UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

BRUNNO ALVES DE ABREU

Exploring Partial Distortion Elimination Techniques in the Sum of Absolute Differences Architecture for HEVC Integer Motion Estimation

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Microeletronics

Advisor: Prof. Dr. Sergio Bampi
Coadvisor: Prof. Dr. Mateus Grellert da Silva

Porto Alegre
May 2019

CIP — CATALOGING-IN-PUBLICATION

Abreu, Brunno Alves de

Exploring Partial Distortion Elimination Techniques in the Sum of Absolute Differences Architecture for HEVC Integer Motion Estimation / Brunno Alves de Abreu. – Porto Alegre: PGMICRO da UFRGS, 2019.

101 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2019. Advisor: Sergio Bampi; Coadvisor: Mateus Grellert da Silva.

1. High Efficiency Video Coding. 2. Integer Motion Estimation. 3. Sum of Absolute Differences. 4. Partial Distortion Elimination. I. Bampi, Sergio. II. Silva, Mateus Grellert da. III. Título.

AGRADECIMENTOS

First of all, I would like to thank my parents, Fernando César Ávila de Abreu and Jaqueline Alves de Abreu, for fully supporting me in my choices and giving me advices when needed.

To all my professors, who have made their best to teach me all they could and helped me to find outside sources to learn even more.

To all my classmates, from my undergraduate and post-graduate studies, with whom I have shared the difficulties, stressful and happy moments we all have been through during our graduation years.

To my research colleagues Guilherme Paim, Leandro Rocha, Eduardo Costa, Gustavo Santana, Thomas Fontanari, Vitor Lima, who have helped me, directly or indirectly, in my maturation as a researcher, and also in the completion of this work. To the colleagues from my laboratory who, even though worked on other research topics, were great companies during coffee time.

Special thanks to my advisor Sergio Bampi, and my co-advisor Mateus Grellert, who have dedicated countless hours in helping me with doubts I had, to work with me in many other side projects, and to encourage me when I lacked the motivation.

ABSTRACT

Digital videos are among the multimedia applications that have been given the most importance in the recent years, leading to the development of better compression techniques at the cost of higher computing requirements. In the most recent video-coding standard, named High Efficiency Video Coding (HEVC), Integer Motion Estimation (IME) is one of the most computing- and memory-intensive steps. IME exploits temporal redundancy by minimizing the rate-distortion cost estimated from two metrics: Sum of Absolute Differences (SAD) and motion vector cost (MV_{Cost}). Partial Distortion Elimination (PDE) techniques may be used to optimize the calculation of the SAD unit itself, avoiding the computation of candidates that will certainly not be selected in the IME. This work explores the impact of employing PDE on circuit area and on energy consumption. Differently from related solutions found in the literature, the designed architectures include the MV_{Cost} in its decision, which enhances the compression efficiency by up to 1.5%. Different designs were also proposed and implemented for the rate-distortion computation unit, in order to discover the best alternative in terms of energy consumption. The architectures were synthesized for ASIC with a 65 nm standard cells library, considering real-input vectors from video sequences to obtain accurate power results. The use of PDE without the MV_{Cost} achieves an average energy reduction of 16.4% for 1080p and of 11.64% for 2160p sequences when compared to non-PDE implementations. By accumulating the MV_{Cost} before SAD, approaches using a multiplexer and a Carry-Save Adder (CSA) were analyzed. Compared to the case that accumulates MV_{Cost} after SAD, average reductions of 17.51% and 5.05% were obtained when using a multiplexer for the HEVC Model (HM) and x265 implementations respectively. The comparisons also showed that using a CSA is the best solution in terms of total energy, with additional reductions of 1.94% (for the HM implementation) and 2.27% (for the x265) when compared to the multiplexer implementation.

Keywords: High Efficiency Video Coding. Integer Motion Estimation. Sum of Absolute Differences. Partial Distortion Elimination.

Explorando Técnicas de Eliminação Parcial de Distorções em Arquiteturas da Soma das Diferenças Absolutas para Estimação de Movimento Inteira no Padrão HEVC

RESUMO

Vídeos digitais estão entre as aplicações multimídia para as quais têm se dado a maior importância nos últimos anos, levando ao desenvolvimento de melhores técnicas de compressão ao custo de maiores requisitos computacionais. No padrão de codificação de vídeo mais recente, *High Efficiency Video Coding* (HEVC), a Estimação de Movimento Inteira (IME) é uma das etapas que demandam maior esforço computacional e mais acessos à memória. A IME explora a redundância temporal, minimizando o custo *rate-distortion* estimado por duas métricas: a Soma das Diferenças Absolutas (SAD) e o custo de vetor de movimento (MV_{Cost}). Técnicas de Eliminação de Distorção Parcial (PDE) podem ser usadas para otimizar o cálculo da unidade de SAD, evitando a computação de candidatos que certamente não serão selecionados na IME. Esse trabalho explora o impacto da utilização de PDE na área e no consumo energético do circuito. Diferentemente de soluções relacionadas encontradas na literatura, as arquiteturas desenvolvidas incluem o MV_{Cost} na decisão, o que melhora a eficiência de compressão em até 1.5%. Diferentes modelos foram propostos e implementados para a unidade que computa o *rate-distortion*, para determinar a melhor alternativa em termos de consumo energético. As arquiteturas foram sintetizadas para ASIC com uma biblioteca de *standard cells* de 65 nm, considerando vetores de entradas reais de sequências de vídeos para obter resultados de potência precisos. O uso da PDE sem o MV_{Cost} atinge uma redução média de energia de 16.4% para sequências 1080p e de 11.64% para sequências 2160p quando comparado a implementações sem PDE. Ao acumular o MV_{Cost} antes da SAD, propostas utilizando um multiplexador e um *Carry-Save Adder* (CSA) foram analizadas. Comparado ao caso que acumula o MV_{Cost} após a SAD, reduções média de 17.51% e 5.05% foram obtidas ao utilizar um multiplexador, considerando as implementações do *HEVC Model* (HM) e do x265 respectivamente. As comparações também mostraram que utilizar o CSA é a melhor solução em termos de energia total, com reduções adicionais de 1.94% (na implementação do HM) e de 2.27% (no x265) quando comparado com a implementação do multiplexador.

Palavras-chave: *High Efficiency Video Coding*, Estimação de Movimento Inteira, Soma das Diferenças Absolutas, Eliminação Parcial de Distorções.

LIST OF ABBREVIATIONS AND ACRONYMS

ABS	Absolute operation
ADC	Analog-to-Digital Converter
AMP	Asymmetric Motion Partition
AMVP	Advanced Motion Vector Prediction
AV1	Alliance for Open Media Video 1
AOMedia	Alliance for Open Media
ASIC	Application Specific Integrated Circuit
AVC	Advanced Video Coding
BD-BR	Bjøntegaard Delta Bit-rate
BD-PSNR	Bjøntegaard Delta Peak Signal-to-Noise Ratio
BMA	Block-Matching Algorithm
CABAC	Context-adaptive Binary Arithmetic Coding
CAVLC	Context-Adaptive Variable-Length Coding
CBB	Candidate Block Buffer
CMOS	Complementary Metal Oxide Semiconductor
CSA	Carry-Save Adder
CTC	Common Test Conditions
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DPB	Decoded Picture Buffer
FHD	Full High-Definition
FIR	Finite impulse response
FiS	First Search

FME	Fractional Motion Estimation
FPGA	Field-Programmable Gate Array
Fps	Frames per second
FS	Full Search
GPP	General Purpose Processor
HD	High-Definition
HEVC	High Efficiency Video Coding
HM	HEVC Test Model
HS	Hexagon Search
IME	Integer Motion Estimation
IP	Internet Protocol
ITU-R	International Telecommunication Union Radiocommunication Sector
JCT-VC	Joint Collaborative Team on Video Coding
JPEG	Joint Photographic Experts Group
JVET	Joint Video Experts Team
ME	Motion Estimation
MOS	Metal Oxide Semiconductor
MSB	Most Significant Bit
MSE	Mean-Squared Error
MV	Motion Vector
MV_{Cost}	Motion Vector Cost
MVP	Motion Vector Predictor
OBB	Original Block Buffer
PCM	Pulse Code Modulation
PDE	Partial Distortion Elimination
Pixel	Picture element

PLE	Physical Layout Estimation
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
QoS	Quality of Service
QP	Quantization Parameter
RD_{Cost}	Rate-Distortion Cost
RDO	Rate-Distortion Optimization
RGB	Red, Green, Blue
RMD	Rough Mode Decision
RQT	Residual Quadtree
RS	Raster Search
RTL	Register-transfer Level
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Transformed Differences
SDF	Standard Delay Format
SEA	Successive Elimination Algorithm
SMP	Symmetric Motion Partition
SS	Star Search
SSE	Sum of Squared Errors
TCF	Toggle Count Format
TU	Transform Unit
TZS	Test Zone Search
UHD	Ultra High-Definition
UMH	Uneven Multi-Hexagon
UVG	Ultra Video Group
VHDL	VHSIC Hardware Description Language

VHSIC	Very High Speed Integrated Circuit
VTM	Versatile Video Coding Test Model
VVC	Versatile Video Coding
WPP	Wavefront Parallel Processing
YCbCr	Luminance, Chrominance Blue, Chrominance Red
ZB	Zettabyte

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The recent advances in semiconductor technology have allowed for the evolution of several different applications. These improvements were accompanied and enabled an increase of the demand for more sophisticated services. Multimedia applications are among the main categories that have evolved in the latest years and, within that range, digital videos have received remarkable importance, as it can be seen from the increased resolutions that have been hitting the market. In addition to that, real-time video content has become more popularized, such as video broadcasting, like Twitch or Youtube. Such services need to work as smoothly as possible to fulfill the Quality of Service (QoS) demands. These increases are further supported by recent (as of February 2019) Cisco market research results (CISCO, 2019), some of which predict that video traffic will represent 82% of all IP traffic by 2022, a 7% increase of what was observed in 2017 (75%). This is aggravated by the fact that the annual IP traffic will likely increase and reach 4.8 ZB (Zettabytes) per year by 2022 – this value was around 1.5 ZB in 2017. Therefore, there is an urgent need for optimizing video applications, in order to alleviate the effects of their growth.

This issue becomes even more significant when we consider embedded devices, e.g. smartphones, tablets, camcorders. Dealing with video applications in systems that depend on limited battery resources is a challenging task, mainly because of the increased requirements of such services, which implies more energy consumption. Moreover, the same predictions from Cisco (CISCO, 2019) indicate that smartphone traffic will exceed PC traffic by 2022: the former will account for 41% of the total IP traffic, while the latter will represent only 19%. More generally speaking, wireless and mobile devices will account for 71% of the total IP traffic by 2022. Therefore, with ever-increasing resolutions in video applications and the increasing share they will represent in the near future, embedded systems will tend to have a smaller battery life. Thus, taking power dissipation and energy into account when developing video applications is of utmost importance to overcome such issues.

Handling digital videos in their uncompressed form requires a huge amount of resources, whether to store or transmit them, becoming prohibitive when dealing with high resolution videos. The storage and transmission requirements, in $bytes$ and $\frac{bytes}{s}$, respectively, of uncoded videos, are given by equations 1.1 and 1.2, where W and H respectively denote the width and height of the video sequence, N refers to the represen-

tation, in bytes, of each pixel, F refers to the frame-rate – in frames per second (fps) –, and t denotes the time duration of the sequence, in seconds.

$$Size = W \cdot H \cdot N \cdot F \cdot t \qquad [bytes] \quad (1.1)$$

$$BitRate = \frac{Size}{t} \qquad [\tfrac{bytes}{s}] \quad (1.2)$$

As a concrete example, we can consider the Ultra High-Definition 4K (UHD 4K) (3840×2160 pixels) video sequence – which is a resolution that is becoming increasingly popular –, recorded at a frame-rate of 30 frames per second (fps), with each of its pixels being represented by 3 bytes (to form the three color channels). A 10-minute video with these specifications would require more than 410 GB to be stored. In order to transmit this video for real-time streaming or broadcasting applications, we would require a bit-rate of more than 710 MB/s. These values become even higher when considering that the International Telecommunication Union Radiocommunication Sector (ITU-R) recommendation for UHD Television (UHDTV) states that resolutions should be increased in both spatial and temporal axes (ITU-R, 2015). Hence, higher frame rates, such as 120 fps, which would quadruplicate the results of the previous example, have to be supported. Thus, due to the prohibitive values of dealing with uncompressed videos, there is an evident need for video compression, in order to alleviate these huge requirements.

Video compression is based on finding redundant information, through several methods, in video frames, and then suppressing most of these redundancies to minimize the number of bits required to represent a video sequence. The main goal of video compression is to make the required storage size and transmission rates more feasible.

However, finding redundant information and taking significant advantage of them is a costly task that demands a significant amount of computational resources. This occurs because modern video encoders perform numerous time-consuming and computing-demanding operations to efficiently compress data, which increases the time and energy required. Aggravated by the increasing demands for higher resolutions and frame-rates, video compression has been receiving major attention in academic and industrial studies.

High Efficiency Video Coding (HEVC) (ITU-T, 2013) is a video coding standard developed by the Joint Collaborative Team on Video Coding (JCT-VC), as a solution for the increasing demands for higher resolution videos. HEVC is the sucessor of the H.264/AVC (Advanced Video Coding) (ITU-T, 2003), and its goal is to double the

compression efficiency, for the same video quality, when compared to its predecessor (SULLIVAN et al., 2012). In practice, however, HEVC manages to achieve an average compression of 39.3% when compared to H.264/AVC (GROIS et al., 2013). These advances became possible due to more well-structured and flexible block partitioning (KIM et al., 2012), complex algorithms, advanced motion vector predictions, and the support for larger block sizes.

The improvements achieved by these new proposals also led to an increase in the computational effort of HEVC: encoders compliant with this standard are 1.2 - 3.2$\times$ more complex when compared to H.264/AVC-compliant encoders (GRELLERT; BAMPI; ZATT, 2016). Thus, although it compresses videos better than H.264/AVC, HEVC has brought an additional issue, especially for embedded devices, due to the higher energy requirements to encode videos in this standard.

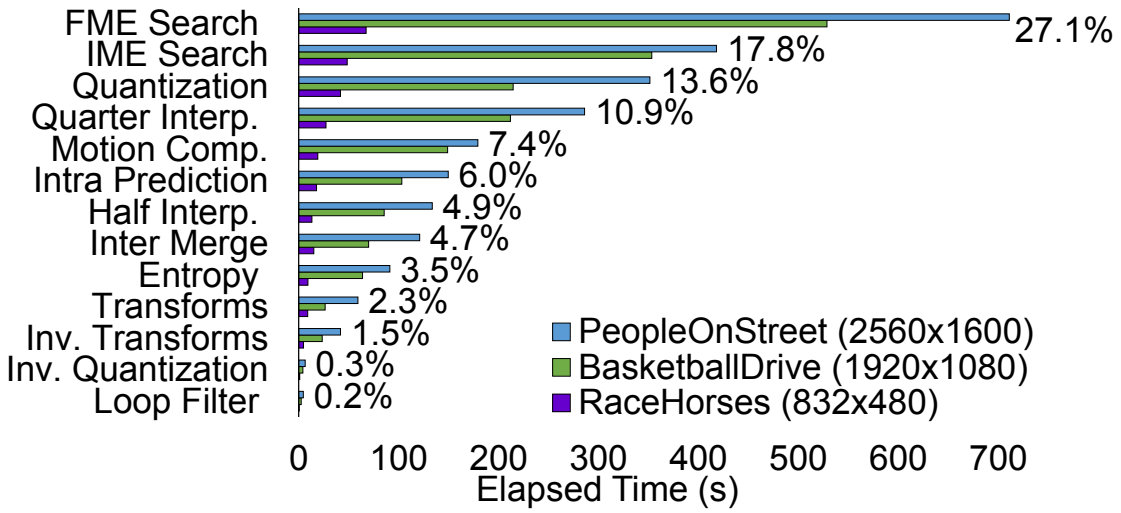
The increased effort required to compress videos represents an additional matter to be considered when dealing with the embedded devices, such as the ones previously cited. The limited battery life demands the encoding task to be executed in the most efficient manner. Processor manufacturers, such as Qualcomm, have recently realized that using simple general purpose processors (GPPs) for video applications would not be efficient to tackle these processing requirements (QUALCOMM, 2014). Therefore, there has been a growing tendency for using dedicated hardware architectures for video compression.

Along with other techniques, such as clock and data gating, designing dedicated architectures is one of the main strategies to mitigate power, energy and timing issues, as they are optimally designed solely for specific applications. Application Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) are two platforms for implementing dedicated architectures. ASICs are designed for specific domains, and represent the class of circuits that contrast the most from software solutions running on GPPs. The use of FPGAs may represent a good balance standing between the spectre of GPPs and ASICs. However, when energy-efficiency is the main design restriction to take into account, designing architectures in ASICs for dedicated execution of certain key compute-intensive algorithms is still the best approach (SCHMITZ; AL-HASHIMI; ELES, 2004).

Motion Estimation (ME) is the most costly step of the HEVC standard, in terms of time. This step is responsible for finding temporal redundancies in video sequences, in order to decrease the bitstream size at the output of the encoder. To take advantage of

most of these redundancies, ME has to be executed several times for each frame in the video sequence. ME is split into two different stages: Integer ME (IME) and Fractional ME (FME). Figure 1.1 shows an analysis for different video sequences performed in (GRELLERT; BAMPI; ZATT, 2016), using the reference software for HEVC – HEVC Test Model (HM) (MCCANN et al., 2013) –, showing that IME and FME are responsible for a large portion of the execution time in the encoding process for the HEVC standard. Considering the ME includes the process of IME, FME, Half and Quarter Interpolations, this analysis shows that it represents about 60.1% of the total encoding process. This value is further validated by (BOSSEN et al., 2012), which also presents a value close to 60% for the ME process.

Figure 1.1: Time percentage of each stage in the video encoding process.



Source: (GRELLERT; BAMPI; ZATT, 2016)

The ME stage also represents a bottleneck in terms of power. According to (BAHARI; ARSLAN; ERDOGAN, 2009), about 77% of the total power was consumed by the ME module of the video encoder analyzed in their study.

In the IME stage, the first step of ME, the encoder attempts to find temporal redundancies by measuring similarity between the block being encoded (original) and blocks (candidates) from previously encoded frames. This similarity is usually measured by using the Sum of Absolute Differences (SAD) metric, along with the Motion Vector Cost (MV_{Cost}) – the cost, in bits, of the current motion vector pointing to the candidate block.

Some techniques are used to decrease the number of cycles required in the SAD calculation, such as the Partial Distortion Elimination (PDE) (CHOI; JEONG, 2009; SEIDEL; BRäSCHER; GüNTZEL, 2015). This technique consists in an early-termination of the SAD calculation, which eliminates cycles of useless candidates in the IME step. How-

ever, related works found in the literature mostly consider algorithms that are unfeasible to work in real encoders. Also, works regarding this technique do not take the MV_{Cost} into account, which leads to compression efficiency degradation and reduced optimization results with respect to the reference HEVC implementation.

Considering the aforementioned issues and the additional ones brought by HEVC, the proposal of this M.Sc. Dissertation is to make a deep analysis in the possible ways to employ PDE techniques along with the MV_{Cost}, mainly by initializing the SAD value with the MV_{Cost} to decrease the cycles requirements even more than conventional PDE usage. Additionally, by using MV_{Cost}, improved compression efficiency results were obtained, which are also presented in this Dissertation.

The main contributions of this work are the following:

- Comparison, in terms of SAD calls and BD-BR, between two important Block Matching Algorithms (BMA) from HEVC-compliant encoders: Test Zone Search (TZS) and Hexagon Search (HS);
- Use of the standard PDE technique in the state-of-the-art HEVC standard and analysis of its impact in 1080p and 2160p video sequences;
- Analysis of the use of PDE along with the MV_{Cost} leading to more precise results;
- Compression efficiency results regarding the use of the MV_{Cost} in the total cost of a block when compared to the cost using only SAD, for HEVC encoders;
- New technique that improves the PDE by including the MV_{Cost}, considering IME algorithms in HEVC standard;
- Two different ways of including the MV_{Cost} in the SAD architecture – along with an adder tree using a Carry-Save Adder (CSA), and independently, with a multiplexer.

This document is structured as follows: Chapter 2 presents a detailed background on the main concepts regarding video coding, quality metrics, power dissipation and more; Chapter 3 details some related works found in the literature that address similar topics regarding SAD, ME and PDE implementations; Chapter 4 presents the methodology used, describing the tools and methods employed for obtaining the reported results; Chapter 5 presents all the PDE analyses proposed in this work, a candidate reordering and the memory design; Chapter 6 presents the cycles, power and energy results, describing and justifying them; and Chapter 7 concludes the report, by summarizing the contributions of the work, and highlighting possible future paths for further research on this topic.

2 BACKGROUND

In order to understand the technical details of the remaining chapters, the introduction of some of the basic related concepts is essential. This chapter details the topics related to the scope of this work, including terms related to video coding, motion estimation, sum of absolute differences, power dissipation and more.

2.1 Fundamentals of Video Coding

The entire digital video process begins when a real-life scene is obtained by a capturing device, e.g. digital video camera, camcorder, etc, through the use of image sensors. These sensors collect light photons, producing an electrical charge – proportional to the number of photons collected – that will further be converted to digital signals by using an analog-to-digital converter (ADC) and produce images (RICHARDSON, 2003). Digital videos are a set of digital images captured sequentially with a high degree of temporal proximity, taking advantage of the limitations of the human vision, which will interpret sequential images as real moving scenes. Each image that composes a digital video is denoted as a frame.

More specifically, a digital image is represented by a rectangular matrix of color elements. This matrix is mapped to an exhibition device, in which each of its elements represents a brightness or color information of a small region of the exhibition device. Each of these small regions is referred to as a picture element (pixel).

Usually, digital videos need to be encoded in a minimum frame-rate to optimize user experience, high enough for the human brain not to perceive the video as individual images. Frame-rate is usually represented in frames per second (fps). Typical values are 30, 60 and 120 fps, the last of which is usually employed in 4K video sequences (ITU-R, 2015).

Several different display resolutions are considered in this work. These resolutions define the number of pixels in each dimension, and they are defined in a width$\times$height form. Table 2.1 presents a set of possible display resolutions and their respective notations employed in this work, for clarification purposes.

A digital video is in its raw form when it is first produced. Due to the fact that working with raw videos is an unfeasible task, as already mentioned, video coding techniques must be applied, which entails the operation of a video encoder and a decoder. The

Table 2.1: Popular display resolutions.

Width × Height	Notation
416×240	240p
832×480	480p
1280×720	720p
1920×1080	1080p
3840×2160	2160p

Source: The Author.

video encoder is responsible for applying compression techniques and transforming a raw video in a sequence of bits – referred to as a video bitstream – according to a specific video standard. Then, the video is combined with other syntax elements at the network layers, to be further sent by a transmitter. That way, generally speaking, a station can receive the data, extract the encoded video bitstream from the network layers and process this bitstream with a decoder that must support the same standard for which the bitstream was generated. This process regenerates the video, which can be displayed or stored for future uses. This process is illustrated in Figure 2.1.

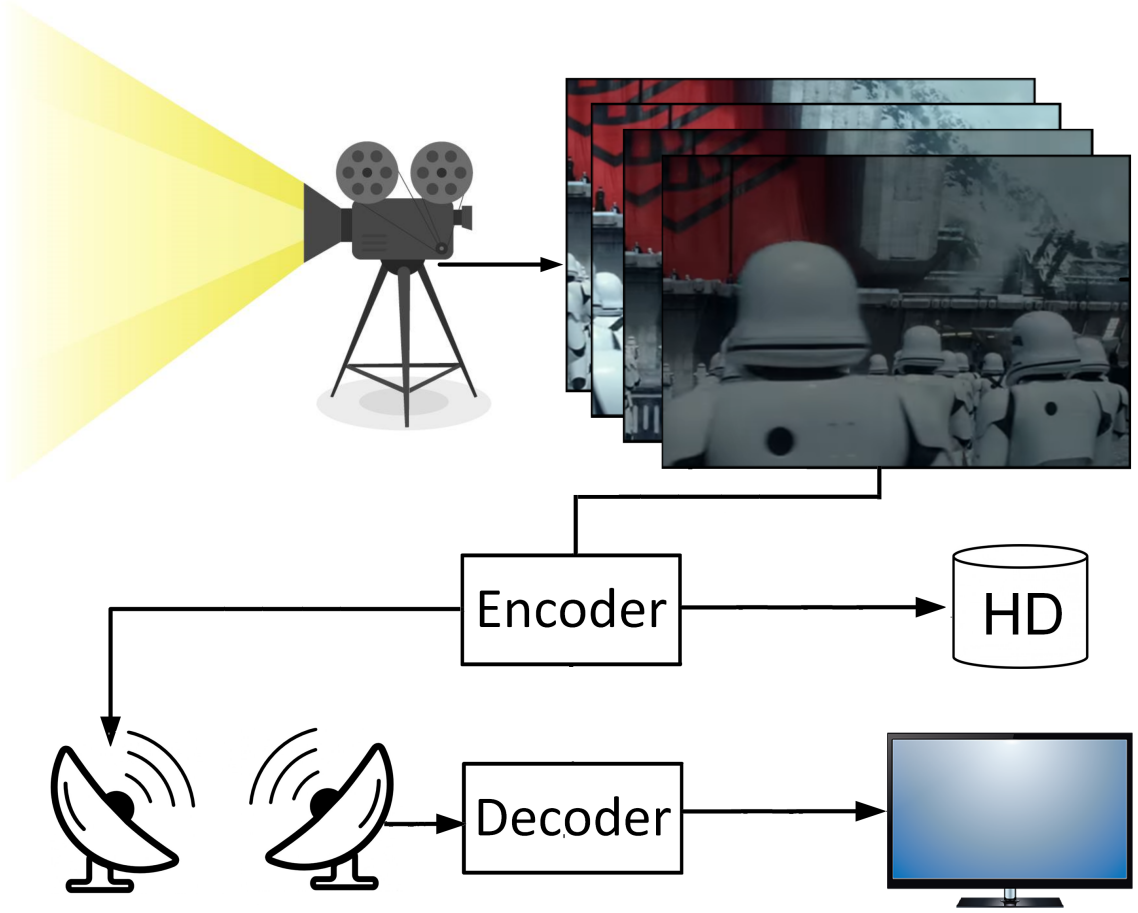
The video coding standard defines only the specifications for which the decoder has to comply. Hence, the encoder can be freely implemented in different ways, by applying different algorithms and in varied hardware platforms, as long as the output bitstream generated by the encoder complies to the standard. In other words, the encoded video bitstream must be able to be processed by any standard-compliant decoder.

2.2 Color Space and Sub-Sampling

Every pixel on a digital video frame can be represented by three color components. A common color space to represent digital images is RGB (Red, Green, Blue). RGB employs three different matrices to represent the three colors, whose choice is based on the three primary colors detected by the human visual system.

A common color space used in video coding, however, is the YCbCr. This color space is split into Y, which is the luminance (luma) component, and two chrominance (chroma) components Cb and Cr, which denote, respectively, the blue component relative to the green component, and the red component relative to the green component. YCbCr is more suitable for video compression due to the fact that the chroma information is totally separated from the luma component. This allows for distinct compression techniques to

Figure 2.1: Simplified scheme of encoding and decoding in video transmission.



Source: The Author

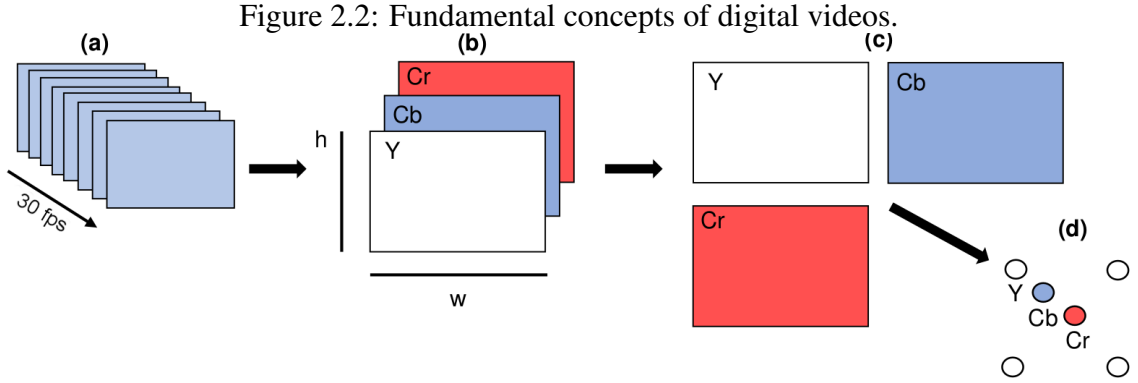
be applied for each of them separately (DINIZ, 2009).

This distinction in compression techniques is important given that the human eye is more sensitive to the luma component than to chroma. Therefore, slightly decreasing the representation of the chroma matrix, for example, would decrease the total size of a video without majorly impacting the image visually. This is denoted as color sub-sampling. Among the several color sub-sampling possibilities, a few of them stand out. 4:2:0 is the most employed one for video compression using the HEVC standard, which consists of using one Cb and one Cr sample for each set of four Luma samples, which, in other words, means that there is an horizontal and vertical sub-sampling in both the Cb and Cr components. 4:2:2 is another possibility to be used, which refers to a sub-sampling in the vertical axis only. Lastly, 4:4:4 refers to no sub-sampling being employed (RICHARDSON, 2003).

Some of the concepts presented so far are shown in Figure 2.2, in which the concepts of temporal resolution (for a 30 fps video sequence) (a), spatial resolution (b), color space (c) – considering the aforementioned YCbCr color space – and color sub-sampling

(d) are illustrated.

Figure 2.2: Fundamental concepts of digital videos.



Source: Adapted from (BUBOLZ, 2018)

2.3 Quality Metrics

Defining the quality of a video is a rather complex task. Many existing metrics attempt to objectively measure video quality, given that most subjective criteria are hard to measure. The most straightforward objective way is by comparing the pixels of the original frame with the generated pixels after the decoding process, which is mostly useful for determining the fidelity of the reconstruction of an image. The most accepted metric, however, is the Peak Signal-to-Noise Ratio (PSNR) (GHANBARI, 2003). The equation for calculating PSNR is shown in 2.1.

$$PSNR_{dB} = 20 \cdot \log_{10} \left(\frac{MAX}{\sqrt{MSE}} \right) \tag{2.1}$$

In Equation 2.1, MAX denotes the maximum representation value of a sample, which equals $2^N - 1$, where N is the number of bits required to represent one sample. MSE refers to the Mean-Squared Error, defined in Equation 2.2, where m and n are the number of pixels in the vertical and horizontal directions of the frame, respectively, and $O_{i,j}$ and $R_{i,j}$ denote the positions within the original and reconstructed frames.

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (O_{i,j} - R_{i,j})^2 \tag{2.2}$$

Based on the PSNR and on the bit-rate value of an encoded video, Gisle Bjøntegaard (BJONTEGAARD, 2001) proposed a model to measure coding efficiency between two different video compressors (encoders). This proposal was based on approximating a rate-distortion (R-D) curve given by the set of bit-rate and PSNR values, from which

a third order logarithmic polynomial fitting has been proposed. This principle generated two metrics: Bjøntegaard delta bit-rate (BD-BR) and Bjøntegaard delta PSNR (BD-PSNR) – both of which are widely used for comparing two models. BD-BR represents an average bit rate difference, measured in %, over the range of four different PSNR values, in a BR vs. PSNR curve. BD-PSNR is the reverse, and denotes the average PSNR difference, measured in %, over the range of four different bit-rate values, in a PSNR vs. BR curve.

2.4 Data Redundancies

Video coding is mainly based on reducing the amount of redundant data in video sequences. In other words, data that are not relevant for the video representation are discarded. Data compression algorithms focus on finding these redundancies, which appear from correlations or repetitions in the video sequence, and exploit them. The four redundancies explored by video codecs are presented below:

- Spatial Redundancy: this mainly refers to correlations and similarities between neighboring or spatially distributed pixels in the same frame. This redundancy occurs due to the fact that spatially close pixels have a tendency of presenting similar values. This redundancy is mainly tackled in the intra-frame prediction stage of the video encoding process;
- Temporal Redundancy: this is the kind of redundancy that costs the most to be explored in modern encoders. It is based on the fact that objects tend to slightly displace from where they originally were in previous frames of a video sequence. Therefore, it would not be necessary to purely encode an entire block of pixels if there is a block very similar to it in a previously encoded frame. This is targeted by the inter-frame prediction, which is one of the most time-consuming tasks in the encoding process (GONZALEZ; WOODS, 2003; AGOSTINI, 2007; GRELLERT; BAMPI; ZATT, 2016);
- Entropic Redundancy: this redundancy is related to the occurrence probability of the coded symbols. It does not have a direct relation to the video content itself, but with the way data is represented by the standard. This is mainly based on representing the most frequent information of a video sequence by using symbols, generating a smaller number of bits in its respective representation;

- Psychovisual Redundancy: it is responsible for exploiting the human visual system limitations – as stated in Section 2.2 regarding color sub-sampling –, by decreasing the least relevant information of an image (MONTEIRO; SANTOS, 2013).

2.5 High Efficiency Video Coding

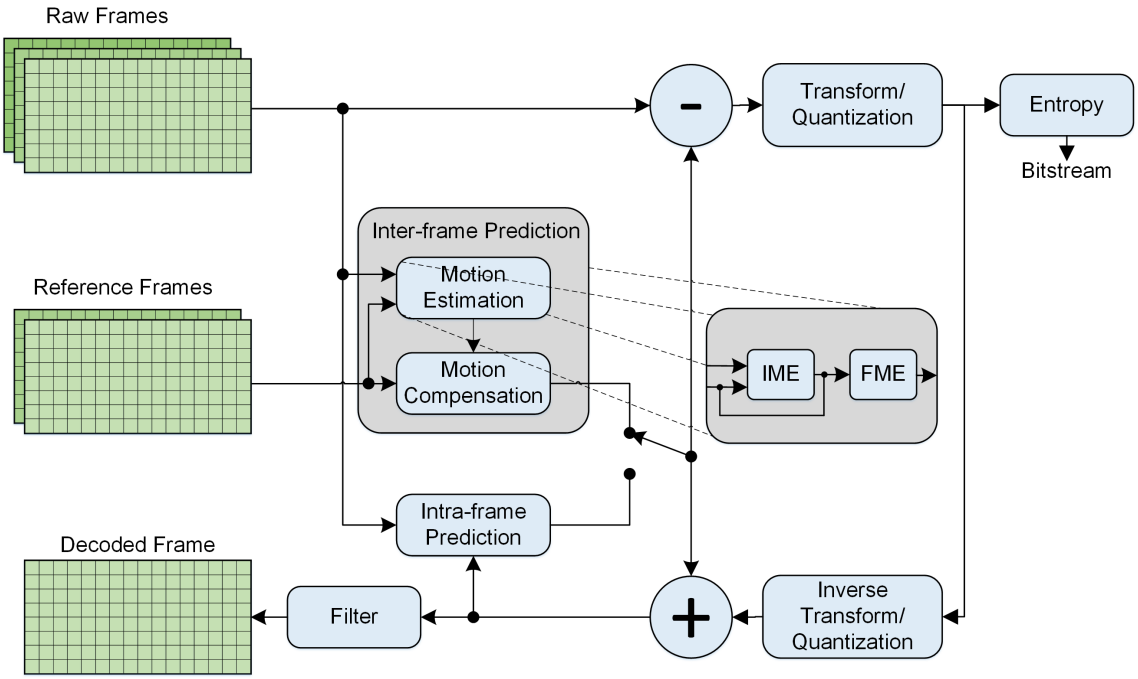
High Efficiency Video Coding (HEVC) (ITU-T, 2013) is a popular video coding standard, focused on targeting higher resolution videos than its predecessor H.264/AVC (Advanced Video Coding) (ITU-T, 2003). HEVC obtains an average compression efficiency of 39.3% (GROIS et al., 2013), for the same video quality, when compared to H.264/AVC, which became possible due to more complex structures and block partitioning (KIM et al., 2012). As an example of its higher complexity, H.264/AVC works with the concept of macroblocks, whose maximum supported size is 16×16, whereas HEVC supports blocks of up to 64×64. Hence, the compression improvements of HEVC led to an increase of 1.2 - 3.2 × in the computational effort to encode videos, when compared to H.264/AVC (GRELLERT; BAMPI; ZATT, 2016). This complexity growth tends to lead to higher time to encode videos and, therefore, higher energy consumption.

Even though several innovations have been introduced by HEVC, this standard still employs a well-established hybrid structure (HABIBI, 1974; FORCHHEIMER, 1981) in its encoder, which has been used since H.261 (ITU-T, 1993). The hybrid term mainly indicates that the encoder uses several different techniques, such as predictions, transforms, quantization, entropy coding, to encode videos. The encoder has an implementation of the decoder within its own functioning, in order to use the locally decoded frames as reference frames – in the same manner that any other separate and dedicated decoder will eventually proceed. This structure is presented in Figure 2.3.

The process starts by splitting each frame of the video sequence to be encoded in blocks called Coding Tree Units (CTUs). The CTU size is fixed for the whole encoding process, and HEVC-compliant encoders usually define this size as 64×64. Each CTU of the frame to be encoded is applied in the stages of the presented hybrid diagram from Figure 2.3.

The blocks are used as input for the inter and intra-frame prediction stages, in order to exploit temporal and spatial redundancies. Based on decisions from the encoder – by choosing between intra or inter-prediction – the encoder generates a predicted block, which is subtracted from the CTU, producing a residual block. The residue serves as an

Figure 2.3: Hybrid structure used in HEVC.



Source: The Author

input for the transform and quantization stages, which discards the most irrelevant information for our visual system. The quantization stage is usually the step that introduces losses to the encoder. The quantized output is sent to the entropy stage, which applies statistical algorithms to generate the output bitstream, to either store or transmit it. The encoded block also needs to be decoded in the encoding scheme – hence the "hybrid" terminology – by applying inverse quantization and inverse transform functions in the encoder, so that the encoded frame can be recovered and stored to be used as reference frame for encoding other frames. These inverse operations are needed because some predictions use information of previously coded frames to find the redundancies in the current frames, so the previously encoded frames need to be ready to be analyzed.

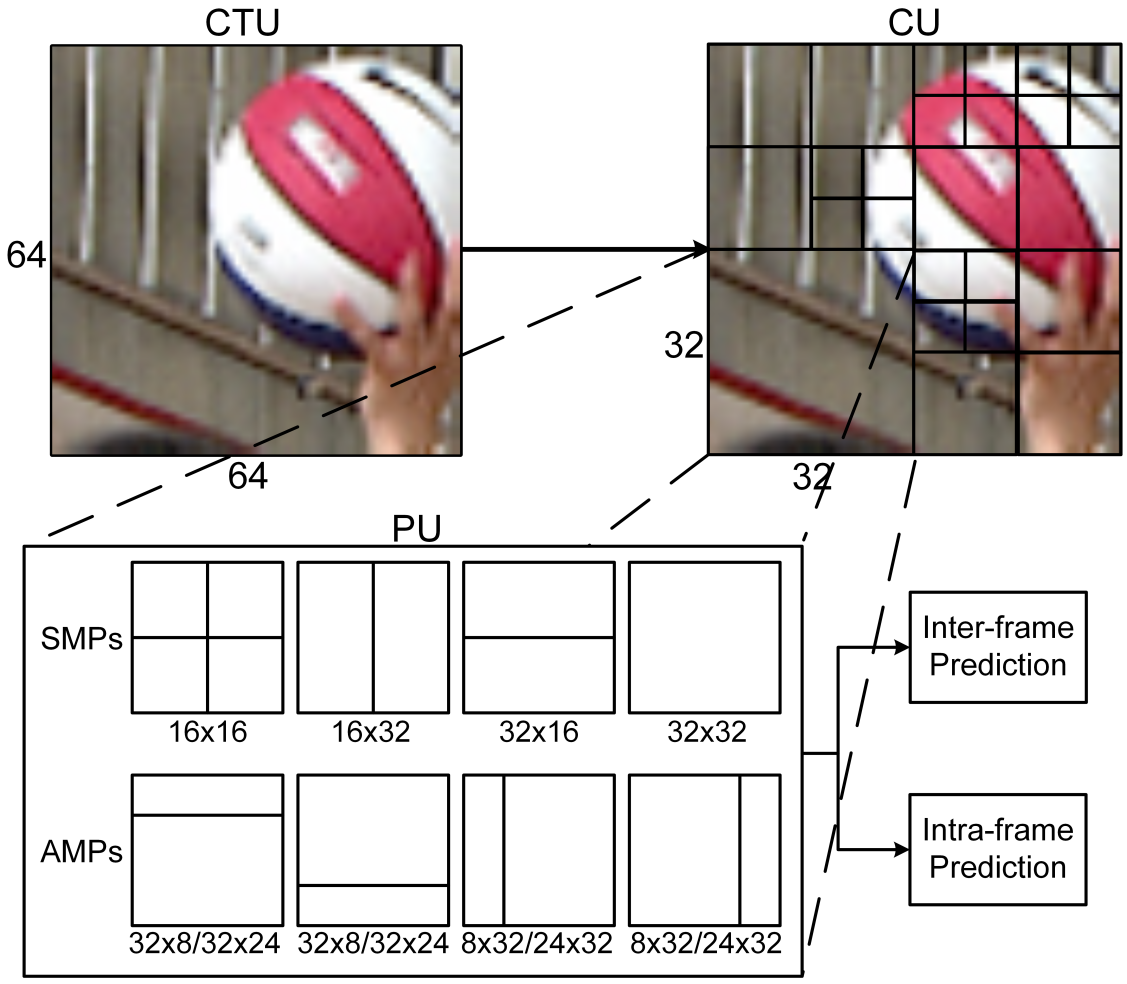
The HEVC standard supports complex data structures apart from CTUs. Each CTU can be encoded in several distinct ways by splitting it into smaller squared blocks, denoted as Coding Units (CUs) in a quad-tree partitioning scheme, and their final partitioning is decided by the use of Rate-Distortion Optimization (RDO), which defines a metric, namely Rate-Distortion Cost (RD_{Cost}), given by Equation 2.3. In this formula, $Distortion$ refers to a similarity metric (which will be explained later in Section 2.7.3; λ_{mode} and R_{mode} refer to the Lagrange multiplier and the cost in bits of the specific mode being used, respectively (GRELLERT, 2018). It should be noted that, in order to determine the number of bits and obtain R_{mode}, it is necessary to go through the stages of

transform, quantization and entropy (GRELLERT, 2018).

$$RD_{Cost} = Distortion + \lambda_{mode} \cdot R_{mode} \tag{2.3}$$

HEVC supports CU sizes of 8×8, 16×16, 32×32 and 64×64. When applying the prediction modes, CUs are further split into Prediction Units (PUs). This gives more flexibility to an encoder, given that objects in a CU may be having different behaviors close to each other, so applying different partitioning schemes and different prediction choices may lead to better results overall. The PUs can be categorized into Symmetric Motion Partitions (SMPs) and Asymmetric Motion Partitions (AMPs), each of which has four possible partitioning schemes. Figure 2.4 shows an example of a CTU being split to the PU level, highlighting the SMP and AMP PU types for a 32×32 CU size.

Figure 2.4: Example of the partitioning scheme in HEVC.



Source: The Author

The number of PU partitions available for a CU depends on its size and the prediction mode. Table 2.2 shows the partitions for each inter and intra-prediction stages of

HEVC, referring to D when the partition is disabled and E when it is enabled.

Table 2.2: Partition sizes employed in each prediction stage, for each CU size.

Mode	CU	2N×2N	2N×N	N×2N	N×N	2N×nU	2N×nD	nL×2N	nR×2N
Intra	64×64	E	D	D	D*	D	D	D	D
	32×32	E	D	D	D*	D	D	D	D
	16×16	E	D	D	D*	D	D	D	D
	8×8	E	D	D	E	D	D	D	D
Inter	64×64	E	E	E	D*	E	E	E	E
	32×32	E	E	E	D*	E	E	E	E
	16×16	E	E	E	D*	E	E	E	E
	8×8	E	E	E	D	D	D	D	D

* Enabled if minimum CU size is greater than 8×8

Source: The Author, adapted from (SILVA, 2014)

In Table 2.2, 2N refers to the CU dimension – hence, N equals half of it. For example, the N×2N PU of a 16×16 CU refers to the 8×16 PU. It should also be noted that N×N is disabled for 8×8 CUs. The reason to that decision, according to (MCCANN et al., 2013), is to reduce bandwidth requirements.

Despite splitting CUs into PUs, the encoder also divides them into Transform Units (TUs), for transform and quantization steps to be applied to the residual block. The different partitions are evaluated so that the encoder can choose which partitioning models achieve a final smaller video bitstream with the smallest quality degradation possible. Ideally, encoders would need to apply intra and inter-frame partitioning algorithms for every possible PU partition in every CU, and test all possible TUs for each of these combinations. Nevertheless, the intense memory accesses and the demanding required computation – which increase the encoding time and energy spent per frame to be encoded – usually lead to encoders employing heuristics for some of the partitions not to be evaluated in some iterations, while obtaining acceptable compression results. These heuristic decisions are left to the encoder developer, to determine whether the PU will be using an intra or an inter-frame partition, and of which type and size of CTU, CUs, PUs and TUs partitions will be employed.

The following subsections briefly describe the intra-prediction, transform, quantization and entropy stages of the HEVC encoding process. The inter-prediction process is described in a subsection of its own, given that this is the main focus of this work.

2.5.1 Intra-frame Prediction

The intra-frame prediction stage of HEVC is responsible for finding spatial redundancies in the encoding process. The sizes supported for this stage were presented in Table 2.2.

Given that it focuses on redundancies occurring spatially, i.e., in neighboring patterns of the same frame, the intra-prediction uses a rationale similar to image compressing algorithms, such as the ones used by the Joint Photographic Experts Group (JPEG) (PENNEBAKER; MITCHELL, 1992).

In HEVC, there are 35 different intra-prediction methods, 33 of which are denoted as directional modes, being mostly useful for repeating patterns on frames, like straight or diagonal lines. Another mode available is the DC, which is based on repeating the average of the samples in the whole block. The remaining one is the planar mode, which is based on repeating information from more than one border.

The HEVC standard defines that not every PU size should evaluate every possible intra-prediction mode, due to the increased amount of computations that would be required. Therefore, the standard defines that only a subset of modes will be evaluated for each PU size, accordingly. This is performed by a heuristical algorithm denoted as Rough Mode Decision (RMD) (SILVA, 2014).

2.5.2 Transform and Quantization

Transform and quantization stages are the two processes that manipulate the residual block produced by the prediction modes.

The transform stage is responsible for translating the blocks to the frequency domain, so that more efficient quantization can be performed. Encoders mostly employ the Discrete Cosine Transform (DCT) on this step. Additionally, transforms can be applied to a variety of block sizes, which are equivalent to the TUs previously mentioned. A Residual Quadtree (RQT) partitioning scheme is employed for each CU in the CTU quadtree, generating TUs which can range from 4×4 to 32×32.

The quantization process is applied right after the transform stage, and introduces losses to the encoder by discarding frequencies not so relevant to the human vision. The frequency range to be discarded is proportional to a parameter called Quantization Parameter (QP). A higher QP implies that more frequencies will be discarded, hence quality

losses and compression gains will occur. Analyses for HEVC standard are usually run with QPs 22, 27, 32 and 37, as stated by the Common Test Conditions (CTC) (SHARMAN; SUEHRING, 2018). The runs with these 4 QPs generate the four points required by the BD-BR and BD-PSNR metrics.

In addition to the transform and quantization stages which will generate blocks for the entropy stage, the encoder also performs inverse quantization and inverse transform stages, to retrieve the original encoded block. These stages are simply inverted versions of the transform and quantization stages previously described.

2.5.3 Entropy Coding

This stage is responsible for properly generating the compressed bitstream to the output of the encoder. This process takes advantage of statistical redundancies – the probability of occurrence of symbols –, applying data compression techniques.

HEVC applies a compression technique denoted as Context-Adaptive Binary Arithmetic Coding (CABAC). This algorithm was also used in H.264/AVC, but not every profile of the previous standard supported it, given that it required more processing. In simpler profiles of H.264/AVC, Context-Adaptive Variable-Length Coding (CAVLC) was employed instead.

2.5.4 Encoder implementations

This section briefly describes some encoder implementations of the HEVC standard, mainly focusing on HEVC Test Model (HM) (MCCANN et al., 2013) and x265 (MULTICOREWARE, 2019a). There are other alternative implementations available on the Internet, some of which are briefly described in this section; however, the literature mostly utilizes these two for analyses and comparisons. Some emerging encoders apart from HEVC are described in Section 2.6.

2.5.4.1 HEVC Test Model Reference Software

The Joint Collaborative Team on Video Coding (JCT-VC) maintains a reference HEVC software – HEVC Test Model (HM) (MCCANN et al., 2013) – to be used for research analysis, which contains the main stages presented in the hybrid structure from

Figure 2.3. The software is written in C++ programming language.

The main goal of HM is to provide a basis upon which experiments can be conducted, by making it easier to verify the coding performance of algorithms and proposals. HM is not meant to be a fast or efficient implementation, but it is the most recommended environment for experiments to be performed.

HM has an extensive documentation on every supported parameter, each of which can be manipulated for analysis. The implementation includes the encoder and the decoder, both compliant to the HEVC standard. As of the time of writing this work, HM is in version 16.20.

2.5.4.2 Efficient implementations of the software encoder

Other implementations of HEVC-compliant encoders can be found, apart from HM. Most of these implementations have faster execution in their default presets when compared to HM.

x265 is an open source project for a fast HEVC encoder, led by MulticoreWare, a provider of video software libraries (MULTICOREWARE, 2019b). The implementation is mainly written in C++ programming language, just as in HM. This is a faster version which uses less partitioning structures and less complex algorithms, in its default preset. Even though the default version is faster than the one in HM, x265 supports the following presets for execution:

- Ultrafast;
- Superfast;
- Veryfast;
- Faster;
- Fast;
- Medium (default);
- Slow;
- Slower;
- Veryslow;
- Placebo.

These high-level presets determine the values of encoding parameters in the x265 software. These encoding parameters include the CTU size, the minimum CU size, faster

modes for inter and intra-prediction, limitations on the number of reference frames to be used, and so on.

x265 supports Wavefront Parallel Processing (WPP) (CHI et al., 2012), which allows for parallel execution of rows of CTUs, without major impacts on the overall quality. Moreover, some parts of its code are written with Assembly instructions, making it even faster and, therefore, convenient, for testing and analyzing. This encoder is considered in some works in the literature, such as (HU et al., 2014; YIN; ZHANG; GAO, 2015; HUANG et al., 2018; LIU; WANG; LI, 2018), due to it being closer to real-time implementations when compared to HM.

HEVC Open Mpeg Encoder (HOMER) (CASAL, 2019) is another alternative for efficient implementations. It is an open-source, real-time and multiplatform video encoder, developed by Juan Casal. The encoder includes every prediction and transform sizes, supports parallelism using WPP and applies every possible intra-frame prediction mode. The encoder does not apply inter-frame prediction for SMPs or AMPs, which reduces encoding complexity. There were no papers found in the literature that present HEVC results using the HOMER encoder.

Kvazaar (VIITANEN et al., 2016a) is an open-source HEVC encoder, developed in C programming language. It is developed by the Ultra Video Group (UVG) (GROUP, 2018). Its main goal is to achieve real-time coding with an efficiency close to HM. Kvazaar is employed in some papers in the literature, such as (VIITANEN et al., 2015; LEMMETTI et al., 2016; VIITANEN et al., 2016b), but none of them focus on PDE solutions, which is the scope of this work.

2.6 Emerging Video Codecs

Other standards apart from HEVC can be found. Among the most popular ones is an encoder software from the group Alliance for Open Media (AOMedia), called AOMedia Video 1 (AV1) (RIVAZ; HAUGHTON, 2019), which is an open and royalty-free video coding standard (MEDIA, 2019). Governing members of the AOMedia include Amazon, Apple, ARM, Google, Facebook, and others. However, only HEVC encoders will be employed in this work, given that AV1 is not considered in the related works that have been found. Moreover, primary analyses performed with the encoder indicated that its default implementation was much slower, given its higher focus on achieving an optimal bit-rate, which would make the analysis much more time-consuming.

Lastly, there is another video standard, which is shaping up to be an evolution of HEVC. Versatile Video Coding (VVC) (BROSS; CHEN; LIU, 2018) is a video coding standard recently developed by the Joint Video Experts Team (JVET) in 2018. VVC aims to achieve an average of 50% in the compression efficiency when compared to HEVC, and its innovations tend to bring even more computational effort for VVC-compliant encoders. Even though this standard is still in development, primary implementations – VVC Test Model (VTM) (BROSS; CHEN; LIU, 2018) – have already been released and are constantly being updated. However, this standard was not considered in this work given that the standard and its reference software are still in process of being stabilized, differently from HEVC and HM, which are already well-developed.

2.7 Inter-frame prediction

Inter-frame prediction is the stage responsible for exploiting temporal redundancies, as mentioned in Section 2.4. The lack of movement and the high frame-rate values usually favor the inter-frame prediction in achieving good compression results. In HEVC, the inter-prediction determines four modes for a PU to be encoded: inter, SKIP, Merge and Pulse Code Modulation (PCM).

When a PU is set to SKIP mode, the encoder has defined that the best choice is to not send residual information to the next stages of the encoding process. This occurs mainly when no movement has been detected in the block being analyzed. The chosen PUs in this mode are always squared blocks. Given that no residual is sent, this mode results in considerable compression gains (SILVA, 2014).

In the Merge mode, the motion parameters of the current PU being encoded are generated based on the information from neighbors, both spatial and temporal. Therefore, the decoder only needs a flag indicating that the mode is Merge and the PU index, so that its neighbors can be obtained and the block can be generated.

The PCM mode consists in sending the raw pixels of the block – instead of a residual one – to the bitstream, which harms the overall compression of the video. However, even though this is defined in the HEVC standard, this mode is not used in the default preset of both HM and x265.

The inter mode is the main one in the inter-prediction stage. It defines that a search should be performed in possibly more than one reference frame (previously encoded frame) to find a block similar to the one being encoded. The main part of this stage

is the Motion Estimation (ME). In the encoder implementation, the inter-frame prediction performs a loop through the PU partitions and through several reference frames – specified by the encoder implementation – in which the predictions will be performed. The inter-frame prediction performs two different prediction types, namely uni- and bi-prediction, to both of which the Motion Estimation (ME) is applied.

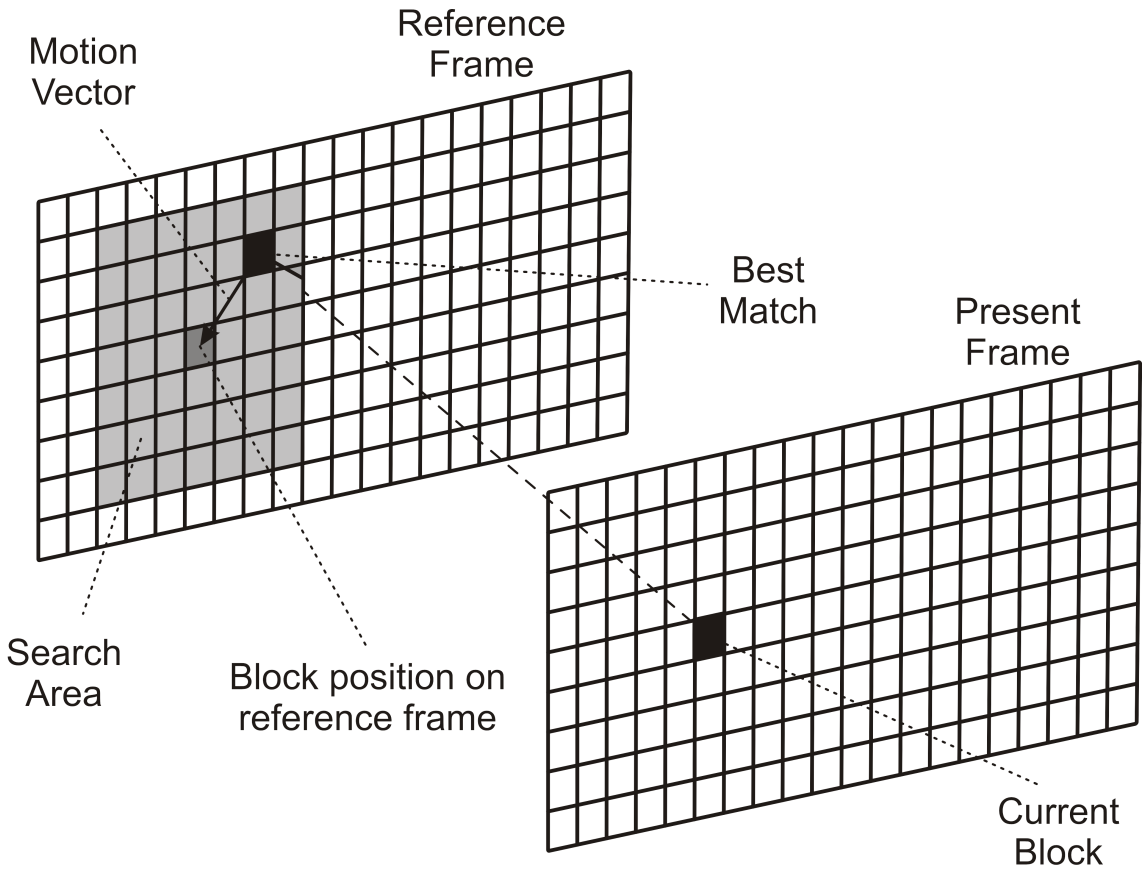
Motion Estimation (ME) is essentially the stage responsible for resolving the temporal redundancies in a video sequence, when inter mode is applied. ME finds the most similar block compared to the one being encoded. Thus, the information needed to be sent to the output of an encoder is just the difference between both blocks (namely, the block of residuals) and a vector pointing to the best matching block – Motion Vector (MV) – so that the decoder has sufficient information, along with the previously decoded frames, to recover the original block. The ME is split into two subsequent stages: Integer Motion Estimation (IME) and Fractional Motion Estimation (FME). These two modules are explained in a more detailed manner in the following subsections, with a special focus on the IME module, given that this is the main encoding stage which is dealt with in this work.

2.7.1 Integer Motion Estimation

Integer Motion Estimation (IME) is the first step of the ME and finds the best matching block between PUs in two different frames (current and reference), using only integer-pixel displacement vectors to compare blocks, by applying a block-matching algorithm (BMA) to the blocks being encoded in the current frame. This search algorithm defines a pattern of positions – represented by motion vectors with integer-pixel x- and y-components – in which the most similar block will be searched in the reference frame. The BMA is only applied in a given search area of the frame, which consists of a window typically smaller than the frame itself, because image patterns tend to slightly displace from the area where they were in a previous frame. Figure 2.5 generically presents the concepts involved in the IME.

Encoders usually define that a BMA will start in a position whose vector is defined by a stage called Advanced Motion Vector Prediction (AMVP). This stage obtains the resulting vectors from ME executions of previously encoded CUs, and checks which of them is the best one to start the search for the current block being encoded. More specifically, a subset of vectors from spatial neighbors – above-left, left, below-left, above

Figure 2.5: Generic search in a previously coded frame.



Source: (PORTO, 2008)

and above-right –, along with temporal neighbors, such as the vector from the co-located block, will decide the starting point of the BMA.

Many different BMAs have been proposed and are employed in current encoder implementations. The following subsections present some of the main BMAs mentioned in the literature.

2.7.1.1 Full Search

The Full Search (FS) is the most naïve BMA implementation in the literature. FS applies the search for the most similar block by displacing to every pixel in the search window, and it always finds the best possible matching block contained in the search window. For that reason, FS obtains an optimal result in temporal redundancy within the search window, resulting in a smaller bitstream at the encoder output.

The number of candidates evaluated by FS is equal, in magnitude, to the area, in pixels, of the search window, given that these are all the possible integer displacements. Due to the large number of candidates being tested, FS requires the higher number of

memory accesses and calculations among all the search algorithms. For that reason, real-time implementations employ other solutions and search heuristics, so that the search is performed for a smaller number of motion vector displacement candidates, while still attempting to find residual blocks very similar to the best possible within the search window considered.

Even though this algorithm is mostly not suitable for real-time implementations, FS has the advantage of not being data-dependent. In other words, all candidates to be evaluated are known from the moment the start position is found, so pre-fetching mechanisms to gather block data can be applied. As it will be seen in the next subsections, the other presented BMAs all depend on block data from previous iterations of the search algorithm being employed.

HM reference software employs the FS algorithm in the bi-prediction stage of the ME stage, by default. The x265 software does not use FS in any of the high-level presets; placebo, which is its most complex preset, uses Star Search (SS), whose behavior will be explained later.

2.7.1.2 Test Zone Search

Test Zone Search (TZS) is the main BMA employed by the HM reference software. TZS represents a better trade-off between time and quality, i.e., even though it does not always find the optimal block inside the search window, it ends up performing a much smaller amount of block comparisons when compared to FS, making it less time-consuming.

The default configuration of TZS is based on a diamond-shaped search, and it is split into four subsequent stages (CRISTANI, 2014):

- **Search Vector Initialization**: this stage checks whether the colocalized vector – pointing to the colocalized candidate block from the reference frame – or the vector resulting from the IME iteration of the 2N$\times$2N partition are better than the current start vector, which was chosen by the AMVP stage. The start search vector is then set as the best one out of the three. Sometimes, these vectors may already have been evaluated, so the encoder performs in the most optimal way to not check these vectors more than once;
- **First Search (FiS)**: in this stage, the main diamond-shaped search is performed, starting from the position defined from the first step. The diamond pattern checks

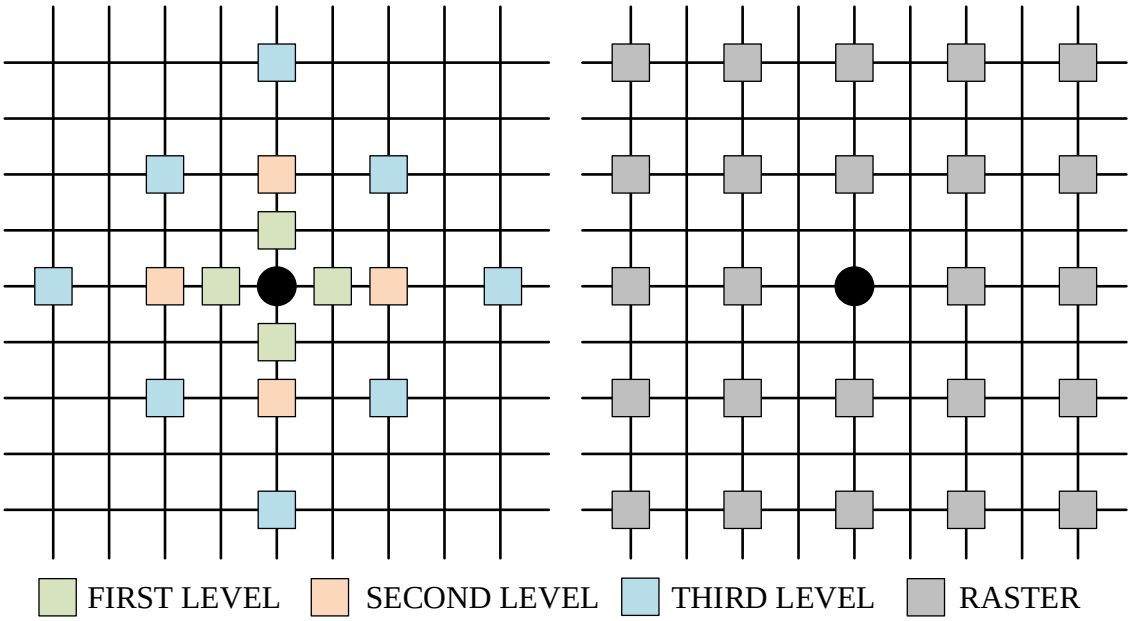
four, eight or sixteen candidates, depending on the number of iterations it requires, which is incremented after each diamond step. FiS ends when the algorithm does not find any better candidates for three iterations. Basically, HM implements a loop with a variable denoted as $iDist$, and this variable is squared at each iteration, increasing in powers of two. This value indicates the distance of the diamond shape from the current center. The higher the value $iDist$ is, the more candidates are evaluated, with a maximum of sixteen candidates per iteration;

- **Raster Search (RS)**: this step searches through the entire search area with a step of pixels between each candidate defined by a variable denoted as $iRaster$. Also, this stage only executes if the vector generated so far by the algorithm is larger than the same value defined by $iRaster$. This variable is set to 5 in the default preset of the encoder. This stage can also be seen as a more generic case of the FS BMA. A RS execution with a step of 1 would imply just in the execution of the FS. This step, just as FS, is also fully independent on its own, so the candidates can be evaluated in parallel, if enough resources are available to perform that;
- **Refinement Search**: the last stage also performs a diamond-shaped search, just like in FiS. However, it performs some iterations of that same algorithm – which is already composed of numerous iterations. This step, as well as the whole IME stage, stops when no better candidates have been found in an entire iteration. The best motion vector from this stage is chosen as the global motion vector of the IME execution, and it is ready for the FME to be applied.

These stages are illustrated in Figure 2.7.1.2 in a simplistic way, highlighting the possible diamond and raster patterns, with a vector starting from the decision in the search vector initialization step.

The diamond shape from TZS may also be replaced by a squared shape, by varying the presets in the HM software. Moreover, several other additional stages may be applied to enhance the block search. In other words, the TZS algorithm can be seen as a template, in which different combinations may be used in each of its steps. The x265 software also implements an algorithm that follows the same stages of the TZS, namely Star Search (SS). This algorithm is executed in the Slow, Slower, Veryslow and Placebo versions of x265. There is a small variation in the Placebo version, however, given that the ME Range parameter – related to the size of the search window, which has been previously explained – is 92, whereas it is defined as 57 for the remaining presets. This specially makes the execution of RS even more costly, given that its number of candidates is proportional to

Figure 2.6: Search shapes for the TZS.



Source: The Author (ABREU et al., 2018)

the number of pixels in the search window.

2.7.1.3 Hexagon Search

Hexagon Search (HS) is the BMA implemented in the medium (default) preset of x265. This algorithm follows a hexagon-shaped pattern, and it has a simpler execution flow than TZS, which is one of the reasons why x265 is so much faster than HM, since IME represents a large portion of the total execution time. HS has three stages, which are described below:

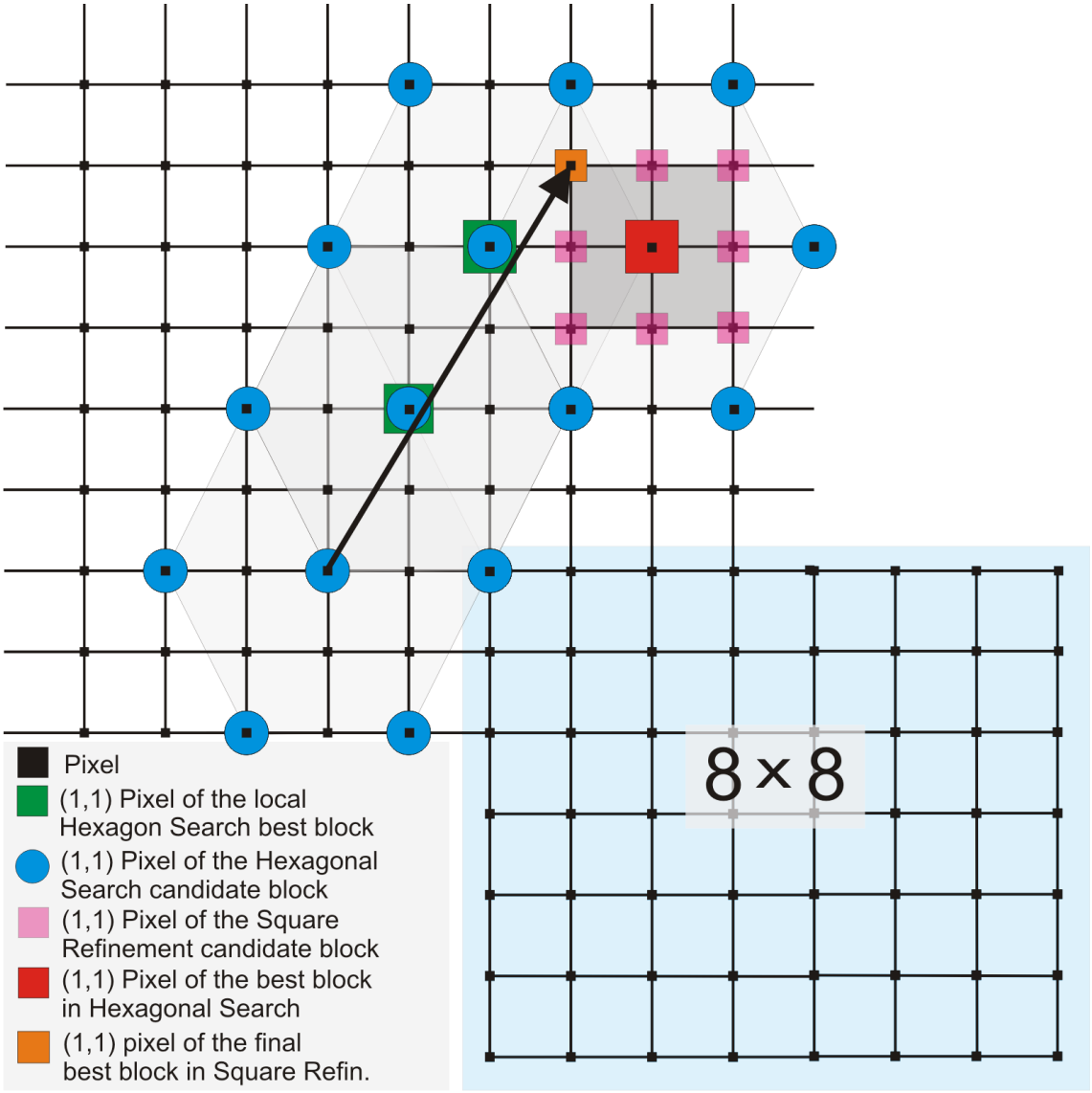
- **Search Vector Initialization**: this stage works similarly to the Search Vector Initialization in the TZS algorithm, only differing in the number of candidates evaluated to decide the initial center of the search;
- **Hexagon**: this step performs a six-point search in a hexagon-shaped format around the center of the search, considering the initial center defined by the previous step. Whenever there is a candidate more similar to the block being encoded than the one in the center, the new center is set as the vector associated with that new candidate. Then, the iteration is applied again, starting from the new center of the search. This stage stops when none of the candidates are better than the current center. The x265 software applies an optimization to this stage: starting from the second iteration of the hexagon search, only three candidates need to be evaluated instead of six, since

the three remaining points will always have been evaluated in the previous iteration;

- **Square Refinement**: after the Hexagon step defines the best candidate, an 8-point square refinement is applied around that point. The final vector value is defined by the best candidate evaluated at this stage. If none of the candidates are better than the current center, then the center defined by the previous step is the best vector.

Figure 2.7 illustrates the whole process of the HS algorithm, considering an 8×8 PU, in which four iterations of the Hexagon are performed. In the last one, no candidates were more similar to the block being encoded than the center, so the square refinement was applied, resulting in a better block found in one of its eight candidates.

Figure 2.7: Search shapes for the HS.



Source: (SILVEIRA et al., 2017)

It should be noted that, for both TZS and HS, the decision of which blocks to

compare heavily depends on the previous stage, as most of the steps start from the best current vector of a previous one. This severely harms pipeline implementations, possibly resulting in pipeline stalls, due to the fact that the decision on the next candidate to be gathered is still executing.

2.7.1.4 Alternative BMAs

Apart from the main BMAs presented in the previous subsections, and from the few BMAs proposals that will be seen in Chapter 3, there are some other BMA implementations in both HM and x265 encoder softwares. These will be briefly explained in the following items:

- **Diamond Search**: this is the BMA employed in the ultrafast preset of the x265 encoder. This pattern simply evaluates four candidates, i.e., above, right, left and below, all of them with a distance of one pixel from the original block, and iterates through this evaluation several times, until similarity conditions are met, or until the search hits the search window borders;
- **Uneven Multi-Hexagon (UMH) Search**: this is implemented in the x265, and it is an adaptation of the BMA used in x264 – the x265 equivalent for H.264/AVC. UMH has a rather complex flow: it starts with a refinement of its predictors, contains a few early-termination mechanisms, uses an adaptive search range based on the variability of the MV candidates, and also uses a shape similar to a hexagon;
- **Successive Elimination Algorithm (SEA)**: also implemented in x265, this BMA is very similar to FS, but attempting to perform it faster, applying different metrics than usual;
- **Selective TZS**: this is a version of TZS with its main stages, but with weaker early termination restrictions, ending the execution earlier when compared to normal TZS. This is implemented in HM as an alternative to TZS, but it is not used by default.

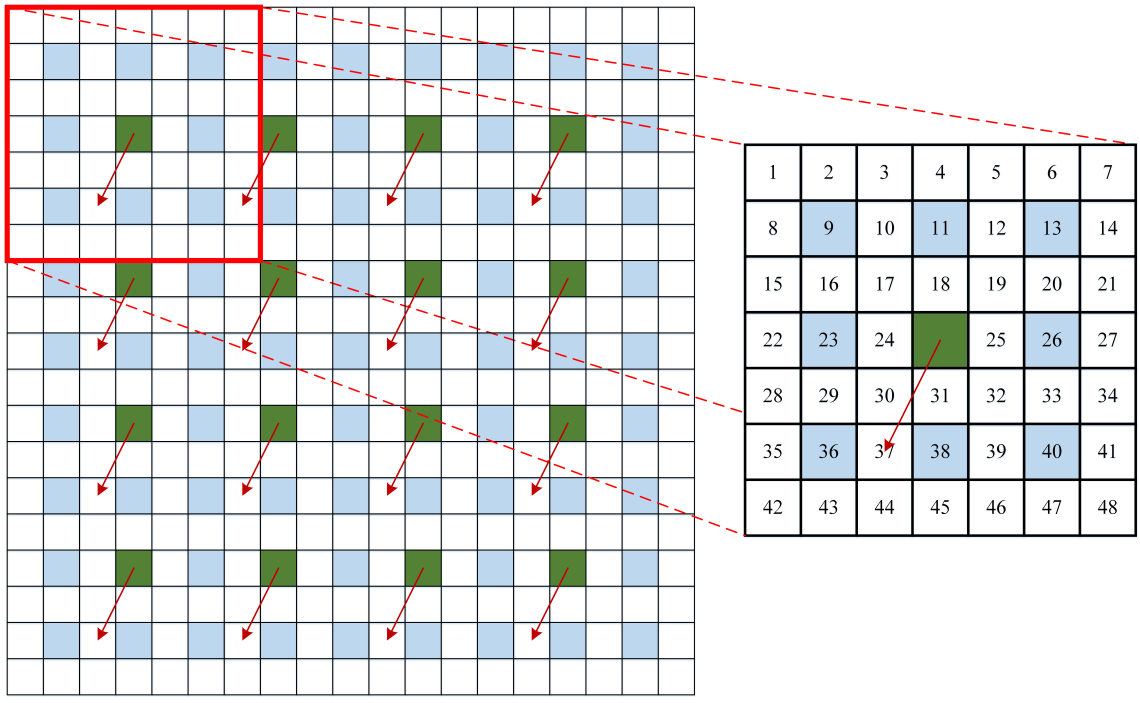
2.7.2 Fractional Motion Estimation

Fractional Motion Estimation (FME) is applied after the IME and it is responsible for finding the best match at the fractional-pixel level, starting from the most similar block found in the IME. Since the frames are formed only by integer pixels, FME requires the

use of an interpolator to estimate fractional pixels positioned between the integer pixels of the image. FME is responsible for increasing image quality in video sequences due to the fact that real-life patterns most frequently move at a rate that is not a good match to the integer-pixel displacement between two simultaneous video captures.

HEVC defines 48 possible candidates to be compared in the FME: 8 half-precision and 40 quarter-precision candidates. Fig. 2.8 presents the set of fractional points needed to gather all the information regarding the 48 fractional blocks. In this figure, green positions correspond to integer pixels, blue positions correspond to half-precision pixels and white positions correspond to quarter-precision pixels. Half and quarter-precision pixels are generated by interpolation using 7-taps and 8-taps FIR filters (RABINER; GOLD, 1975), depending on the specific point. The highlighted partition shows the 48 possible points to which comparisons will be performed.

Figure 2.8: FME fractional pixels window for a 4×4 PU.



Source: The Author

In x265 and HM, only a subset of these candidates are evaluated. In both encoders, only 8 half-precision and 8 quarter-precision candidates are evaluated. The execution first evaluates the 8 half precision pixels, and based on the best one, the 8 quarter precision candidates surrounding the best half precision candidate are evaluated.

2.7.3 Metrics for Block Similarity

Several metrics can be applied to determine the degree of similarity between two blocks. They differ from each other in ease of implementation, efficiency and result accuracy, i.e. how precisely they can define the similarity between the blocks. The main metrics used to estimate block similarity in video codecs are shown in the next subsections.

2.7.3.1 Sum of Absolute Differences

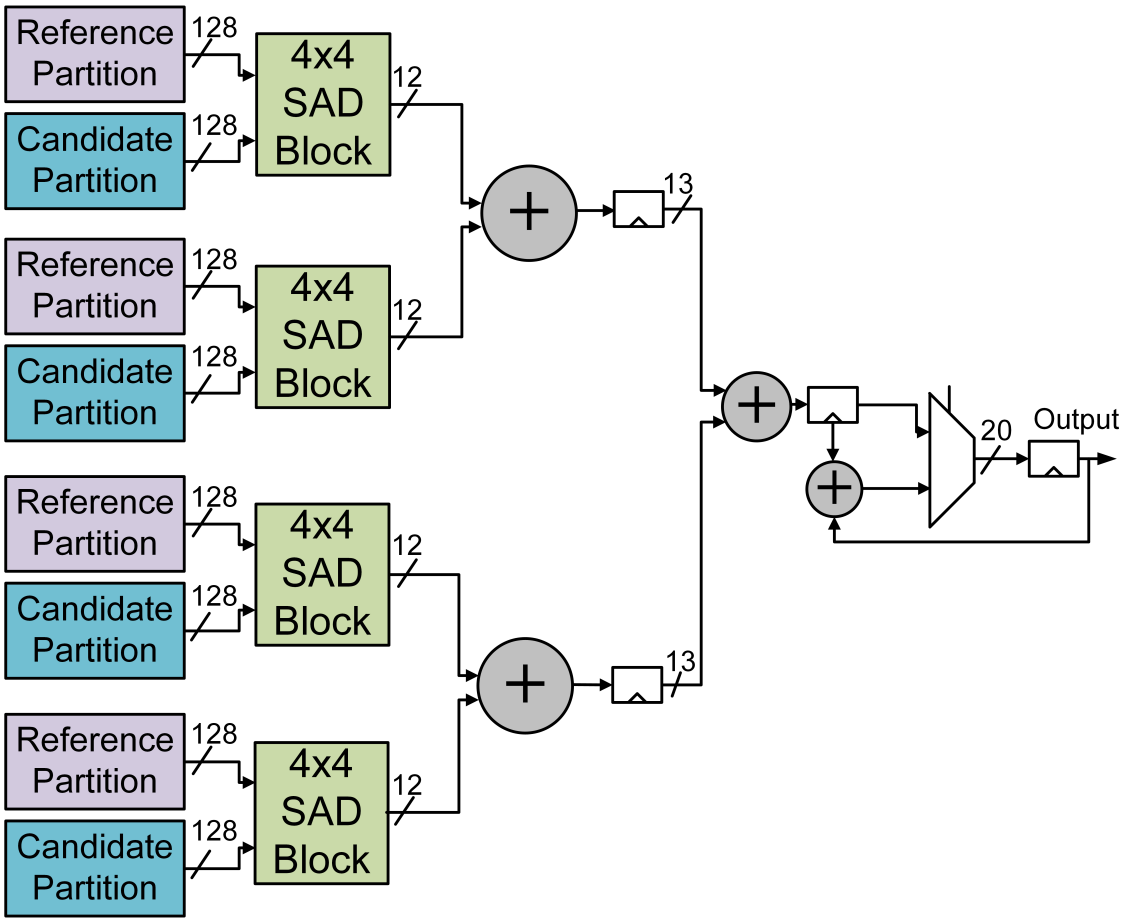
Sum of Absolute Differences (SAD) is the simplest similarity metric used in the video encoding process and it is applied by calculating the differences between the co-localized pixels of a current and a candidate block, performing an absolute operation in these differences, and then adding the values. SAD is employed in the IME and is also one of the most used metrics in the video encoding process, representing, on average, 22.4% of the encoding time in the HM reference software (ABREU et al., 2017). The complete formula is given by 2.4, in which O and R denote the current and the candidate blocks, respectively; m and n refer to the width and height of the blocks (correspondent to the shape of the PU being considered).

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |O_{i,j} - R_{i,j}| \tag{2.4}$$

SAD architectures are mostly implemented using subtractors, absolute operators, and an adder tree, with an accumulator on the output so that SAD for bigger blocks can also be calculated. Most architectures use pipeline schemes so that several sum stages can be done concurrently and the resulting critical path is shortened. An 8×8 SAD architecture is shown in Fig. 2.9, in which the sizes are presented in bits.

The presented architecture calculates SAD for an 8×8 block in each cycle after the pipeline is filled. The 4×4 SAD blocks correspond to simple adder trees, including the subtractors and absolute operators. The 2-1 multiplexer is used to extend the block sizes possibilities so that receiving 8×8 blocks several times would allow the architecture to calculate SAD for larger block sizes. Considering a generic SAD architecture, the number of cycles required to calculate an $W \times H$ block is given by Equation 2.5. In this formula, $input_W$ is the input width, in bytes, of all the candidate partitions; W and H refer to the width and the height of the PU to be calculated, respectively; and P_{depth}

Figure 2.9: 8×8 SAD architecture.



Source: The Author

corresponds to the number of pipeline levels in the architecture.

$$cycles = P_{depth} + \frac{(W \cdot H)}{input_W(bytes)} \tag{2.5}$$

It should be noted that an increase in the minimum block size of the architecture implies in more area for the dedicated parallel hardware operators, resulting in power and energy increases. However, small architectures spend more time calculating SAD for larger blocks, since the number of accumulations needed is higher. SAD is a bottleneck in terms of power when used in ME architectures. The work in (MIYAKOSHI et al., 2005) demonstrates that SAD computation accounts for 59%-66% of the total power dissipation in ME architectures.

2.7.3.2 Sum of Absolute Transformed Differences

The Sum of Absolute Transformed Differences (SATD) is a more complex metric for similarity evaluation during the inter-prediction process. SATD is calculated by

taking the frequency transform of the differences between pixels of current and candidate blocks. Equation 2.6 presents the formula to calculate SATD. SATD represents, on average, 27.1% of the encoding time (SILVEIRA et al., 2017). In Equation 2.6, $T_{i,j}$ refers to the coordinate (i, j) of the block after being transformed to the frequency domain, and m and n denote the block dimensions, just as in SAD.

$$SATD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |T_{i,j}| \tag{2.6}$$

In HEVC, SATD is the metric used by default during the FME, and Hadamard is the transform function used for the space-to-frequency transformation. Although SATD is a more accurate metric to determine the most efficient block match, its hardware architectures have a more complex implementation when compared to SAD architectures.

2.7.3.3 Sum of Squared Errors

Sum of Squared Errors (SSE) is one of the most complex metrics to measure similarity in the HEVC standard. This is due to the fact that its implementation requires the use of squared operators, which are more costly in terms of hardware. Therefore, even though this operation is more precise in finding similarity than SAD, encoders usually avoid applying it. The operation is shown in Equation 2.7, where $O_{i,j}$, $R_{i,j}$, m and n have the same meaning as in SAD.

$$SSE = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (O_{i,j} - R_{i,j})^2 \tag{2.7}$$

HM and x265 do not apply this metric in the ME process, by default. SSE is only used after the inter-frame prediction stage occurs, in code portions that are not as frequently executed.

2.7.4 Motion Vector Cost

In order to define the similarity of blocks, the three aforementioned metrics can be used. However, the end goal of an encoder is to decrease the output bitstream size, while keeping the quality level of the video.

It is known that very similar blocks will generate residual blocks with many of its matrix values as zeros (0's), which will likely be more optimized in the transform and

quantization stages when compared to candidate blocks that were not as similar. These high quantities of zeroes will likely enhance the entropy algorithms that will be able to represent the output bitstream with less bits.

However, it is important to notice that the motion vector (MV), i.e., the vector pointing to the best candidate block, also needs to be sent to the decoder through the bitstream, so that it can retrieve the original block. Therefore, the bit cost of the MV – MV_{Cost} – also needs to be taken into account when calculating the total cost of an IME execution.

In other words, if a less similar candidate block (with a higher SAD/SATD/SSE value) has a motion vector with a bit cost smaller than another candidate block's motion vector, and the bit difference of the motion vector compensates the smaller similarity of the candidate block, the less similar one should be chosen, given that it decreases the final bitstream size.

The total cost of a block in modern encoders is represented by Equation 2.8. In this equation, $Distortion$ refers to the value of the chosen similarity metric, e.g., SAD, SATD, SSE, etc.

$$RD_{Motion} = Distortion + MV_{Cost} \tag{2.8}$$

The MV_{Cost} value is calculated in encoders as an estimate of how much the entropy algorithm will optimize it, so it does not need to go through the transform, quantization and entropy stages. This value denotes a prediction of the bit cost of the motion vector, to assist the entropy stage with motion vectors with a smaller bit cost, decreasing the final size of the bitstream.

The value of MV_{Cost} is given by Equation 2.9. In this formula, λ_{Motion} denotes the Lagrange multiplier, which depends on the chosen QP, and R_{Motion} is associated to the number of bits to encode the MV (SULLIVAN; WIEGAND, 1998).

$$MV_{Cost} = \lambda_{Motion} \cdot R_{Motion} \tag{2.9}$$

2.8 Partial Distortion Elimination in SAD

The main idea of partial distortion elimination (PDE) techniques is to early-terminate the calculation of block candidates whose distortion values will certainly exceed the cur-

rent best one. Therefore, the earlier we find out that a candidate will not be the best one, the less cycles we need in the overall execution. This optimization can be used in any distortion metric, as long as there are ways of prematurely eliminating the candidates. This work, however, focuses on using PDE in SAD architectures, given that this is the most popular one for IME.

The use of PDE is possible in SAD due to the implementation scheme of the architectures. Due to the large block sizes supported by HEVC, SAD units for this standard usually do not compute the entire block in a single cycle, requiring several iterations to calculate them. Therefore, PDE acts when there is a way to find out in fewer cycles that a candidate is worse than the current best one.

More details on PDE is presented in related works from Chapter 3. The PDE techniques described and proposed in this work are presented in Chapter 5.

2.9 Power Dissipation in CMOS Circuits

Power dissipation is one of the main parameters to be considered when designing digital circuits. Within this context, peak power is used as one of the main aspects of the reliability analysis of circuits. However, the most critical factor is the average power consumption of the circuits that operate during a time interval (CHANDRAKASAN; BRODERSEN, 1995). Power dissipation in CMOS circuits is split into dynamic and static power. These power sources will be detailed in the following subsections.

2.9.1 Dynamic Power

Dynamic power is the main source of power dissipation, even though the differences when compared to static power are becoming smaller in newer technologies that involve larger integration scales. Dynamic power can be split into switching power and short-circuit power, both of which are presented below.

2.9.1.1 Switching Power

This portion of the dynamic power is a result of the charging and discharging of the capacitances of the circuit, as a consequence of the switching activity of CMOS circuits (RABAEY, 1996).

Switching power in a logic is measured based on Equation 2.10, where we assume that a load capacitance C_L will be charged or discharged based on a change in the gate output. In Equation 2.10, A denotes the output node activity, measured in events/second, V_{dd} is the input source voltage, f is the frequency of operation. It may be more convenient to use a probability value α, which represents the node activity factor, along with f, given that most of the circuit nodes will not change at every clock cycle.

$$P_{switching} = C_L \cdot A \cdot V_{dd}^2 = \alpha \cdot f \cdot C_L \cdot V_{dd}^2 \tag{2.10}$$

Most tools for measuring power consider α as a fixed value. This is employed as a methodology for power extraction in many works, which does not lead to accurate results, given that the input variations depend on the application for which the developed architecture will be used.

2.9.1.2 Short-Circuit Power

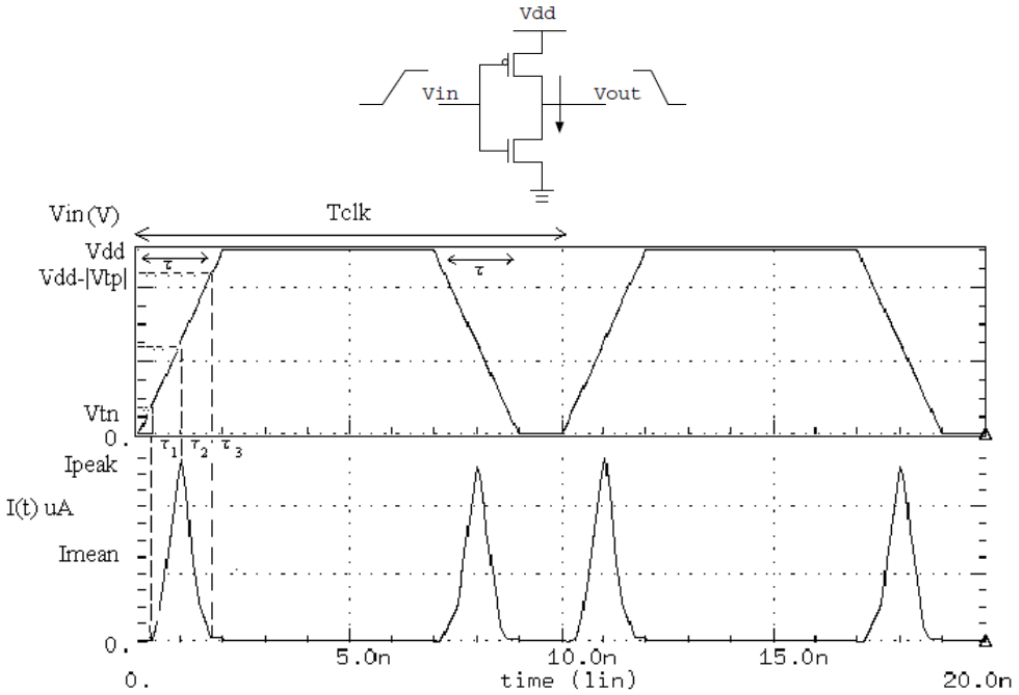
Short-circuit is part of the dynamic power, and it occurs when a direct current flows through a path from the voltage supply to ground, in a CMOS network. This happens when a static CMOS circuit is switched on by an input signal with rising and fall times different from zero, which leads to the PMOS and NMOS transistors of both networks being active simultaneously for a short period of time. This situation leads to the direct current to flow directly to ground.

This concept is seen in Figure 2.10, using the example of an inverter gate. As it can be seen, in the rise transition, the NMOS transistor starts to drive when the input voltage V_{in} is equal to the threshold voltage V_{tn} of the NMOS. The PMOS transistor will stop conducting when V_{in} is equivalent to the difference between V_{dd} and the threshold voltage of the PMOS transistor V_{tp}.

Short-circuit power can be expressed by Equation 2.11, where I_{CC} is the average short-circuit current, and β is a factor related to the input signal transition time to the cell peak short-circuit current.

$$P_{SC} = \beta \cdot I_{CC} \cdot V_{dd} \tag{2.11}$$

Figure 2.10: Short-circuit current for an inverter gate.

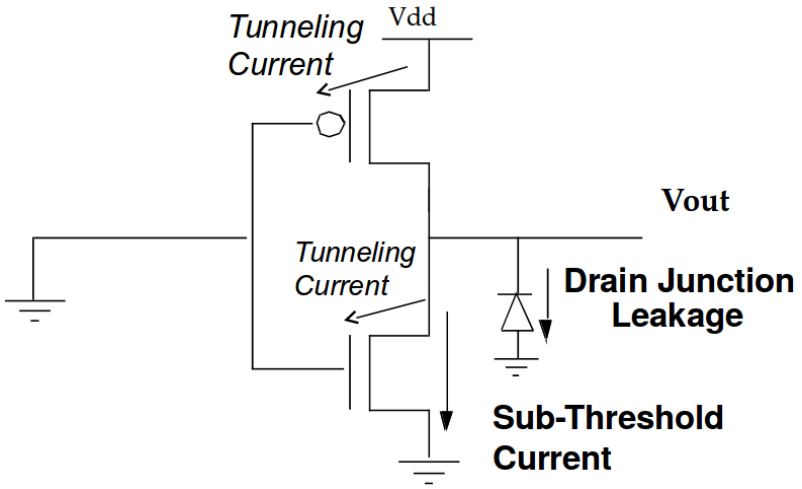


Source: (Costa, Eduardo Antonio César da, 2002)

2.9.2 Static Power

Static power occurs due to leakage, even when the circuit is in an idle state. Leakage currents in CMOS transistors are due to three factors: subthreshold transistor currents, reversely biased diodes and gate-to-channel or gate-to-drain tunneling effects (RABAEY, 1996). These factors can be seen in Figure 2.11, using an inverter gate as an example.

Figure 2.11: Sources of leakage currents, in a CMOS inverter gate.



Source: (RABAEY, 1996)

Static power is given by Equation 2.12, where I_{static} is the static current.

$$P_{static} = I_{static} \cdot V_{dd} \tag{2.12}$$

2.9.3 Total Power

Taking the dynamic – switching and short-circuit – and static power into account, we can define the total power as in Equation 2.13.

$$P_{total} = P_{dyn} + P_{static} = P_{switching} + P_{SC} + P_{static} \tag{2.13}$$

When considering the characteristics of a digital integrated circuit, in which there are N nodes, each of which containing its own capacitive loads C_{L_i}, we can extend this model, based on the equations presented in the previous subsections, obtaining the formula in Equation 2.14.

$$P_{total} = \sum_{i=1}^{N} \alpha_i \cdot f \cdot C_{L_i} \cdot V_{dd}^2 + \sum_{i=1}^{N} \beta_i \cdot f \cdot V_{dd} + \sum_{i=1}^{N} I_{static} \cdot V_{dd} \tag{2.14}$$

3 RELATED WORKS

For this dissertation, an extensive research in related works has been performed to find related proposals in the literature. This chapter will present and describe works regarding analyses of SAD, SATD, IME or entire ME works, and works focused on PDE techniques.

3.1 SAD and SATD works

Several works are focused on implementations of SAD and SATD modules, in several different ways.

Silveira *et al.* (SILVEIRA et al., 2017) propose a low-power SAD architecture using adder compressors. Structures for 3-2, 4-2, 5-2 and 7-2 adder compressors are presented in the work. This work is an extension of the one presented in (SILVEIRA et al., 2016). Based on a throughput analysis, the work decided a frequency for the architecture to be synthesized, to simulate a real-time scenario. The work also employed a power estimation methodology using real-input vectors from the encoder, which increases the accuracy of the final power results. The work obtained a power and energy reductions of 25.5%, on average, when compared with the SAD architecture using conventional adders. However, the work does not present any optimizations in order to decrease the number of SAD computations.

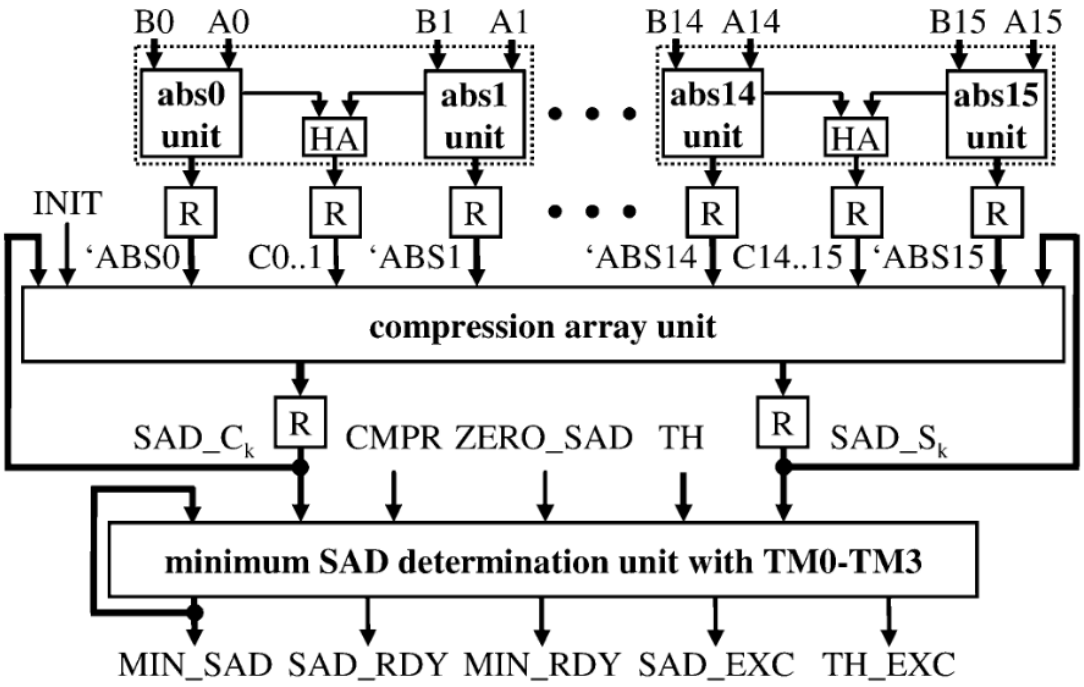
Abreu et al. (ABREU et al., 2017) present an extensive analysis on different absolute operators implementations to be used in SAD architectures. This analysis comprised ten different architectural models for the absolute operator, some of which have been based on related works found. The work compared all the versions with the baseline implementation – using the "abs" macrofunction from the synthesis tool. The work performed implementations for a SAD architecture considering 8×1 modules, with and without the use of pipeline. The target frequency for the synthesis was set based on the work from (SILVEIRA et al., 2017). The architectures were also fed with real-input vectors from the encoder software, to obtain accurate power results. The best design achieved a power reduction of 20.4% and a 7.8% reduction in area.

Kumm *et al.* (KUMM; KLEINLEIN; ZIPF, 2016) propose an architecture for computing SAD in an efficient manner, for FPGA devices. The architecture is based on a configurable adder/subtractor model where both inputs can be negated. The synthesis

was performed for Xilinx Virtex-6 FPGA. The work managed to achieve higher maximum frequency results when compared to related works presented, while decreasing the required Look-up Table (LUT) usage. Overall, the proposed circuit reduces the resources in 17.4%. However, no analysis is presented regarding frequencies of real-time scenarios, nor any optimizations in an encoder-level are presented.

Vanne *et al.* (VANNE et al., 2006) propose a high-performance SAD architecture for ME. Even though the work is focused on previous standards as well, a more detailed analysis on the SAD architecture is presented. Some absolute difference operators are considered in this paper (VASSILIADIS et al., 1998; JEHNG; CHEN; CHIUEH, 1993) in order to present an improved absolute difference module. The absolute differences is based on inverting one of the inputs, adding the values and maintaining or inverting the output based on the carry-out bit of the addition. Moreover, correction bits are required to be added in the compression array unit, to make up for the incorrect result of the current absolute operation. The paper also employs a compression array unit with carry save adders (CSA) trees for the SAD module. The full module proposed by Vanne *et al.* is shown below, in Figure 3.1.

Figure 3.1: Vanne *et al.* proposed 3-stage SAD architecture.



Source: (VANNE et al., 2006)

However, the work does not present any power results for the presented modules. Also, even though early-termination mechanisms have been presented, the analysis does

not consider its real impact in cycles, power and timing. Also, the work does not take MV_{Cost} into account, which leads to less efficient compression.

Shah *et al.* (SHAH; DALAL, 2018) present a SAD system with parallel memory. An extensive analysis on hardware proposals for different BMAs and SAD architectures is presented. However, no analysis is presented regarding SAD optimizations. Also, the synthesis was performed for FPGA and power results were not presented.

Yufei *et al.* (YUFEI; XIUBO; QIN, 2007) propose a high-performance low cost SAD architecture. The work present the absolute stages, the compression array unit, and describes in a detailed manner how each component of the SAD tree was implemented. The architecture manages to achieve considerable area gains (15.3% - 22.9%) when compared to related works. However, no optimizations in the context of decreasing cycles were presented. Also, the work does not present power results, and no analyses are performed in the context of specific video encoders.

Dang *et al.* (DANG; CHANG; KIM, 2014) propose a power-efficient SAD unit for H.264/AVC encoder, by approximating several most significant bits (MSBs) into a single one, while managing to not degrade the video quality. The architecture presented power results considering effect of fan-it, fan-out and interconnections, which lead to more accurate results. An improvement of 18% was obtained when compared to the architecture from (VANNE et al., 2006). However, the work was proposed for the predecessor H.264/AVC standard, which leads to the fact that it may not be thoroughly suited for more recent standards, such as HEVC. Also, no PDE techniques were employed, which could improve the results even more.

Soares *et al.* (SOARES et al., 2016) propose energy-efficient SATD architectures for HEVC based on approximate computing concepts. A pruning-based algorithm is proposed, to remove least significant coefficients and the associated terms, to decrease even more the operation complexity. The architectures with varied number of discarded coefficients were synthesized for 45nm CMOS standard cells library, and a maximum of 52.7% of power was saved with the pruning techniques.

Seidel *et al.* (SEIDEL et al., 2016) present two approaches for calculating SATD for any N×N block. The proposals use a transpose buffer and a linear buffer. As it could be seen from the results, the scalability of the method using linear buffers was the best in terms of area. However, when using Low-Vdd/High-Vt synthesis, the most energy-efficient architecture was the one using transpose buffers.

3.2 IME/ME works

Some works also propose complete IME/ME solutions, which usually contain SAD modules within them.

Chen *et al.* (CHEN et al., 2006) design an architecture for ME, considering SAD as the similarity metric. Two IME modules are presented, focused on low and high-resolution applications. Partial SAD techniques are employed to reduce the circuit area of the search mechanism. The synthesis results estimated a gate count of 330.2k, running at a target frequency of 108 MHz. The proposed architecture achieves 720p real-time throughput, considering a single reference frame. This consideration harms the video quality, given that encoders usually employ more reference frames per IME execution. Furthermore, power results were not presented.

Yap *et al.* (YAP; MCCANNY, 2003) implement a ME architecture for the FS BMA. SAD architecture is included within the proposal. However, the analysis considers previous and outdated video coding standards. Furthermore, the analysis considers an unfeasible BMA for real-time encoding of high resolution videos.

Porto *et al.* (PORTO et al., 2010) propose a ME architecture using 4-2 and 8-2 adder compressors (similar to the ones used in (SILVEIRA et al., 2017) to implement the adder tree of SAD architecture. The BMA employed in the algorithm was the Quarter Sub-sampled Diamond Search with Dynamic Iteration Control. The architecture was synthesized to FPGA devices and to a 180nm CMOS standard cell library. The maximum target frequencies for FPGA and ASIC were 213.3 MHz and 287.3 MHz, respectively. Processing rate gains of up to 12% were obtained when compared to their previous implementation.

Sanchez *et al.* (SANCHEZ; PORTO; AGOSTINI, 2013; SANCHEZ et al., 2015) propose hardware-friendly IME algorithms and respective VLSI designs. The proposals are named spread and iterative search (S&IS) and low density and iterative search (LD&IS). Tese proposals are based on employing parallel diamond search executions. Some improvements were performed for the proposals to have regular memory accesses. The work achieves 1080p real-time throughput, considering one reference frame, which causes quality degradation. The synthesis was performed for FPGA at 210 MHz and for ASIC 90 nm standard cells library at 42.3 MHz (to target 1080p). The ASIC synthesis for LD&IS and S&IS obtained power results of 12.5mW and 13.5mW, respectively.

3.3 PDE Works

Some works can be found in the literature focusing on PDE techniques.

Chiu *et al.* (CHIU; SIU, 2006) propose an approach similar to PDE in ME, using a Successive Elimination Algorithm (SEA). The goal is to eliminate as many redundant operations as possible. The paper early-terminates SAD calculations. However, no architectures are implemented in order to obtain power, energy and area results. The analysis also does not consider state-of-the-art BMAs such as TZS, and does not consider the effects of PDE in newer standards, such as HEVC. Moreover, no considerations of the MV_{Cost} value is presented, as it considers that the distortion cost is only composed by SAD.

Choi *et al.* (CHOI; JEONG, 2009) propose a PDE algorithm based on correlations found in the distances between the blocks and the SAD values. The proposal attempts to change the order of the pixels to calculate partial SAD, operating with the pixels with higher contributions to the total SAD, as to converge to an approximation of the true SAD faster. To do that, the work performs a relationship between the SAD values and the distances from the mean value of a block. Experimental results show improvements of about 45% compared to typical PDE. However, this analysis considers a full scanning BMA, and it becomes unfeasible for state-of-the-art encoders. Also, changing the order of the pixels may lead to memory issues, as this rearranging scheme may be critical and influence on the throughput. Also, no architectures were presented, and therefore it is not possible to define the power effects of the proposals. Lastly, MV_{Cost} was not considered in the total distortion cost.

Lee *et al.* (LEE et al., 2009) propose a PDE algorithm employing a mechanism of maximum error constraint. This proposal introduces losses to the encoder. Also, the paper also considers the FS algorithm. The use of FS in such analysis tends to be optimistic, due to the fact that it evaluates every candidate of the window, including the ones that are not even close to similar. Also, no architectures are implemented with this proposal, so only theoretical results are presented.

Seidel *et al.* (SEIDEL; BRäSCHER; GüNTZEL, 2015) evaluate the use of PDE for H.264/AVC. The implementation of the SAD architecture along with the modifications in the control unit to include PDE verification incurred in an increase in area of up to 10.9%. The energy requirements of using SAD with PDE decrease in 47.94%. However, even though an architecture was implemented to verify the energy reductions of the

PDE usage, exhaustive BMAs were considered, which may not reflect actual real-time implementations. The fact that the predecessor standard H.264/AVC was employed also indicates that these results may be outdated, given the different block partitions that are evaluated in newer standards, which may impact PDE results.

Several other works regarding SAD, SATD or ME architectures have been found. However, most of these works do not present power results, or do not run the synthesis for ASIC circuits. Moreover, even the ones that present power results for ASIC are not viable for comparison, given that they do not focus on PDE techniques, which is an optimization in SAD. Within the scope of PDE works found in the literature, to the best of our knowledge, none of them implement PDE techniques considering the HEVC standard, and considering higher resolution video sequences (4K). Moreover, the related works found do not consider the MV_{Cost} in the calculation of the total cost of the blocks, which leads to a loss in compression efficiency in the overall encoder, whose results will be presented in the next chapters. Moreover, considering the MV_{Cost} along with SAD allows for different architectural possibilities that improve PDE in different ways, two of which will be presented in Chapter 5. This new technique and discovery is important given that the IME stage is one of the bottlenecks of the encoding process and decreasing cycles without severe additional costs is important, especially when the compression efficiency is increased when compared to PDE implementations without the MV_{Cost}.

3.4 Literature Summary

In order to summarize the related works, we present their general characteristics, contrasting them with the ones from the work presented in this report. This is shown in Table 3.1. We analyze whether the related works employed SAD as the similarity metric, considered MV_{Cost} as well, used the HEVC standard, presented power results, used real-input vectors for any power analysis and employed PDE techniques. As it can be seen, most works mainly lack the use of the MV_{Cost} in the total distortion cost and do not present power results. Also, most of the ones that present power values do not consider real-input vectors from video sequences. Moreover, all the works that analyze PDE techniques do not take MV_{Cost} into account, and only one of them includes power results. It is important to notice that the work in (VANNE et al., 2006) includes a similar technique to PDE, but the logic is not exactly the same as employed in the other related works, so it is represented with an approximation symbol. Also, the work in (SEIDEL et

al., 2016) considers an analysis considering standards beyond HEVC, so it is also marked with a specific notation.

Table 3.1: Characteristics of Related Works.

	SAD	MV_{Cost}	**HEVC**	**ASIC**	**FPGA**	**Power Results**	**Real-input Vectors**	**PDE**
(SILVEIRA et al., 2016)	✓	-	✓	✓	-	✓	✓	-
(SILVEIRA et al., 2017)	✓	-	✓	✓	-	✓	✓	-
(ABREU et al., 2017)	✓	-	✓	✓	-	✓	✓	-
(KUMM; KLEINLEIN; ZIPF, 2016)	✓	-	-	-	✓	-	-	-
(VANNE et al., 2006)	✓	-	-	✓	-	-	-	≈
(SHAH; DALAL, 2018)	✓	-	-	-	✓	-	-	-
(YUFEI; XIUBO; QIN, 2007)	✓	-	-	-	-	-	-	-
(DANG; CHANG; KIM, 2014)	✓	-	-	✓	-	✓	-	-
(SOARES et al., 2016)	-	-	✓	✓	-	✓	-	-
(SEIDEL et al., 2016)	-	-	✓[1]	✓	-	✓	-	-
(CHEN et al., 2006)	✓	✓	-	-	-	-	-	-
(YAP; MCCANNY, 2003)	✓	-	-	✓	-	✓	-	-
(PORTO et al., 2010)	✓	-	-	✓	✓	-	-	-
(SANCHEZ; PORTO; AGOSTINI, 2013)	✓	-	✓	✓	-	✓	-	-
(CHIU; SIU, 2006)	✓	-	-	-	-	-	-	✓
(CHOI; JEONG, 2009)	✓	-	-	-	-	-	-	✓
(LEE et al., 2009)	✓	-	-	-	-	-	-	✓
(SEIDEL; BRäSCHER; GüNTZEL, 2015)	✓	-	-	✓	-	✓	-	✓
This work	✓	✓	✓	✓	-	✓	✓	✓

[1] Beyond HEVC.

Source: The author

4 METHODOLOGY

In order to obtain BD-BR, cycles and power results for the proposals, several considerations and decisions were made, and several different tools to perform the required tasks were used. The following sections detail the methodology used in each step of this work.

4.1 BD-BR analysis implementations using the software encoders

The analysis in this work was performed with both the HM and x265 encoder softwares. The use of both encoders was important given that they contain the main implementations of TZS and HS, respectively. A first comparison was performed in order to decide which BMA to use for the remaining analyses, and HS was the initial choice, so only x265 was going to be used. However, for the main proposal – which includes the use of SAD along with MV_{Cost} – that will be shown in the next chapter, it was decided to run the simulations for both HM and x265. This is mainly due to the fact that HM is the reference software, and it is more used by the literature for academic purposes. This work used version 16.9 of HM. Even though 16.20 is the last version of the software so far, no significant changes in the results should occur, as no abrupt modifications have been made in the modules being targeted in this work.

All the BD-BR results were obtained by running simulations using the baseline as the default implementation of the respective encoder, and running the modified version of the encoder, for QPs 22, 27, 32 and 37, in accordance to the CTCs. Moreover, benchmark video sequences, also recommended by the CTC, have been employed. Some 4K video sequences have been also considered, from the Ultra Video Group (UVG) (GROUP, 2018), to increment the analysis to higher resolutions. The video sequences employed are shown in Table 4.1, both from the CTC and from UVG.

For the BD-BR analysis, apart from the main parameters available from both encoders, we needed to introduce two additional ones. The variables and functions shown below are from the HM reference software implementation. However, the implementations for the x265 software are fairly similar, apart from their specific names, as the same operations are performed.

- **useMVCost**: this parameter has a Boolean type, and disables or enables the MV_{Cost}

Table 4.1: Video sequences considered in the analysis.

	Resolution	Video Sequence
CTC	240p	BasketballPass
		RaceHorses
	480p	BasketballDrill
		PartyScene
	720p	FourPeople
		SlideEditing
	1080p	Cactus
		ParkScene
		BasketballDrive
		BQTerrace
		Kimono
		Tennis
UVG	2160p	Beauty
		Bosphorus
		HoneyBee
		Jockey
		ReadySteadyGo
		YachtRide

Source: The author

in the encoder. Therefore, instead of the total distortion being composed of the sum of SAD with MV_{Cost}, it will only consist of SAD. MV_{Cost} is added after the SAD value is calculated, when it is added to the return value of the SAD function. $useMVCost$ will simply set MV_{Cost} as zero before it is added to the SAD value;

- **useMVCostFME**: it has similar functionality to $useMVCost$. However, this is used to specifically disable MV_{Cost} in the FME operation. As both HM and x265 use SATD for the FME, in their default presets, this variable disables the use of MV_{Cost} with SATD.

Then, by running the encoders by enabling and disabling the desired parameters, a set of PSNR and bit-rate values were obtained, for each QP. A BD-BR calculator received these values as inputs, to calculate the degradations in the compression efficiency.

4.2 Cycles count

In order to obtain the cycles count of each of the proposed cases, this was performed in two different ways.

Our first analysis of employing PDE in SAD, without considering MV_{Cost}, was performed in the x265 encoder. The relative vector position of the current candidate of the IME execution was extracted – regardless of its absolute position on the reference frame, so that we could apply some scheme of candidate reordering, that will be explained in the next chapter –, along with data of the 8×1 SAD chunks of the specific candidate. Due to the fact that SAD is executed in other portions of the encoder software, we had to limit the data extraction through the use of flags that dictate that SAD will be extracted only within the IME execution. Also, given that there are several different SAD functions in x265, we extracted data from the functions *sad_x3* and *sad_x4*, which are the ones executed in the HS BMA. These functions respectively calculate SAD for three – half of the hexagon pattern – and four – half of the square pattern – candidates, for optimization purposes. It is important to emphasize that, in order to easily extract the data from the encoder, we needed to turn off the assembly functions for SAD, given that it would require unnecessary work to properly extract the SAD chunks from assembly code. This led to a slightly slower execution of the encoder. The extraction was performed as shown in Table 4.2.

Table 4.2: Chunk model for estimating PDE for SAD.

	Data
Coordinate	0×-1
SAD Chunks	83 85 86 103 93 85 ... 97
SAD Chunks	98 113 79 93 93 88 ... 93
⋮	⋮
Coordinate	1×-1
SAD Chunks	95 93 86 100 82 82 ... 80
SAD Chunks	90 105 95 86 86 86 ... 83
⋮	⋮

Source: The author

These extractions were further analyzed through the use of a Python script, that is responsible for reading the input files that contain the pre-calculated 8×1 SAD chunks for each candidate, and apply normal SAD and SAD with PDE, to obtain the total number of cycles. It is important to notice that the PU area is unnecessary, given that we can infer it

from the number of 8×1 SAD chunks contained in one specific line.

The second analysis, which included the MV_{Cost} value, was an improved version of the first one. Due to the fact that we were already aware that purely extracting the data and post-analyzing it with a script would lead to heavy files, we decided to implement the whole cycles analysis inside the encoder software. For this, we implemented the entire logic inside the same SAD functions – but now including the HM software as well – and using auxiliary variables for storing the SAD chunks and determining the effects of our several PDE proposals with MV_{Cost}. The auxiliary variables were needed so we did not change the overall execution of the encoder, given that we were not interested in making changes that would lead to different candidates being calculated or different prediction modes being taken, when compared to the default implementation.

4.3 Power Estimation Methodology for ASIC Design Flow

The methodology used to obtain accurate power results for our architectures was based on the one used in works such as (SILVEIRA et al., 2017). The method uses a design flow based on using real-input vectors from a specific application, in order to better estimate the dynamic power dissipation.

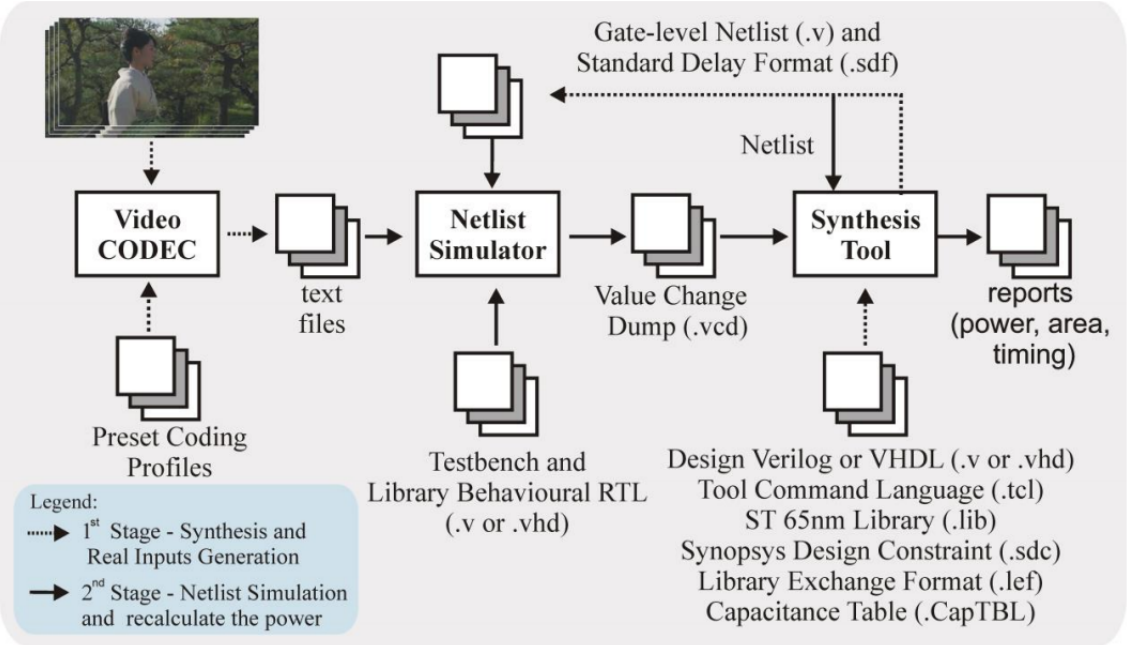
It is known that signal transitions cause dynamic power dissipation in digital integrated circuits. However, the estimations performed by commercial synthesis tools, such as Cadence Genus Synthesis Solution (CADENCE, 2019), are fairly pessimistic, rarely representing the actual behavior of the specific application for which the circuits were proposed. This is because such tools consider probalistic default input values to estimate the circuit total power dissipation.

Due to that, we require a more accurate method to obtain power, so that it represents, in a more precise manner, the real behavior, providing the designer with more reliable results, enabling the optimization of critical power-hungry modules.

The proposed methodology employed in this work to obtain the power results is shown in Figure 4.1. The whole process begins by describing the proposed architectures in some hardware description language. At the same time, we can use the video codecs – HM, x265, or some other encoder implementation – to extract the inputs from benchmark videos with the register-transfer level (RTL) description of the proposed architectures, which will be synthesized, in this case, using Genus, generating a netlist of the circuit, in Verilog hardware language. This netlist contains elements from the standard cell li-

brary employed. In our case, the architectures were synthesized for a 65 nm commercial standard-cell library from ST, at 1.0 V voltage supply Also, a file named Standard Delay Format (SDF) is generated, which contains the specific delays for gates and nets and the area, power and timing reports. A simulation tool, such as Cadence Incisive, simulates this generated netlist along with the testbench files implemented – which read the input data from the video codec. This process generates a dump file, which can be either in Value Change Dump (VCD) or Toggle Count Format (TCF) file formats.

Figure 4.1: Methodology used to obtain accurate power results for video applications.



Source: The author (submitted)

In addition to that, Cadence Genus Synthesis Solution (CADENCE, 2019) has a mode denoted as physically-aware layout estimation (PLE). This mode takes into account the gate interconnection penalties in terms of area, power and timing. This is performed by estimating the length of the circuit nets and taking the load capacitance effects into account for the power dissipation. PLE considers a relatively pessimistic layout routing estimation, however.

The analysis with PLE requires the use of Library Exchange Format (LEF) files, which contain the physical layout information of the library, essential for these interconnections to be taken into account. The LEF macro includes the capacitance of the inner library cell. The tech LEF comprises the process metal capacitance for the estimation of the interconnection capacitance. A capacitance table file (CapTBL) can also be made available by the library, which contains descriptions of the technology capacitances in a fine-grained way, by considering process variations.

4.4 SAD Input Vector Acquisition

We also needed to obtain inputs from the video sequences which would feed the architectures, to better estimate power results, using the methodology explained in the previous section. The inputs were obtained from the HM reference software. To obtain these, we analyzed the inputs of our architectures, disregarding their implementation. Considering that our architecture would calculate the distortion costs of SAD (and possibly MV_{Cost}), we needed to extract data of these values from the encoder. However, given that our input width was already previously defined as 8×1, we generated eight binary outputs of each candidate and original block, totalizing 16 output files, each of which contains data from one pixel of each respective block. The pixel values are 8-bit long.

The decision on using an 8×1 SAD architecture was reinforced by the work in (SILVEIRA et al., 2017), which also employed a SAD architecture with the same width. Moreover, every PU area defined in the HEVC process is divisible by eight, which implies that, with proper memory management, no hardware would be unused at any time, as there would not be remaining samples of less than 8 bytes to be fed to the circuit.

Besides extracting SAD and MV_{Cost} data, we also needed to extract data regarding the number of accumulations that would be required for a specific PU. This would allow the architecture to properly manage when the register should stop accumulating and start a new PU. This value was extracted by obtaining the PU area and dividing it by the chunk size (8, in this case), and the formula used is shown in Equation 4.1.

$$\#accumulations = \frac{(W \times H)}{input_W(bytes)} \tag{4.1}$$

Equation 4.1 is similar to Equation 2.5 apart from the initial number of cycles due to the pipeline latency, i.e., the terms W, H and $input_W$ all have the same meaning. The number defined in Equation 4.1 is simply the number of accumulations for any block, given its dimensions and the input width of the architecture, supposing that pipeline is full (or supposing there is no pipeline) and the operation is constantly running. Table 4.3 presents the number of accumulations required per PU partition of the HEVC process, considering an 8×1 SAD architecture.

It is important to mention that SMP and AMP partitions from Table 4.3 are not considered in the default preset of the x265. This encoder only considers 2N×2N partitions, whereas HM considers all the possible partitions.

After the SAD values from both candidate and original block, MV_{Cost} and PU

Table 4.3: Number of accumulations required to calculate SAD for every PU partition in HEVC, for an 8×1 SAD architecture.

PU	Area (pixels)	# Accumulations
64×64	4096	512
64×32	2048	256
32×64	2048	256
64×48	3072	384
48×64	3072	384
64×16	1024	128
16×64	1024	128
32×32	1024	128
32×16	512	64
16×32	512	64
32×24	768	96
24×32	768	96
32×8	256	32
8×32	256	32
16×16	256	32
16×8	128	16
8×16	128	16
16×12	192	24
12×16	192	24
16×4	64	8
4×16	64	8
8×8	64	8
8×4	32	4
4×8	32	4

Source: The author

accumulations were generated accordingly, we obtained the inputs, which would later be fed to the testbench of each respective architecture.

4.5 Description of the Architectures

After obtaining the cycles reduction results, which will be presented in the next chapters, we decided to implement the architectures for all the different proposals of our work. Every architecture was described using VHSIC Hardware Description Language (VHDL). The correspondent testbenches were described in Verilog. These testbenches are responsible for reading the input files from the SAD values, the MV_{Cost} values (when MV_{Cost} was employed) and the file that contains the number of chunks, so that the test-

bench knows when to stop accumulating the current block.

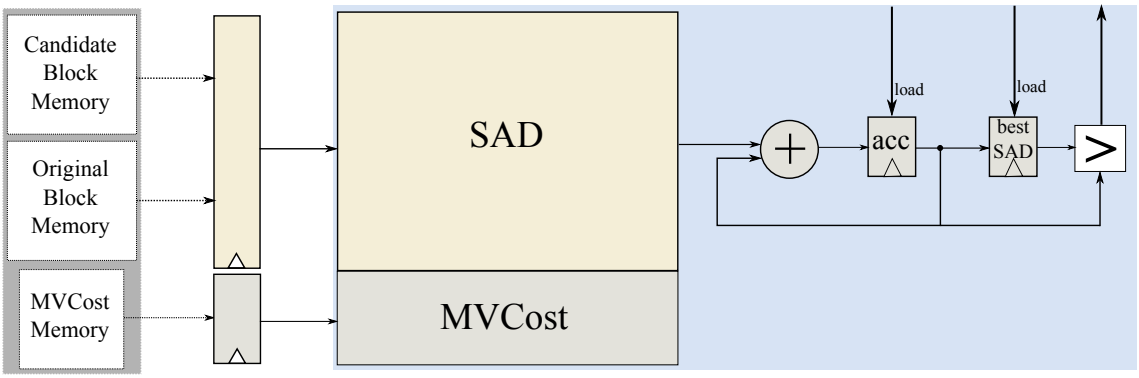
All the architectures descriptions were based on a default implementation of SAD using conventional adders for the tree. The implementation of the absolute operators and the subtractors was based on the results obtained from our previous work (ABREU et al., 2017).

Due to the fact that we were focused on obtaining power-aware results, the architectures were employed without pipeline. Additionally, due to the fact that we are dealing with data-dependent BMAs, i.e., both depend on data from previous block executions at each iteration, we opt not to employ pipeline, given that this would result in occasional stalls that could harm the circuit throughput.

This disadvantage of pipeline implementations can be exemplified if we consider a generic SAD architecture with full pipeline implementation, in an adder level (no pipeline is implemented inside the conventional adders), and considering the HS BMA. It was explained in Chapter 2 that HS evaluates six (or three) candidates around the current center, and if any of them has a smaller cost value than the current center, this will be the new center of the search for the next iteration. However, in order to reliably calculate the new center of the search, the last remaining SAD chunk of the last candidate has to be calculated, and therefore we would not know at that moment what the next candidate to be sent to the pipeline would be. Hence, pipeline stalls will happen between each iteration. This logic would also occur in the FiS and Refinement Search stages of TZS – RS would not present these issues given that it is not data-dependent.

Based on this decision, the only registers contained in the architecture are input registers, the accumulation register and the best SAD/cost register. A generic model of the architectures employed is shown in Figure 4.2. Depending on the model, MV_{Cost} may not be included.

Figure 4.2: Generic model of the architectures in this work.



Source: The author

5 ANALYSIS AND ARCHITECTURE DESIGN

5.1 Choice of Block-Matching Algorithm

Given that this work is focused on performing an optimization on SAD architectures within the IME execution, it is important to choose a BMA to be used as a case study. In order to decide the best one, we can consider the number of candidates evaluated – which consequently dictates the execution time – and the quality results of each one.

We decided to make a comparison between TZS and HS, which are the default BMAs employed in the two encoder softwares HM and x265, respectively. Both TZS and HM are prominent alternatives for implementation in IME architectures, so a preliminary analysis was performed in this work in order to assess the computational requirements of each one. A similar analysis was already performed for a previous work. However, that analysis compared both BMAs in their respective encoders. This consideration leads to an unfair comparison, given that a BMA may present much higher cycles results just because of the number of partitions evaluated by that encoder. Therefore, this analysis was performed in the same encoder software (x265) for both BMAs so that there was no interference from using different mode decisions in distinct encoders. For TZS, we used the adaptation implemented by the x265 software, SS, based on the algorithm coded in the HM software model. This adapted algorithm is very similar to the one in HM, as it presents the same TZS stages explained in the previous sections.

The results displayed in Table 5.1 show the number of 8×1 SAD calls required to encode one second of six Full-High Definition (FHD) (1080p) and six Ultra-High Definition (UHD) 4K (2160p) sequences using the TZS and HS algorithms, as well as the reduction obtained when using HS. This analysis was performed for the random access configuration of x265, and the default preset was considered for every other parameters apart from the IME algorithm. The 8×1 size was used to standardize the number of calculations from both algorithms, since counting only the total SAD operations performed would not consider the size of the blocks in each calculation. This size has also been used in several related works regarding SAD implementations. The cycles count was obtained for a QP of 32. Table 5.1 also shows the BD-BR decrease resulting from replacing the TZS for the HS. For this quality analysis, the results were run for QPs 22, 27, 32 and 37, as stated by the CTC.

Table 5.1: Number of 8×1 SAD calculations for different BMAs (QP=32) and BD-BR increase of the HS (using TZS as anchor).

		# SAD calculations (1s)			
Resol.	**Video**	**TZS** (10^6)	**HS** (10^6)	**Red.** (%)	**BD-BR** (%)
1080p	BasketballDrive	14.11	2.39	83.1	0.11
	BQTerrace	6.36	2.24	64.8	0.56
	Cactus	6.37	1.81	71.6	1.27
	Kimono	5.24	1.14	78.2	0.66
	ParkScene	1.81	0.82	54.7	0.55
	Tennis	9.32	1.27	86.4	4.1
	Average	7.2	1.61	73.13	1.21
2160p	Beauty	134.7	23.46	82.6	0.84
	Bosphorus	47.64	18.44	61.3	≈ 0
	HoneyBee	21.54	14.97	30.5	1.48
	Jockey	85.88	19.83	76.9	3.14
	ReadySteadyGo	69.09	18.74	72.9	1.46
	YachtRide	111.19	22.61	79.7	2.9
	Average	78.34	19.68	67.32	1.64

Source: Paper published by the author (ABREU et al., 2018)

As it can be observed from Table 5.1, even though the TZS method is much more costly in terms of SAD calculations than HS – about 4.5× and 4× for FHD and 4K videos, respectively –, the compression efficiency of using TZS is higher, surpassing 1.5%, on average for 4K videos. At first, we considered this a negligible BD-BR increase, and in the primary analysis on PDE without MV_{Cost}, we considered the x265 encoder.

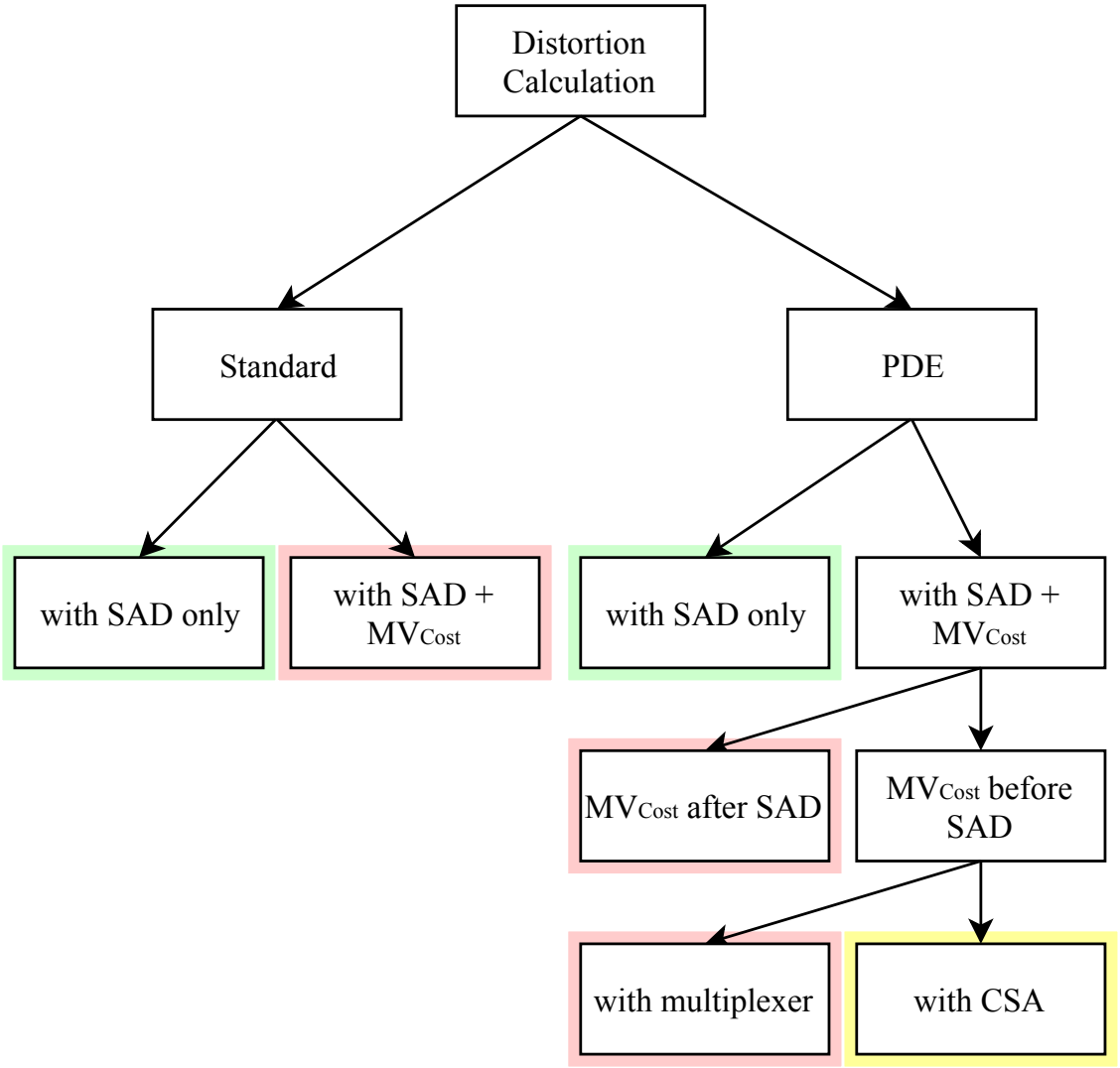
However, many members of video-coding scientific community consider this BD-BR increase not negligible, especially for applications concerned with coding efficiency. In addition, in order to maintain our analysis more faithful to the CTC, we decided to include the TZS algorithm in the remaining experiments, which is the default search method of the HM reference software. Therefore, in our latest approaches including MV_{Cost}, we perform the analysis for both HM and x265 softwares.

5.2 Partial Distortion Elimination Proposals

The following subsections will detail each one of the proposals from this work. A general flow chart of the possible solutions analyzed is presented in Figure 5.1. This flow is split into the Standard (non-PDE) implementations, which do not employ any PDE

solutions, containing the accumulations of SAD and of SAD with MV_{Cost}. The PDE solutions are also presented, including the already existent proposal of PDE with SAD, and the newly proposals of PDE with SAD and MV_{Cost}. Even though PDE with SAD has already been proposed, related works have only considered naïve BMA implementations, such as FS, and no data-dependent BMAs have been analyzed. We also analyzed the impacts of PDE with SAD in 4K video sequences, to deepen the already-existent proposal.

Figure 5.1: Flow chart of solutions for calculating distortion.



Source: The author

The solutions with the same colors in the flow chart from Figure 5.1 are implemented with similar architectures. For different colors, different architectural solutions are required. As it can be seen, the PDE architectures in fact are employed with similar components when compared to architectures without PDE. The main difference resides in the fact that the data from the next PUs will be requested to the memory module, which will act differently.

We begin by presenting the MV_{Cost}-Oblivious Implementations (both without and with PDE), considering only SAD. The remaining subsections present the MV_{Cost}-aware approaches.

5.2.1 MV_{Cost}-Oblivious Implementation

The first analysis performed disregarded the use of MV_{Cost} in the total distortion cost of a block being calculated in the IME stage. Similar analysis is already presented in some related works such as (CHOI; JEONG, 2009; SEIDEL; BRäSCHER; GüNTZEL, 2015), as mentioned in Chapter 3. However, only the work in (SEIDEL; BRäSCHER; GüNTZEL, 2015) presented any form of architecture implementation. Moreover, none of these works considered the use of more recent BMA proposals, and did not consider higher resolutions such as 4K, in this case.

The PDE technique, in this case, basically consists in what was already defined in Chapter 2. Due to how SAD is implemented, using conventional adders in an adder tree scheme, the smaller block chunks are accumulated at each cycle. This accumulation is stored in an accumulator register, which is compared to an overall SAD register, that contains the smallest SAD value of the whole IME iteration. This comparison is performed at every time. PDE consists in prematurely ending the SAD calculation of the current candidate w.r.t. to the block being encoded, when the accumulator register has a value higher than the best current SAD value. Therefore, the architecture would not need to keep calculating candidates whose SAD is already worse than the best current one, which incurs in the accumulation of unnecessary chunks.

This logic can be presented in the example from Table 5.2. This example considers the middle of an execution, with a best SAD register already with a value different from the initial one, and a PU of 8×8. This PU requires eight 8×1 SAD chunks to be accumulated, so that the entire SAD can be calculated.

In the beginning of the execution, the best SAD register is initialized as the maximum SAD value possible. This maximum value was determined from a worst case scenario, considering the largest block size of 64×64. Therefore, considering that every pixel difference from the two blocks being compared is maximum – for an 8-bit pixel –, then we have a maximum SAD value of $(2^8 - 1) \cdot 64 \cdot 64 = 1044480$. This value can be stored in a register of 20 bits. Therefore, we initialized the best SAD register with all 20 bits as '1' logic level.

Table 5.2: Example case of comparison between standard and PDE approaches of MV_{Cost}-Oblivion implementations, for an 8×8 PU.

Best SAD = 280, PU = 8×8			
Cycle	**SAD Chunk**	**Standard**	**PDE**
0	73	0	0
1	87	73	73
2	79	160	160
3	82	239	239
4	84	321	321
5	82	405	DONE
6	80	487	DONE
7	98	567	DONE
8	-	665	DONE
9	-	DONE	DONE

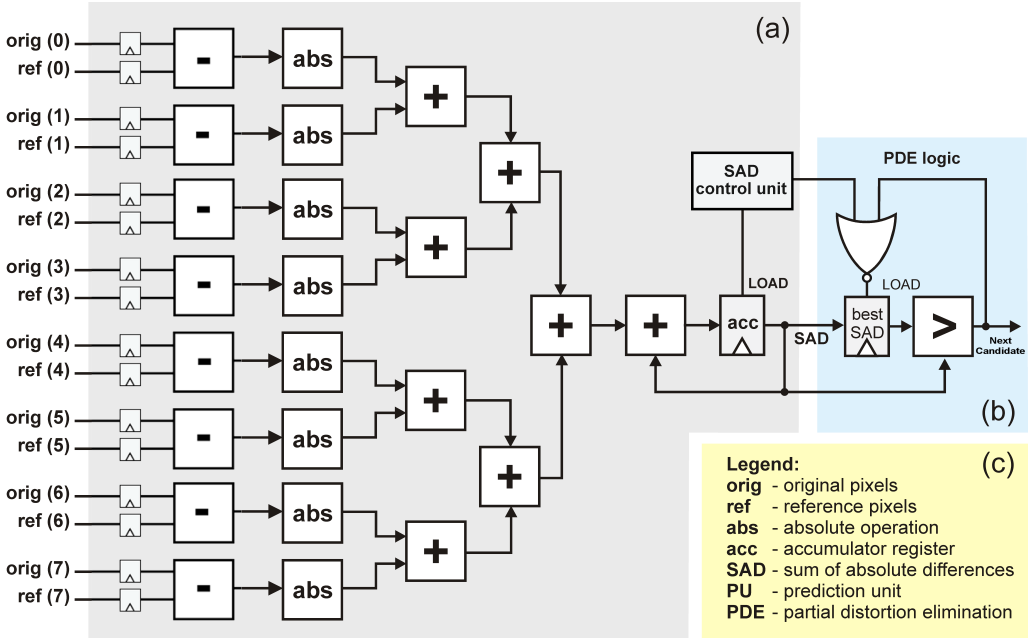
Source: The author

The architecture for this module is presented in Figure 5.2.

It is important to notice that no significant additional hardware resources are required to implement this module. This is because the comparator module is already part of well-established SAD architectures, given that this comparison is performed at the end of the SAD calculation. However, this PDE analysis performs this comparison in all cycles and sends output data to the memory controller indicating that the next PU should be retrieved.

This solution is more suitable for a hardware implementation of the IME module than for the encoder software, due to the fact that a conditional operation would be needed after each chunk is calculated, and it would be costly in terms of software performance. The algorithm for PDE is more clearly illustrated in Algorithm 1. However, in a hardware implementation, the comparator logic works in parallel with the SAD calculation itself, thus this solution becomes viable. In this algorithm, $bestSAD$ refers to the best current SAD which receives a value N from a current iteration, $currentSAD$ is the current SAD value being accumulated, $chunkSize$ is the size, in pixels, of a SAD chunk, $chunkNr$ is a counter that increments every time an absolute difference of a pixel is accumulated to $currentSAD$, and the value of $chunkNr$ is limited to the value of $chunkSize$; $isCandBetter$ is a variable that indicates whether the current candidate for which SAD is being calculated is a better one than the current best. Also, $nrColumns$ and $nrRows$ consist of the width and height of the PU, and $org[x][y]$ and $ref[x][y]$ refer to the pixel position inside the PU, for the original and reference (candidate) blocks,

Figure 5.2: (a) 8×1 SAD Architecture. (b) PDE logic module with an added NOR gate for the best SAD register load operation.



Source: Paper published by the author (ABREU et al., 2018).

respectively.

$bestSAD = N;$

$currentSAD = 0;$

$chunkSize = 8;$

$chunkNr = 1;$

$isCandBetter = 1;$

for ($x = 0;\ x < nrColumns;\ x + 1$ **) {**

  for ($y = 0;\ y < nrRows;\ y + 1$ **) {**

    $chunkNr+ = 1;$

    $currentSAD+ = abs(org[x][y] - ref[x][y]);$

    if $chunkNr == chunkSize$ **then**

      $chunkNr = 1;$

      if $currentSAD >= bestSAD$ **then**

        $isCandBetter = 0;$

      end

    end

  }

}

Algorithm 1: PDE in an MV_{Cost}-oblivious approach, if it was proposed in a software encoder.

When designing a solution with PDE, we have to account for the fact that, if we waste many cycles finding out that the current SAD being calculated is higher than the current best SAD, the next candidate is possibly already being sent to the architecture, so attempting to send a signal for the output to send new data would be useless. Due to the fact that there is an accumulator, which means one cycle is wasted before knowing whether the signal requesting data ahead of time should be sent to the output, we only achieve cycles reduction if we discover that the SAD being calculated is useless at least two cycles before, for an architecture without pipeline. Therefore, this solution is most suitable for energy-efficient architectures, due to the fact that the more pipeline levels we have, the sooner we will have to figure out whether the SAD is useless, making it difficult to achieve any effective reduction. Therefore, we implemented SAD with PDE for an architecture without pipeline.

Additionally, we have considered the input width of the architecture as 8 bytes (16 considering both blocks being compared), just as shown in Chapter 2 regarding SAD architectures. It is important to notice that using larger input widths would lead to higher circuit area and power dissipation. Moreover, we would have to accumulate partial SADs in larger chunks. Hence, there would be fewer cycles available for optimization when using PDE.

A script was employed to perform this cycles analysis, as described in Chapter 4. The results of this PDE approach will be presented in Chapter 6.

5.2.2 Compression impacts of disregarding MV_{Cost} in the total distortion cost

The issue of this primary analysis without the MV_{Cost} is that it is imprecise: simply extracting the SAD values from the encoder software without removing the MV_{Cost} from the whole process would result in inaccurate values. For example, if we extract a best SAD value of a specific iteration, and we obtain the cycles calculations without/with PDE of a given candidate, even though the results would be correct locally, they would not reflect what the encoder was actually executing. This is due to the fact that the encoder, after the calculation of the SAD value – outside of its function and outside of where data was being obtained – adds the value of the MV_{Cost} to the total cost of a block. These imprecisions could lead to another candidate being chosen different from the chosen candidate of our script analysis, that only considers SAD. This difference could escalate to different MVs being chosen, leading to different AMVPs being decided as initial search

points for the next blocks, and so on.

We also need to account for the fact that disregarding MV_{Cost} leads to a loss in compression efficiency when compared to the default case. Both HM and x265 encoders add the value of MV_{Cost} by default, in the IME stage.

Usually, works tend to consider only the SAD cost as part of a block total cost, which brings considerably less compression efficiency in the overall video sequence. An analysis was performed to compare the BD-BR increase of ignoring the MV_{Cost} in the total cost when compared to a full implementation that considers both SAD and MV_{Cost} in the total cost.

From this point, we decided to include the HM reference encoder – along with x265 – for the analyses and proposals, due to the compression , and to stay faithful to the CTC. The analysis considered the two variables described in Chapter 4: $useMVCost$ and $useMVCostFME$. We ran the configurations disabling both parameters, which disables MV_{Cost}, and enabling them, which is equivalent to the default encoder. This analysis was performed for both encoders, using the TZS and HS BMAs, for several video sequences, and the BD-BR percentages are presented in Table 5.3. The PSNR and bit-rate values were obtained from QPs 22, 27, 32 and 37. Also, for a more general analysis, we considered the runs for several video sequences from different resolutions.

Table 5.3: BD-BR percentages of encoding a video without considering the MV_{Cost} in the total distortion value.

		BD-BR	
Video		**HM**	**x265**
240p	BasketballPass	1.28%	0.9%
	RaceHorses	1.21%	0.3%
480p	BasketballDrill	0.82%	1.53%
	PartyScene	0.77%	0.31%
720p	FourPeople	0.16%	1.12%
	SlideEditing	0.41%	$\approx 0\%$
1080p	Cactus	0.55%	1.12%
	ParkScene	0.49%	0.99%

Source: The author (accepted for publication)

The results show that the difference in bitrate is not negligible, especially for applications concerned with compression efficiency. This degradation tends to be smaller for higher resolutions, in the case of HM. This may be due to the influence of the MV_{Cost} in the total distortion cost: given the tendency of larger PU blocks being chosen more often with a resolution increase, there is also a tendency for higher SAD values to be

achieved. Therefore, MV_{Cost} will not be as influential. The x265 encoder software presents a slightly different behavior, which may be due to different mode decisions, variations from the video sequences, or simply the tendency of its BMA being much more simple, resulting in different decisions being taken which may lead to blocks with different MV_{Cost} being chosen. Therefore, even though there are different behavior between both encoders, the compression efficiencies can be significant when certain types of applications are considered, so including MV_{Cost} in the total distortion cost is important.

5.2.3 MV_{Cost}-Aware Implementation

Given that loss in compression efficiency is present in the way SAD with PDE is calculated in the previous case (and in related works in general), we decided to propose an improved method of including MV_{Cost}, to increase the compression results.

The following subsections present a new proposal performed with PDE, which includes the use of the MV_{Cost} in the total distortion cost. Some details would change in the implementation: now, the register containing the best SAD value should be referred to as the best distortion register, given that it no longer contains the value of SAD only, as it also includes the MV_{Cost}. Some possibilities arise with the inclusion of MV_{Cost}: we can choose whether MV_{Cost} should be included before or after the SAD calculation. Different architectural ways of including the MV_{Cost} inside SAD modules are also explored.

The results for all these analyses and proposals will be included in Chapter 6.

5.2.3.1 MV_{Cost} accumulation after SAD

The first method of including the MV_{Cost} value is to treat it as an additional value to be accumulated in the accumulator register after SAD is entirely calculated. This is the trivial proposal, as it is in accordance to the implementation of the encoders, which accumulate MV_{Cost} after the SAD calculation, as presented in the pseudo-algorithm 2. The algorithm presents a generic case that illustrates both x265 and HM encoders, given that they both include MV_{Cost} after SAD calculation.

An example case of the behavior is presented in Table 5.4. From this analysis, we can see that the value of MV_{Cost} may not even be accumulated in the PDE case, as long as the sum of the SAD chunks already calculated is large enough to surpass the current best SAD + MV_{Cost} value stored in the best distortion register. The fact that MV_{Cost} is

```
bestCost = N;
currentCost = 0;
chunkSize = 8;
chunkNr = 1;
isCandBetter = 1;
for ( x = 0; x < nrColumns; x + 1 ) {
    for ( y = 0; y < nrRows; y + 1 ) {
        chunkNr+ = 1;
        currentCost+ = abs(org[x][y] − ref[x][y]);
        if chunkNr == chunkSize then
            chunkNr = 1;
            if currentCost >= bestCost then
                isCandBetter = 0;
            end
        end
    }
}
if isCandBetter! = 0 then
    currentCost+ = MV_Cost;
    if currentCost >= bestCost then
        isCandBetter = 0;
    end
end
```

Algorithm 2: PDE in an MV_{Cost}-aware approach, accumulating MV_{Cost} after the SAD chunks.

not included in every calculation does not bring any degradation, given that it must be accumulated only when the current SAD has not yet surpassed the best distortion.

5.2.3.2 MV_{Cost} accumulation before SAD

Another method can be employed, which is mainly based in including the MV_{Cost} before the SAD calculation, as opposed to the encoder software implementation. Even though this proposal may be intuitively equivalent to the other one, we have to consider the magnitude of the MV_{Cost} value when compared to a single SAD chunk. Primary analysis and extractions showed that the value of MV_{Cost} is usually higher than a single 8-byte SAD chunk; therefore, this may be used to obtain an optimization.

The main idea is: if we add MV_{Cost} before the rest of the SAD value, we can obtain a head start with a higher magnitude value than if we first accumulate the SAD chunks (as in the previous implementation). This may lead to less SAD chunks being required to be accumulated before finding out that the current candidate is a worse one. This is more convenient, especially because in this case, with the inclusion of MV_{Cost} –

Table 5.4: Example case of comparison between standard and PDE approach of a MV_{Cost}-Aware implementation, with accumulation after SAD, for an 8×8 PU.

Best Distortion Cost = 298, PU = 8×8			
Cycle	**Chunk**	**Standard**	**PDE**
0	78	0	0
1	83	78	78
2	87	161	161
3	93	248	248
4	77	341	341
5	83	418	DONE
6	83	501	DONE
7	88	584	DONE
8	215	672	DONE
9	-	887	DONE
10	-	DONE	DONE

● MV_{Cost} chunk.
Source: The author

as opposed to SAD-only implementations –, we also have a higher best distortion cost, so it is better if we start the accumulation with a higher value.

To illustrate it, we present an example in Table 5.5. It can be seen that the MV_{Cost} chunk arrives to be accumulated before the SAD chunks, which decreases the number of SAD chunks that need to be accumulated before finding out that the candidate is a worse one.

Based on this explanation, this logic will likely present better results than the one presented in the previous subsection. In order to implement this in a hardware architecture, two possible ways have been analyzed, both of which slightly differ in the logic of the accumulation: with a multiplexer and with a Carry-Save Adder (CSA), which will be presented in the following subsections.

Even though the changes between the two modes may seem insignificant, the number of times a SAD calculation is performed in the encoder is huge, and this stage shares a representative portion of the total encoding time and power, as shown in the previous chapters. Therefore, even though we are attempting to perform small modifications, these reductions will escalate and lead to significant results, which will be shown in Chapter 6.

(a) MV_{Cost} accumulation before SAD with Multiplexer

The architecture for this case is presented in Figure 5.3-a. In this case, the accumulation of MV_{Cost} occurs independently from the accumulation of the SAD chunks;

Table 5.5: Example case of comparison between standard and PDE approach of a MV_{Cost}-Aware implementation, with accumulation before SAD, for an 8×8 PU.

Best Distortion Cost = 298, PU = 8×8			
Cycle	**Chunk**	**Standard**	**PDE**
0	215	0	0
1	90	215	215
2	87	305	305
3	93	392	DONE
4	77	485	DONE
5	83	562	DONE
6	83	645	DONE
7	88	728	DONE
8	78	816	DONE
9	-	894	DONE
10	-	DONE	DONE

● MV_{Cost} chunk.
Source: The author

in other words, in a given distortion calculation, we first accumulate MV_{Cost} in the first operational cycle, and the next remaining cycles will accumulate the SAD chunks, accordingly.

This works through the logic explained in the previous paragraphs. It is important to notice that this architecture can also be employed in the case of the inclusion of MV_{Cost} after SAD. The only changes here are in the order in which data is received from the memory. Therefore, the power analysis in Chapter 6 will use this architecture for the approaches of MV_{Cost} after SAD and MV_{Cost} before SAD with multiplexer.

(b) MV_{Cost} accumulation before SAD with Carry-Save Adder

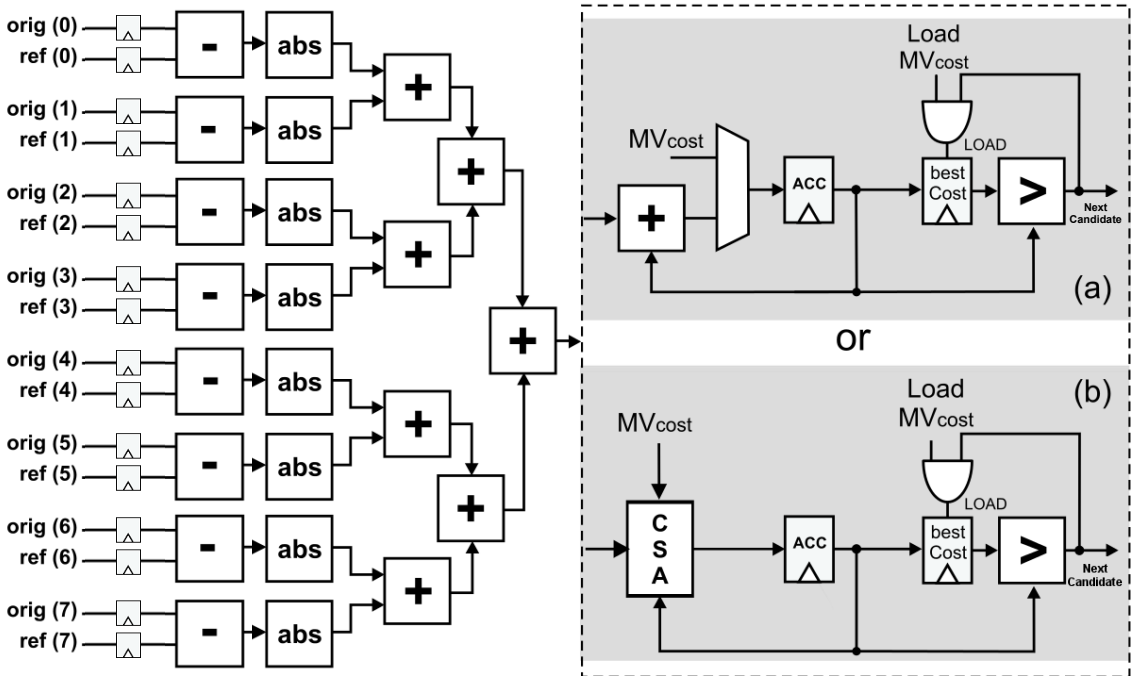
This case is an attempt to perform even better than the previous one. The use of a CSA to add more than two inputs is to include the MV_{Cost} along with the SAD tree. Even though this introduces a dependency context to the system, we are now able to add MV_{Cost} along with the first SAD chunk, which may give slightly better cycles results.

However, given our premise of adding MV_{Cost} along with the first SAD chunk, it may be the case that sometimes the SAD chunk is not ready in the output of the encoder to be accumulated along with it. This may happen when we consider previous cost calculations that have discovered late that the current result was worse than the best one. More specifically, the memory module, by default, is supposed to send the data of the new data blocks whenever the current one has been entirely sent. So, the $NextPU$ data may be

useless if it discovers too late that the current distortion is already worse, given that the memory module may already be sending the data. These cases may introduce an increase in the number of cycles, especially to this CSA mode, given that we have to wait for the first SAD chunk to have arrived to the input of the CSA, whereas the previous case with the multiplexer can add the MV_{Cost} independently.

The architecture for this case is presented in Figure 5.3-b. We made some analyses to find out the number of bits used in the represensation of MV_{Cost}, and the maximum number found was 11. Therefore, considering that the width of the adders increase by one in each level of the SAD tree, we replaced the normal adder of the third level of the tree, which is an 11-bit adder, for the CSA.

Figure 5.3: Architectures for the calculation of the total distortion cost, considering a MV_{Cost}-aware approach, accumulating MV_{Cost} before SAD with (a) a multiplexer or (b) a Carry-Save Adder.



Source: The author

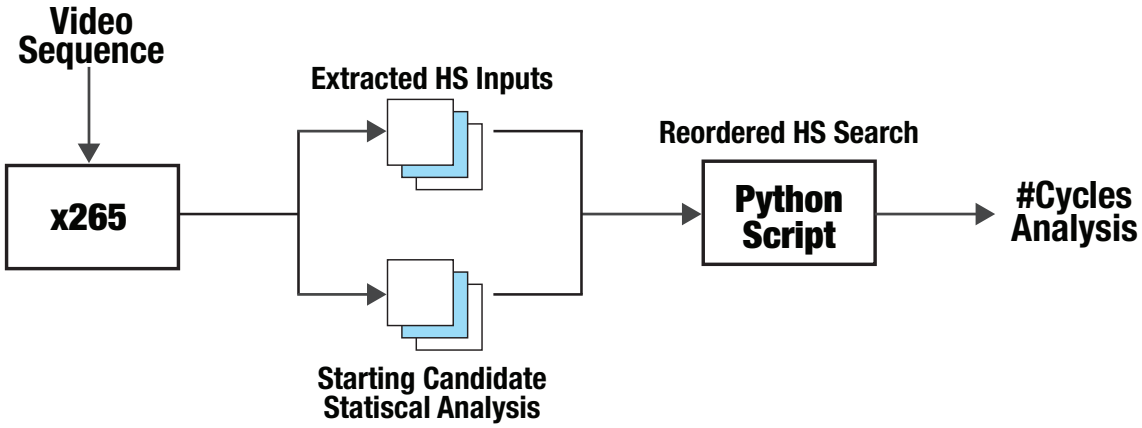
5.3 Candidate Reordering

PDE methods, in general, could be optimized even more, if we guaranteed that a very similar candidate was evaluated before less similar ones. For example, considering the HS BMA, we could change the order in which the six candidates are evaluated (reorder their SAD calculation), so that we evaluate the best ones first. This would be beneficial

because if we calculate candidates with smaller SAD values before, we may reach this SAD value in the calculation of the remaining ones before, hence saving cycles.

However, given that there is no way to know before-hand unless we actually performed the calculations, we decided to perform some candidate reordering based on occurrence probability. To do this, we considered an analysis with the HS BMA, and counted how many times each of the six candidates were the best in an overall execution, in each hexagon iteration. These counts were run for every FHD and 4K video sequence, for one second of execution of each video, and the average results were obtained, which gives us a candidate ordering to be performed when each hexagon iteration occurs. Therefore, we may possibly obtain better PDE reductions, given that the best candidates will likely be evaluated in the first iterations, and thus the other candidates may not calculate the entire SAD for their respective blocks. The analysis performed for the candidate reordering scheme is briefly presented in Figure 5.4.

Figure 5.4: Cycles analysis with candidate reordering approach.



Source: The author

The findings of this analysis will be shown in Chapter 6.

5.4 Memory Organization

Given that the ME module is focused on intensively gathering data from reference frames, and that memory communication can be a computation bottleneck in data-intensive operations, it is important to design a memory model to verify the feasibility of designed architectures.

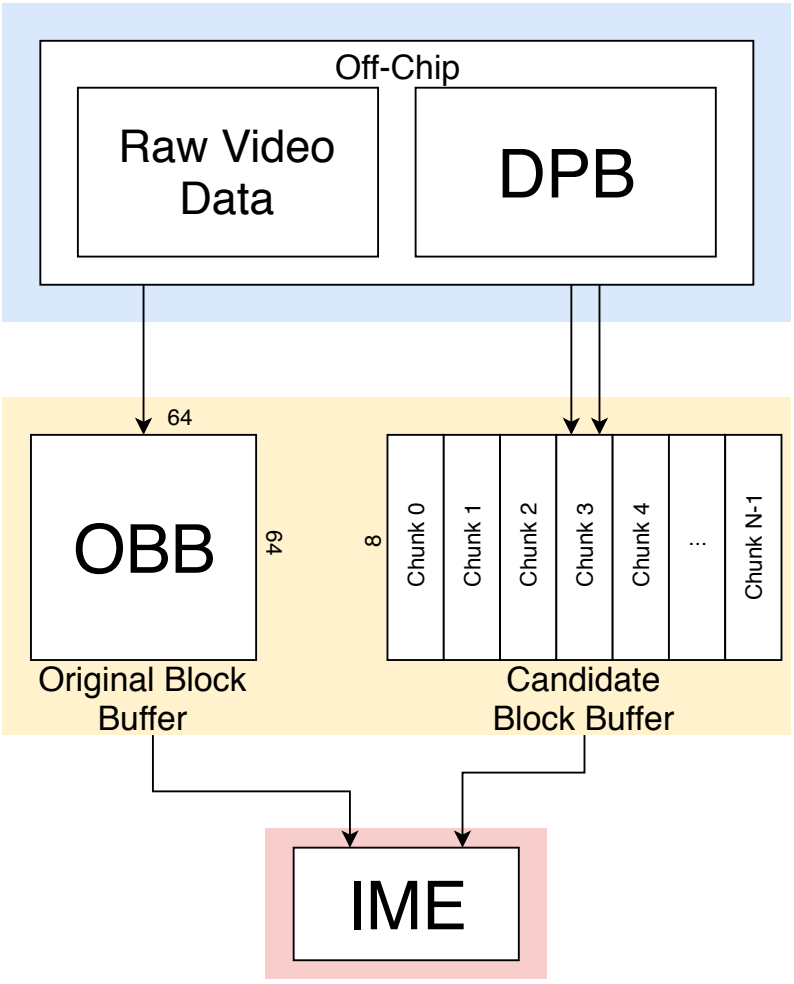
A precise model is highly dependent of the memory hierarchy in each system, but such in-depth analysis is out of the scope of this work. To simplify this discussion, the

following assumptions are considered:

- An off-chip memory contains data from the frame being encoded, and this memory is able to send data of the next CTU to be encoded to an on-chip buffer before the operation starts;
- An off-chip memory unit (possibly the same one mentioned above) contains data from the reference frame before the beginning of the current IME execution. This is commonly referred to as Decoded Picture Buffer (DPB).

In some cases, the DPB is also implemented as an on-chip memory, but this would not affect our estimates regardless. Based on the assumed off-chip memory, two local on-chip memories were idealized: the original and the candidates blocks buffers (OBB and CBB respectively). The generic model considering SAD is shown in Figure 5.5.

Figure 5.5: Memory model.



Source: The Author

The OBB contains the data from the CTU being currently encoded, which will be used for several IME executions (given that the 64×64 CTU can be interpreted as four

32×32 CU, and so on). Given that this value is fixed as 64×64, this module contains 64 lines of 64 bytes each. Data from this memory buffer will be sent to the SAD module in chunks of 8×1, according to what the encoder dictates.

The CBB was optimized in order to minimize its area. This module will contain data from the current SAD chunk being sent to the SAD module, and from the next SAD chunk of the same block. Therefore, two lines of 8 bytes each are used to store data of the current block. However, due to the PDE technique, which causes an early termination of the SAD operation, it is not exactly known when data from the next block will be required. Therefore, the next block to be calculated must also be stored speculatively to prevent execution stalls in such cases.

A second remark is that, due to the candidate reordering scheme, the next candidate to be computed may also vary, so we also need to store the first chunk of all the possible candidates to be calculated in advance.

To circumvent this drawback, the first chunk of every possible candidate in the next iteration must be pre-fetched before it starts. The number of pre-fetched chunks are different depending on whether HS or TZS is being used.

Due to the optimizations performed by x265, we know that three of the candidates are never evaluated, given that they have already been checked in the last iteration, due to the common points inherent to the hexagon shape. Therefore, only three candidates are possible, for each possible better candidate in the search. Before the execution of the last candidate, we already know whether any previous candidates have presented better SAD values than the current center. Therefore, in the worst case, we would need data from the three possible candidates of the current best one in that iteration and from the three possible candidates of the last iteration, whose SAD result has not yet been calculated. This leads to six lines which need to be stored before the calculation of the last candidate of that iteration.

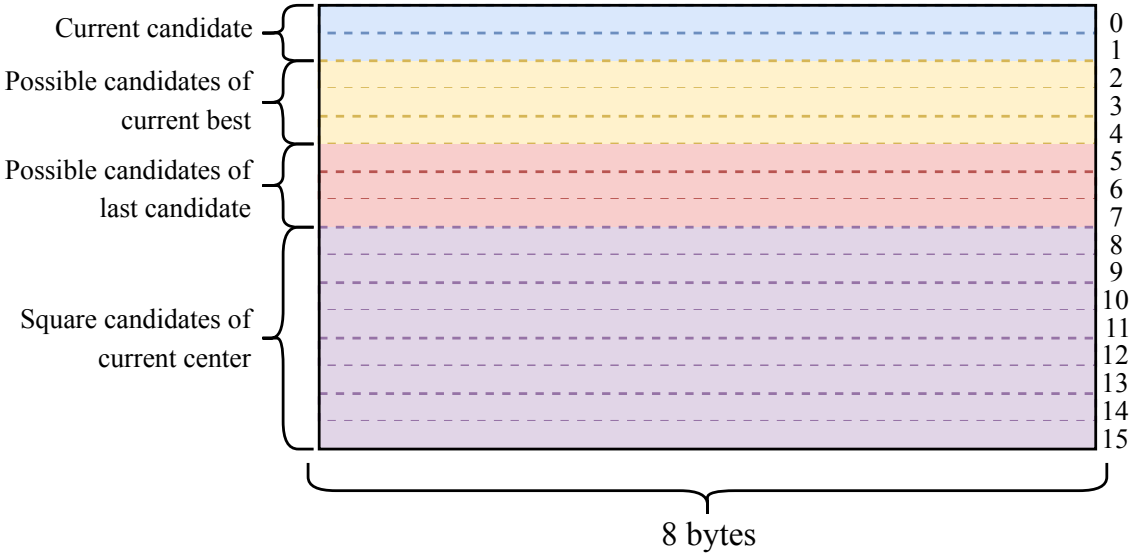
Additionally, it is also possible that no candidates are better than the current center, and the last one may not be either. This would require us to keep the data of the first chunk of each of the eight candidates of the Square Refinement step. Given that it only executes around the current center, we already know which candidate would be necessary. Considering these two possibilities, the memory requires 14 extra lines of 8 bytes each, apart from the two 8-byte lines of the current candidate being calculated.

The logic is the same when considering TZS. We should consider every possible step of this specific BMA. In FiS and Refinement Search, we should consider that,

depending on the iteration being performed (which depends on the $iDist$ value, as explained in Chapter 2), more candidates can be possibly evaluated in the next steps of the search. Given that no trivial optimizations are performed in evaluating less candidates in TZS like in HS, mainly due to the different shapes, we consider the worst case scenario of 16 candidates being evaluated in this case. Following the same logic of saving the 16 possible candidates of the best current one and the 16 possible candidates of the possible best one (which is the last candidate), we require 32 lines of 8 bytes each, for the diamond-shaped search steps. When considering that RS may be executed, we also leave one additional line of 8 bytes (given that the candidates in RS have fixed positions). Along with that, the two lines of the current candidate block being calculated are also required, resulting in a total of 35 lines of 8 bytes each.

In conclusion, the memory buffer required for the candidate block in TZS would be slightly larger than the one in HS, due to the higher number of possibilities. The proposed memory buffers for HS and TZS are shown in Figures 5.6 and 5.7, respectively.

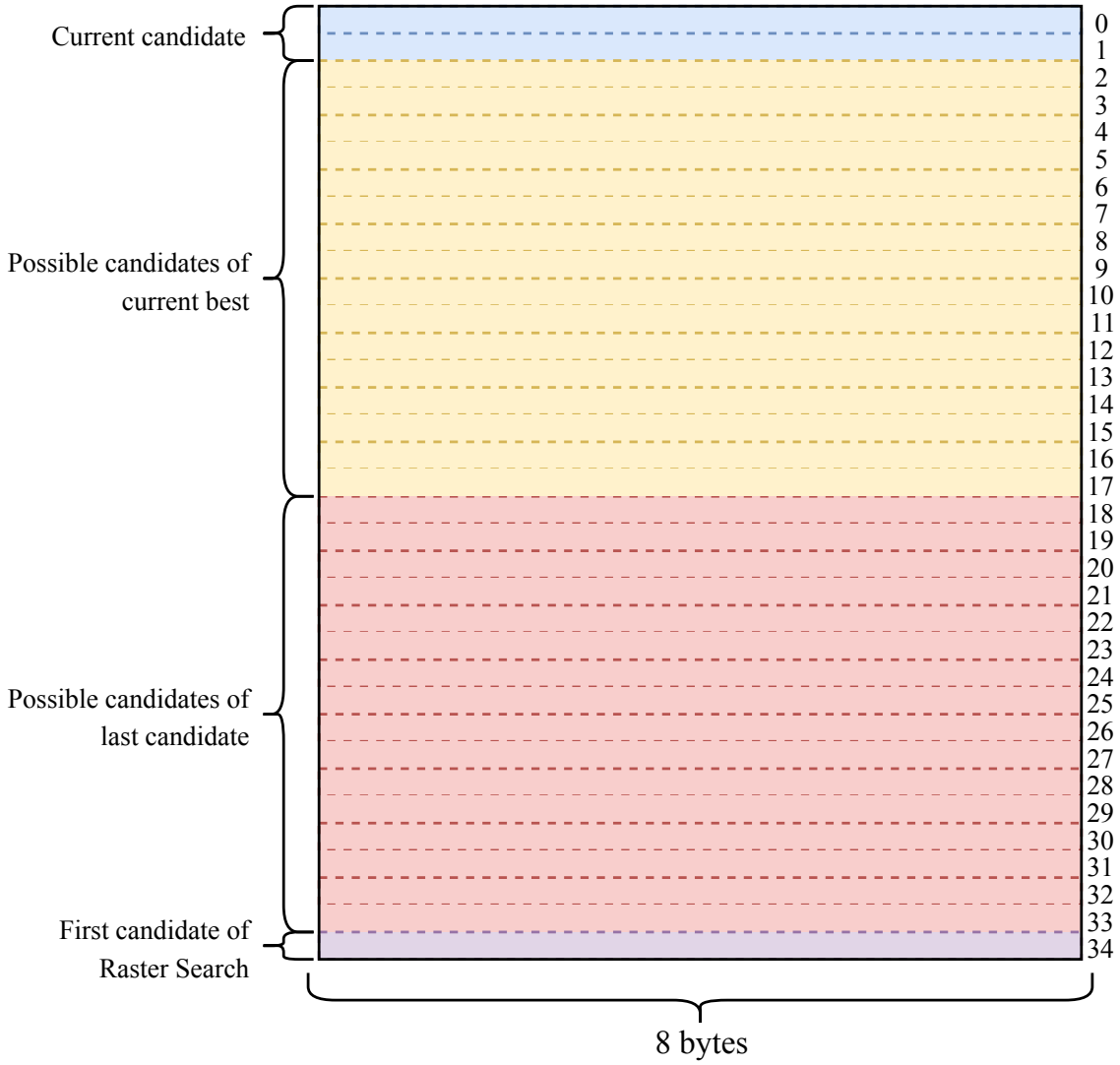
Figure 5.6: Proposed memory buffer for HS.



Source: The author

When taking the bus size into account, for both HS and TZS, the memory must provide at least 8 bytes per cycle (1 chunk per cycle). In a worst case scenario of the HS algorithm, when the last candidate of a specific iteration is being evaluated, we would require that all candidates of the current best, the possible candidates considering that the last candidate of the hexagon step would be chosen, and the candidates of the worst case scenario of the square refinement to be retrieved. Therefore, 14 lines of 8 bytes each for possible next candidates, totalizing 112 bytes, would be required for HS. Using

Figure 5.7: Proposed memory buffer for TZS.



Source: The author

the same logic for the worst case scenario in TZS, 264 bytes would be required. It is important to notice that this amount would only be required when we do not know which candidate of the next iteration would be evaluated first – because of candidate reordering that could change the order, depending on the iteration. Also, these values could be obtained throughout the whole execution of the current block being calculated, so the obtainment of the predicted chunks could be spread. Therefore, without applying any candidate reordering, we would only require 3 lines of 8 bytes each in HS – totalizing 24 bytes –, each of which are related to the first chunk of the hexagon step of the next possible iteration considering that the current best is the chosen one, the first chunk considering that the last candidate will be the chosen one, and the first line of the first square candidate. For TZS, in this scenario that we know the candidates order, we would require 3 lines of 8 bytes as well, for the next possible diamond step of the current best, the next possible

diamond step of the last candidate, and the first candidate of the raster search.

The memory containing MV_{Cost} values is also not an issue. This can be treated as the same external memories containing frame data, as the possible MV_{Cost} values are fixed and usually calculated prior to the execution of the IME stage. The x265 encoder already maintains the possible values calculated; therefore, a similar implementation will not bring latency issues.

6 RESULTS AND DISCUSSIONS

This chapter presents the results of the work developed in this research. We begin by presenting the results of the PDE proposals in terms of number of cycles. Next, we present the power and energy results of the architectures designed for the PDE proposals.

6.1 PDE Cycles Results

The cycles analysis was performed as presented in Chapters 4 and 5. The primary analysis of PDE without MV_{Cost} was obtained by extracting the inputs in the model previously presented in Table 4.2, for the HS BMA. The MV_{Cost}-aware analysis was performed by introducing the calculation of the cycles within each encoder, for both the HS and TZS BMAs.

6.1.1 MV_{Cost}-oblivious

This analysis was performed with x265. Before extracting the inputs to calculate the number of cycles of both no-PDE and PDE approaches, we decided to apply the candidate reordering scheme based on the analysis performed in HS. This analysis counted the number of best candidates in each of the HS iterations, and in the square refinement step. Then, we decided to reorder the SAD chunks in the model previously presented (the one from Table 4.2), based on the best candidates found.

The candidate reordering analysis is presented in Table 6.1. It was discovered that, for the medium preset of x265, HS performs, at most, 28 iterations in its hexagon stage. After this, the hexagon step reaches the borders of the search window or does not find any better candidates than the current center, which are the two conditions for it to stop. The candidates evaluated in the hexagon step are always the ones with coordinates $(-2, 0)$, $(-1, -2)$, $(-1, 2)$, $(1, -2)$, $(1, 2)$ and $(2, 0)$, w.r.t. to the current center of the search. The square refinement steps evaluates the candidates $(-1, 0)$, $(-1, -1)$, $(0, -1)$, $(1, -1)$, $(1, 0)$, $(1, 1)$, $(0, 1)$, $(-1, 1)$. The original order of the hexagon and square refinement step are presented in the first two lines of Table 6.1. The remaining lines show the reordering based on the average of all 1080p and all 2160p videos run for one second, for a QP of 32, for the first ten iterations of the hexagon step and for the square refinement (which has

only one iteration). The order presented is the average case of the videos.

Table 6.1: Reordered candidates using the HS BMA.

Iteration	Order
Hexagon Orig.	$(-2,0),(-1,-2),(-1,2),(1,-2),(1,2),(2,0)$
Square Orig.	$(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(1,1),(-1,1),(1,-1)$
Hexagon 1	$(2,0),(-2,0),(-1,2),(1,2),(-1,-2,),(1,-2)$
Hexagon 2	$(2,0),(-2,0),(1,2),(-1,2),(-1,-2,),(1,-2)$
Hexagon 3	$(2,0),(-2,0),(1,2),(-1,2),(-1,-2,),(1,-2)$
Hexagon 4	$(2,0),(-2,0),(1,2),(-1,2),(-1,-2,),(1,-2)$
Hexagon 5	$(2,0),(-2,0),(1,2),(-1,2),(-1,-2,),(1,-2)$
Hexagon 6	$(2,0),(-2,0),(1,2),(-1,-2),(-1,2,),(1,-2)$
Hexagon 7	$(2,0),(-2,0),(1,2),(-1,-2),(-1,2,),(1,-2)$
Hexagon 8	$(2,0),(-2,0),(1,2),(-1,-2),(-1,2,),(1,-2)$
Hexagon 9	$(-2,0),(2,0),(-1,-2),(-1,2),(1,-2,),(1,2)$
Hexagon 10	$(-2,0),(2,0),(-1,-2),(-1,2),(1,2,),(1,-2)$
⋮	⋮
Square	$(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(1,1),(-1,1),(1,-1)$

Source: The author

The inputs from the IME using the x265 software were extracted and used as input to a script that calculates the number of cycles wasted in the normal case (no PDE) and the optimized case. The results without candidate reordering are presented in Table 6.2. The simulations were performed for 30 frames of six FHD (1080p) and six UHD 4K (2160p) video sequences, considering the default preset of the x265 encoder software.

This analysis was performed with 4K videos given that this contribution had not yet been presented in the literature, and the newly results for higher resolution videos were never explored.

Based on the presented results, we obtained a reduction in the number of cycles of 16.41% and 11.64%, on average, for 1080p and 2160p videos, respectively. These results slight differ from related works that employ PDE like (CHOI; JEONG, 2009; SEIDEL; BRäSCHER; GüNTZEL, 2015). (CHOI; JEONG, 2009) obtained reductions of up to 45%, but the work uses the FS algorithm and QCIF (144p). Also, even though the percentages – for the average of 1080p videos – are lower than the 36.1% obtained by (SEIDEL; BRäSCHER; GüNTZEL, 2015), our work considers a more realistic scenario using a fast BMA and the more recent HEVC coding standard, which is different from H.264/AVC employed in (SEIDEL; BRäSCHER; GüNTZEL, 2015). Additionally, the solution in (SEIDEL; BRäSCHER; GüNTZEL, 2015) considers a SAD input width of four pixel comparisons, which is smaller than the size used in this work, and represents

Table 6.2: Number of cycles for the HEVC Hexagon Search algorithm execution using the baseline SAD and with PDE optimization.

		# Cycles ($\times 10^6$)		
Resolution	**Video**	**Baseline**	**w/ PDE**	**Reduction** (%)
1080p	BasketballDrive	986.53	850.07	13.83
	BQTerrace	733.18	482.27	34.22
	Cactus	773.94	603.98	21.96
	Kimono	1023.47	907.66	11.31
	ParkScene	722.71	579.98	19.75
	Tennis	1236.37	1153.62	6.69
	Average	912.7	762.93	16.41
2160p	Beauty	3850.09	3770.07	2.08
	Bosphorus	2938.8	2509.77	14.6
	HoneyBee	2419.54	2127.5	12.07
	Jockey	3230.77	2897.88	10.3
	ReadySteadyGo	2904.01	2246.27	22.65
	YachtRide	3702.43	3281.3	11.37
	Average	3174.27	2805.46	11.64

Source: The author (ABREU et al., 2018)

an advantage when predicting an useless candidate. However, it reduces the throughput in two times when compared with our eight pixel comparisons PDE.

Also, FS is unfeasible for real-time solutions with higher resolutions such as UHD, due to the high number of candidates evaluated. The better results obtained in that analysis may be explained due to the fact that highly different candidates are being evaluated, given that the FS evaluates every candidate in the search area. Recent BMA solutions, such as HS and TZS, tend to calculate SAD for candidates that have better chances of being similar, and do not perform an extensive search.

Therefore, the contributions in this first analysis are in the fact that the actual PDE reductions are not as good as it was considered, when popular and non-trivial BMAs are employed. Moreover, the slightly worse results from 1080p to 2160p are also noticeable. However, these results may be explained due to variability purposes of the video sequences, given that the 2160p videos are not from the recommended CTC, whereas the 1080p videos are.

The approach applying candidate reordering was run and analyzed, and its results are presented in Table 6.3. The PDE results varied depending on the video: some of them presented slightly worse results, and some optimized the number of cycles. On average,

this solution did not bring satisfactory results. This may be explained by the use of an average case between video sequences to obtain the best candidates of each step. Using such an approach ignores the individual characteristics of each video sequence, given that there is no pattern in which movement is occurring between several videos. Moreover, we considered the average of the iterations in an isolated manner, which also may have contributed to the worse results. In other words, the order of the best candidates of one iteration likely depends on the previous best candidates of other iterations, given that there is no tendency that a search may lead to a certain direction and go to the opposite one in another iteration. However, considering these approaches would implicate in higher complexity, given that a tree of decisions would be required, and the ordering would depend on the previous best ones, we decided not to use candidate reordering for the remaining analyses.

Table 6.3: Number of cycles for the HEVC Hexagon Search algorithm with PDE and candidate reordering.

		# Cycles ($\times 10^6$)		
Resolution	**Video**	**w/ PDE**	**w/ PDE and Reordering**	**Reduction (%)**
1080p	BasketballDrive	850.07	850.24	-0.02
	BQTerrace	482.27	482.51	-0.05
	Cactus	603.98	603.94	0.01
	Kimono	907.66	907.14	0.06
	ParkScene	579.98	580.19	-0.04
	Tennis	1153.62	1151.85	0.15
	Average	762.93	762.64	0.02
2160p	Beauty	3770.07	3769.91	≈ 0
	Bosphorus	2509.77	2510.73	-0.04
	HoneyBee	2127.5	2127.72	-0.01
	Jockey	2897.88	2895.42	0.08
	ReadySteadyGo	2246.27	2239.05	0.32
	YachtRide	3281.3	3279.33	0.06
	Average	2805.46	2803.69	0.07

Source: The author

With that mentioned, a promising solution to deal with a generic case in video sequences for candidate reordering is to apply machine learning techniques. This will be explored in future works.

6.1.2 MV_{Cost}-aware

For the MV_{Cost}-aware approach, we performed an analysis to obtain the number of cycles (within both HM and x265 encoders) required for each of the presented cases:

- Without PDE (a);
- With PDE, using a multiplexer:
  - Accumulating MV_{Cost} after the SAD chunks (based on the encoder software normal execution flow) (b);
  - Accumulating MV_{Cost} before the SAD chunks (c);
- With PDE, accumulating MV_{Cost} along with the first SAD chunk, using a CSA (d).

For this analysis, we used both HM and x265 softwares, with the TZS and HS algorithms, and ran the simulations for 30 frames of several video sequences for different resolutions, for a QP of 32. We also considered the SAD architectures without pipeline for this analysis. The cycles results are shown in Table 6.4 for HM, and in Table 6.5 for x265, as well as cycles reductions for the versions presented.

Table 6.4: Cycles requirements of different approaches of including MV_{Cost}, using HM.

		# Cycles ($\times 10^9$)				**Red. (%)**	
Video		**(a)**	**(b)**	**(c)**	**(d)**	**(c) w.r.t. (b)**	**(d) w.r.t. (c)**
240p	BBPass	1.06	0.79	0.65	0.65	17.72	0
	RaceHorses	2.22	1.55	1.37	1.36	11.61	0.73
480p	BBDrill	6.55	4.79	4.05	4.07	15.45	-0.5
	PartyScene	6	4.26	3.84	3.83	9.86	0.26
720p	FourPeople	8.96	6.66	4.91	5.02	26.27	-2.24
	SlideEditing	10.1	5.67	4.14	4.28	26.98	-3.38
1080p	Cactus	28.78	21.34	17.86	17.97	16.31	-0.62
	ParkScene	24.3	19.73	16.59	16.65	15.91	-0.36
					Avg.	17.51	-0.76

Source: The author (Accepted for publication)

Cycles reductions with respect to the version without PDE (a) are already known by the literature in works such as (CHOI; JEONG, 2009; LEE et al., 2009; SEIDEL; BRäSCHER; GüNTZEL, 2015; ABREU et al., 2018), so these reductions are not summarized in Table 6.9. All versions reduce cycles when compared to the baseline case (a). When comparing (c) – MV_{Cost} accumulated before SAD – w.r.t. (b), which accumulates

Table 6.5: Cycles requirements of different approaches of including MV_{Cost}, using x265.

		# Cycles ($\times 10^6$)				Red. (%)	
Video		**(a)**	**(b)**	**(c)**	**(d)**	**(c) w.r.t. (b)**	**(d) w.r.t. (c)**
240p	BBPass	34.55	19.73	18.66	18.58	5.42	0.43
	RaceHorses	48.4	39.62	38.24	37.45	3.48	2.07
480p	BBDrill	173.57	127.48	122.74	121.15	3.72	1.3
	PartyScene	162.81	108.92	106.21	104.9	2.49	1.23
720p	FourPeople	286.11	174.23	165.61	164.68	4.95	0.56
	SlideEditing	285.03	100.32	86.6	95.91	13.68	-10.75
1080p	Cactus	813.17	618.94	600.12	593.27	3.04	1.14
	ParkScene	759.02	596.52	575.19	570.42	3.58	0.83
					Avg.	5.05	-0.41

Source: The author

MV_{Cost} after SAD, we obtain an additional average reduction of 17.51% in HM and a reduction of 5.05% in x265, which indicates that accumulating MV_{Cost} before the SAD chunks – as opposed to the normal flow of the encoder software – results in a reduced number of cycles. When comparing the version with CSA (d) with respect to (c), we found out that (d) has an increased number of cycles of, on average, 0.76% and 0.41% for HM and x265, respectively. This happens because, even though version with CSA accumulates MV_{Cost} along with the first SAD chunk, it has the disadvantage of having to wait for the first chunk to arrive before accumulating it, while (c) can accumulate MV_{Cost} before the chunks arrive.

The high reduction differences between encoders has two main reasons. The first one is that the mode decisions of both encoders are different, which may slightly differ the results. The second one is the most significant one, and it is mainly the fact that the x265 encoder evaluates less candidates than HM, given that HS has less complex stages than TZS. This leads to TZS evaluating several distant candidates with small similarities, which may also lead to an effective head-start when considering MV_{Cost}.

6.2 Determination of the target frequency

In order to define the target frequency for the architectures to be run, we opted to simulate a real-case scenario, for illustrative purposes – given that the main contributions of this work are in the number of cycles, which are independent from the frequency.

Therefore, this does not mean that the architecture will necessarily be run in that frequency, as it depends on the requirements of the whole encoder. However, considering a realistic case will lead to an accurate behavior of the architectures. We decided to obtain a target frequency from an analysis performed in our previous work in (SILVEIRA et al., 2017). The analysis performed in that work is briefly presented here, for clarification purposes and given that it justifies the use of the target frequency.

This analysis was conducted with the x265 software, given that it is the closest one to a real-time scenario. This consisted in counting the number of 8×1 SAD computations in several different configurations of x265, using the HS BMA. The objective of this analysis is to obtain the best configuration of the encoder software, with the most feasible frequency estimation results (which is equivalent to the target frequency) while attempting not to degradate the compression efficiency. Due to the high amount of different parameters that can be varied in x265 and given the time that would require to run the combinations of all parameters, the configurations were run for a subset of the parameters only. The parameters varied are shown below:

- **Reference frames**: the number of reference frames to be used in the ME stage;
- **Early Skip**: this parameter breaks the execution of the CTU tree before it is completely analyzed, given specific circumstances. This impacts mainly the encoding time;
- **Maximum Iterations**: this parameter is a custom one, which early-terminates the Hexagon stage of HS before a number of iterations specified by this parameters's value;
- **Test Local Motion Vector Predictors (MVPs)**: custom parameter, which tests whether local MVPs in the beginning of the search, such as median and neighbor MVPs, will be compared to the start point (which comes from the result of AMVP) to begin the search in some other point;
- **Square Refine**: this is also a custom parameter, which decides whether the square refinement stage will be executed or not.

Based on multiple runs of the encoder, three configurations stood out as the best, which are shown in Table 6.6. The c0 configuration is the default implementation of the encoder.

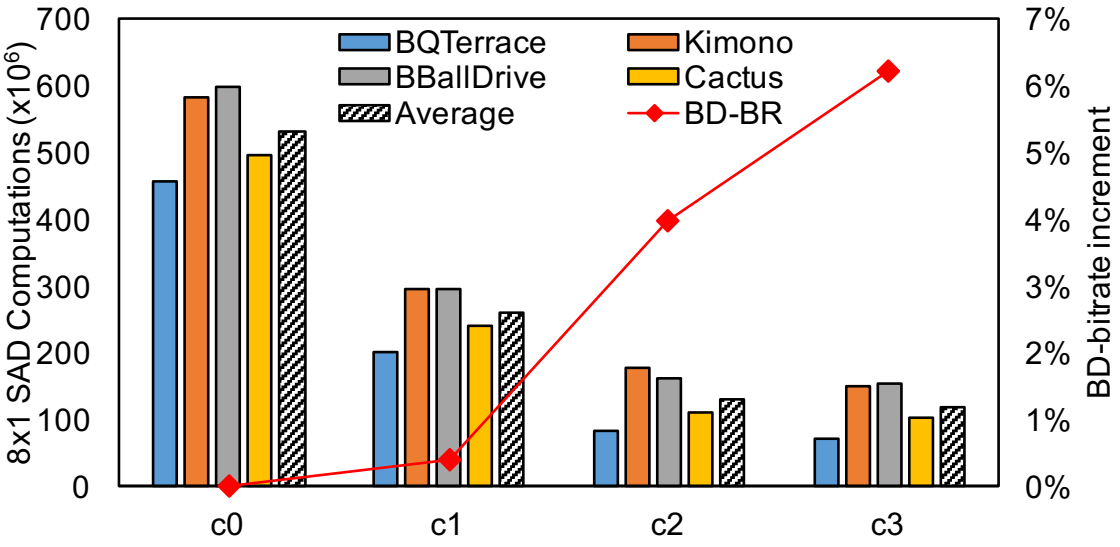
The throughput and BD-BR results of this analysis, for these four configurations, are shown in Figure 6.1.

Table 6.6: Configurations used to achieve different SAD computation targets.

	x265 parameters				
Config. Index	Reference Frames	Early Skip	Max Iter.	Test Local MVP	Square Refine
c0	3	D	–	E	E
c1	3	D	6	D	D
c2	3	E	3	E	D
c3	2	E	15	E	D

Source: (SILVEIRA et al., 2017)

Figure 6.1: Throughput analysis performed in (SILVEIRA et al., 2017).



Source: (SILVEIRA et al., 2017)

Based on Figure 6.1, the default configuration c0, although it does not present quality degradation, requires a high target frequency. On the other hand, c1 requires almost half the frequency in c0, with a slight increase in BD-BR. Configurations c2 and c3 present high BD-BR increase, with an additional decrease in the required frequency.

For both MV_{Cost}-oblivious and MV_{Cost}-aware analyses, we used the configuration from c1, and employed a target frequency of 300 MHz, given that it presents the most acceptable frequency value with a quality drop of 0.4%.

6.3 Power and Energy Results

6.3.1 MV_{Cost}-oblivious

The SAD architecture of this case – which was previously presented in Figure 5.2 – was described in VHDL and synthesized to low-power 65 nm commercial standard-cell library from ST at 1.0 V voltage supply, using Cadence Genus Synthesis Tool (CADENCE, 2019). The synthesis was run, as mentioned, for a 300 MHz operating frequency target, using real-input vectors from the aforementioned benchmark video sequences and used as stimuli to obtain more accurate power estimation results.

Table 6.7 shows the leakage (leak.), dynamic (dyn.) and total power dissipation and the energy consumption per operation results for an 8×1 SAD hardware architecture.

Table 6.7: Power and energy/operation synthesis results for the 8×1 SAD architecture @ 300MHz.

	Video	Power Dissipation (µW) Leak.	Dyn.	Total	Energy per Op. (pJ)
1080p	BasketballDrive	3.31	699.10	702.41	2.34
	BQTerrace	3.31	705.27	708.59	2.36
	Tennis	3.31	701.06	704.37	2.35
	Cactus	3.31	704.73	708.04	2.36
	Kimono	3.31	701.04	704.35	2.35
	ParkScene	3.31	708.04	711.34	2.37
2160p	HoneyBee	3.31	701.74	705.05	2.35
	Bosphorus	3.31	698.56	701.87	2.34
	ReadySteadyGo	3.31	709.74	713.05	2.38
	Jockey	3.31	697.69	701.00	2.34
	YachtRide	3.31	716.57	719.88	2.40
	Average	3.31	703.96	707.27	2.36

Source: The author (ABREU et al., 2018)

Finally, based on the average energy consumption per operation and the average number of cycles for both 1080p and 2160p video sequences, we estimate the total energy consumption of each solution, which is presented in Table 6.8. Based on the results presented, we obtained a reduction in the number of cycles (and energy) of 16.41% and 11.64% on average, for 1080p and 2160p videos, respectively, which are equivalent to the cycles reductions, in magnitude.

Table 6.8: SAD Average energy results for processing 30 frames of FHD and UHD video sequences using x265.

	Energy Consumption		
Resolution	**Baseline (mJ)**	**w/ PDE (mJ)**	**Savings (%)**
1080p	2.15	1.8	16.4
2160p	7.49	6.62	11.6

Source: The author (ABREU et al., 2018)

6.3.2 MV_{Cost}-aware

For the MV_{Cost}-aware approach, the two distortion architectures from Figure 5.3 were also described in VHDL and synthesized to a low-power 65 nm commercial standard-cell library from ST at 1.0 V voltage supply, at 300 MHz, based on the same presented analysis. The synthesis was also performed for ASIC design flow using Cadence Genus Synthesis Tool (CADENCE, 2019), and real-input vectors were also considered for the power extraction.

Table 6.9 shows the power dissipation results for the two 8×1 SAD architectures, with multiplexer and with CSA, as well as the energy reduction of comparing (d) with (c), for both HM and x265. Reductions of (c) w.r.t. (b) have not been presented given that both versions use the same architecture (the one with the multiplexers), so the energy reductions are equal, in magnitude, to the cycle reductions presented in the same Table. The energy reduction results have shown that, even though (c) beats (d) in cycles, the architecture with CSA (which was proposed for the predictions from (d)) dissipates slightly less power, resulting in an average reduction of 1.94% and 2.27% of (d) w.r.t. (c), in terms of energy for HM and x265, respectively.

Table 6.9: Power results and energy reductions of the proposed MV_{Cost}-aware architectures @ 300 MHz.

		Power (nW)						Energy	
		Mux			CSA			Reduction (%)	
	Video	**Leak.**	**Dyn.**	**Total**	**Leak.**	**Dyn.**	**Total**	**(d)-(c)[1]**	**(d)-(c)[2]**
240p	BBPass	10.86	1364.56	1375.42	10.35	1343.17	1353.51	1.68	2.01
	RaceHorses	10.86	1397.26	1408.12	10.36	1377.8	1388.16	2.18	3.45
480p	BBDrill	10.84	1370.28	1381.12	10.35	1326.37	1336.72	2.53	4.47
	PartyScene	10.87	1447.03	1457.9	10.36	1409.16	1419.52	2.89	3.83
720p	FourPeople	10.87	1023.31	1034.18	10.34	978.53	988.87	2.24	4.92
	SlideEditing	10.9	1249.81	1260.71	10.39	1254.97	1265.37	-3.79	-11.16
1080p	Cactus	10.87	1234.19	1245.06	10.39	1198.78	1209.16	2.28	3.99
	ParkScene	10.9	1329.03	1339.92	10.41	1250.8	1261.21	5.53	6.65
	Avg.	10.87	1340.15	1351.02	10.38	1301.48	1311.86	1.94	2.27

[1] reduction of version (c) w.r.t. to (d) in HM

[2] reduction of version (c) w.r.t. to (d) in x265

Source: The author

7 CONCLUSION

This work presented several analyses of components related to one of the most time and energy-consuming tasks in the HEVC video coding process: the IME. This dissertation focused on the core of IME architectures, which is the SAD module.

First, basic video coding and power dissipation concepts were introduced and detailed, to ease the description of the content of the remaining sections. This background chapter focused on the inter-prediction stage of the video encoding process in HEVC, which contains the ME stage.

A detailed related works chapter was presented, giving an overview of literature works regarding video coding. The summarization of the related works was divided in SAD/SATD, IME/ME, and PDE. Under a more detailed analysis, the author pointed out that most of the previous works did not present any power results or corresponding architectures to their proposals. Also, some of the works did not consider state-of-the-art search algorithms to perform their analyses, and did not take the MV_{Cost} into account, which harms compression efficiency results.

Then, the methodology chapter presented the main considerations taken for obtaining the desired results for the analyses and proposals, which were presented in Chapter 5.

The first analysis presented was the choice of BMA to be used in this work, as a study case. For our primary analysis, we decided to use HS, due to its smaller number of SAD calculations. However, a different route was taken in the MV_{Cost}-aware proposals, taking quality into account with a higher degree. Therefore, given that BD-BR results of this analysis presented a BD-BR increase of up to 4.1% of using HS in comparison with TZS, we decided to consider the use of TZS. So, the MV_{Cost}-aware analysis included the use of both HS and TZS, running the simulations for both x265 and HM encoders.

The PDE proposals were then presented. Two main approaches were analyzed and described in Chapter 5: MV_{Cost}-oblivious and MV_{Cost}-aware, both of which can be employed with or without the use of PDE. They mainly differ in the resulting compression efficiency, given that considering MV_{Cost} may lead to reduced BD-BR results – this analysis was also presented in Chapter 5. Inside the scope of MV_{Cost}-aware proposals with PDE, we have analyzed three possibilities: MV_{Cost} accumulated after SAD, and MV_{Cost} accumulated before SAD with a multiplexer or with a CSA. Each of these approaches have their own advantages and limitations. Several architectures were implemented for

each of these models, considering an 8×1 SAD module.

A candidate reordering scheme was also employed, in order to increase the benefits of the PDE technique. In addition to that, the memory was modeled, considering the high dependency of the analyzed BMAs on previous blocks.

In the results chapter, we presented cycles results of all the PDE approaches, performing comparisons between them and with the non-PDE method. For the MV_{Cost}-oblivious methods, PDE obtained a reduction in the number of cycles of, on average, 16.41% and 11.64%, for FHD (1080p) and UHD 4K (2160p) video sequences, respectively, when compared to non-PDE implementation. When taking MV_{Cost} into account, we compared the versions that accumulate the MV_{Cost} before SAD with the one that accumulates it after (which is the normal flow of the encoder software). We found that we obtained additional average reductions of 17.51% and 5.05% with the multiplexer version, respectively for both HM and x265, besides the reductions already established for the normal use of the PDE. These results were slightly better (0.76% and 0.41% less cycles, for HM and x265, respectively) than the version with CSA.

In order to have most accurate power and energy estimates for the developed hardware, the architectures were synthesized for 65 nm CMOS ST cell-library using real input vectors. The results for the MV_{Cost}-oblivious approach presented the same reductions, given that both present the same SAD datapath architecture. In the MV_{Cost}-aware results, we found out that, even though the CSA approach presents slightly worse results in terms of cycles, it beats the multiplexer version energetically, given that its respective architecture presents reduced power when compared to the multiplexer version. Therefore, the CSA version was the best in terms of energy, with average reductions of 1.94% and 2.27% for HM and x265, respectively, besides the normal energy reductions of PDE.

To conclude, new proposals that optimize the number of cycles required to calculate the distortion of blocks in the ME – which is the most time-consuming step in the video encoding process – were presented in this dissertation. These contributions are important to alleviate the bottleneck that this stage represents. As future work, it is intended to introduce other concepts, such as the use of machine learning, to early-terminate the ME stage based on features obtained before this stage, using decision support tools, e.g., decision trees and random forests.

REFERENCES

ABREU, B. et al. Exploiting absolute arithmetic for power-efficient sum of absolute differences. In: **2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2017. p. 522–525.

ABREU, B. et al. Exploiting Partial Distortion Elimination in the Sum of Absolute Differences for Energy-Efficient HEVC Integer Motion Estimation. In: **2018 31st Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2018. p. 1–6.

AGOSTINI, L. V. **Desenvolvimento de Arquiteturas de Alto Desempenho dedicadas à compressão de vídeo segundo o Padrão H.264/AVC**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2007.

BAHARI, A.; ARSLAN, T.; ERDOGAN, A. T. Low-Power H.264 Video Compression Architectures for Mobile Communication. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 19, n. 9, p. 1251–1261, Sep. 2009. ISSN 1051-8215.

BJONTEGAARD, G. Calculation of average PSNR differences between RD-curves. In: **Doc. VCEG-M33 ITU-T Q6/16**. [S.l.: s.n.], 2001.

BOSSEN, F. et al. HEVC Complexity and Implementation Analysis. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1685–1696, Dec 2012. ISSN 1051-8215.

BROSS, B.; CHEN, J.; LIU, S. **Versatile Video Coding**. Liubliana, Slovenia: [s.n.], 2018.

BUBOLZ, T. L. A. Decisões Rápidas de Estruturas de Particionamento na Transcodificação de Vídeo Homogênea no Padrão HEVC. In: **Trabalho de Bacharelado - UFPEL**. [S.l.: s.n.], 2018.

CADENCE. **Cadence EDA tools**. 2019. Available from Internet: <https://www.cadence.com/>.

CASAL, J. **Homer HEVC**. 2019. Available from Internet: <http://homerhevc.com/>.

CHANDRAKASAN, A. P.; BRODERSEN, R. **Low Power Digital CMOS Design**. [S.l.]: Kuwer Academic Publishers, 1995.

CHEN, C.-Y. et al. Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 53, n. 3, p. 578–593, March 2006.

CHI, C. C. et al. Parallel Scalability and Efficiency of HEVC Parallelization Approaches. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1827–1838, Dec 2012. ISSN 1051-8215.

CHIU, M.; SIU, W. New results on exhaustive search algorithm for motion estimation using adaptive partial distortion search and successive elimination algorithm. In: **2006 IEEE International Symposium on Circuits and Systems**. [S.l.: s.n.], 2006. p. 4 pp.–3981. ISSN 0271-4302.

CHOI, C.; JEONG, J. New sorting-based partial distortion elimination algorithm for fast optimal motion estimation. **IEEE Transactions on Consumer Electronics**, v. 55, n. 4, p. 2335–2340, November 2009. ISSN 0098-3063.

CISCO. Cisco Visual Networking Index: Forecast and Trends, 2017-2022. February 2019. Available from Internet: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf>.

Costa, Eduardo Antonio César da. **Operadores Aritméticos de Baixo Consumo para Arquiteturas de Circuitos DSP**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2002.

CRISTANI, C. R. Investigação da Estimação de Movimento para o Novo Codificador de Vídeo HEVC em Vídeos de Ultra Alta Definição. In: **Trabalho de Bacharelado - UFPEL**. [S.l.: s.n.], 2014.

DANG, T. L. D.; CHANG, I. J.; KIM, J. a-SAD: Power Efficient SAD Calculator for Real time H.264 Video Encoder Using MSB-Approximation Technique. **Proceedings of the 2014 international symposium on Low power electronics and design - ISLPED '14**, ACM, v. 1, n. c, p. 259–262, 2014. Available from Internet: <http://dl.acm.org/citation.cfm?doid=2627369.2627650>.

DINIZ, C. **Arquitetura de hardware dedicada para a predição intra-quadro em codificadores do padrão H.264/AVC de compressão de vídeo**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2009.

FORCHHEIMER, R. Differential Transform Coding - A New Hybrid Coding Scheme. **Proc. Picture Coding Symp. (PCS-81)**, p. 15–16, June 1981.

GHANBARI, M. **Standard Codecs: Image Compression to Advanced Video Coding**. [S.l.]: Institution Electrical Engineers, 2003. ISBN 0852967101, 9780852967102.

GONZALEZ, R.; WOODS, R. **Processamento de imagens digitais**. Edgard Blucher, 2003. ISBN 9788521202646. Available from Internet: <https://books.google.com.br/books?id=d3MnAgAACAAJ>.

GRELLERT, M. **Machine learning mode decision for complexity reduction and scaling in video applications**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2018.

GRELLERT, M.; BAMPI, S.; ZATT, B. Complexity-scalable HEVC encoding. In: **2016 Picture Coding Symposium (PCS)**. [S.l.: s.n.], 2016. p. 1–5. ISSN 2472-7822.

GROIS, D. et al. Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders. In: **Picture Coding Symposium**. [S.l.: s.n.], 2013. p. 394–397.

GROUP, U. V. **Test Sequences**. 2018. Available from Internet: <http://ultravideo.cs.tut.fi/>.

HABIBI, A. Hybrid Coding of Pictorial Data. **IEEE Transactions on Communications**, v. 22, n. 5, p. 614–624, May 1974. ISSN 0090-6778.

HU, Q. et al. Analysis and optimization of x265 encoder. In: **2014 IEEE Visual Communications and Image Processing Conference**. [S.l.: s.n.], 2014. p. 502–505.

HUANG, Y. et al. An MCMC based Efficient Parameter Selection Model for x265 Encoder. In: **2018 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2018. p. 1–5. ISSN 2379-447X.

ITU-R. Parameter values for ultra-high definition television systems for production and international programme exchange. **Recommendation ITU-R BT.2020**, Oct 2015.

ITU-T. Video Codec for Audiovisual Services at p x 64 kbits. **ITU-T Recommendation H.261**, 1993.

ITU-T. Advanced video coding for generic audiovisual services. **ITU-T Recommendation H.264**, 2003.

ITU-T. High Efficiency Video Coding. **ITU-T Recommendation H.265**, 2013.

JEHNG, Y.-S.; CHEN, L.-G.; CHIUEH, T.-D. An efficient and simple VLSI tree architecture for motion estimation algorithms. **IEEE Transactions on Signal Processing**, v. 41, n. 2, p. 889–900, Feb 1993. ISSN 1053-587X.

KIM, I. et al. Block Partitioning Structure in the HEVC Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1697–1706, Dec 2012. ISSN 1051-8215.

KUMM, M.; KLEINLEIN, M.; ZIPF, P. Efficient sum of absolute difference computation on FPGAs. In: **2016 26th International Conference on Field Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2016. p. 1–4. ISSN 1946-1488.

LEE, S. et al. Fast partial distortion elimination based on a maximum error constraint for motion estimation. In: **2009 16th IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2009. p. 1549–1552. ISSN 1522-4880.

LEMMETTI, A. et al. AVX2-optimized Kvazaar HEVC intra encoder. In: **2016 IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2016. p. 549–553. ISSN 2381-8549.

LIU, Z.; WANG, L.; LI, X. Rate Control Optimization of X265 Using Information from Quarter-Resolution Pre-Motion-Estimation. In: **2018 25th IEEE International Conference on Image Processing (ICIP)**. [S.l.: s.n.], 2018. p. 3623–3627. ISSN 2381-8549.

MCCANN, K. et al. **HM10: High Efficiency Video Coding Test Model (HM10) Encoder Description**. Geneva, Switzerland: [s.n.], 2013.

MEDIA, A. for O. **Alliance for Open Media**. 2019. Available from Internet: <https://aomedia.org/>.

MIYAKOSHI, J. et al. A Low-Power Systolic Array Architecture for Block-Matching Motion Estimation. **IEICE Transactions**, v. 88-C, p. 559–569, 04 2005.

MONTEIRO, E. C.; SANTOS, L. A. S. Modernização dos Sistemas Audiovisuais do Governo Baseada em Software Livre. In: **ConSempro - Congresso Serpro de Tecnologia e Gestão aplicadas a Serviços Públicos**. [S.l.: s.n.], 2013.

MULTICOREWARE. **HEVC x265 Encoder**. 2019. Available from Internet: <https://bitbucket.org/multicoreware/x265/>.

MULTICOREWARE. **MulticoreWare**. 2019. Available from Internet: <https://multicorewareinc.com/>.

PENNEBAKER, W. B.; MITCHELL, J. L. **JPEG Still Image Data Compression Standard**. 1st. ed. Norwell, MA, USA: Kluwer Academic Publishers, 1992. ISBN 0442012721.

PORTO, M. et al. Motion Estimation Architecture Using Efficient Adder-Compressors for HDTV Video Coding. **Journal of Integrated Circuits and Systems**, v. 5, n. 1, p. 78–88, Mar 2010.

PORTO, R. E. C. **Desenvolvimento arquitetural para estimação de movimento de blocos de tamanhos variáveis segundo padrão H.264/AVC de compressão de vídeo digital**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2008.

QUALCOMM. Qualcomm Snapdragon 4K Ultra HD. 2014. Available from Internet: <https://www.qualcomm.com/media/documents/files/snapdragon-4k-datasheet.pdf>.

RABAEY, J. M. **Digital Integrated Circuits: A Design Perspective**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN 0-13-178609-1.

RABINER, L. R.; GOLD, B. **Theory and Application of Digital Signal Processing**. Englewood Cliffs, New Jersey, USA: Prentice Hall, 1975.

RICHARDSON, I. E. G. Video Codec Design : Developing Image and Video Compression Systems. 03 2003.

RIVAZ, P. d.; HAUGHTON, J. **AV1 Bitstream Decoding Process Specification**. 2019.

SANCHEZ, G.; PORTO, M.; AGOSTINI, L. A hardware friedly motion estimation algorithm for the emergent HEVC standard and its low power hardware design. In: **2013 IEEE International Conference on Image Processing**. [S.l.: s.n.], 2013. p. 1991–1994. ISSN 1522-4880.

SANCHEZ, G. et al. Hardware-friendly HEVC motion estimation: new algorithms and efficient VLSI designs targeting high definition videos. **Analog Integrated Circuits and Signal Processing**, v. 82, n. 1, p. 135–146, Jan 2015. ISSN 1573-1979. Available from Internet: <https://doi.org/10.1007/s10470-014-0342-9>.

SCHMITZ, M. T.; AL-HASHIMI, B. M.; ELES, P. **System-Level Design Techniques for Energy-Efficient Embedded Systems**. Norwell, MA, USA: Kluwer Academic Publishers, 2004. ISBN 1402077505.

SEIDEL, I.; BRäSCHER, A. B.; GüNTZEL, J. L. Combining Pel Decimation with Partial Distortion Elimination to Increase SAD Energy Efficiency. In: **2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)**. [S.l.: s.n.], 2015. p. 177–184.

SEIDEL, I. et al. Energy-efficient SATD for beyond HEVC. In: **2016 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2016. p. 802–805. ISSN 2379-447X.

SHAH, N. N.; DALAL, U. D. Register array-based sum of absolute difference processor with parallel memory system for fast motion estimation. **IET Computers Digital Techniques**, v. 12, n. 3, p. 95–104, 2018. ISSN 1751-8601.

SHARMAN, K.; SUEHRING, K. **Common test conditions**. 2018. Available from Internet: <https://hevc.hhi.fraunhofer.de/>.

SILVA, M. G. d. **Computational effort analysis and control in High Efficiency Video Coding**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2014.

SILVEIRA, B. et al. Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 64, n. 12, p. 3126–3137, Dec 2017. ISSN 1549-8328.

SILVEIRA, B. et al. Power-efficient sum of absolute differences architecture using adder compressors. In: **2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. [S.l.: s.n.], 2016. p. 340–343.

SOARES, L. B. et al. A novel pruned-based algorithm for energy-efficient SATD operation in the HEVC coding. In: **2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2016. p. 1–6.

SULLIVAN, G. J. et al. Overview of the High Efficiency Video Coding (HEVC) Standard. **IEEE Trans. Circuits Syst. Video Technol.**, v. 22, n. 12, p. 1649–1668, Dec 2012. ISSN 1051-8215.

SULLIVAN, G. J.; WIEGAND, T. Rate-distortion optimization for video compression. **IEEE Signal Processing Magazine**, v. 15, n. 6, p. 74–90, Nov 1998. ISSN 1053-5888.

VANNE, J. et al. A High-Performance Sum of Absolute Difference Implementation for Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 16, n. 7, p. 876–883, July 2006. ISSN 1051-8215.

VASSILIADIS, S. et al. The sum-absolute-difference motion estimation accelerator. In: **Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204)**. [S.l.: s.n.], 1998. v. 2, p. 559–566 vol.2. ISSN 1089-6503.

VIITANEN, M. et al. Kvazaar HEVC encoder for efficient intra coding. In: **2015 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2015. p. 1662–1665. ISSN 0271-4302.

VIITANEN, M. et al. Kvazaar: Open-Source HEVC/H.265 Encoder. In: **Proceedings of the 2016 ACM on Multimedia Conference**. [S.l.]: ACM, 2016. p. 1179–1182.

VIITANEN, M. et al. Live demonstration: Run-time visualization of Kvazaar HEVC intra encoder. In: **2016 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2016. p. 454–454. ISSN 2379-447X.

YAP, S. Y.; MCCANNY, J. V. A VLSI architecture for advanced video coding motion estimation. **Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors**, v. 2003-Janua, p. 293–301, 2003. ISSN 10636862.

YIN, S.; ZHANG, X.; GAO, Z. Efficient SAO coding algorithm for x265 encoder. In: **2015 Visual Communications and Image Processing (VCIP)**. [S.l.: s.n.], 2015. p. 1–4.

YUFEI, L.; XIUBO, F.; QIN, W. A High-Performance Low Cost SAD Architecture for Video Coding. **IEEE Transactions on Consumer Electronics**, v. 53, n. 2, p. 535–541, May 2007. ISSN 0098-3063.