profiler, perf, Valgrind tool suite profiler named Callgrind, and C/C++ timestamps are the most used open source C/C++ profiler for Linux operating systems. It is suggested to use Callgrind and VTune Profiler if a GUI is required. In addition, both profilers provides a detailed profile with higher accuracy. Furthermore, gprof and pref are the lighter profilers that require much less memory. However, it is difficult to target a specific portion of the software to profile as it does not allow specification inside the program. Finally, timestamps is the lighter profile most suitable to target a particular area of the software. In this thesis, Callgrind and timestamps are used since the target platforms are ARM-based limited resource embedded platforms. The Callgrind was chosen over vtune as it supports non-Intel architectures. Finally, the use of timestamps can consume a lot of development time because some changes must be done in the code but they are very useful if profile needs a high accuracy and/or if the application uses multi threads. If timestamps are needed, it is interesting to create a library to simplify the way to define the code to measure and to move the measured sections/modules to a new version of the decoder.

From above mentions requirement set the following factors are listed that should be considered when selecting the profiler:

- Operating system compatibility: The profiler must have compatibility with the operating system used in the project.
- Memory requirements: It is one of the main factors considered for a limited resource platform. A profiler can not be used if a higher memory is required for operating than the platform has.

- Open source: It is important to have an open source solution for scientific research as it is developed and used by a wide range of communities.
- GUI availability: It provides visual representation of the profiling results. It can help to see the profile results easily.
- The profile accuracy: The preciseness of the profile results is dependent on the application requirement. Some profiler provides very accurate results and some are rough results.

In conclusion, it is highly recommended to carry out a computational load analysis, adjusting the level of detail in accordance with the impact of each block on the total share. It is interesting to define a standard test set (sequences and configurations) and try to automate the process in order to minimise the impact on the working flow, easily adapt to new versions of the reference code and/or to facilitate its interoperability between different hardware platforms, if necessary.

6.1.4 Evaluation and adaptation to the most suitable techniques for parallel processing

There are various parallelism techniques used for video coding in scientific literature, apart from the use of hardware accelerators, which is discussed in the next section, this work has focused on the use and application of four widely known methods, all explained in Chapter 3:

- Parallelising the video stream processing (coarse-grain parallelism)
    - o Frame-level parallelism
    - o Tile-level parallelism
    - o Wavefront Parallel Processing
- Optimising the implementation of algorithms (fine-grain parallelism)
    - o Using Single Instruction Multiple Data (SIMD) operations

As far as the parallelisation of the video stream is concerned, a second subdivision is needed. On the one hand, the parallelisation of the decoding process (frame), where the inter-frame dependencies shall be maintained, but the execution of the whole decoder (including the CABAC decoding, the main serial element) is parallelised. On the other hand, there are the methods that parallelise intra-frame decoding (tiles and WPP). Here,

the dependencies within the frame have to be managed, including those within the video stream itself, which also affects the encoder.

In these cases, the coding of the functionality can be a complex task, mainly because of the need to maintain a correct and efficient (non-blocking) synchronisation of the internal decoding pipeline. These parallelism is done usually using threads and once implemented, it is often easily portable and the performance improvement can be very significant if multiple processing cores are available. This effect has been clearly stated in Chapter 5.

Regarding the direct writing of sections of code in low-level language, usually using SIMD instructions, as mentioned throughout this Thesis, it can be very effective, as has been shown in the results Chapter. However, this optimisation is strongly dependent on the technology used and the compatibility of the libraries that support them. This lack of cross-platform interoperability is being mitigated by the development of some libraries such as SIMDe for the case of ARM platforms, but a direct translation is still far from being achieved. This kind of optimisation needs a big effort so it is very important to select the most suitable modules. To select these modules the profile defined in the previous step in the methodology is essential.

The Table 6.1 summarises all these impressions. The expected improvement factor should be understood as a mere orientation based on the experience during the development of this thesis. Furthermore, in the case of coarse-grained parallelisation, it has to be taken into account that its use can be combined and is highly dependent on the platform hardware.

Table 6.1. Summary of the approximate impact on performance improvement, indicative development time, main dependencies and interoperability for the different techniques considered.

Improvement type	Development time	Expected improvement	Main dependencies	Interoperativity
Frame	Medium	40 to 90%	Software	High
Tiles	High	15 to 80%	Software	High
WPP	High	25 to 90%	Software	High
SIMD	Very high	Up to 95%	Hardware, Software	Medium

6.1.5 Algorithm acceleration on heterogeneous platform

In this Thesis, a GPP+GPU-based heteregenous platform was selected as target SoC. Therefore, suggestions are provided for the future GPU-based acceleration and implementation of video applications. During the technical development of this Thesis. All of them

should be normally necessary in a GPP-to-GPU migration of code[1]. The effort to move some modules to the GPU is high so it is important to identify the most interesting ones before starting the process. Some points should be taken into account:

- Algorithm analysis in order to identify the level of parallelism. GPUs are able to execute some simple algorithms in parallel but it require to redesign the code and probably a new data ordering. To know the details of the algorithm is critical.
- Design data access for GPU. exploring different memory allocation and data transfer methods is important because each one have different features. The bandwidth needed to move data from CPU to GPU and vice versa is very important because the performance of the global implementation is reduced by the time expended for these copies. A preliminary analysis of this performance in interesting before starting the algorithm moving to the GPU.
- Kernel distribution, the code of the GPU is organised in kernels and it is important an optimised distribution
- Task schedule. In order to optimise the performance of data movement, a ping pong buffer to paralyse the execution and the copy of the data is recommended.

6.1.5.1 Algorithm redesign and data ordering

GPU cores have a different architecture and run with the instruction from a governor CPU. Therefore, for GPU implementation, the possibility of algorithm redesign is a likely scenario. Redesigning a code allows it to fit more precisely to the target architecture (GPU in this case), and to exploit all the advantages of the new hardware. However, this task will undoubtedly be one of the most costly in terms of manpower time and the level of expertise of the person in charge of this work. As can be seen, it is necessary to redesign a complex algorithm to have a deep knowledge not only of how the algorithm is implemented, but also of the particularities of the architecture. This is necessary to successfully undertake this phase of the design methodology.

For example, and as stated in this Thesis, the pixel scanning pattern in VVdeC ALF was resigned for GPU implementation. During the algorithm redesign, it had to be considered that the data should be linearly organised depending on the processing order. This must be followed as GPU threads, within a thread block, can easily access the

[1]In this case, it is understood that the functionality to be migrated has a significant entity or complexity, usually greater than what would be allocated to a GPU-type accelerator.

data without causing a bottleneck. Then, data ordering should be optimised for GPU implementation[2]. It should be considered to reduce the amount of data transfer while data ordering is performed. It will reduce the complexity of implementation, the time needed to transfer data and the energy consumption. In addition, the GPU execution time will be reduced as an efficient data ordering approach facilitates coalescent access of the data by GPU threads.

In conclusion, this phase is crucial when working with GPU, and the success of adopting this technology depends to a large extent on the skill of the staff in charge of code migration. Alternatively, there are methodologies based on the automatic generation of code according to graphical descriptions [9], [121]. However, these solutions often do not perform significantly better in terms of the overall performance of the application. This aspect is not limited to this sub-section, but is common to the following sub-sections.

6.1.5.2 Design data access for GPU

GPU parallelism of video applications begins after algorithm redesign and data ordering are completed. The main challenge of parallelism using GPU comes from data access because GPU does not have access to CPU memory, although it executed the operations with the instruction received from CPU. In this situation, first, the memory allocation needs to be addressed for achieving efficient data transfer between CPU and GPU.

There are four types of memory allocation supported in CUDA: 1) pageable memory, 2) pinned memory, 3) mapped memory, and 4) unified memory. Here, the memory allocation in pageable memory causes the greatest delay in transferring data between CPU and GPU because it is located on the hard drive. On the other hand, the fastest data transfer between CPU and GPU can be achieved by allocating memory to pinned memory. The data transfer is fastest in this allocation type due to the fact that two data transfers between pageable and pinned memory are discarded. Moreover, mapped memory can be considered if GPU has less memory. It enhances transfer rates of PCIe by avoiding data transfers between CPU and GPU. However, this increases the execution time of the kernel. Lastly, unified memory allocation makes the data accessible by CPU and GPU, which makes it easy to program. However, data transfer is slower than pinned memory, as two copies are needed under the hood to perform the memory allocation: 1) unified memory to pinned memory and 2) pinned memory to GPU.

A comparison of these four methods is provided in Table 6.2 for simple integer array summation operation (c[x] = a[x]+b[x]) with 1048576 array elements (x) [120]. Here,

[2]Data ordering of VVdeC ALF is presented in Section 4.6.1.1.

for pinned memory allocation, less kernel execution time and comparatively less memory transfer are needed. Although memory transfer time is zero in mapped memory allocation, the kernel execution time is almost 10 times for this simple example, which may increase much for complex problem. Therefore, it is suggested that the type of memory allocation should be chosen depending on the target application and platform.

Table 6.2. A comparison of four memory allocation methods (source:[120]).

Memory type	Memory transfer time (ms)	Kernel execution time (ms)	Total time (ms)
Pageable	19.43	0.44	19.87
Pinned	4.26	0.44	4.70
Mapped	0.00	3.72	3.72
Unified	1.96	16.13	18.09

6.1.5.3 Kernel distribution

GPU starts executing the program after all data are transferred and available. The program that is executed on GPU is called the kernel [79]. As it has been aforementioned, it is very important to efficiently design a kernel to properly exploit the GPU architecture. One kernel is executed in parallel by threads, where threads are members of a thread block. Moreover, a thread block is processed by one GPU processing unit SM. SM execute a thread block with the unit of warp, where each warp contains 32 threads. All member threads of a warp share the same data and execution instruction. Therefore, SM of GPU must be considered for designing a kernel, and it is recommended that threads per block should be multiplied by warp size 32. However, a maximum of 1024 threads per block is supported by CUDA.

6.1.5.4 Task schedule

Lastly, task scheduling should be considered to manage the execution of multiple CPU cores and GPU for maximising the performance of the target application. The main bottleneck of CPU+GPU is data availability during program execution. To overcome this limitation, the use of multiple memory buffers is recommended as it will allow multiple threads to access the data at the same time. This point, proposing to scheduling using two memory buffer is presented in Section 4.6.2.4.

Table 6.3 provides a general overview based on the experience accumulated during the development of this Thesis. Thus, guessing an assessment for different types of actions (operations), of how deep the knowledge of the architecture and hardware needs to be, an estimation of the development time, and additional dependencies, if any.

Table 6.3. Summary of the different operations suggested to be considered for GPU-based implementation and their evaluation.

Operation	Knowledge level	Time to development	Dependencies
Algorithm redesign	Very high	Very high	Knowledge about the algorithm
Data ordering	Low	Medium	Knowledge about the algorithm
Memory allocation	Low	Medium	CUDA instruction set
Data transfer	Low	Low	CUDA instruction set
Kernel distribution	Medium	High	CUDA instruction set, Knowledge about algorithm
Task schedule	Medium	High	CUDA instruction set, Knowledge about algorithm

6.1.6 Validation

Validating the modified implementation is crucial to assess the precision and quality of the video decoder. For video applications, PSNR, MSE, and SSIM metrics are recommended to be tested to check the quality of videos against its reference. The optimal value of PSNR is infinite, MSE is zero, and SSIM is one, i.e. when two videos are exactly the same. In this work, for measuring these metrics, Vooya raw video player has been used, which also provides different visual tools to check the video. Vooya supports Linux, Windows, and Mac operating systems, but is open source for Linux operating systems only. Moreover, the MD5 message-digest algorithm hash function can be used to compare two videos. MD5 provides a 128-bit hash value for each unique file. The hash value is changed even if the video is slightly modified. Therefore, it is only suggested to use this method for comparing two videos if the exact same video is required after optimisation. In addition, this method does not provide any information about the differences. This methodology is intended to keep the exact same quality of the raw videos after its use, where the trade-off between video quality and performance improvement using lossy optimisation is not considered. However, that can be easily defined by the application requirements using different values of PSNR, MSE, and SSIM metrics.

Chapter 7

Results and contributions of the Thesis

This chapter summarises the results obtained in this thesis work and lists the contributions that have been generated during its development (publications, contributions to research projects, and direction of student's work).

7.1 Objectives of the Thesis

The objectives of the doctoral thesis were presented in Chapter 1. As a reminder, the main objective of this doctoral thesis is to define new design methodology to implement efficient video decoders on GPP+GPU based embedded heterogeneous platform. This objective was achieved by parallel exploiting the multicore technology of GPP's, SIMD optimisation, and GPU accelerator, where the decoder was redesigned for the parallel processing of video sequences. This methodology was validated through the implementation of different video decoders and different versions of the same video decoder compatible with the VVC standard on different platforms.

The secondary objectives are the following:

- To provide the profile and performance analysis of the decoder that is necessary to target the decoder module for its acceleration.
- To analyse the parallel processing abilities of the VVC decoder blocks and establish a design methodology to accelerate the applications of video processing over heterogeneous platforms.
- To analyse the energy consumption and the memory usage analysis on embedded heterogeneous platforms.

This Thesis concludes by addressing these objectives in the following Section they are summarised according to their respective contribution.

7.2 Contributions of the Thesis

This section presents the results achieved during the doctoral thesis that satisfied the objectives summarised in Chapter 1. The most relevant contributions are highlighted below.

- During the development of this doctoral thesis, different versions of various VVC decoders were migrated for different types of hardware, including VTM v8.0, 10.2, VVdeC v0.2, 1.0, 1.1, 1.2, 1.1, 1.3. These decoders were initially designed for the x86 architecture. In this study, these decoders were configured for widely used ARM based architecture used in different consumer electronics devices, and numerous embedded heterogeneous platforms. This contribution enhanced the versatile use of the state-of-the-art VVC video decoders.

- A detailed analysis of the above mentioned VVC decoders was performed where the coarse-grain and fine-grain profiles are performed to identify computationally heavy decoder modules with the aim of accelerating the decoder. This study was published in a peer-reviewed scientific journal [16]. This information was helpful in accelerating VVC decoders during this doctoral thesis and will also be useful to other researchers.

- The above mentioned version of the VVdeC decoders was optimised using Neon-based SIMD for ARM-based embedded platforms. This optimisation resulted in an average speedup of $\times 2$ in the total decoding time. This work was published in an international peer-reviewed journal [17]. In addition, a similar approach was adapted by the VVdeC decoder in its version 1.4.

- Different versions (0.2 and 0.3) of the OpenVVC decoder were studied and developed the Neon-based SIMD implementation of OpenVVC v1.0 for the ARM architecture. This work achieved an average speedup of $\times 1.8$ on embedded platforms, which led to a joint publication [18] (currently under review process) between the INSA IETR laboratory in Rennes (France) and the GDEM CITSEM of UPM.

- Further, the VVdeC decoder was accelerated using a CPU+GPU based approach. Here, a methodology was developed to process VVdeC ALF block using GPU. An

average speedup of ×2 was achieved for this module which causes an increase of up to 20% in the total decoding time on an embedded heterogeneous platform Xavier. This work was the result of international collaboration between INESC-ID in IST, Lisbon Portugal, and GDEM of UPM, where a joint journal article [19] was submitted to a peer-reviewed international journal and is currently under review process.

- The energy consumption and memory usage analysis were conducted on embedded heterogeneous platforms, which are two important factors for resource-constrained mobile platforms. The results of these analyses will be helpful in choosing the right platform for the video processing application. Furthermore, the results obtained for the analysis of energy consumption show that a similar energy consumption per frame was achieved for proposed CPU+GPU based approach compared to the CPU based approach. This means that more performance was achieved using almost the same energy consumption.

7.3 Works published related with the Thesis

The results obtained in this doctoral thesis have resulted in two publications and two submitted papers, currently under review process to journals included in the JCR index and two in international conferences with peer review. In this doctoral Thesis, we mainly focused on journals rather than conferences. We avoided travel due to the COVID-19 pandemic. The publications in journals are summarised below:

- A. Saha, M. Chavarrías, F. Pescador, Á.M. Groba, K. Chassaigne, P.L. Cebrián, "Complexity Analysis of a Versatile Video Coding Decoder over Embedded Systems and General Purpose Processors," Sensors 2021, 21, 3320. https://doi.org/10.3390/s21103320. Quartile Q2 and impact factor 3.847 (2021).

This article [16] presents a detailed complexity analysis of the VTM8.0 based VVC decoder on the HGPP-based Ryzen and resource-constrained embedded Xavier platform. Here, a detailed analysis of the new VVC standard was performed to support future implementation and essential optimisations. In addition, the ALF, DBF, and IP blocks were profiled in depth. This served as a starting point for characterising VVC decoders with the only implementation available at the time. Moreover, a comparison of the correlations between the computational load of the decoder blocks over two different types of architecture was shown. Finally, the obtained results were compared with other VVC implementations published in scientific journals.

- A. Saha, M. Chavarrías, V. Aranda, M. J. Garrido and F. Pescador, "Implementation of a Real-time Versatile Video Coding Decoder based on VVdeC over an Embedded Multi-core Platform," in IEEE Transactions on Consumer Electronics, 2022, doi: 10.1109/TCE.2022.3202512. Quartile Q2 and impact factor 4.414 (2021).

In this article [17], the profile of the VVdeC v0.2 decoder was shown for the implementation with and without SIMD optimisation in the X-series architecture based on the HGPP and embedded Xavier platform. Here, a optimisation technique was introduced to accelerate the VVdeC v0.2 decoder over Xavier using Neon-based SIMD optimisation. Lastly, the obtained results were compared with other research available at that time.

- A. Saha, W. Hamidouche, M. Chavarrías, G. Gautier, F. Pescador, and I. Farhat, "Performance Analysis of Optimized Versatile Video Coding Software Decoders on Embedded Platforms" [Online]. Available: `https://arxiv.org/abs/2206.15311` (under review process in IEEE Transactions on Consumer Electronics).

This study [18] focused on accelerating the OpenVVC v1.0 decoder using frame-tile parallelism and Neon-based SIMD optimisation for two embedded platforms. Here, different frame-tile configurations of the OpenVVC v1.0 decoder were studied. In addition, the results of performance, energy consumption, and memory usage analysis were compared for different configurations of the OpenVVC v1.0 decoder and VVdeC v1.3 decoder on Xavier and Nano.

- A. Saha, N. Roma, M. Chavarrías, T. Dias, F. Pescador and V. Aranda, "GPU-based Parallelisation of a Versatile Video Coding Adaptive Loop Filter in Resource-Constrained Heterogeneous Embedded Platform, (under review process in Journal of Real-Time Image Processing)"

This article [19] proposed a GPU-based parallelisation technique to accelerate the VVdeC v1.3 decoder by accelerating ALF. Here, the redesign of the VVdeC ALF algorithm was carried out to exploit the GPU architecture. In addition, this proposed implementation uses GPU simultaneously with the multithreading feature and SIMD optimisation of CPU. Lastly, energy consumption analysis was reported and found that the proposed solution achieved higher performance using almost the same energy per frame.

International conferences with peer review process: There are not many conferences published during the development of this Thesis, because COVID19 does not allow travelling.

- R. Medina, A. Saha, M. Floriano, M. Chavarrías and F. Pescador, "Porting Adaptive Multiple Transforms of a Versatile Video Coding decoder using OpenMP," 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), 2019, pp. 138-139, doi: 10.1109/ICCE-Berlin47944.2019.8966176.

This article [21] introduced a GPU based approach to accelerate the VVC AMT feature on top of OpenHEVC. Here, AMT was processed on GPU and the rest of the decoding tasks were processed on CPU on Xavier. This was a preliminary work done before the standardisation of the VVC standard.

- M. F. Vázquez, A. Saha, R. M. Morillas, M. C. Lapastora and F. P. d. Oso, "Work-in-Progress: Porting new Versatile Video Coding transforms to a heterogeneous GPU-based technology," 2019 International Conference on Compliers, Architectures and Synthesis for Embedded Systems (CASES), New York, USA, 2019, pp. 1-2.

This article [22] discussed OpenMP [131] based implementation of the VVC AMT feature on top of OpenHEVC over three different architectures. Here, POSIX [132] based approach was replaced by OpenMP based implementation. This was a preliminary work done when the VVC standard was under development.

7.4 Other results related to the Thesis

This section presents other results and funding related to the research work carried out: the development of new international collaboration between the GDEM group of CITSEM UPM and INESC-ID in IST Lisbon, strengthen the existing international collaboration between the GDEM group and INSA RENNS, the direction of different end-of-degree projects and master thesis by the doctoral candidate, and the scholarships received during the development of the thesis.

7.4.1 Research projects

The work developed during this doctoral thesis has been closely related to one research project of which the GDEM group was a part. This provided funding for the development of the doctoral Thesis.

IVME – Immersive Visual Media Environments (30-12-2016 to 29-12-2019, TEC2016-75981-C2-2-R, Ministerio de Economía y Hacienda)[11]. The goal of this project is to

develop omnidirectional digital video processing techniques, which enable users to provide novel immersive visualisation and interaction experiences. In addition, one activity of the project was related to the optimisation of video decoders.

7.4.2 Collaboration during the Thesis with the INSA in Rennes

This doctoral candidate collaborated with the INSA IETR laboratory in Rennes (France) between 30 July 2021 and 30 January 2022. This collaboration strengthened the historic relationship between the INSA IETR laboratory in Rennes (France) and GDEM of UPM. The INSA IETR laboratory in Rennes (France) developed the OpenVVC decoder and GDEM optimised the OpenVVC decoder using SIMD for ARM-based platforms. The OpenVVC decoder was accelerated on average $\times 1.9$ on Xavier and $\times 1.7$ Nano using Neon-based SIMD instructions. This collaboration resulted in a joint publication that was submitted to an international journal [18].

7.4.3 Collaboration and stay during the Thesis at the INESC-ID in IST

During the doctoral study, the doctoral candidate spent a research stay in the INESC-ID laboratory of the Instituto Superior Técnico (IST), University of Lisbon Portugal between September 15th 2021 and December 17th 2021. The professor responsible for the stay was Dr. Nuno Roma. This research stay aimed to acquire knowledge about parallel programming mainly using GPU. Here, GPU architecture and CUDA programming API were explored and extensively studied in the area of video decoder application to accelerate VVdeC. In addition, the student attended the "Parallel and Heterogeneous Computing Systems" course during the stay. This course provided the fundamentals of parallel programming and introduced different types of parallel programming techniques. To summarise, the result obtained from this collaboration was the hybrid implementation of the VVdeC decoder using CPU+GPU, which achieved an average speedup $\times 2$ in the ALF filtering time and an average speedup $\times 1.1$ of the total decoding time compared to the SIMD optimised solution in CPU. Moreover, this collaboration resulted in a co-authored article in the journal [19]. Finally, this research stay started a collaboration between INESC-ID, IST and GDEM, UPM. This research stay is also one of the requirements for PhD international mention. This stay was funded by Santander Research Scholarships (see Section 7.4.5).

7.4.4 Supervision of Final Degree Projects

Different works were carried out by students from Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación (ETSIST)-UPM that have led to respective Final Projects and Master's Thesis in regarding to some of the milestones that were developed during the research work associated with the doctoral thesis.

The doctoral candidate has supervised the following Bachelor Degree Project:

- Manuel Floriano Vázquez, "Integración de nuevos algoritmos de descodificación de vídeo digital HEVC y VVC sobre la plataforma heterogénea de NVIDIA Jetson AGX Xavier". Completed in July 2019 (Mark 10/10).
- Sergio Baz López, "Integración de técnicas de procesamiento paralelo en un descodificador de vídeo de última generación sobre la plataforma Jetson AGX Xavier". Completed in October 2020 (Mark 10/10).
- Víctor Aranda López, "Integración de técnicas de optimización y aceleración sobre un descodificador Versatile Video Coding en sistemas empotrados heterogéneos". Completed in July 2021 (Mark 10/10).
- Kheyter Augusto Chassaigne, "Integración de técnicas de procesamiento paralelo en el descodificador de vídeo de última generación VVC, sobre la plataforma Jetson AGX Xavier". Completed in September 2021 (Mark 10/10).

The doctoral student has supervised the following Master's Degree Thesis:

- Víctor Aranda López, "Analysis, optimization and testing of the Versatile Video Coding codec over edge devices for IoT-based applications". Completed in July 2022 (Mark 10/10).

These works mentioned above are closely related to the research work carried out in this doctoral thesis.

7.4.5 Scholarships and awards obtained

During the development of the thesis, the following scholarships were obtained:

- Santander Research Scholarships: "Programa propio de I+D+i de la UPM. Ayudas al personal investigador en formación predoctoral contratados o becados OTT" - September 2021. The duration of the scholarship was 3 months in Lisbon.

Chapter 8

Conclusions and future work

8.1 Conclusions

The optimisation of complex applications with high computational demanding using limited resources platforms is a challenge and the developers spend a lot of time in this process. To reduce the time to market of these applications, the definition of an optimisation methodology can be considered an interesting contribution to the state of the art. In this methodology special attention must be taken with the use of the selected hardware architectures based usually in GPPs and accelerators as GPUs.

In this PhD an efficient way to implement VVC decoders on resource-constrained embedded heterogeneous platforms based on ARM processors, where multithreading based parallelisation of CPU, SIMD optimisation and GPU are used concurrently, were the main objective. In order to handle the added complexity imposed by the state-of-the-art VVC decoder, the proposal is to optimise the use of threads with the parallelisation tools included in the standard (multi frame, tiles, WPP) and to parallelise efficiently the most computationally demanding modules with higher parallel processing ability using GPU and the rest of the decoder modules using the data-level parallelisation based on the SIMD instructions of the General Propose CPU.

These techniques have been applied to a limited resources heterogeneous platform called Xavier (see Section 4.2.1.2) using an state of the art open source decoder VVdeC (see Section 2.5.2) with excellent results achieving real-time performance for HD, full HD and some ultra high definition sequences defined in the standard. An average speed-up of $\times 2$ is obtained with this platform and this implementation.

To achieve these goals several optimisations phases have been defined, first, fine-grain

and coarse-grain profiling of the VVC decoders was carried out first. Next some automatic course-grain techniques was applied. The speed-up obtained is limited so additional techniques must be applied. Therefore, VVdeC was optimised using Neon-based SIMD instructions available in the the ARM architecture. The results obtained was very significant and real time performance is obtained for some standard sequences. Then, an hybrid approach was used in order to take advantage of the GPUs integrated in the platform. In CPU+GPU implementation, the VVdeC ALF was accelerated using GPU and other decoding blocks were processed in CPU. ALF was chosen to be processed in GPU due to the fact that it was found the most time-consuming block of VVC decoders on heterogeneous platforms even after SIMD optimisation was applied. Moreover, ALF algorithm has a high degree of parallel processing ability, as it involves several repetitive arithmetic operations. In this optimisation phase, it is critical to analyse the data transfer efficiency between GPP and GPU because it can spend more time than the improvement achieved with the parallel processing of the algorithm.

Using the experience obtained after the optimisation process, other platform called Nano was selected (see section 4.2.1.2) with lower resources (memory size, number of cores or memory bandwidth), and the same decoder was optimised with the same techniques obtained similar results with an average speed-up of $\times 1.7$. This result was obtained with a lower effort than needed in the previous platform. In this platform the use of GPU doesn't generate an improvement due to the low bandwidth between GPU and GPU.

In order to generalise the optimisation process, other VVC decoder implementation called OpenVVC (see Section 2.5.3) has been optimised for both platforms using the same techniques in a shorter period of time. In this platform an average speed-up of $\times 1.8$ is obtained for Xavier and $\times 1.7$ for Nano. Again, these results are similar to achieved with the previous decoder.

Finally, with the experiences obtained, the methodology presented in chapter 6 was developed to optimise algorithms using the hybrid approach where the fine-grain SIMD optimisation and CPU+GPU based hardware accelerator were used along with the default coarse-gain optimisation. This methodology allows the developers to accelerate the optimisation process of video applications using heterogeneous platforms and it could be applied to other complex algorithms in similar heterogeneous platforms.

Additionally to the methodology to optimise the performance, other critical factors for resource-constrained embedded platforms have been analysed in this work: 1) memory usage and 2) energy consumption. Both parameters have been measured and compared for different decoder implementations and platforms. A comparison was made between the memory usage and energy consumption of the VVdeC and OpenVVC decoders (see

Section 5.10 and 5.11.1). Additionally, the energy consumption analysis of the CPU+GPU based hybrid implementation was conducted and compared with CPU implementation. This information will be useful in determining the most suitable platform for the video processing application and for future optimisations.

To better highlight the presented novelties, the main contributions of this dissertation are summarised as follows:

- Fine-grain and coarse-grain profiling of the VVC decoders was performed on HGPP and EGPP based platforms. This insight was essential in speeding up the VVC decoders for this doctoral dissertation and will be beneficial to other studies.
- Optimisation of the VVC-based video decoders: VVdeC and OpenVVC was accelerated using SIMD instructions for heterogeneous embedded platforms based on the ARM architecture. Moreover, an average $\times 2$ and $\times 1.8$ speedup was achieved on two ARM architecture-based embedded platforms for the VVdeC and OpenVVC decoder implementations, respectively. Here, EP and ALF were the most benefited decoder blocks by SIMD instructions for both OpenVVC and VVdeC decoder, which also were the highest computational demands decoder blocks. It means that the same methodology can be used for different decoder implementations and different platforms.
- A hybrid approach was implemented using the fine-grain SIMD instructions and CPU+GPU based hardware accelerator along with the default coarse-gain optimisation. In the CPU+GPU based implementation, data accessing pattern for ALF filtering has been redesigned, different data transfer methods have been evaluated and the most efficient data transfer method has been identified. The selection of the data transfer method between GPP and GPU is critical to decide if the use of GPUs can help to improve the performance.
- An analysis of energy consumption and memory usage over embedded heterogeneous platforms has been carried out. The results of the energy consumption analysis presented that almost the same energy per frame was consumed to achieve better performance using CPU+GPU based hybrid approach than CPU based approach. Although GPU normally consumes very high energy, the proposed implementation compensated energy per frame by achieving better decoder performance optimisation.
- The proposed solutions have been compared by means of the decoding performance, energy consumption, and memory usage of VVdeC and OpenVVC decoders on different heterogeneous platforms.

In summary, a methodology was proposed where at the beginning the platform and video standard were selected. Then, a detailed analysis of the decoder was performed to identify the most computationally intensive modules for optimisation using a coarse- and fine-grain profile. A hybrid approach was implemented using the fine-grain SIMD instructions and CPU+GPU based hardware accelerator along with the default coarse-gain optimisation.

A complete decoding solution using two decoder implementations was provided using the proposed methodology that maximises the use of resources on embedded heterogeneous platforms. In general, an average speedup of $\times 2.1$ was achieved for the proposed approach on embedded heterogeneous platforms (see Section 5.9). Here, real-time decoding was achieved for all HD resolution sequences with 1280×720, FHD sequences with 1920×1080, and some UHD sequences with 3840×2160 over Xavier platform based on ARM architecture. Similar speed-up has been achieved with the platform Nano, obtaining real time performance for all HD resolution sequences with 1280×720 and some FHD sequences with 1920×1080.

This work has been acknowledged by several publications in scientific forums and journals. All these contributions have been summarised in Chapter 7, Section 7.3.

8.2 Future work

Finally, and as a conclusion to this dissertation, the following points are presented as proposals for future work. These items should be considered as lines of work that extend or complement the thesis from two points of view: a technical one (generation of energy consumption models or migration of other blocks from the decoder to the accelerators), and a second one that includes alternative approaches to some parts of the proposed design methodology (exploration alternative parallel coding techniques or the use of alternative accelerator architectures).

- Platforms including other accelerating architectures like FPGA or DSP could be considered and explored to execute the different blocks of state-of-the-art VVC decoders. These architectures were widely used for the preceding HEVC standard [86]-[89],[98]-[101]. Moreover, it is found from the work published in scientific literature that FPGA has proven to be efficient hardware used to run some blocks such as TX and IP [86]-[89]. The use of other architectures will not only accelerate the decoding speed but also increase the versatility of the decoder.
- Other concurrent code management techniques, like OpenMP, could be considered

to handle the intrinsic parallelism of state-of-the-art VVC decoders. OpenMP was already used as main parallel task manager in previous video decoders. In [133]-[135], HEVC decoders were instrumented using OpenMP. OpenMP-based source code is proposed for future work to accelerate VVC decoder due to the fact that most of the VVC decoder blocks are similar or based on same principle of HEVC decoder blocks.

- Moving other blocks of VVC decoder to GPU is suggested for future work. For example, DBF could be considered to be accelerated using GPU because it did not benefit much from SIMD optimisation. In addition, several works [95]-[97] published in scientific literature accelerated HEVC DBF using GPU. As VVC DBF is similar to HEVC DBF, it is recommended to use GPU for processing VVC DBF.

- Design a model based on energy-efficient machine learning techniques that dynamically adapts power usage according to the needs of both the environment (video demand) and video performance. Applications such as video decoding are likely to improve their impact on energy usage through machine learning models by having easy and continuous access to parameters such as performance or energy consumption. The GDEM research group, where this Thesis has been conducted, is currently working on the development of machine learning models that dynamically adjust to different scenarios in order to reduce energy consumption.

- Integration of the methodology proposed in this thesis in a tool that allows automating the execution of all or some of the proposed steps. In this case, tools based on automatic code generation from high-level descriptions of both the algorithm and the hardware could be considered as possible solutions. Such approaches have proven to be very effective in reducing design time and providing the solution with a high degree of modularity. However, they often have a negative impact on the final performance of the application.

Bibliography

[1] R.D. Caballar, "Battle of the Video Codecs: Coding-Efficient VVC vs. Royalty-Free AV1," [Online]. Available:https://spectrum.ieee.org/battle-video-codecs-hevc-coding-efficiency-vvc-royalty-free-av1.

[2] "Versatile Video Coding (VVC)," [Online]. Available:https://jvet.hhi.fraunhofer.de/.

[3] 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, vol. 22, no. 12, pp. 1649-1668, Dec. 2012, doi: 10.1109/TCSVT.2012.2221191.

[4] C. Feldmann, "Versatile Video Coding hits major milestone," [Online]. Available: https://bitmovin.com/compression-standards-vvc-2020.

[5] A. Wieckowski, G. Hege, C. Bartnik, C. Lehmann, C. Stoffers, B. Bross and D. Marpe, "Towards A Live Software Decoder Implementation For The Upcoming Versatile Video Coding (VVC) Codec," 2020 IEEE International Conference on Image Processing (ICIP), 2020, pp. 3124-3128. doi: 10.1109/ICIP40778.2020.9191199.

[6] T. Amestoy, P. Cabarat, G. Gautier, W. Hamidouche, and D. Menard, "OpenVVC: a Lightweight Software Decoder for the Versatile Video Coding Standard." publisher arXiv, 2022 [Online]. Available: https://doi.org/10.48550/arXiv.2205.12217.

[7] B. Zhu, S. Liu, Y. Liu, Y. Luo, J. Ye, H. Xu, Y. Huang, H. Jiao, X. Xu, X. Zhang and C. Gu, "A Real-Time H.266/VVC Software Decoder," 2021 IEEE International Conference on Multimedia and Expo (ICME), 2021, pp. 1-6, doi: 10.1109/ICME51207.2021.9428470.

[8] Y. Serpa, "Why are GPUs So Powerful?" [Online]. Available: https://towardsdatascience.com/

the-ai-illustrated-guide-why-are-gpus-so-powerful-99f4ae85a5c3#:~:text=This%20is%20throughput.,traveling%20by%20car.

[9] M. Chavarrías, 2017. "Aportaciones metodológicas para el diseño de descodificadores de vídeo de última generación sobre plataformas Multi-DSP," E.T.S.I. y Sistemas de Telecomunicación, Universidad Politécnica de Madrid, PhD thesis. Available: https://oa.upm.es/47193/.

[10] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, C. Sanz and P. Paz, "An FPGA-Based Architecture for the Versatile Video Coding Multiple Transform Selection Core," in IEEE Access, vol. 8, pp. 81887-81903, 2020, doi: 10.1109/ACCESS.2020.2991299.

[11] "Project IVME: Immersive Visual Media Environments", Jan. 2017 - Dec. 2019, [Online]. Available:https://www.citsem.upm.es/en/research/projects?view=projects&task=show&id=62.

[12] "Project MR-UHDTV: Mixed Reality over Ultra High Definition Television", Duration Jan. 2014 - Dec. 2017, [Online]. Available:https://www.citsem.upm.es/en/research/projects?view=projects&task=show&id=54.

[13] "Project H2B2VS: HEVC - Hybrid Broadcast Video Services", Duration Sep. 2012 - Apr. 2015, [Online]. Available:https://www.citsem.upm.es/en/research/projects?view=projects&task=show&id=13.

[14] "VTM VVC reference software" [Online]. Available:https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM.

[15] "Fraunhofer HHI VVdeC software repository," [Online]. Available: https://github.com/fraunhoferhhi/vvdec.

[16] A. Saha, M. Chavarrías, F. Pescador, Á.M. Groba, K. Chassaigne and P.L. Cebrián, "Complexity Analysis of a Versatile Video Coding Decoder over Embedded Systems and General Purpose Processors," Sensors 2021, 21, 3320. https://doi.org/10.3390/s21103320.

[17] A. Saha, M. Chavarrías, V. Aranda, M. J. Garrido and F. Pescador, "Implementation of a Real-time Versatile Video Coding Decoder based on VVdeC over an Embedded Multi-core Platform," in IEEE Transactions on Consumer Electronics, 2022, doi: 10.1109/TCE.2022.3202512.

[18] A. Saha, W. Hamidouche, M. Chavarrías, G. Gautier, F. Pescador, and I. Farhat, "Performance Analysis of Optimized Versatile Video Coding Software Decoders on Embedded Platforms" [Online]. Available:https://arxiv.org/abs/2206.15311.

[19] A. Saha, N. Roma, M. Chavarrías, T. Dias, F. Pescador and V. Aranda, "GPU-based Parallelisation of a Versatile Video Coding Adaptive Loop Filter in Resource-Constrained Heterogeneous Embedded Platform," (submitted)

[20] "OpenHEVC," [Online]. Available:https://github.com/OpenHEVC/openHEVC.

[21] R. Medina, A. Saha, M. Floriano, M. Chavarrías and F. Pescador, "Porting Adaptive Multiple Transforms of a Versatile Video Coding decoder using OpenMP," 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin), 2019, pp. 138-139, doi: 10.1109/ICCE-Berlin47944.2019.8966176.

[22] M. F. Vázquez, A. Saha, R. M. Morillas, M. C. Lapastora and F. P. d. Oso, "Work-in-Progress: Porting new Versatile Video Coding transforms to a heterogeneous GPU-based technology," 2019 International Conference on Compliers, Architectures and Synthesis for Embedded Systems (CASES), 2019, pp. 1-2.

[23] Dr. T. Hussain, "Multimedia Computing." Google Books, Booksclinic Publishing, 21 Oct. 2020, https://books.google.com/books/about/MULTIMEDIA_COMPUTING.html?id=pk4EEAAAQBAJ.

[24] "Image Processing 101 Chapter 1.2: Color Models" [Online]. Available:https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-models/#:~:text=A%20color%20space%20identifies%20a,on%20the%20RGB%20color%20model.

[25] D. Brunner, "Frame Rate: A Beginner's Guide", [Online]. Available: https://www.techsmith.com/blog/frame-rate-beginners-guide/.

[26] F. Romano, "Frame Rate History — Why Speeds Vary", [Online]. Available: https://vanillavideo.com/articles/history-frame-rates-why-speeds-vary/.

[27] N. Surana, "Video Resolutions: What they are, Different Types, and their Pixel Size", [Online]. Available: https://typito.com/blog/video-resolutions/.

[28] A.S. NAGARAGHATTA, 2019. "Algorithms and methods for video transcoding," Robert Gordon University [online], PhD thesis. Available: https://rgu-repository.worktribe.com/output/842023.

[29] ITU-T, “Recommendation H.120 : Codecs for Videoconferencing Using Primary Digital Group Transmission,” pp. 7–9, 1988.

[30] ITU-T, “Recommendation H.261: Video Codec for Audiovisual Services at p64 kbits,” 1993.

[31] ISO/IEC JTC 1/SC 29. "Programme of Work — Allocated to SC 29/WG 11, MPEG-1 (Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s)," 2009.

[32] ITU-T, “Recommendation H.262: Transmission of Non-Telephone Signals,” 1994.

[33] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 560–576, July 2003.

[34] Microsoft Corp. (April 28, 1997) Microsoft Licenses Duck Corp.’s TrueMotion 2.0 To Bring Television-Quality Video to the PC Archived 2008-04-30 at the Wayback Machine, Retrieved on 2009-08-11

[35] "On2’s VP3 Codec Available Via QuickTime 5’s Component Download Feature" [Online]. Available:`https://web.archive.org/web/20071203061516/http://www.on2.com/index.php?id=486&news_id=397`.

[36] G. Bjøntegaard, T. Davies, A. Fuldseth and S. Midtskogen, "The Thor Video Codec," 2016 Data Compression Conference (DCC), 2016, pp. 476-485, doi: 10.1109/DCC.2016.74.

[37] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi, C. -H. Chiang, Y. Wang, P. Wilkins, J. Bankoski, L. Trudeau, N. Egge, J. -M. Valin, T. Davies, S. Midtskogen, A. Norkin and P. de Rivaz, "An Overview of Core Coding Tools in the AV1 Video Codec," 2018 Picture Coding Symposium (PCS), 2018, pp. 41-45, doi: 10.1109/PCS.2018.8456249.

[38] L. Guo, O. C. Au, M. Ma, Z. Liang and P. H. W. Wong, "A Novel Analytic Quantization-Distortion Model for Hybrid Video Coding," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 5, pp. 627-641, May 2009, doi: 10.1109/TCSVT.2009.2017403.

[39] D. Karwowski, ”Precise Probability Estimation of Symbols in VVC CABAC Entropy Encoder,” in IEEE Access, vol. 9, pp. 65361-65368, 2021. doi: 10.1109/ACCESS.2021.3075875.

[40] J. Chen, Y. Ye and S. Kim, "JVET-Q2002-v3: Algorithm description for Versatile Video Coding and Test Model 8 (VTM 8)." Jvet 17th Meeting: Brussels, BE, 7–17 January 2020.

[41] X. Zhao, S. -H. Kim, Y. Zhao, H. E. Egilmez, M. Koo, S. Liu, J. Lainema and M. Karczewicz, "Transform Coding in the VVC Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 3878-3890, Oct. 2021, doi: 10.1109/TCSVT.2021.3087706.

[42] M. Koo, M. Salehifar, J. Lim and S. -H. Kim, "Low Frequency Non-Separable Transform (LFNST)," 2019 Picture Coding Symposium (PCS), 2019, pp. 1-5, doi: 10.1109/PCS48520.2019.8954507.

[43] R. Ghaznavi-Youvalari and J. Lainema, "Joint Cross-Component Linear Model For Chroma Intra Prediction," 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), 2020, pp. 1-5. doi: 10.1109/MMSP48831.2020.9287167.

[44] Y. -J. Cho, J. -H. Ko, H. -G. Yu, J. -H. Lee, D. -J. Park and S. -H. Jun, "Adaptive motion vector resolution based on the rate-distortion cost and coding unit depth," 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 1000-1003, doi: 10.1109/IAdCC.2014.6779460.

[45] F. Bossen, K. Sühring, A. Wieckowski and S. Liu, "VVC Complexity and Software Implementation Analysis," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 3765-3778, Oct. 2021. doi: 10.1109/TCSVT.2021.3072204.

[46] H. Liu, L. Zhang, K. Zhang, H. C. Chuang, Y. Wang and J. Xu, "Two-Pass Bi-Directional Optical Flow Via Motion Vector Refinement," 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 1208-1211, doi: 10.1109/ICIP.2019.8803763.

[47] M. Aklouf, M. Leny, F. Dufaux, and M. Kieffer, "Low Complexity Versatile Video Coding (VVC) for Low Bitrate Applications," 2019 8th European Workshop on Visual Information Processing (EUVIP), Roma, Italy, Oct. 2019; pp. 22-27, doi: 10.1109/EUVIP47703.2019.8946261.

[48] H. Gao, X. Chen, S. Esenlik, J. Chen and E. Steinbach, "Decoder-Side Motion Vector Refinement in VVC: Algorithm and Hardware Implementation Considerations," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 8, pp. 3197-3211, Aug. 2021, doi: 10.1109/TCSVT.2020.3037024.

[49] T. Lu, F. Pu, P. Yin, S. McCarthy, W. Husak, T. Chen, E. Francois, C. Chevance, F. Hiron, J. Chen, R. -L. Liao, Y. Ye and J. Luo, "Luma Mapping with Chroma Scaling in Versatile Video Coding," 2020 Data Compression Conference (DCC), 2020, pp. 193-202, doi: 10.1109/DCC47342.2020.00027.

[50] M. Karczewicz, N. Hu, J. Taquet, C. -Y. Chen, K. Misra, K. Andersson, P. Yin, T. Lu, E. François and J. Chen, "VVC In-Loop Filters," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 3907-3925, Oct. 2021, doi: 10.1109/TCSVT.2021.3072297.

[51] K. Andersson, K. Misra, M. Ikeda, D. Rusanovskyy and S. Iwamura, "Deblocking filtering in VVC," 2021 Picture Coding Symposium (PCS), 2021, pp. 1-5, doi: 10.1109/PCS50896.2021.9477477.

[52] J. Yang, B. Du and T. Tang, "Improved method of deblocking filter based on convolutional neural network in VVC," 2020 IEEE/CIC International Conference on Communications in China (ICCC), 2020, pp. 764-769, doi: 10.1109/ICCC49849.2020.9238791.

[53] C. -M. Fu, C. -Y. Chen, Y. -W. Huang and S. Lei, "Sample adaptive offset for HEVC," 2011 IEEE 13th International Workshop on Multimedia Signal Processing, 2011, pp. 1-5, doi: 10.1109/MMSP.2011.6093807.

[54] P. Lakshmi Amruthavalli and P. Nalluri, "A Review on In-Loop Filters for HEVC and VVC Video Coding Standards," 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), 2022, pp. 997-1001, doi: 10.1109/ICACCS54159.2022.9784992.

[55] P. Bordes, F. Galpin, T. Dumas and P. Nikitin, "Revisiting the Sample Adaptive Offset post-filter of VVC with Neural-Networks," 2021 Picture Coding Symposium (PCS), 2021, pp. 1-5, doi: 10.1109/PCS50896.2021.9477457.

[56] C. -Y. Tsai, C. -Y. Chen, T. Yamakage, I. S. Chong, Y. -W. Huang, C. -M. Fu, T. Itoh, T. Watanabe, T. Chujoh, M. Karczewicz and S. -M. Lei, "Adaptive Loop Filtering for Video Coding," in IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 934-945, Dec. 2013, doi: 10.1109/JSTSP.2013.2271974.

[57] J. Erfurt, W. Lim, H. Schwarz, D. Marpe, and T. Wiegand, "Extended multiple feature-based classifications for adaptive loop filtering," in APSIPA Transactions on Signal and Information Processing, vol. 8, E28, 2019, doi:10.1017/ATSIP.2019.19.

[58] X. Wang, H. Sun, J. Katto and Y. Fan, "A Hardware Architecture for Adaptive Loop Filter in VVC Decoder," 2021 IEEE 14th International Conference on ASIC (ASICON), 2021, pp. 1-4, doi: 10.1109/ASICON52560.2021.9620332.

[59] "HM HEVC reference software" [Online]. Available:`https://vcgit.hhi.fraunhofer.de/jct-vc/HM`.

[60] "Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video," [Online]. Available: `https://ffmpeg.org/`.

[61] "GPAC: Multimedia Open Source Project," [Online]. Available: `https://gpac.wp.imt.fr/`.

[62] A. Wieckowski, C. Lehmann, B. Bross, D. Marpe, T. Biatek, M. Raulet, and J. Le Feuvre, "A Complete End to End Open Source Toolchain for the Versatile Video Coding (VVC) Standard," 2021 Proceedings of the 29th ACM International Conference on Multimedia, Association for Computing Machinery, New York, NY, USA, 3795–3798.

[63] "OpenVVC software repository," [Online]. Available:`https://github.com/OpenVVC/OpenVVC`.

[64] "VLC media player: VideoLAN, a project and a non-profit organization," [Online]. Available: `https://www.videolan.org/`.

[65] T. Amestoy, P. Cabarat, G. Gautier, W. Hamidouche and D. Menard, "OpenVVC: a Lightweight Software Decoder for the Versatile Video Coding Standard," arXiv preprint arXiv:2205.12217, 2022.

[66] “Tencent O266dec decoder library,” [Online]. Available: `https://github.com/TencentCloud/O266player`.

[67] D. Vandevoorde and N. M. Josuttis, "C++ Templates", The Complete Guide. Addison-Wesley, 2003.

[68] B. Zhu, S. Liu, Y. Liu, Y. Luo, J. Ye, H. Xu, Y. Huang, H. Jiao, X. Xu, X. Zhang and C. Gu., "A Real-Time H.266/VVC Software Decoder," 2021 IEEE International Conference on Multimedia and Expo (ICME), 2021, pp. 1-6, doi: 10.1109/ICME51207.2021.9428470.

[69] "Intel® Instruction Set Extensions Technology," [Online]. Available: `https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html`.

[70] "Neon", [Online]. Available: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon.

[71] "ARM Advanced SIMD (NEON) Intrinsics and Types in LLVM", [Online]. Available: https://blog.llvm.org/2010/04/arm-advanced-simd-neon-intrinsics-and.html.

[72] "Learn the architecture - Optimizing C code with Neon intrinsics", [Online]. Available: https://developer.arm.com/documentation/102467/0100/Check-your-knowledge.

[73] S. Gudumasu, S. Bandyopadhyay, and Y. He, "Software-based versatile video coding decoder parallelization," in Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20), Association for Computing Machinery, New York, NY, USA, 202–212, 2020. doi: 10.1145/3339825.3391871.

[74] M. Koziri, P. K. Papadopoulos, N. Tziritas, N. Giachoudis, T. Loukopoulos, S. U. Khan and G. I. Stamoulis, "Heuristics for tile parallelism in HEVC," 2017 25th European Signal Processing Conference (EUSIPCO), 2017, pp. 1514-1518, doi: 10.23919/EUSIPCO.2017.8081462.

[75] T. Amestoy, W. Hamidouche, C. Bergeron and D. Menard, "Quality-Driven Dynamic VVC Frame Partitioning for Efficient Parallel Processing," arXiv preprint: https://doi.org/10.48550/arXiv.2012.14792.

[76] J. Wang and X. Zhou, "Wavefront parallel processing based on POSIX threads," 2016 IEEE International Conference on Consumer Electronics-China (ICCE-China), 2016, pp. 1-5, doi: 10.1109/ICCE-China.2016.7849744.

[77] J. Sancho, P. Sutradhar, G. Rosa, M. Chavarrías, A. Perez-Nuñez, R. Salvador, A. Lagares, E. Juárez and C. Sanz, "GoRG: Towards a GPU-Accelerated Multiview Hyperspectral Depth Estimation Tool for Medical Applications," Sensors 2021, 21, 4091. https://doi.org/10.3390/s21124091

[78] K. Punithakumar, P. Boulanger and M. Noga, "A GPU-Accelerated Deformable Image Registration Algorithm With Applications to Right Ventricular Segmentation," in IEEE Access, vol. 5, pp. 20374-20382, 2017, doi: 10.1109/ACCESS.2017.2755863.

[79] "CUDA TOOLKIT," [Online]. Available: https://developer.nvidia.com/cuda-toolkit.

[80] P. Gupta, "CUDA Refresher: The CUDA Programming Model," [Online]. Available: https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/.

[81] "IEEE Standard Verilog Hardware Description Language," in IEEE Std 1364-2001 , vol., no., pp.1-792, 28 Sept. 2001, doi: 10.1109/IEEESTD.2001.93352.

[82] "FPGA vs GPU, What to Choose?," [Online]. Available: https://hardwarebee.com/fpga-vs-gpu-choose/.

[83] A. Kammoun, W. Hamidouche, P. Philippe, O. Déforges, F. Belghith, N. Masmoudi and J. -F. Nezan, "Forward-Inverse 2D Hardware Implementation of Approximate Transform Core for the VVC Standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 11, pp. 4340-4354, Nov. 2020, doi: 10.1109/TCSVT.2019.2954749.

[84] H. M. Waidyasooriya, M. Hariyama, H. Iwasaki, D. Kobayashi, Y. Omori, K. Nakamura, K. Nitta and K. Sano, "OpenCL-Based Design of an FPGA Accelerator for H.266/VVC Transform and Quantization," 2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS), 2022, pp. 1-4, doi: 10.1109/MWSCAS54063.2022.9859281.

[85] H. Azgin, E. Kalali and I. Hamzaoglu, "An Efficient FPGA Implementation of Versatile Video Coding Intra Prediction," 2019 22nd Euromicro Conference on Digital System Design (DSD), 2019, pp. 194-199, doi: 10.1109/DSD.2019.00037.

[86] S. Abdellah, S. Youcef and D. Lamine, "Implementation of HEVC intra 4×4 prediction on FPGA," 2015 Science and Information Conference (SAI), 2015, pp. 1160-1164, doi: 10.1109/SAI.2015.7237291.

[87] H. Azgin, A. C. Mert, E. Kalali and I. Hamzaoglu, "An efficient FPGA implementation of HEVC intra prediction," 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1-5, doi: 10.1109/ICCE.2018.8326332.

[88] E. Kalali and I. Hamzaoglu, "FPGA implementations of HEVC Inverse DCT using high-level synthesis," 2015 Conference on Design and Architectures for Signal and Image Processing (DASIP), 2015, pp. 1-6, doi: 10.1109/DASIP.2015.7367262.

[89] V. Viitamäki, P. Sjövall, J. Vanne and T. D. Hämäläinen, "High-level synthesized 2-D IDCT/IDST implementation for HEVC codecs on FPGA," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050323.

[90] X. Han, S. Wang, S. Ma and W. Gao, "Optimization Of Motion Compensation Based On GPU And CPU For VVC Decoding," 2020 IEEE International Conference on Image Processing (ICIP), 2020, pp. 1196-1200, doi: 10.1109/ICIP40778.2020.9190708.

[91] J. Ma, F. Luo, S. Wang and S. Ma, "Flexible CTU-level parallel motion estimation by CPU and GPU pipeline for HEVC," 2014 IEEE Visual Communications and Image Processing Conference, 2014, pp. 282-285, doi: 10.1109/VCIP.2014.7051559.

[92] H. Igarashi, F. Takano and T. Moriyoshi, "Highly parallel transformation and quantization for HEVC encoder on GPUs," 2016 Visual Communications and Image Processing (VCIP), 2016, pp. 1-4, doi: 10.1109/VCIP.2016.7805520.

[93] L. -p. He and S. Goto, "A high parallel way for processing IQ/IT part of HEVC decoder based on GPU," 2014 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), 2014, pp. 211-215, doi: 10.1109/ISPACS.2014.7024454.

[94] S. Radicke, J. -U. Hahn, Q. Wang and C. Grecos, "A Parallel HEVC Intra Prediction Algorithm for Heterogeneous CPU+GPU Platforms," in IEEE Transactions on Broadcasting, vol. 62, no. 1, pp. 103-119, March 2016, doi: 10.1109/TBC.2015.2505401.

[95] Y. Wang, X. Guo, X. Fan, Y. Lu, D. Zhao and W. Gao, "Parallel In-Loop Filtering in HEVC Encoder on GPU," in IEEE Transactions on Consumer Electronics, vol. 64, no. 3, pp. 276-284, Aug. 2018, doi: 10.1109/TCE.2018.2867812.

[96] D. F. de Souza, A. Ilic, N. Roma and L. Sousa, "HEVC in-loop filters GPU parallelization in embedded systems," 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015, pp. 123-130, doi: 10.1109/SAMOS.2015.7363667.

[97] D. F. de Souza, N. Roma and L. Sousa, "Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 4993-4997, doi: 10.1109/ICASSP.2014.6854552.

[98] S. Lee, J. Song, W. Lee, D. Kim, J. Kim and S. Lee, "DSP based programmable FHD HEVC decoder," 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 972-973.

[99] P. Rodriguez, F. Balseiro, M. Chavarrías, F. Pescador and M. Garrido, "A DSP-based HEVC decoder implementation using RVC-CAL and native OpenHEVC code," 2015 International Symposium on Consumer Electronics (ISCE), 2015, pp. 1-2, doi: 10.1109/ISCE.2015.7177782.

[100] Y. Zhang, R. Fan, C. Zhang, G. Wang and Z. Li, "SIMD acceleration for HEVC encoding on DSP," 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2017, pp. 1719-1725, doi: 10.1109/APSIPA.2017.8282310.

[101] M. Chavarrías, F. Pescador, M. J. Garrido, E. Juárez and C. Sanz, "A multicore DSP HEVC decoder using an actor-based dataflow model," 2015 IEEE International Conference on Consumer Electronics (ICCE), 2015, pp. 370-371, doi: 10.1109/ICCE.2015.7066449.

[102] "FPGA vs. GPU vs. CPU: Which Is the Best Choice for Your Application?," [Online]. Available: https://www.raypcb.com/fpga-vs-gpu-vs-cpu/#:~:text=A%20GPU%20can%20perform%20general,right%20choice%20for%20your%20needs.

[103] R. Rachita, "Graphics processing unit (GPU) Market," [Online]. Available: https://www.alliedmarketresearch.com/graphic-processing-unit-market.

[104] I. Cutress, "AMD Zen Microarchiture Part 2: Extracting Instruction-Level Parallelism," [Online]. Available: https://www.anandtech.com/show/10591/amd-zen-microarchiture-part-2-extracting-instructionlevel-parallelism/5.

[105] "AMD Ryzen™ Threadripper™ Processors," [Online]. Available: https://www.amd.com/en/products/ryzen-threadripper.

[106] "Intel Core i9," [Online]. Available: https://www.profesionalreview.com/intel/intel-core-i9/.

[107] "Intel® Core™ i9-10900X X-series Processor," [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/198019/intel-core-i910900x-xseries-processor-19-25m-cache-3-70-ghz.html.

[108] "NVIDIA Jetson AGX Xavier Developer Kit, User Guide," DA_09403_003, December 17, 2019, [Online]. Available: https://developer.nvidia.com/jetson-agx-xavier-developer-kit-user-guide.

[109] "NVIDIA Jetson Nano Developer Kit, User Guide," DA_09402_004, January 15, 2020, [Online]. Available: https://developer.nvidia.com/embedded/dlc/Jetson_Nano_Developer_Kit_User_Guide.

[110] "JETSON STORE," [Online]. Available: https://store.nvidia.com/en-us/jetson/store/?page=1&limit=9&locale=en-us.

[111] "Valgrind's Tool Suite," [Online]. Available: https://www.valgrind.org/info/tools.html.

[112] "Callgrind: a call-graph generating cache and branch prediction profiler," [Online]. Available: https://valgrind.org/docs/manual/cl-manual.html.

[113] SSE for NEON library, "sse2neon", [Online]. Available: https://github.com/DLTcollab/sse2neon.

[114] E. Nemerson, "Transitioning SSE/AVX code to NEON with SIMDe," [Online]. Available: https://simd-everywhere.github.io/blog/2020/06/22/transitioning-to-arm-with-simde.html.

[115] "ARM Compiler armasm User Guide Version 5.06," [Online]. Available: https://developer.arm.com/documentation/dui0473/m/vfp-instructions/vadd--floating-point-.

[116] S. J. Kim, W. Joo and D. H. Kim, "Raster scan waveform compensation control for enhancing the orthogonality of images in SEM," in Microscopy, vol. 62, no. 4, pp. 475-484, Aug. 2013, doi: 10.1093/jmicro/dft024.

[117] N. V. Sunitha, K. Raju and N. N. Chiplunkar, "Performance improvement of CUDA applications by reducing CPU-GPU data transfer overhead," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), 2017, pp. 211-215, doi: 10.1109/ICICCT.2017.7975190.

[118] M. Bayati, M. Leeser and N. Mi, "Exploiting GPU Direct Access to Non-Volatile Memory to Accelerate Big Data Processing," 2020 IEEE High Performance Extreme Computing Conference (HPEC), 2020, pp. 1-6, doi: 10.1109/HPEC43674.2020.9286174.

[119] "C++ API Routines: cudaMallocHost", [Online]. Available: http://horacio9573.no-ip.org/cuda/group__CUDART__HIGHLEVEL_ge439496de696b166ba457dab5dd4f356.html.

[120] R. P. Ponnuraj, "CUDA — Memory Model," [Online]. Available: https://medium.com/analytics-vidhya/cuda-memory-model-823f02cef0bf.

[121] K. Desnos, 2014. "Memory Study and Dataflow Representations for Rapid Prototyping of Signal Processing Applications on MPSoCs," IETR - Institut d'Électronique et des Technologies du numéRique, INSA de Rennes, ⟨NNT : 2014ISAR0004⟩. ⟨tel-01127297⟩, PhD thesis. Available: https://theses.hal.science/tel-01127297.

[122] "vooya :: raw Video Sequence Player," [Online]. Available: https://www.offminor.de/.

[123] "function md5," [Online]. Available: https://www.md5.cz/.

[124] "time(1) — Linux manual page," [Online]. Available: https://man7.org/linux/man-pages/man1/time.1.html.

[125] "Software-Based Power Consumption Modeling" [Online]. Available: https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-325/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0AG0HA.

[126] F. Bossen, J. Boyce, X. Li, V. Seregin, and K. Sühring, "JVET Common Test Conditions and Software Reference Configurations for SDR Video," Document JVET-N1010, JVET of ITU-T, Geneva, Mar 2019.

[127] "CMake - Cross Platform Makefile Generator," 2000-2021 Kitware, Inc. and Contributors [Online]. Available: https://cmake.org/.

[128] "Auto-vectorization in GCC" [Online]. Available: https://gcc.gnu.org/projects/tree-ssa/vectorization.html.

[129] "Automatic Vectorization, GCC Autovectorization flags" [Online]. Available: https://www.codingame.com/playgrounds/283/sse-avx-vectorization/autovectorization.

[130] "ARM NEON for C++ Developers" [Online]. Available: http://const.me/articles/simd/NEON.pdf.

[131] "OpenMP - The OpenMP API specification for parallel programming" [Online]. Available: https://www.openmp.org/

[132] "IBM POSIX threads explained - A simple and nimble tool for memory sharing," [Online]. Available: http://lsi.vc.ehu.es/pablogn/docencia/ISO/Act7%20Hilos/IBM%20posix%20threads.pdf.

[133] M. Chavarrías, F. Pescador, M. J. Garrido, M. Pelcat and E. Juárez, "Design of multicore HEVC decoders using actor-based dataflow models and OpenMP," 2016 IEEE International Conference on Consumer Electronics (ICCE), 2016, pp. 287-288, doi: 10.1109/ICCE.2016.7430616.

[134] F. Pescador, M. Chavarrías, M. Garrido, J. Malagón and C. Sanz, "Real-time HEVC decoding with OpenHEVC and OpenMP," 2017 IEEE International Conference on Consumer Electronics (ICCE), 2017, pp. 370-371, doi: 10.1109/ICCE.2017.7889358.

[135] M. Chavarrías, F. Pescador, M. J. Garrido, E. Juárez and C. Sanz, "A multicore DSP HEVC decoder using an actorbased dataflow model and OpenMP," in IEEE Transactions on Consumer Electronics, vol. 61, no. 2, pp. 236-244, May 2015, doi: 10.1109/TCE.2015.7150599.