Ricardo António da Silva Rocha

Load-balancing for Parallel HEVC Video Encoding

Dissertação de Mestrado em Engenheira Electrotécnica e de Computadores

Fevereiro 2016

· U C ·

Universidade de Coimbra

· U C ·

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Load-balancing for Parallel HEVC Video Encoding

Ricardo António da Silva Rocha

Coimbra, February 2016

· U C ·

Load-balancing for Parallel HEVC Video Encoding

Supervisor:

Luis A. da Silva Cruz

Jury:

Henrique José Almeida da Silva

Vítor Manuel Mendes da Silva

Fernando José Pimentel Lopes

Dissertation submitted in partial fulfillment of the requisites for the obtention of the Master of Science in Electrical and Computer Engineering.

Coimbra, February 2016

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Luís A. da Silva Cruz, for the useful advice and help through my work and writing process of this master thesis. His support was essential for its success.

I would also like to thank my laboratory colleagues and friends that provided me a good working environment and supported me during my work.

I also would like to thank my parents for their endless support, keeping my motivation intact during this phase.

Finally, I must express my sincere and profound gratitude to my girlfriend. Her emotional support and advices were important for me to conclude my work.

Resumo

A norma de codificação de vídeo HEVC (High Efficiency Video Coding) surge como sucessora da norma H.264/AVC, em resposta ao rápido aumento do volume de informação em vídeo com resoluções HD e UHD. A nova norma inclui ferramentas para suportar implementações paralelas, como *slices*, *tiles* e *wavefront parallel processing.* Este trabalho propõe um método que ajusta automaticamente a geometria das *tiles* durante o processo de codificação de vídeo, para equilibrar a distribuição de carga entre as unidades processadoras , num cenário em que cada *tile* é processada por uma unidade distinta. A solução baseia-se em quatro métodos de estimação de complexidade e um algoritmo que ajusta dinâmicamente as *tiles* de acordo com a complexidade estimada. Os resultados mostram que o método proposto é capaz de aumentar a eficiência de paralelização, quando comparado com *tiles* com tamanhos iguais, obtendo-se em média uma redução do tempo de processamento de 6.1% com um aumento de BD-rate entre 0.82% e 1.67%.

Palavras-chave: Codificação de Vídeo, High-efficiency Video Coding, Ferramentas orientadas a parallelismo, Tiles, Equilíbrio de Carga

Abstract

The High-efficiency Video Coding (HEVC) standard arises as the successor to the H.264/AVC standard, in response to the rapid increase in HD and UHD digital video information volume. The new standard includes tools to support parallel implementations, such as slices, tiles and wavefront parallel processing. This work proposes a method that automatically adjusts the tile geometry during the video encoding process, to balance the workload distribution among the processing units, assuming that each tile is processed by a different unit. The solution comprises four complexity estimation methods and an algorithm to dynamically adjust the tile geometry. Results show that the proposed method is able to improve the parallelization efficiency compared to the use of same-sized tiles, saving on average 6.1% of the processing time at the cost of an increase in BD-rate between 0.82% and 1.67%.

Keywords: Video Coding, High-efficiency Video Coding Standard, Parallel-oriented Tools, Tiles, Load-balance

Contents

List of Acronyms

AMVP	Advanced Motion Vector Prediction
ATS	Average Time Saving
AVC	Advanced Video Coding
CABAC	Context Adaptive Binary Artithmetic Coding
CB	Coding Block
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DBF	Deblocking Filter
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
ETZS	Enhanced Test-Zone Search
FPS	Frames Per Second
FS	Full Search
GOP	Group of Pictures
HD	High-Definition
HEVC	High-efficiency Video Coding
JCT-VC	Joint Collaborate Team on Video Coding

LCBF	Linear Combination of Bits per Frame
LCBG	Linear Combination of Bits per GOP
LCTF	Linear Combination of encoding Times per Frame
LCTG	Linear Combination of encoding Times per GOP
MADIT	Mean Absolute Difference from the Ideal Time
MV	Motion Vector
OWF	Overlapped Wavefront
PB	Prediction Block
PSU	Parallel Speed Up
PU	Prediction Unit
QP	Quantization Parameter
SAO	Sample Adaptive Offset
SI	Spatial Index
SL	Simple Linear
TB	Transform Block
TI	Temporal Index
TU	Transform Unit
UHD	Ultra High-Definition
WPP	Wavefront Parallel Processing

List of Figures

List of Tables

1 Introduction

Digital video representation has been continuously growing in the last decades. Video content with high resolution requires large amounts of data to be represented, making the use of video compression unavoidable in video storage and transmission applications.

Advances in technology lead to the appearence of devices capable of recording and reproducing high-resolution video content. Nowadays even mobile devices, such as smartphones and tablets, enable the user to view high-definition (HD) videos with resolutions of 720p and 1080p and there is a constant demand for better quality and the emergence of ultra-high definition (UHD) video, with resolutions going up to 4K or even 8K. Some products, such as monitors and TVs, that support UHD resultions such as 3840x2160, are already being comercialized [9]. This makes more evident the need for more efficient video coding techniques.

The H.264/AVC standard, which is the most used and widely distributed video codec, can handle HD resolution video but it isn't optimized to encode UHD video content [2]. This limitation lead to the development of the sucessor of H.264/AVC, the High Efficiency Video Coding (HEVC) standard.

The Joint Collaborative Team on Video Coding (JCT-VC) finished the first standardized version of HEVC in 2013. The main goal of this new standard is to reduce the bitrate, compared to the H.264/AVC standard, by half while maintaining the same visual quality. This bitrate reduction, however, comes at a cost of about two to ten times more computational effort, specially on the encoder side [10].

To deal with the high costs in terms of computational workload, the HEVC standard offers novel tools to support parallel processing. In an era where multi-core devices are becoming the norm, the use of parallelism is highly beneficial.

When using parallel processing, a key issue is the parallel efficiency. Maximum efficiency is achieved when all tasks take exactly the same time to complete [11] and the idle time is reduced. This applies to HEVC encoding using the parallel processing tools supported as well. Among the parallel tools supported, HEVC defines tiles, which divide a video frame into various parts that can be processed independently from each other. In order to maximize the throughput, it's important that all parts have the same encoding complexity. Thus, the challenge is to distribute the frame encoding workload evenly among the available processing units. However, partioning a frame into independently encodable sections, breaks coding dependencies and, consequently, introduces losses in coding efficiency. Therefore the number of partitions within a picture must be minimized.

Some published works propose methods to optimize specific performance parameters of HEVC parallel encoding. While the main focus of most of these works is on achieving better parallel efficiency, some authors take a different approach and aim to reduce the losses in coding efficiency introduced by the usage of the parallel tools of HEVC [4]. This work is focused on improving the HEVC parallel efficiency through load-balancing.

1.1 Objectives

This thesis proposes a method to dynamically adjust the tile geometry during HEVC encoding of a video sequence. The goal is to define the partitions in a way that ensures a balanced encoding workload distribution among all processing units. The algorithm starts by estimating the complexity of encoding basic visual data units. Various complexity estimation methods were proposed and evaluated. Then, based on the estimated complexities, a load distribution algorithm defines the tile geometry that achieves the most even computational complexity distribuition among all processing units.

1.2 Text Organization

Chapter 2 will give a brief summary of video coding basics and briefly describe the HEVC standard as well as its parallel-oriented tools. Chapter 3 describes and analyses problems that result from the usage of those tools, as well as proposed solutions from other works in the literature. Chapter 4 focuses on the encoding complexity estimation problem and describes some early experiments in an attempt to develop an estimation model. Chapter 5 describes four proposed complexity estimation methods and an algorithm to dynamically adjust the tile geometry based on the estimated encoding complexity. Chapter 6 presents the experimental evaluation of the proposed solution, listing results for the load-balancing efficiency, estimated parallelism speed-up and the impact on coding efficiency, as well as comparing them to similar works. Chapter 7 concludes this work by summarizing the method proposed and its results and lists some future work topics.

2 Introductory video coding concepts and the HEVC standard

2.1 Basic concepts of video coding

Before describing in detail the major aspects of the new HEVC standard, this section will give a brief explanation of the basic concepts of video coding. The main purpose of video coding is to compress video data by exploiting temporal and spatial redundancies of the video signals [1], as well as irrelevancy of parts of the visual data. Spatial redundancy is used in intra-frame encoding, where temporal redundancy is exploited in inter-frame encoding. Frames encoded using exclusively intra encoding are called I-frames, while frames that include inter encoded blocks are either called P-frames or B-frames. P-frames are encoded using references from previous frames (in display order), while B-frames can use references from both previous and subsequent frames. The encoding process could be performed for the entire frame at once. However for high resolution video this would require a lot of computational power, in addition to be less robust to errors. Therefore, the frame is partitioned into smaller blocks which are considered the basic coding unit.

The inter-frame encoding aims to explore temporal redundancy. A video sequence can be efficiently coded by estimating the motion between frames and compensating the effect of that motion during prediction [1]. For each block in the frame that is being encoded, the block-based motion estimatior searches for the best matching block in one or more reference frames. The displacement of the best matching block is then represented by a motion vector (MV). Afterwards, motion compensation uses those vectors to build a prediction frame/block.

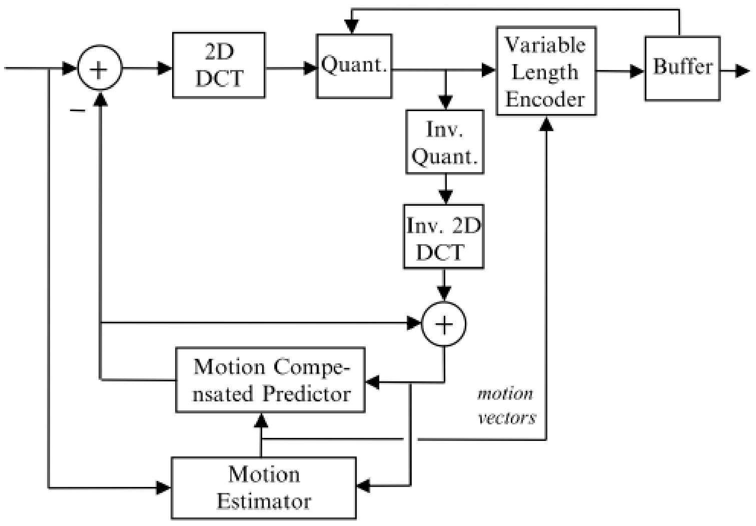

Figure 2.1: Generic hybrid encoder (source: [1])

Intra-frame encoding exploits spatial redundancies within a frame, analyzing samples adjacent to the block to be encoded, to be used as reference data for prediction in regions where inter-frame prediction is not used, either because the current frame is a I-frame or because it is more efficient than the inter-frame prediction.

After generating the prediction, using either of the previously described methods, the encoder subtracts that prediction from the original frame. The result is called frame/block residual. This residuals are then processed using a 2D transform. Usually encoders use an approximation of the Discrete Cosine Transform (DCT) to transform the residuals. The results are then quantized, where the amount of information discarded can be controlled via a Quantization Parameter (QP). Finally the encoder applies entropy coding to the transformed and quatized residuals, to further reduce the data volume. This step is based on the probability of occurrence of the encoded symbols and uses codes such as Huffman or arithmetic encoders.

Frames of a video sequence are usually grouped into sets usually called Groups of Pictures (GOP). These GOPs contain, in general one intra-encoded frame while the rest is comprised of inter-encoded frames. Figure 2.2 shows a representation of a typical GOP structure. The arrows indicate the prediction directions for each frame. A GOP can either be an open GOP, which allows B and P-frames to have references outside of the GOP, or a closed GOP where all the references must be inside the same GOP.

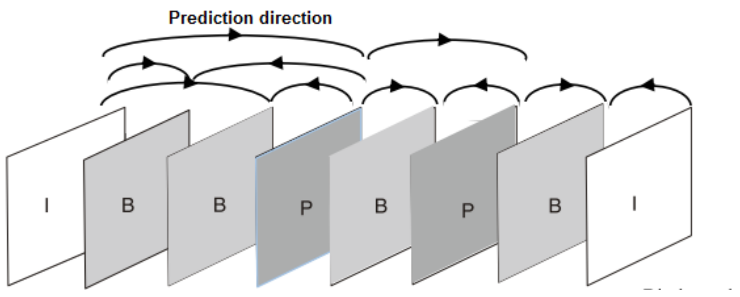

Figure 2.2: Typical structure of a GOP (source: [1])

This section described how a generic video encoder works. The recent video coding standards, including HEVC, follow all these steps to compress video data. In the next section, the tools and coding structures specific to HEVC are explained in more detail.

2.2 The HEVC standard

The Joint Collaborate Team on Video Coding (JCT-VC) proposed a new video codec, with a goal to reduce the bitrate of the current standard by half while maintaining the same visual quality. In 2013, the HEVC reached its first final version. Improvements and extensions are still being prepared, for instance, to support HDR encoding.

HEVC uses the same hybrid approach as previous video coders, since H.261 [2], based on intra- and inter-frame prediction and 2-D transforms. Figure 2.3 shows the block diagram of a hybrid model for a video encoder according to the HEVC standard. However, in order to obtain better results in compression rate, while keeping the same visual quality, some of the encoding tools used in previous standard were improved and some new features were added to the HEVC standard, as described in more detail in the next sections.

2.2.1 Coding Structures

The basic coding structure of HEVC is the Coding Tree Unit (CTU), replacing the macroblock used in the previous standards. CTUs consist of luma and chroma Coding Tree Blocks (CTBs), with each luma CTB being of size LxL and each chroma CTB of size L/2xL/2. L can assume three values: 16, 32 and 64 samples. The selection of the CTB sizes is highly related to the video characteristics. For example, for HD or UHD videos it's reasonable to use bigger CTB sizes [2].

Each CTB can be partitioned into smaller blocks, called Coding Blocks (CB) according to a quad-tree structure (figure 2.4). The root of the quadtree is at the CTU level, meaning

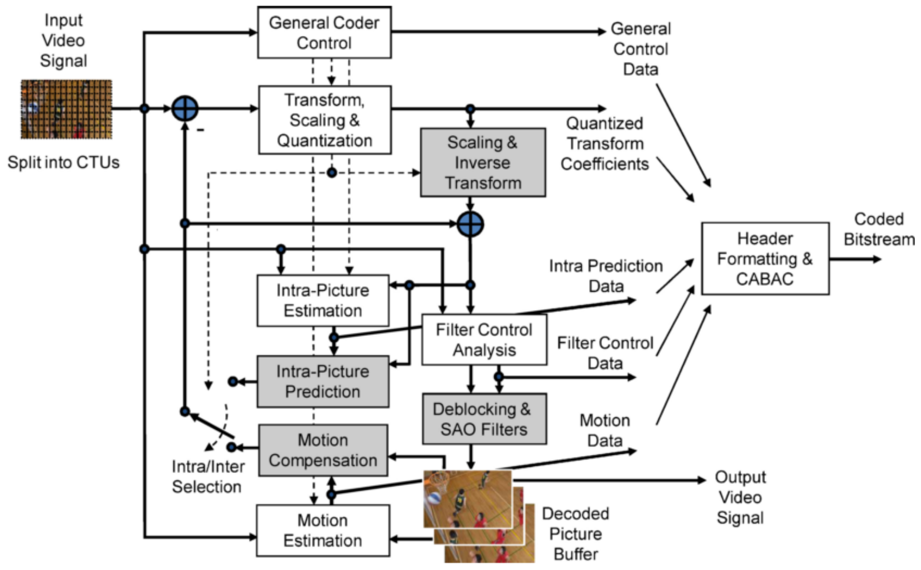

Figure 2.3: Block diagram of the HEVC video encoder (source: [2])

that the largest supported luma CB size is the size of a luma CTB. Each CB can then be further split into smaller CBs in a similar way until the minimum allowed size for a luma CB is reached (8x8 samples). The combination of a luma CB plus two chroma CBs and some syntax elements form the Coding Unit (CU).

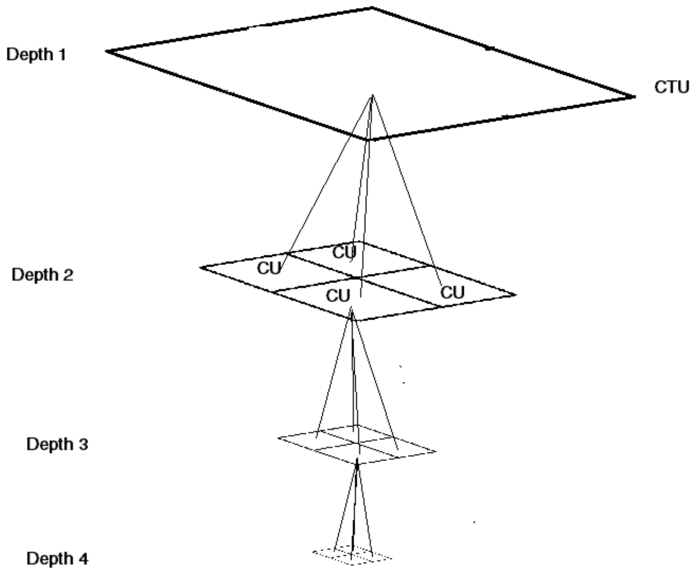

Figure 2.4: CTU quadtree structure

The prediction mode (intra or inter) used in each CU is signalled in the bitstream. If the prediction mode is signaled as intra, the CU can assume two different configurations for its Prediction Units (PU): either a 2Nx2N or a NxN configuration (figure 2.5, dotted line). The former means that the whole CU defines the PU, while the latter can only be applied if the CU size matches the minimum allowed size (8x8). In that case, the PUs have the size of 4x4 samples.

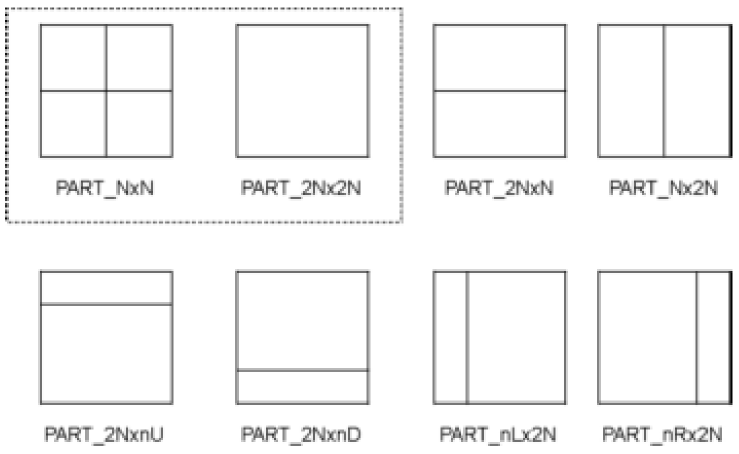

Figure 2.5: PU partitioning modes (source: [3])

If the CU is signaled with inter prediction, it can use the same configurations as with intra-frame prediction with the same rules. However there are additional PU configurations, as shown in figure 2.5, for the intra-frame prediction. Those configurations can either be symmetric or assymetric. The latter can only be used in CUs with size greater than 16x16 pixels.

For residual encoding, a CU can be split into Transform Units (TU), which cotains luma and chroma Transform Blocks (TB). The luma TB, as well as the chroma TBs, can be identical to the respective CB residual, or it can be further split into smaller TBs. The residuals that make up each TB are transformed using integer approximations of either the DCT or the Discrete Sine Transform (DST).

2.2.2 Intra-frame Prediction

Intra-frame prediction supports 33 directional (or angular) prediction modes, as well as one DC and one planar mode (figure 2.6). Directional modes predict the current PU by performing an extrapolation on samples from adjacent already encoded PUs [2]. Figure 2.7 showns an example for the directional mode 21.

Planar prediction and DC prediction can be used as an alternative. Planar prediction generates the prediction for the current PU by a weighted average of four reference samples. DC mode, on the other hand, performs its prediction calculating the mean of its reference samples.

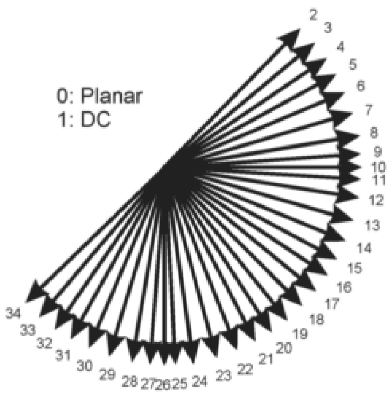

Figure 2.6: Intra-frame prediction modes (source: [2])

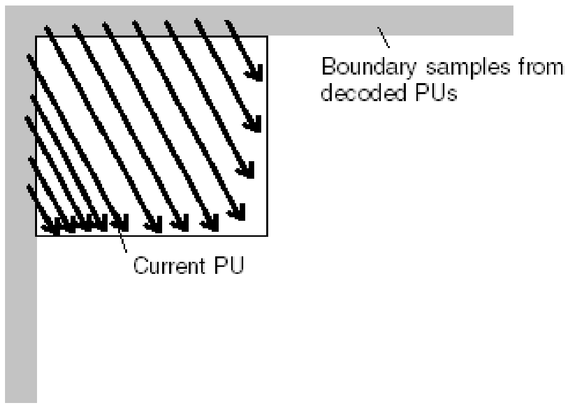

Figure 2.7: Example of the directional mode 21 (adapted from: [2])

Testing all 35 available modes would find the best prediction mode for intra-prediction. However, doing so requires a lot of computational effort which would be highly inefficient. Instead, several algorithms have been developed to decrease the number of prediction modes that need to be tested. For instance, the authors of [12] propose a solution in which the prediction modes are chosen based on edge information of the current PU.

2.2.3 Inter-frame Prediction

Inter-frame prediction, as mentioned before, is based on motion compensated temporal prediction, which requires a previous motion estimation step. This is one of the most important encoding tools in any video codec as it is the one responsible for achieving high bitrate savings. However, it requires a lot of computational effort. In HEVC, with the upgraded, more complex coding structures, inter-frame leads to even higher computational workloads compared to H.264/AVC [3].

Block-based motion estimation defines a search area in the reference frame, usually centered around the same position of the unit that's being processed in the current frame[13]. Then it searches for the block that best matches the current one, within the search area. With the best match found, a motion vector is defined poiting from it towards the position of the current block. Limiting the search area leads to less comparisons needed to be performed and thus better computational efficiency. The HEVC reference software, HM, uses two different approaches for block matching: Full Search (FS) and Enhanced Test Zone Search (ETZS) [14].

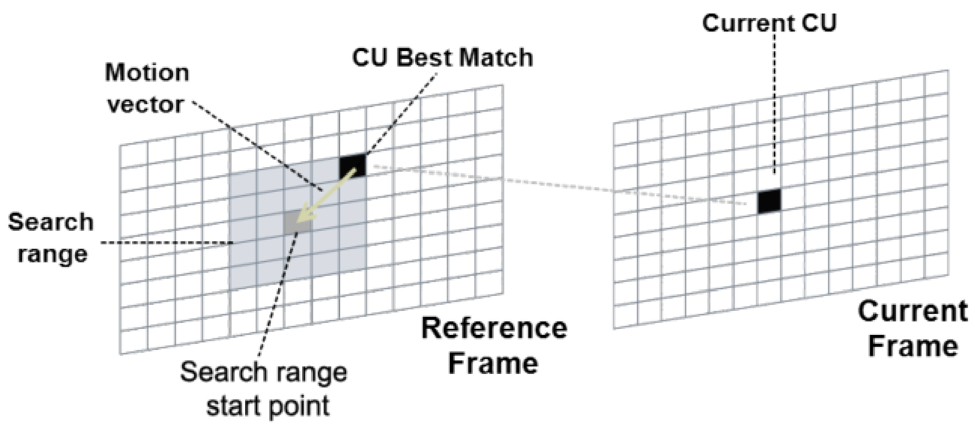

Figure 2.8: Motion Estimation (source: [4])

Motion estimation is done at the PU level. For each PU one of three prediction modes is selected: inter, merge or skip. For inter mode, Advanced Motion Vector Prediction is used. This method selects the best motion vector from a candidate list to predict the motion vector of the current PU [5]. This list contains two spatial motion candidates and one temporal candidate (figure 2.9). This way only the difference between the current MV and the predicted MV is encoded.

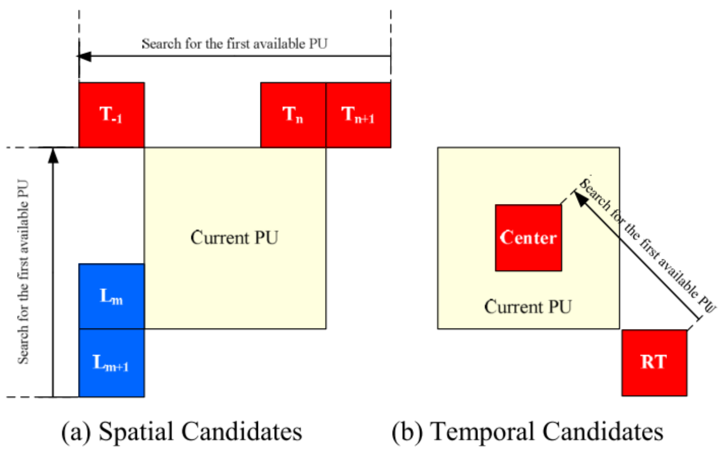

Figure 2.9: Illustration of the AMVP candidates (source [5].)

For merge mode the MVs are directly inherited from temporally or spatially neighbouring PUs. Skip mode is treated as a special case of the merge mode but it works in a similar way.

It is important to note that more than one reference frame can be used for inter prediction. The reference frames are tracked using two lists, List 0 and List 1. List 0 is used for P-frames, while B-frames use both List 0 and List 1 [15]. For each PU, one MV is defined for P-frames and two MVs for B-frames.

2.2.4 Transform and Quantization

The result of the intra or inter-frame prediction is a residual frame. This residual will be the input for the transform module. HEVC uses an integer approximation of the 2D Discrete Cosine Transform to generate coefficients that will be quantized afterwards. The HEVC standard also uses an integer approximation of the 2D Discrete Sine Transform (DST) in 4x4 intra-predicted TUs [2].

The quantization process depends on a parameter set before the encoding starts, the Quantization Parameter (QP), which can be changed during encoding to, for example, control the bitrate [16]. Depending on its value, more or less information is discarded during the transform coefficient quantization. The QP varies between 0 and 51 and an increase by 6 means double the quantization step size. The higher its value, the more information is lost during the quantization step.

2.2.5 Entropy Coding

Entropy coding is a form of lossless compression used at the last stage of video encoding after the video has been reduced to a series of syntax elements [17]. For entropy encoding, the HEVC standard adopts the Context Adaptive Binary Arithmetic Coding (CABAC) as its main tool. The core algorithm of CABAC, in relation to the previous standard, remains unchanged [2]. The CABAC involves three main functions: binarization, context modeling and arithmetic coding.

Binarization maps the syntax elements to binary symbols. Several different binarization processes are used in HEVC, including Unary (U), Truncated Unary (TU), kth-order Exp-Golomb (EGk), and Fixed Length (FL) [17]. The process selection is based on the type of syntax element. Context modeling is used in terms of estimation of the probability of the binary symbols resulting from binarization. This dynamic probability estimation is a key factor to the efficiency of the CABAC coding [2]. The context model for each binary symbol can be chosen based on several properties, like type of syntax element or information on luma/chroma. The context can be switched after each symbol. HEVC uses an improved probability update method from H.264/AVC [17]. Arithmetic coding efficiently maps the binary symbols to bits, according to the estimated probabilities.

2.2.6 In-Loop Filters

HEVC uses two filters: the deblocking filter (DBF) and a Sample Addaptive Offset (SAO) filter. The DBF is applied at the block boundaries, in order to reduce visual artifacts resulting from block-based coding. It's similar to the DBF used in the H.264/AVC standard. The SAO filter, however, is new to HEVC. It modifies the decoded samples by conditionally adding an offset value based on values in look-up tables transmitted by the encoder.

2.3 Parallel processing oriented tools

Encoding HD and UHD video requires a lot of computational power and time, due to the large amount of data to be processed. To face that problem, current multimedia processing devices are progressively adopting architectures and algorithmic solutions that enable parallel processing.

The HEVC standard includes tools to support parallel-oriented processing, for both encoding and decoding. These tools allow for different regions of a frame to be processed independently.

Slices, Tiles and Wavefront Parallel Processing (WPP) are the parallel processing functions proposed by the standard. This chapter will describe these tools and address their advantages and problems.

2.3.1 Slices

Slices are data structures that define groups of CTUs, that can be coded and reconstructed independently [2], thus allowing parallel simultaneous encoding and decoding. They contain the CTUs in raster scan order. A slice can either contain a section of a picture (figure 2.10) or the entire picture. By default, there is always one slice.

There are three types of slices: I-slices, P-slices and B-slices. I-slices are coded using intra-prediction only. P-slices contain inter encoded CTUs with motion compensated prediction from one reference frame, while B-slices contain inter encoded CTUs with motion compensated prediction from two reference pictures. Since a frame is formed by one or more slices, some restrictions apply. An I-frame can only contain I-slices, while a P-frame or a B-frame can contain combinations of all slice types.

Slices are self-contained, which means they can be parsed from the bitstream and their samples can be decoded without needing any additional data from other slices in the same picture. Their main purpose is allowing resynchronization in case of data loss and packetization.

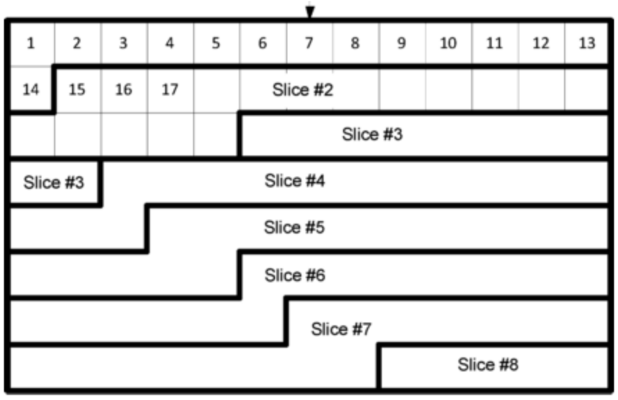

Figure 2.10: An example of a picture divided in various slices (source: [6])

Because of how they are formed, slices usually form regions with a lower level of spatial correlation within the picture, resulting in a high losses of coding efficiency. Additionally each slice has an associated slice header which adds some overhead, reducing the encoding efficiency.

2.3.2 Wavefront Parallel Processing (WPP)

When WPP is enabled, each slice is divided into rows of CTUs [7]. The first row is encoded in a normal way and the next row starts as soon as at least two CTUs of the previous one have been encoded. This repeats for all subsequent rows (figure 2.11). Also, WPP does not change the regular raster scan order. Because dependencies are not broken the rate-distortion loss of a WPP bitstream is small compared to a regular bitstream [7].

However, due to data dependencies, it's not possible to start or finish processing several CTU rows at the same time. This introduces some parallization inneficiencies.

There's enhancement of WPP called Overlapped Wavefront (OWF) [7], which mitigates some parallezation inefficiencies of the former. In this enhancement, when a CTU row has finished being processed and no more unprocessed rows are available, the processing of the next picture can start immediately instead of having to wait until the remaining rows are

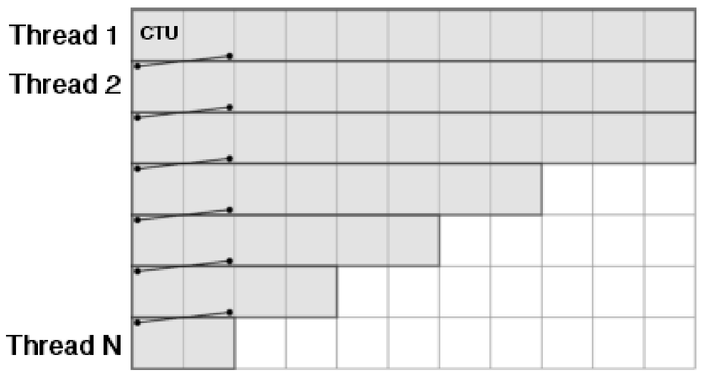

Figure 2.11: Wavefront Parallel Processing (adapted from [7])

finished (figure 2.12). For this technique to work, some constraints have to be applied to motion vectors, such as ensuring that all reference pictures are available and limiting the maximum vertical lenght of the MVs.

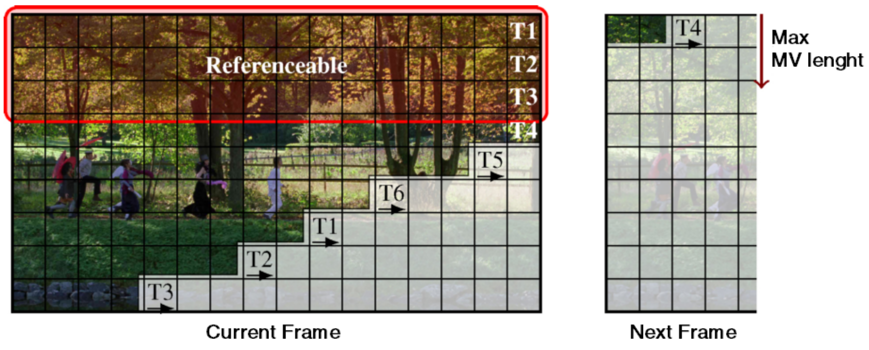

Figure 2.12: Representation of OWF (adapted from [7])

Overall WPP has the lowest amount of coding losses of all parallel-oriented tools [7]. Additionally, it doesn't introduce visual artifacts at the boundaries. However, due to large dependence of data between the various rows, it's hard to expect large time savings when using WPP [8]. WPP has low parallel efficiency, directly related to data dependencies. The reason for that is since CTU rows have to wait for at least two CTUs from the previous rows to finish, the processing units they are asigned to will be in idle state during that time.

2.3.3 Tiles

Tiles split the picture into rectangular regions. They form independently encodable and decodable partitions, but require adding additional header information [2]. A slice can contain multiple tiles (mostly when the slice represents the entire picture). Just like slices, Tiles are independent. Thus, they don't require any communication between the processors for CTU endoding and decoding. This means that all coding dependencies, such as motion vectors, entropy coding context and intra-frame neighbouring samples are broken at the boundaries [6]. The filtering stage, however, can be performed across the tile boundaries to reduce visual artifacts [18]. Slices and tiles can coexist but if a frame is divided into multiple slices, each slice must belong to a unique tile. In other words, a slice boundary cannot cross a tile boundary [2].

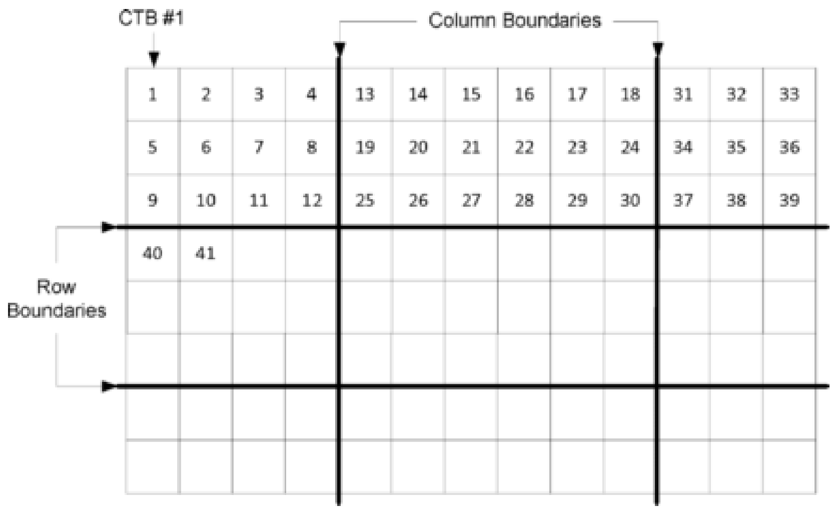

Figure 2.13: A picture divided into tiles (source: [6])

Unlike slices, boundaries of tiles alter the encoding order of CTUs which gives a bit more felixibility in terms of partitioning, particulary since it allows defining regions of interest [19]. As a result, the correlation between pixels in a tile can be higher when compared to slices. In addition, there's no need to encode header information for each individual tile, because this information is already contained within the slice header. [6]. For example, for the case of one slice (the entire frame) divided into four tiles, only one header is encoded that contains all information on the four tiles.

The usage of tiles does not cause a big loss of coding efficiency compared to, for example, slices [20]. Additionally having no dependencies between tiles allows them to be processed independently. Therefore, tiles present characteristics to be an effective tool to explore parallel solutions and, for this reason, will be the focus of this work.

3 Parallel Tools of HEVC - the load balancing problem

With the increasing availability of multi-core devices and the widespread adoption of high and ultra-high resolution video, not to mention 3D/multiview video, the potential of parallel video encoding should be considered.

As seen in chapter 2, HEVC includes tools to actively support the use of parallel processing: slices, tiles and WPP. By distributing the workload of the HEVC encoder among the various available processing units, the total encoding time can be greatly reduced which overall can improve the power efficiency [21]. On the other hand, parallezation comes with some constraints, namely the risks of video quality losses and larger energy consumption if the parallelization of the encoding is not properly designed.

One important issue when using the parallel tools of HEVC is the distribution of the workload in a selective manner among multiple processing cores. If all cores are balanced in terms of workload, all of them take a similar amount of time to complete their tasks which is crucial to fulfil the imposed deadlines and avoid problems such as chip failure due to uneven heat dissipation.

3.1 Workload Unbalance with Tiles

Although HEVC supports other parallel encoding tools, this work will focus on the use of tiles, as they provide more flexibility in adjusting the load among processing units. Tiles divide a frame into rectangular partitions that can be encoded independently. It is possible to vary the number of tiles as well as the position of their boundaries.

The simpler tile configuration uses tiles of the same size, i.e., a configuration in which all tiles contain the same number of CTUs. Theoretically, this would lead to a balance in encoding complexity as all tiles would have the same number of blocks to be processed. However, most of the time, that does not happen as it is common for a video sequence to experience a lot of intra-frame variation in its content, causing some areas of the frame to be more complex to encode than others. For example, a certain part of a video frame contains an action scene with more movement. This results in the frame encoding complexity not being uniformly distributed.

For a better illustration of the problem, three HD video sequences, shown in table 3.1, were encoded using the HM 16.0 reference software, with a QP of 32 and the Low Delay P configuration. Four uniformly spaced tiles were defined. Note that due to the resulution of the videos used, the tile configuration is not exactly uniform. For each frame, there are 30 CTUs per row and 17 CTUs per column. Since 17 is an odd number, it is not possible for all tiles to have the same number of CTUs. Therefore, the uniform tile configuration used in this case places the boundaries on position 15,8 (figure 3.1). The images of the video frames, with the respective tile configuration, were obtained using the software *Gitl HEVC Analyser* [22].

Sequence	Resolution	Frame count	FPS
BasketballDrive	1920x1080	500	50
Jockey	1920x1080	600	120
ParkScene	1920x1080	240	24

Table 3.1: Video sequences used to study the problem related to uniform tiles.

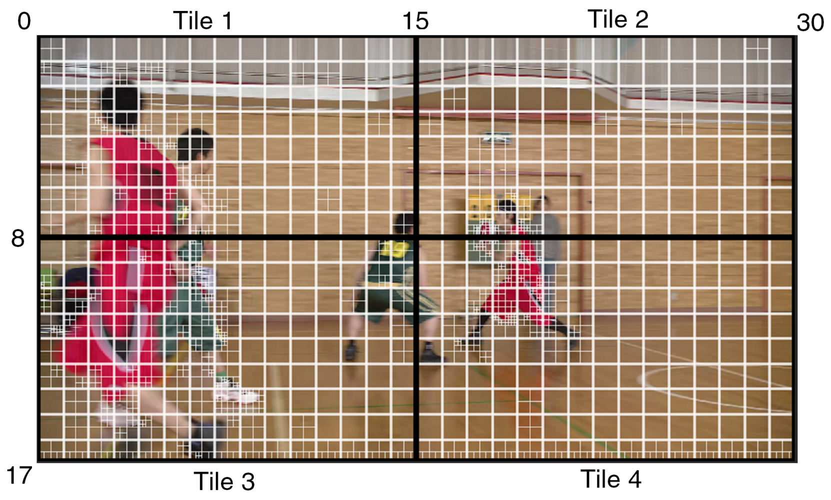

Figure 3.1: Video frame divided into four uniformly-spaced tiles

It's important to note that the HEVC reference software, unlike the actual HEVC encoder, is not optimized for real-time applications. That is the reason why the resulting encoding times in all simulations are given in seconds. However, it reflects the encoder's behavior if it was applied in an optimized implementation.

As stated before, frames of a video sequence can experience an uneven encoding complexity distribution. To illustrate this problem, figure 3.2 shows a video frame from the sequence *BasketballDrive*, divided into four uniformly-spaced tiles, and the encoding times of its CTUs.

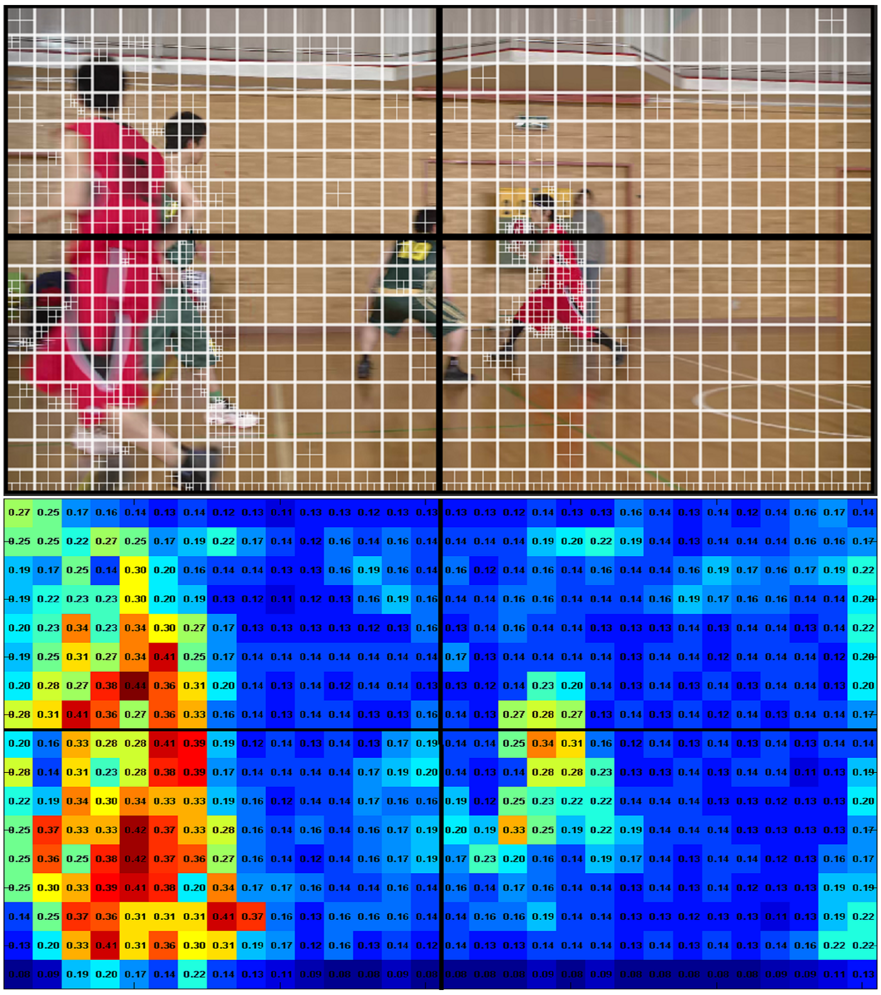

Figure 3.2: 6th frame of the sequence *BasketballDrive* with a uniformly-spaced tile configuration (top) and the respective encoding times (in seconds) for each CTU (bottom).

By analyzing the figure, it is evident that the left area of the frame contains more CTUs with higher encoding times, whereas the right side has low encoding times on the majority of its CTUs. From this analyis it is safe to assume that tiles 1 and 3 (top left and bottom left, respectively) will take longer to encode compared to tiles 2 and 3 (top right and bottom right, respectively). Table 3.2 summarizes the distribution of encoding complexities for the three considered video sequences. The total encoding time for a tile is calculated by the sum of the encoding times of each CTU contained in it.

Sequence	**Tile**	**Encoding Time Total (seconds)**	**Complexity %**
BasketballDrive (6th frame)	1	23.26	25.4
	2	18.09	19.7
	3	30.00	32.7
	4	20.36	22.2
Jockey (6th frame)	1	16.27	22.8
	2	18.80	26.4
	3	16.99	23.8
	4	19.27	27.0
ParkScene (6th frame)	1	18.53	26.1
	2	16.41	23.1
	3	19.64	27.6
	4	16.44	23.2

Table 3.2: Summary of the encoding complexity distribution

From the observation of the presented values, it is evident that the encoding complexity is not uniformly distributed. In the first sequence, tile 3 represents approximately 33 % of the total encoding complexity of the frame, while tile 2 only represents about 20 %. For the second sequence, tiles 2 and 4. For the third sequence, tiles 1 and 3.

The analysis above refers to only one frame. For a better overview of the described problem, the analysis will be expanded. The graphics in figure 3.3 represent the encoding times of each tile, for the first 100 frames of each considered sequence. The theoretical ideal time for each tile is represented too, which is calculated by dividing the total time of the respective frame by four.

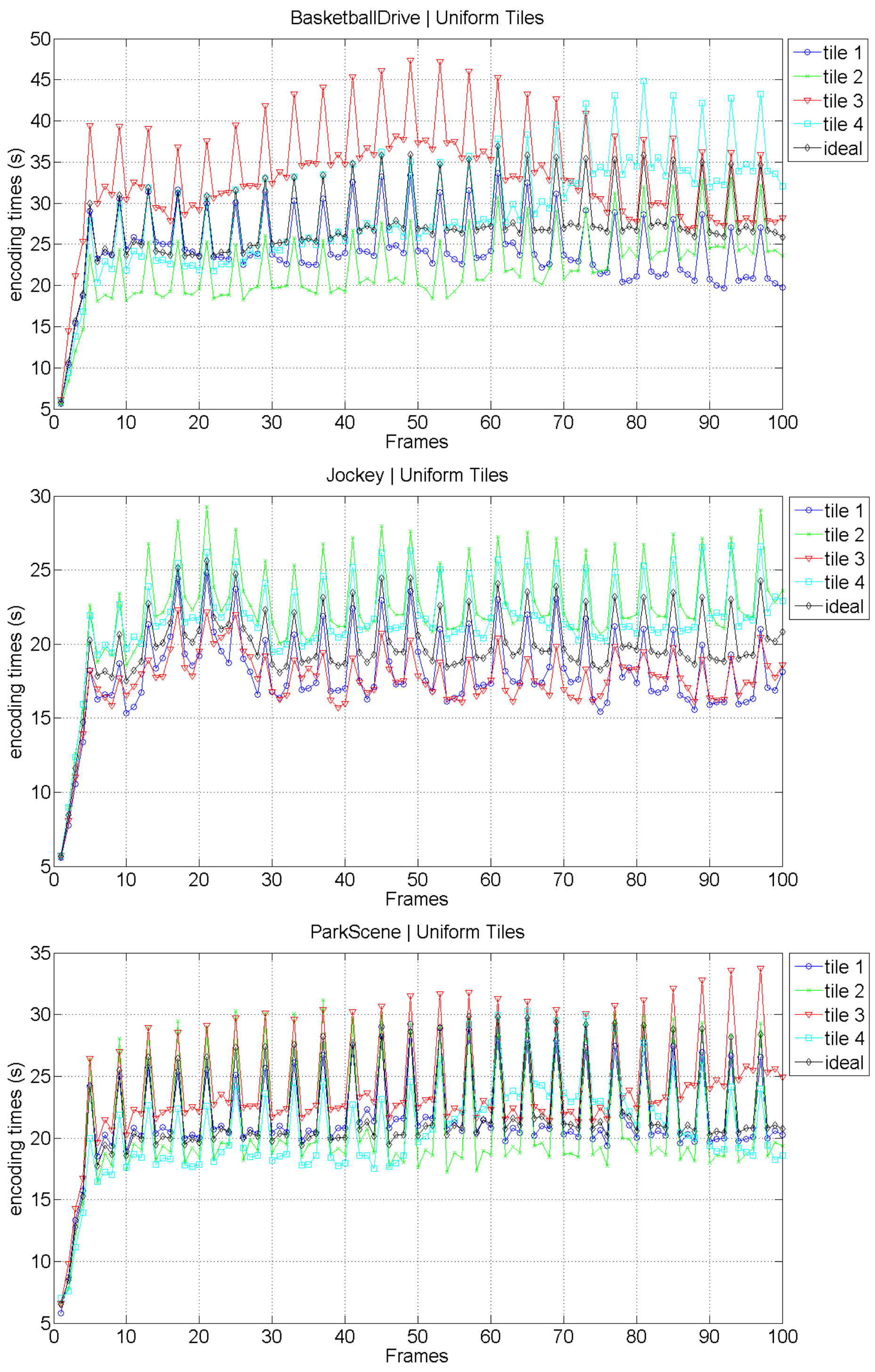

Figure 3.3: Encoding times per tile for the first 100 frames

The graphics show that in all three sequences, the encoding times deviate from the theoretical ideal time. This deviation is significant for the sequence *BasketballDrive*, with tile 3 having a difference of approximately 10 seconds from the ideal time on frame 40, 25.83 seconds. For the sequences *Jockey* and *ParkScene* the deviation from the ideal time is noticeable too. To complement this data, figure 3.4 shows the encoding time distribution per tile, in terms of percentage, for the first 100 frames of each sequence.

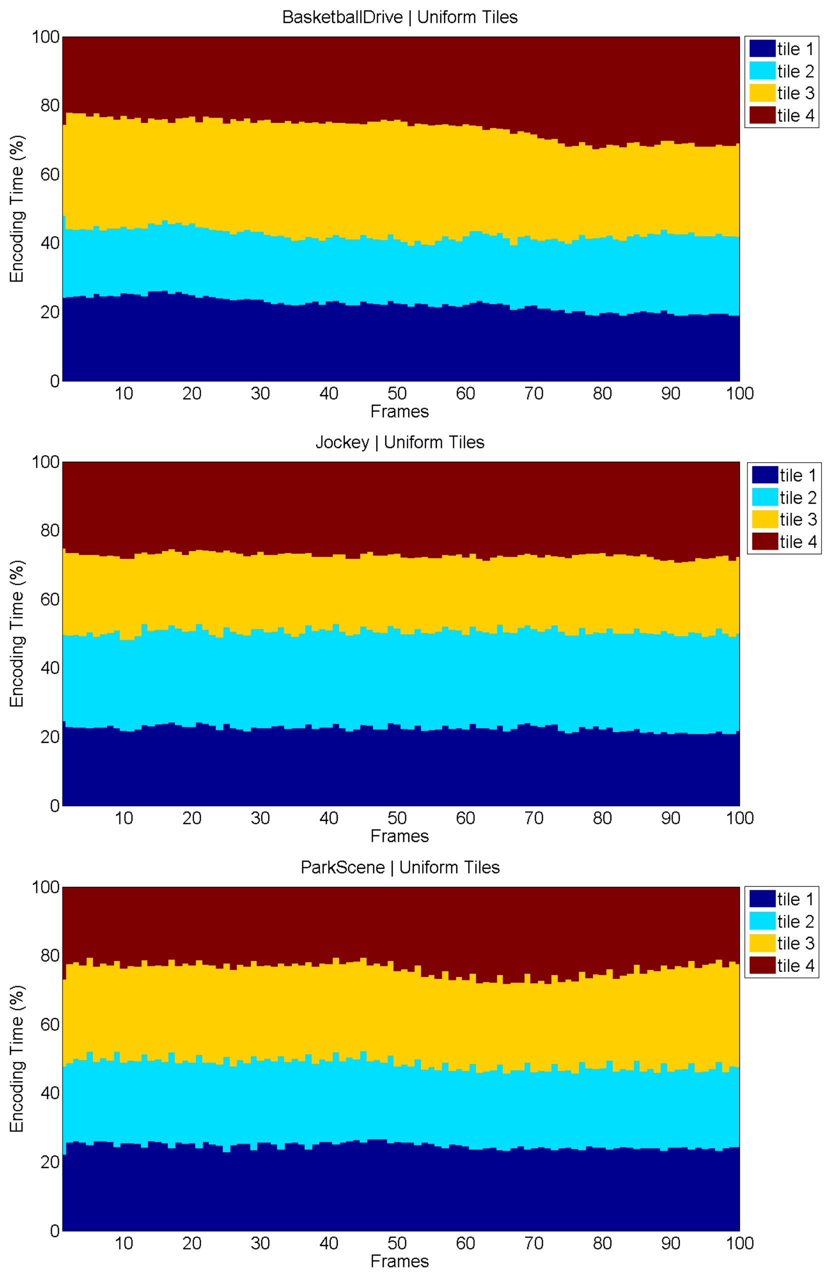

Figure 3.4: Encoding complexity distribution per tile

If we assume that each tile is assigned to a single processing unit, the load distribution using a tile configuration with same-sized tiles is not optimal. For instance, for the sequence *BasketballDrive*, the processing units assigned to tiles 2 and 4 would experience a lot of idle time, which equals wasted resources and longer processing times.

The experimental results presented in this section show that uniformly spaced tiles would lead to less than optimal results for certain video sequences. Therefore it's important to use an adaptive tile size adjustment method that ensures an even computational load distribution among the processing units.

3.2 Related Work

Researchers have been studying alternatives to better use parallel tools, such as tiles, to obtain a better workload balance. Several solutions have been proposed by various authors.

In [8], the authors propose a load-balancing method, that adjusts the number of CTUs per slice or tile, based on estimates of the complexity of prediction modes. The model used considers the normalized computational complexities, listed in table 3.3 based on the CU sizes and prediction modes used.

CU size	Merge	Inter	Intra
64x64	109	760	52
32x32	42	280	16
16x16	9	71	3
8x8	2	19	1

Table 3.3: Workload factors for the complexity estimation model (source [8]).

The complexities listed are used to estimated the CTU complexities according to equation 3.1.

$$
\begin{aligned}
CC_i(l) &= \textstyle\sum_S \sum_M CEM(s,m) \times CHK(s,m|l) \\
S &= \{s|64 \times 64, 32 \times 32, 16 \times 16, 8 \times 8\} \\
M &= \{m|MERGE, INTER, INTRA\} \\
CHK(s,m|l) &= \begin{cases} 1 & \text{if for } l\text{, the selected } s \text{ and } m \text{ apply} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.1}
$$

$CC(l)$ represents the predicted complexity of the l-th CTU. $CEM(s,m)$ is the workload factor of the CTU based on its size, s, and the prediction mode, m. The number of CTUs assigned to each slice or tile is then adjusted to evenly distribute the total encoding complexity. This algorithm obtained an average speed-up, in relation to a uniform partition, of 3.81% (maximum 11.48% and minimum of -0.67%) for tiles and an average of 12.05% (maximum 19.70% and minimum 1.33%) for slices.

The work described in [21] proposes a novel software architecture with a different approach. Instead of adjusting the tile sizes, it controls the workload of each one. Given some definitions by the user, the software automatically alters some parameters of the encoder in order to balance each tile's workload. Their goal, however, is in terms of power savings and not specifically optimizing the parallelization speed-up.

In [4] an Adaptive Tiling Algorithm for the HEVC standard is proposed. The main goal here is to reduce the encoding losses due to the use of tiles. The work automatically adjusts the tiles' boundaries to break context as little as possible. This proposed solution tries to minimize quality loss at the cost of a smaller speed-up gain. For a 2x2 tile partition scheme and four processing units, the algorithm only managed to get speed-ups of roughly 2, which is very low, well below the theoretical maximum of 4.

An additional study was presented in the same work, which related the number of tiles used with the impact on coding efficiency. The conclusion was that the higher the number of tiles, the more the loss in coding efficiency (see Table 3.4). This is due to the larger number of broken dependencies that happen when using more tiles. Results showed losses in coding efficiency up to 5.45% for Class A videos and up to 6.12% for Class B videos. Therefore it's important to keep the number of tiles to a minimum in order to preserve coding efficiency.

Class	**Nr. of Tiles (rows x cols)**	**Efficiency Loss** (%)
A	2x2	0.5
	2x4	0.8
	4x4	1.3
B	2x2	0.5
	2x4	1.0
	4x4	1.9

Table 3.4: Encoding efficiency losses for various tile sizes. Values taken from [4].

Considering the results of these previous state-of-the-art works, the work described in this thesis if focused on the use of tiles with the 2x2 configuration (4 tiles). This choice minimizes the coding efficiency losses and is the most simple tile partition to optimize. The next two chapters will describe in detail a proposed solution for this problem.

4 Tile Encoding Computational Complexity Estimation

When using tiles, they have to be defined prior to encoding a frame. However, since the encoding complexities of the CTUs in a frame are unknown before the actual encoding, it wouldn't be possible to correctly define the tile boundaries to fulfil the goal of balancing the workload among each tile.

To solve this problem, the encoding complexity has to be estimated. Some visual information dependent variables were studied, to understand if they could be used to estimate the encoding complexity of the CTUs. In this section, we use the Pearson's linear correlation coefficient, defined in equation 4.1, to calculate the correlation between the studied variables.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.1}$$

4.1 Spatiotemporal Index

The Spatiotemporal Index is comprised of two factors: the Spatial Index (SI) and the Temporal Index (TI) [23]. The SI measures the amount of spatial detail of a picture, while the TI measures the amount of temporal changes in a sequence of pictures. The Spatial Index is computed using the Sobel Filter. Each frame (luma plane) is filtered with the Sobel Filter and, afterwards, the standard deviation is calculated over the pixels of the filtered frame. The maximum value of this standard deviation measures the spatial detail of the scene. Refer to equation 4.2

$$SI = max(std_{space}[Sobel(F_n)]) \tag{4.2}$$

where F_n represents the nth frame of the sequence.

The Temporal Index is based on the inter-frame difference, which is basically the difference between pixel values with the same spatial location but in sucessive frames. TI is calculated by computing the standard deviation of the inter-frame differences, as shown in equations 4.3 and 4.4

$$TI = max(std_{space}[M_n(i,j)]) \tag{4.3}$$

where

$$M_n(i,j) = F_n(i,j) - F_{n-1}(i,j) \tag{4.4}$$

$F_n(i,j)$ represents the pixel located on the coordinates i, j of frame n.

An existing script [24] was modified in order to calculate the values for SI and TI at CTU level. SI_{CTU} represents the SI value of a CTU and is computed using an equation adapted from 4.2:

$$SI_{CTU} = max(std_{CTU}[Sobel(F_n)]) \tag{4.5}$$

The sobel filter is still applied to the whole frame, but the standard deviation is calculated individually for each CTU. This is similar for the TI, as the pixel differences are calculated for the whole frame but the standard deviation is done for each CTU. TI_{CTU} represents the TI of a CTU and its equation, adapted from 4.3, is the following:

$$TI_{CTU} = max(std_{CTU}[M_n(i,j)]) \tag{4.6}$$

where

$$M_n(i,j) = F_n(i,j) - F_{n-1}(i,j) \tag{4.7}$$

In order to evaluate the viability of the SI and TI in estimating the encoding complexity, some studies had to be carried out. On a software-level, the best indicator of the encoding complexity is the elapsed encoding time. Therefore, if the SI and TI are correlated to the encoding times, this means that they can be used for the estimation of the encoding complexities.

For the following experiments, the same video sequences used in chapter 3, presented in table 3.1 were used, under the same encoding conditions as described in that chapter. The first experiment consisted of studying the relationship between SI and TI and the encoding times of a single frame. Figure 4.1 shows the encoding times, as well as the SI and TI values per CTU of the sixth frame of the sequence *BasketballDrive.* At first sight it is evident that both SI and TI aren't directly related to the encoding times. CTUs of the frame with higher values of SI/TI don't match the CTUs with higher encoding times.

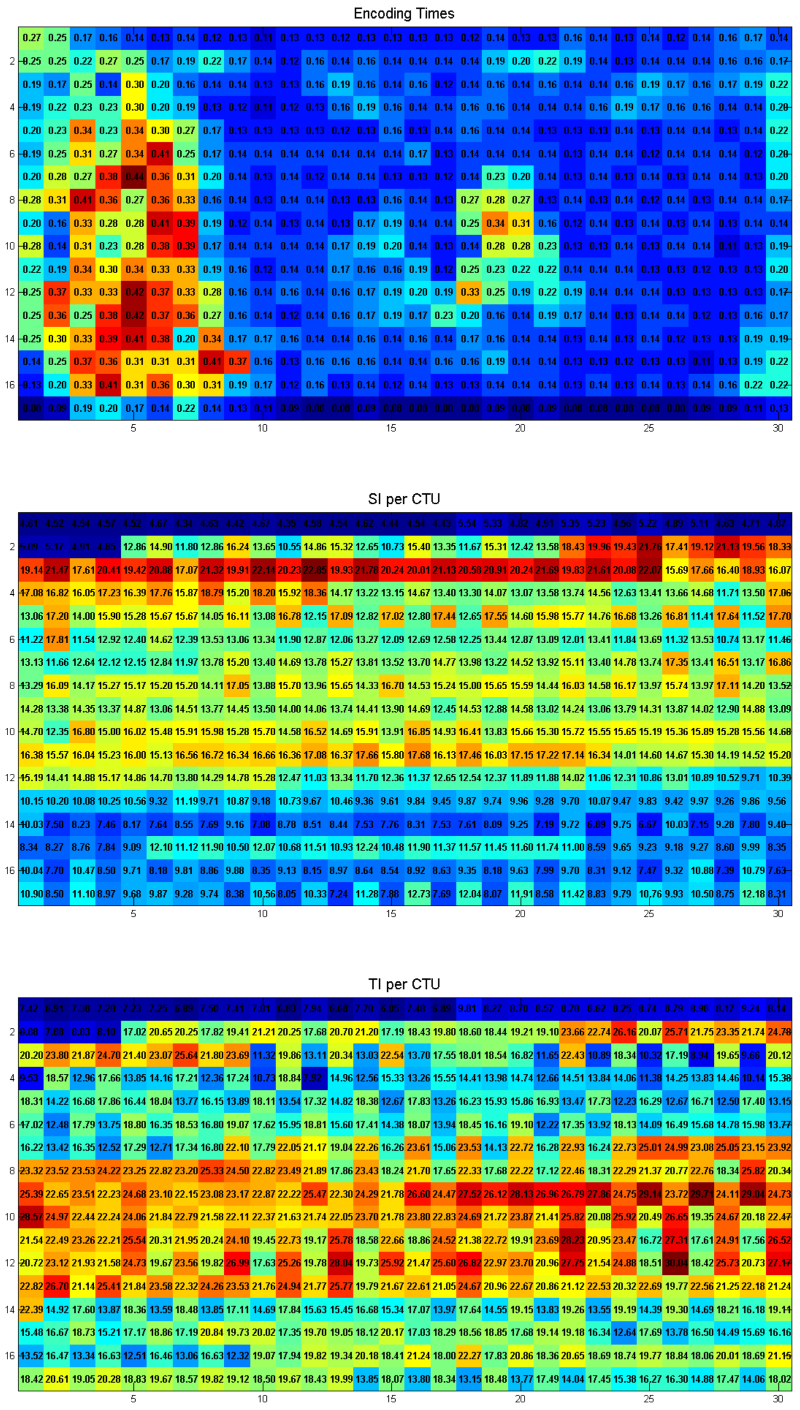

Figure 4.1: Encoding times, SI and TI values for the sixth frame of *BasketballDrive*

Table 4.1 shows the correlation coefficients between encoding times and the SI/TI values for the sixth frame of each used test sequence. It confirms the low correlation between encoding times and SI/TI, with values not higher than 0.3.

Sequence	**Correlation**	
(Sixth frame)	SI	TI
BasketballDrive	0.041	0.124
Jockey	0.240	0.290
ParkScene	-0.122	-0.025

Table 4.1: Correlation coefficients between encoding times and SI/TI for the test sequences (sixth frame)

For a more extended analysis, the second experiment consisted in repeating the previous experiment for the first 100 frames of each sequence. The correlation coefficients between encoding times and the SI/TI values are presented in table 4.2.

Sequence	**Correlation**	
100 frames	SI	TI
BasketballDrive	0.2135	0.419
Jockey	0.318	0.355
ParkScene	0.047	0.17

Table 4.2: Correlation coefficients between encoding times and SI/TI for the test sequences (first 100 frames)

In this case the correlation is higher. However the correlation coefficients are below 0.5, which means the correlation between encoding times and SI/TI is stil low. From these experiments we can conclude that the Spatiotemporal Index is a poor factor to estimate the encoding complexity.

4.2 Encoding Times and Bitrate

As mentioned before, at software-level, the elapsed time is a good indicator of the encoding complexity. Encoding times of already encoded frames are accessible. Therefore, the encoding times of previous encoded frames can possibly be used to estimate the encoding times of the current frame.

To evaluate the potential of using encoding times of previous frames as a way to estimate the encoding complexity, the correlation between encoding times of successive frames was studied. The same test sequences as in the previous section, under the same encoding conditions, were used. Table shows the correlation between encoding times of the CTUs in the 8th frame of each sequence and the co-located CTUs in the respective 6th and 7th frames.

Sequence	**Correlation**	
(8th Frame)	**6th Frame**	**7th Frame**
BasketballDrive	0.89	0.94
Jockey	0.89	0.94
ParkScene	0.83	0.90

Table 4.3: Correlation coefficients between encoding times of CTUs in the 8th frame and the co-located CTUs in the 6th and 7th frames for the test sequences

.

The results presented show that the correlation between encoding times of co-located CTUs in successive frames is very high. Between the 7th and 8th frame the correlation coefficient is above 0.9, while between the 6th and 8th frame it is slightly below 0.9. It is safe to say that the encoding times from previously encoded frames can be used for estimating the encoding complexity.

The bitrate is another potential indicator for estimating the CTU complexities. Each CTU of a frame requires a certain amount of bits to be encoded. Table 4.4 shows, for each sequence, the correlation coefficient between the amount of bits required to encode each CTU of the 6th frame and their respective encoding times.

Sequence (6th frame)	**Correlation**
BasketballDrive	0.771
Jockey	0.84
ParkScene	0.77

Table 4.4: Correlation coefficients between bitrate and encoding times of CTUs in the 6th frame for the test sequences

The results show that the correlation between encoding times and the amount of bits used to encode a CTU is high. Since the encoding times, as seen by the previous results, can be used to estimate the encoding complexity and the bitrate is highly correlated to the encoding times, then both can be used for the same complexity estimation purpose.

The results presented in this chapter show that the encoding times and the bitrate from already encoded frames can be used to estimate the encoding complexity, while the Spatiotemporal Index (SI and TI) is not a good way to do so. The next chapter will describe in detail the various methods used for complexity estimation as well as the proposed load-balancing method using tiles.

5 Load Balancing through Tile Geometry Adjustment

When distributing the processing of tiles among processors, to be encoded in parallel, the workload is an important factor to consider. If it's well distributed among the processing units used, this will translate into better efficiency in terms of speed-up.

To solve the problem of load balance when using parallelism tools in HEVC, such as Tiles, a method is proposed that dynamically adjusts the geometry of the tiles. The goal is to change the tile boundaries in order to obtain a better load distribution per tile during the encoding process. In case of each tile being assigned to a different processor to be encoded, this prevents some processors from being overloaded.

The method presented in this work comprises two steps: Complexity Estimation and Tile Geometry Adjustment. The first step estimates the encoding complexities of the frame to be encoded. Then the tile geometry is adjusted according to the estimated values. Figure 5.1 presents a simple diagram of our proposed method.

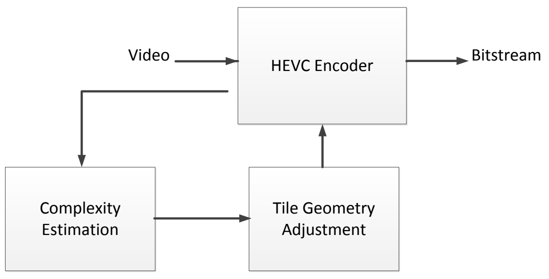

Figure 5.1: The proposed method.

5.1 CTU Encoding Complexity Estimation

As seen in the previous chapter, the encoding times and bitrate of previously encoded frames can be used to estimate the encoding complexity of the current frame CTUs. In this section four methods are presented, based these variables, to estimate the encoding complexities of each CTU in a frame. They are described in more detail below.

5.1.1 Total Encoding Time

In this method each CTU's complexity is estimated directly from the sum of the encoding times of co-located CTUs in all frames of the previously encoded GOP, as shown in equation 5.1.

$$\hat{C}_{m,n,t} = T_{m,n,t-1} + T_{m,n,t-2} + ... + T_{m,n,t-G} \tag{5.1}$$

$\hat{C}_{m,n,t}$ represents the estimated complexity of the CTU on position m, n, $T_{m,n,t}$ is the complexity of the CTU located in the same position on frame in the instant t and G is the GOP size.

The purpose of this method is to estimate the complexity based on an entire GOP, as this takes in consideration the evolution of the encoding complexity. Therefore, the adjustment of the tile geometry is performed at the start of each new GOP, except for the first one since there are no previously encoded frames. For a more simple presentation, the name of this method is shortened as **TET**.

5.1.2 Linear Combination of Encoding Times

This method estimates the complexity of each CTU by performing a weighted linear combination of the encoding times of the co-located CTUs in the two previously encoded frames (5.2).

$$\hat{C}_{m,n,t} = \beta_1 T_{m,n,t-1} + \beta_2 T_{m,n,t-2} \tag{5.2}$$

β_1 and β_2 are calculated according to equation 5.3.

$$\beta_1 = \frac{|\Delta T_{m,n,t-1}|}{|\Delta T_{m,n,t-1}| + |\Delta T_{m,n,t-2}|}; \beta_2 = 1 - \beta_1 \tag{5.3}$$

where

$$\Delta T_{m,n,t-1} = T_{m,n,t-1} - T_{m,n,t-2}$$

$$\Delta T_{m,n,t-2} = T_{m,n,t-2} - T_{m,n,t-3}$$

β_1 and β_2 determine which of the previous two frames has more weight in the estimation. For example, if $\Delta T_{m,n,t-1} < \Delta T_{m,n,t-2}$ then $\beta_1 < \beta_2$, meaning the frame on $t-1$ will have a greater weight on the complexity estimation that the frame on $t-2$. Only B or P frames can be used for the estimation process, I-frames are ignored due to the complexities being significantly different.

This method can be applied either at the start of each GOP (except the first one) or for each frame, except the first three B or P frames since this estimation process requires three previously encoded frames to work. This mmethod is named **Linear Combination of Encoding Times per GOP** (**LCTG**) if applied at the start of each GOP, and **Linear Combination of Encoding Times per Frame** (**LCTF**) if applied after each frame.

5.1.3 Linear Combination of Bits

This approach is similar to the previous one, except that it performs the estimation based on the amount of bits used to encode each CTU instead of the encoding times. While encoding times can vary slightly, depending on the machine the software runs on, the number of bits used to encode a CTU does not depend on that. The equations are shown below (5.4 and 5.5)

$$\hat{C}_{m,n,t} = \beta_1 B_{m,n,t-1} + \beta_2 B_{m,n,t-2} \tag{5.4}$$

$$\beta_1 = \frac{|\Delta B_{m,n,t-1}|}{|\Delta B_{m,n,t-1}| + |\Delta B_{m,n,t-2}|}; \beta_2 = 1 - \beta_1 \tag{5.5}$$

where

$$\Delta B_{m,n,t-1} = B_{m,n,t-1} - B_{m,n,t-2}$$

$$\Delta B_{m,n,t-2} = B_{m,n,t-2} - B_{m,n,t-3}$$

$B_{m,n,t}$ is the amount of bits used to encode the CTU located at coordinates m, n on a frame at instant t. The formula was adapted from the previous method because, as seen before, the correlation between the bits used to encode a CTU and their respective encoding time is high.

Simiar to the Linear Combination of Encoding Times, only B or P-frames can be used for the estimation process. This method can be applied either at the start of each GOP (except the first one) or for each frame, except the first three B or P frames. This method is named **Linear Combination of Bits per GOP** (**LCBG**) if applied at the start of each

GOP, and **Linear Combination of Bits per Frame** (**LCBF**) if applied at the start of each frame.

5.1.4 Simple Linear

The extrapolation approach assumes a linear evolution of the CTU complexities. Having the data from the co-located CTUs from the previous two frames, the estimation is performed applying a linear extrapolation as seen in equation 5.6.

$$\hat{C}_{m,n,t} = T_{m,n,t-1} + (T_{m,n,t-1} - T_{m,n,t-2}) = 2.T_{m,n,t-1} - T_{m,n,t-2} \tag{5.6}$$

This method is always applied for each frame, except the first two P or B frames since this method requires at least two encoded frames. Once again, I frames cannot be used for the estimation. For a more simple presentation, the name is shortened as **SL**.

In the next section, the actual tile adjustment algorithm will be described in detail.

5.2 Tile Geometry Adjustment

This section proposes an algorithm to automatically ajust the tile boundaries to balance the encoding complexity among the four tiles. Being a 2x2 tile configuration, there are two boundaries, one horizontal and one vertical, and their intersection forms a point (figure 5.2). W and H represent the width and height, respectively, in CTUs of the video frame.

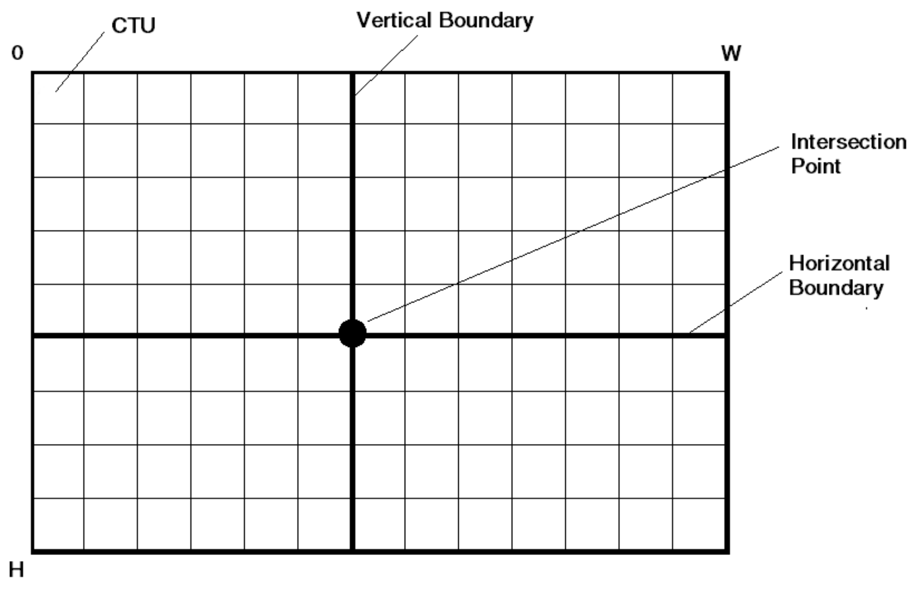

Figure 5.2: The tile boundaries and their intersection point.

The algorithm comprises these four steps:

- selection of a starting point for the tile configuration;

- define an additional set of points. These points are called **candidate points**;

- test all tile configurations that are defined with the starting point and the candidate points;

- select the point that guarantees the optimal tile configuration.

Figure 5.3 shows the flowchart for the algorithm, that will be described in more detail afterwards.

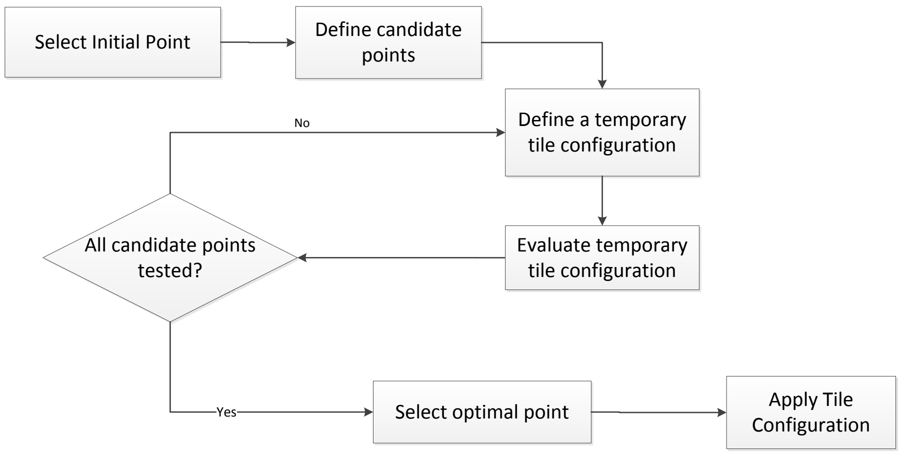
Select Initial Point
Define candidate points
Define a temporary tile configuration
No
Evaluate temporary tile configuration
All candidate points tested?
Yes
Select optimal point
Apply Tile Configuration

Figure 5.3: Flowchart of the algorithm for adjusting the tile geometry.

Following the complexity estimation step, the goal is to adjust the tile configuration so that the total encoding complexity of all CTUs within a tile is approximately equal to 25% of the encoding complexity of the whole frame

To find the optimal positions for the new tile boundaries, the most accurate method would do an exaustive search of all possible interception points. However, in order to limit the number of points to check, an alternate method is proposed. For the first step, an initial point for the tile boundaries is selected. The selection is done so that the sum of all estimated CTU complexities of all columns on the left of the point is similar to the sum of the columns on the right. In a similar way, the sum of all rows above the point has to be similar to the sum of all rows below it. Considering $colS_m$ to be the sum of the estimated CTU complexities of column m and $rowS_n$ to be the sum of the estimated CTU complexities of row n, the goal is to select a point located on column i and row j, so that the conditions in equation 5.7 occur.

$$\begin{gathered} \textstyle\sum_{m=1}^{i} colS_m \approx \sum_{m=i+1}^{W} colS_m \\ \textstyle\sum_{n=1}^{j} rowS_n \approx \sum_{m=j+1}^{H} rowS_n \end{gathered} \tag{5.7}$$

For a more simple presentation, consider

$$\sum_{m=1}^{i} colS_m = LeftSum_i$$

$$\sum_{m=i+1}^{W} colS_m = RighSum_i$$

$$\sum_{n=1}^{j} rowS_n = TopSum_j$$

$$\sum_{m=j+1}^{H} rowS_n = BottomSum_j$$

Starting with $i = 1$ and $j = 1$, in each iteration the algorithm calculates the difference between $LeftSum_i$ and $RightSum_i$, as well as the difference between $TopSum\ BottomSum$. Refer to figure 5.4 and equation 5.8.

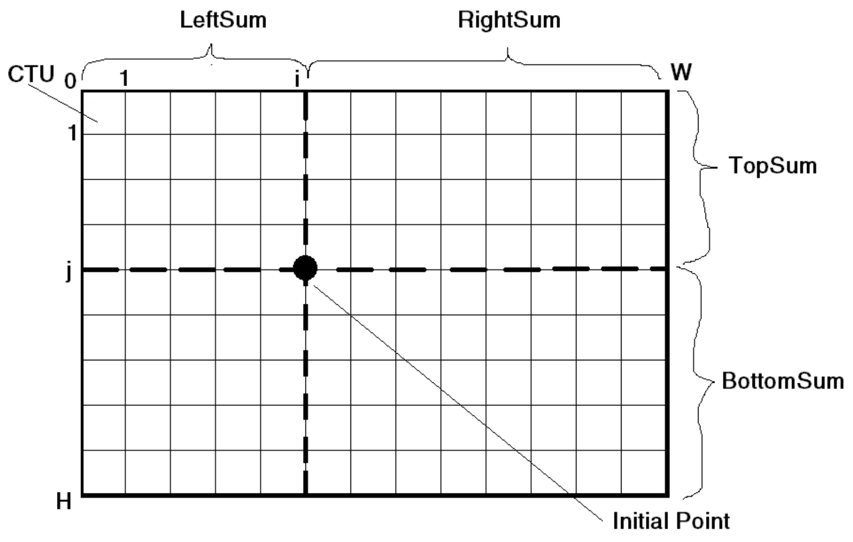

Figure 5.4: Illustration of the initial point selection.

$$\begin{aligned} D_c(i) &= |LeftSum_i - RightSum_i| \\ D_r(j) &= |TopSum_j - BottomSum_j| \end{aligned} \tag{5.8}$$

The algorithm stops when $D_c(i + 1) \geq D_c(i)$ and $D_r(j + 1) \geq D_r(j)$. The values of i and j that ensure this, form the initial point. This process is described in the pseudo-code shown in Algorithm 1.

From the initial point, a set of candidate points is defined. Those candidate points are neighbouring points located around the initial point (figure 5.5). The algorithm then proceeds to evaluate the candidate points as well as the initial point itself.

Algorithm 1 Initial Point Selection

```
Input: Width $W$, Height $H$, estimated CTU complexities $\hat{C}_{m,n}$
  for each column $m$ and row $n \in \hat{C}_{m,n}$ do
      $colS_m \leftarrow$ sum of CTU complexities of column $m$
      $rowS_n \leftarrow$ sum of CTU complexities of row $n$
  end for
  $i \leftarrow 1$
  $j \leftarrow 1$
  $colDiff \leftarrow |\sum colS_i - \sum_{m=i+1}^{W} colS_m|$
  $rowDiff \leftarrow |\sum rowS_j - \sum_{n=j+1}^{H} rowS_n|$
  $minDiff \leftarrow false$
  while $i < W$ and $minDiff == false$ do
      $i \leftarrow i + 1$
      $D_c \leftarrow |\sum_{m=1}^{i} colS_m - \sum_{m=i+1}^{W} colS_m|$
    if $D_c \geq colDiff$ then
      $minDiff \leftarrow true$
    else
      $colDiff \leftarrow D_c$
    end if
  end while$minDiff \leftarrow false$
  while $j < H$ and $minDiff == false$ do
      $j \leftarrow j + 1$
      $D_r \leftarrow |\sum_{n=1}^{j} rowS_n - \sum_{n=j+1}^{W} rowS_n|$
    if $D_r \geq rowDiff$ then
      $minDiff \leftarrow true$
    else
      $rowDiff \leftarrow D_r$
    end if
  end while
  return $i, j$
```

Each point (initial and candidates) represent one possibility for the interception of the new tile boundaries. The goal is to choose the optimal point among them, which is the one that grants the better distribution of the encoding complexity among all tiles. In an ideal situation, each tile would have 25% of the frame's total encoding complexity. Therefore, the algorithm creates, for each point, a temporary tile division and evaluates how far from the ideal value the total encoding times for each tile are (equations 5.9, 5.10 and 5.11).

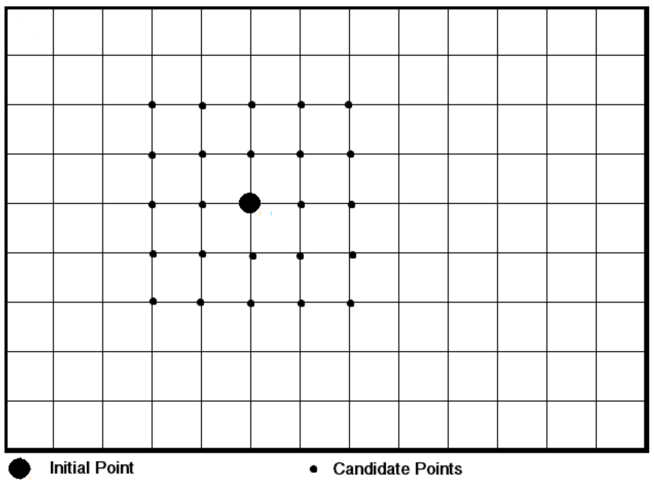

Figure 5.5: The initial point and the resulting candidate points for the new tile boundaries

$$D_N(P) = |TL_N - \frac{\hat{C}_{tot}}{4}|, N = 1, 2, 3, 4 \tag{5.9}$$

$$TL_N = \sum_{c \in TileN} \hat{C}(c) \tag{5.10}$$

$$D_{Tot}(P) = \sum_{N=1}^{4} D_N(P) \tag{5.11}$$

TL_N represents the encoding complexity of the tile N, $\hat{C}(c)$ represents the estimated encoding complexity of the CTU c, $\hat{C}_{tot}$ is the sum of all estimated CTU complexities and P represents the point that is used for the temporary tile division. This step is repeated for each point described above. The optimal point is the one with the lowest value for D_{Tot}, or as shown in equation 5.12.

$$P_{opt} = arg \min_{P} D_{Tot}(P) \tag{5.12}$$

P_{opt} defines the point where the new tile boundaries will intercept. This will be the configuration for the next frame or GOP. This process is described in Algorithm 2.

Algorithm 2 Optimal Point selection

```
Input: Estimated CTUs $\hat{C}_{i,j}$, Complexity total $\hat{C}_{tot}0$, Candidate Points $P$
  for each candidate point, $P$ do
      $TL_N \leftarrow$ define tiles $TL_{1...4}$ with boundaries passing through $P$
      $D_N \leftarrow$ calculate $|TL_N - \frac{\hat{C}_{tot}}{4}|$, $N = 1, 2, 3, 4$
     if $sum(D_N) < prevSum$ then
      $P_{opt} = P$
      $prevSum = sum(D_N)$
     end if
  end for
return $P_{opt}$
```

The optimizing partitioning is then used by the encoder to define the new tile configuration. If the new optimal partitioning is the same as the previous one, in other words, if the tile configuration remains the same, no values are transmitted to the encoder. This prevents an unecessary addition of header information. The next chapter will present the results and analysis of this algorithm.

6 Experimental Results

All experiments described next were conducted on a clustered computer based on Intel XeonE5520 (2.27 GHz) processors running on Windows Server 2008 HPC operating system. For the experimental setup, the algorithm described in chapter 5 was implemented in the HM 16.0 reference software. Then, for each proposed complexity estimation method, a modified version of the software was created in order to test the method. All video sequences used are class B (1920x1080), since the use of parallelism is only beneficial on high definition videos as those usually require a lot of processing time and power. The Low Delay P encoding configuration was chosen, as it guarantees that the encoding order of the frames is the same as the playback order. No Class A videos (2560x1600) were used because the Low-Delay configuration does not support it [25].

Sequence Name	**Frame Count**	**FPS**
BasketballDrive	500	50
BQTerrace	600	60
Cactus	500	50
Jockey	600	120
Kimono	240	24
ParkScene	240	24

Table 6.1: Used video sequences

Each video sequence was encoded using four tiles in a 2x2 arrangement. For the initial configuration we use equally-sized tiles. All videos were then encoded, using the configuration suggested by the Common Test Conditions [3]. For parallelism-evaluation purposes, it is assumed that each tile is assigned to a single processing unit. To evaluate how well the algorithm proposed in the previous chapter performs, we will analyse it from three different perspectives: effectiveness in load-balancing, the estimated parallel speed up and the impact in coding efficiency. The results are then compared to existing works described in the literature.

6.1 Effectiveness of the proposed Load-Balancing Algorithm

The proposed algorithm automatically adjust tiles to aim for a better workload distribution.

For an efficient distribuition of workload, as stated before, the main goal is for tiles to have similar encoding complexities to maximize the parallel efficiency. The ideal value would be 25% of the total frame time per tile. To evaluate how distant each tile is from that value, when adjusted using the proposed algorithm, we use the **Mean Absolute Difference from the Ideal Time** (**MADIT**), which is defined in equation 6.1.

$$MADIT = \frac{1}{NF}\sum_{f=1}^{NF}\sum_{N=1}^{4}|T_{N,f} - \frac{FT_f}{4}| \tag{6.1}$$

NF is the number of frames of the video sequence, $T_{N,F}$ the time used to encode tile N of frame f and FT_f is the total time used to encode frame f. In short, higher values of MADIT mean worse distribution of the encoding complexity among the tiles and, therefore, less load-balancing efficiency. Table 6.2 shows the values for the MADIT for various test sequences (QP=32), using the various methods for complexity estimation described in chapter 5. The results are compared with the uniform tile partition.

Sequence	**MADIT (seconds)**						
	LCTG	**LCTF**	**LCBG**	**LCBF**	**TET**	**SL**	**Uniform**
BasketballDrive	6.4057	6.0814	6.4490	**6.0545**	6.5464	8.6056	17.6234
BQTerrace	4.48	4.39	4.53	4.36	**3.76**	7.88	3.90
Cactus	8.53	**7.69**	8.69	7.82	7.95	11.07	13.81
Jockey	5.3088	**4.7605**	5.3046	4.9764	5.2205	6.8844	9.5922
Kimono	**4.4570**	4.6235	4.5239	4.7062	4.4245	6.4850	6.1146
ParkScene	6.5515	6.5060	6.9133	6.4969	**5.9686**	9.4553	7.0617

Table 6.2: MDIT for each test sequence

From the obtained results we can see that by performing an adjustment of the tile geometry is, most of the time, more efficient than the uniform tile configuration. The worst complexity estimation method is clearly the SL method, which has worse results than the uniform for the *BQTerrace*, *Kimono* and the *ParkScene* sequences and has worse results than every other complexity estimation method for all sequences.

To better compare with the experiments done in chapter 3 regarding the use of same-sized tiles, figure 6.1 presents the encoding times per tile, adjusted with the proposed algorithm, for the test sequences, *BasketballDrive*.

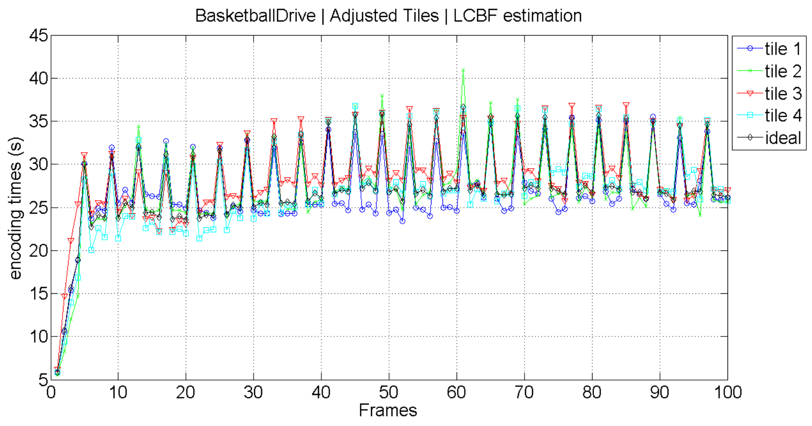

Figure 6.1: Encoding Times per tile for the first 100 frames of the sequence *BasketballDrive*.

By observing the graphics, we can conclude that the encoding times for each tile are close to the ideal value (black line), meaning that the tile geometry adjustment achieves a better workload balance. Note that the encoding time peaks correspond to frames at the start of each GOP, as they are encoded with more information than the rest of the frames in the GOP. Note that only the first frame of the whole sequence is an I-frame, which explains the low encoding times at the beginning. Figure 6.2 gives additional proof of the better workload balance for the same sequence. For the graphics of the other test sequences, refer to Appendix B.

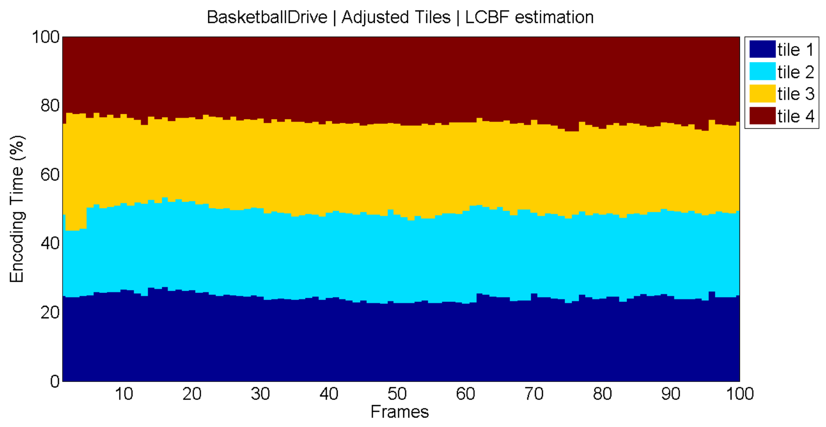

Figure 6.2: Distribuition of the workload per tile for first 100 frames of the sequence *BasketballDrive*.

For a better illustration of the tile adjustment, figure 6.3 represents 3 frames with the respective adjusted tile boundaries from the sequence *BasketballDrive*.

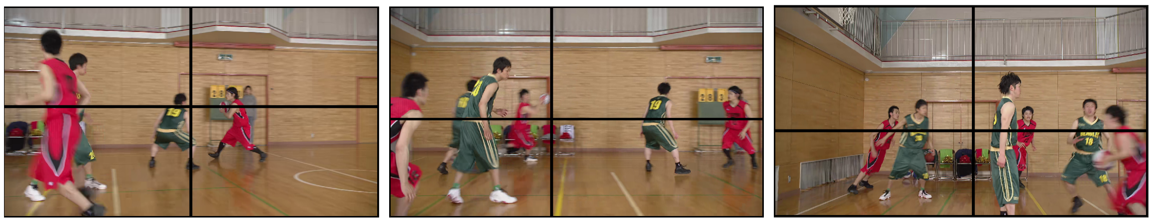

Figure 6.3: Adjustment of the tile boundaries for the sequence *BasketballDrive* for frames 4, 40 and 311.

The first picture, corresponding to frame 4, the boundaries are uniform. This is because it's still the beginning of the sequence and thus the configuration is still the initial one. In the second picture, however, the vertical boundary has shifted towards the left side of the picture while the horizontal one went slighly down. This is due to the center of the action being focused more on the left side, making it more complex in terms of encoding. In the third picture that action scene is more concentrated in the right section, therefore the boundaries shifted to the right side.

The results obtained show that the proposed algorithm is able to effectively adjust the tile boundaries to get a better workload balance. However, the effetiveness can be better or worse depending on the video sequence. Overall, the adjustment of the tile geometry with the proposed algorithm obtains better results in terms of load-balancing compared to the use of same-sized tiles.

6.2 Estimated Parallel Speed Up

As it was shown in the previous section, the proposed algorithm is able to achieve improvements over the uniform tile partition in terms of workload distribution, by dynamically adjusting the tile geometry. Consequently this will lead to better parallelism results. However, the HM reference software does not support tools such as multithreading to directly test and measure the parallel speed-up. Therefore, this has to be done via an estimation.

The Parallel Speed-up, PSU, can be calculated by diving the elapsed time of the encoding process when run on a single processor, sequentially, by the elapsed time when run on various processors (equation 6.2)

$$PSU = \frac{T_{sequential}}{T_{parallel}} \tag{6.2}$$

During the video encoding process, only the frame encoding can be done in parallel. Thus, $T_{parallel}$ can be divided into $ParallelTime$ and $ExtraTime$ as seen in equation 6.3.

$$PSU = \frac{T_{sequential}}{(ParallelTime) + (ExtraTime)} \tag{6.3}$$

$TotalTime$ is the elapsed time used to encode the whole video sequentially. $ParallelTime$ is the time needed to encode all frames if their processing was done in parallel. Assuming that all four tiles of a frame begin their encoding at the same time and the next frame cannot start processing until the whole current frame is encoded, $ParallelTime$ corresponds to the elapsed time of the frame encoding when done in parallel. This is calculated by only considering, for each frame, the encoding time of the tile with the largest value for that frame and then adding them(equation 6.4).

$$ParallelTime = \sum_{f=1}^{NF} \max_{N}(TL_{N,f}), N = 1, 2, 3, 4 \tag{6.4}$$

NF means Number of Frames and $TL_{N,f}$ is the encoding time of tile N of frame f. Note that only the frame encoding is done in parallel, the rest of the video encoding process is always done sequentially. This part is represented by $ExtraTime$ and is calculated as shown in equation 6.5.

$$ExtraTime = T_{sequential} - \sum_{f=1}^{NF} \sum_{N=1}^{4} TL_{N,f} \tag{6.5}$$

Note that for four tiles and four processors, $PSU \leq 4$. This limit represents the perfect result for this case, which in reality is never achieved. The goal is to make the speed up as close as possible to it.

Another term used in this section, to show and compare results, is the Average Time Saving (ATS) [20]. It is calculated using the following equation:

$$ATS(\%) = \frac{T_{uni} - T}{T_{uni}} \times 100 \tag{6.6}$$

T_{uni} represents the elapsed time of encoding a video sequence, in parallel, using same-sized tiles. T is the elapsed time of the same procedure, but with the tile boundaries adjusted using the proposed method. This value can, alternatively be calculated using equation 6.7.

$$ATS(\%) = \frac{PSU - PSU_{uni}}{PSU} \times 100 \tag{6.7}$$

PSU is the parallel speed-up obtained using the proposed method and PSU_{uni} is the speed-up obtained using uniformly-spaced tiles. To prove that both equations are equal, replacing PSU by the definition in equation 6.2, we get equation 6.8.

$$\frac{PSU - PSU_{uni}}{PSU} = \frac{T}{T} - \frac{T}{T_{uni}} \tag{6.8}$$

The expression on the right can be written as shown below, obtaining equation 6.9.

$$\frac{T.T_{uni} - T^2}{T.T_{uni}} = \frac{T_{uni} - T}{T_{uni}} \tag{6.9}$$

Therefore, it is proven that the ATS value can be calculated using the PSU values, according to equation 6.7.

The results for the tested video sequences are summarized on Table 6.3, showing the PSU values obtained for each estimation method and for using same-sized tiles. For the ATS, we show the best value (corresponding to the best PSU) for each sequence.

Sequence	**PSU**						**PSU (Uni)**	**ATS (%)**
	LCTG	**LCTF**	**LCBG**	**LCBF**	**TET**	**SL**		
BasketballDrive	3.56	3.56	3.57	**3.58**	3.55	3.46	3.07	14.25
BQTerrace	3.56	3.57	3.56	3.56	**3.60**	3.37	3.57	0.84
Cactus	3.34	**3.41**	3.34	3.39	3.40	3.25	3.23	5.28
Jockey	3.55	**3.59**	3.56	3.58	3.57	3.48	3.34	6.96
Kimono	**3.63**	3.62	3.61	3.61	3.62	3.52	3.46	4.68
ParkScene	3.46	3.45	3.44	3.46	**3.50**	3.31	3.34	4.57
Average	3.52	3.53	3.51	3.53	**3.54**	3.40	3.33	6.1

Table 6.3: Estimated Speed Ups using adjusted tiles and uniformly-spaced tiles

Results show that by adjusting the tile boundaries, with the proposed method, we can obtain an improvement over the usage of same-sized tiles. The TET estimation method has the best average PSU, with 3.54, while the SL method has the lowest, 3.40. It is still an improvement over the average PSU value obtained when using same-sized tiles (3.33). Considering the best estimation method for each sequence, it is possible to obtain an average ATS value of 6.1%. *BasketballDrive* has the highest ATS value (14.25%) while *BQTerrace* has the lowest (0.84%). The reasons for the low value is the fact that using same-sized tiles guarantees an already large PSU value (3.57), making further improvements difficult.

Adjusting the tile boundaries seems to have a good impact in load-balancing, capable of speeding up the encoding process if used in a parallel environment. However, as stated before, the usage of tiles breaks coding dependencies on their boundaries. This leads inevitably to a loss in coding efficiency.

6.3 Impact in Coding Efficiency

Tiles have an impact in the coding efficiency of HEVC, due to boundaries breaking coding dependencies. That impact needs to be analyzed to evaluate if it's viable to use tiles with the proposed method.

The results regarding coding efficiency are measured in terms of BD-rate increase [26], compared to the reference with no tile partitions (less is better). Table 6.4 summarizes the results for the used test sequences for each estimation method. The underlined values correspond to the estimation method that obtained the best PSU value (table 6.3).

Sequence	**BD-rate (%)**						**BD-rate (uniform)(%)**
	LCTG	**LCTF**	**LCBG**	**LCBF**	**TET**	**SL**	
BasketballDrive	**1,0**	1,1	**1,0**	1,1	**1,0**	1,4	0,8
BQTerrace	0,4	0,5	**0,3**	0,6	0,4	1,2	0,3
Cactus	**0,2**	0,7	0,3	0,7	0,3	1,1	0,3
Jockey	**1,8**	2,4	**1,8**	2,4	1,9	3.4	1,5
Kimono	**1,1**	1,4	**1,1**	1,4	**1,1**	1,7	1,0
ParkScene	0,5	0,9	**0,4**	0.9	**0,4**	1,2	0,3
Average	0.83	1.17	**0.82**	1.18	0.83	1.67	0.7

Table 6.4: BD-rate results for the test sequences

Results show that dynamically adjusting the tile boundaries causes a larger impact in coding efficiency compared to the use of same-sized tiles. The LCBG estimation method has the lowest increase in BD-rate, with 0.82%) while the SL has the highest, 1.67%. In comparison, using same-sized tiles causes an average increase in BD-rate of 0.7%. This means that the difference, in terms of BD-rate increase, between our proposed method and using same-sized tiles is not higher than 0.97%. Considering the improvement in the ATS, this difference is not very significant. It is interesting to note that for the *Kimono and the ParkScene* sequences, the methods that achieve the best result in terms of ATS have the lowest increase in BD-rate as well.

6.4 Comparison with other works

In this section the obtained results are compared with other existing solutions, for a better evaluation of their performance. Both the ATS and BD-rate results are compared with the correspondent values from [8] (labeled as **A**) and [4] (labeled as **B**). The solution proposed in A focuses in the improvement of parallel efficiency, that is, improving the ATS. On the other hand, the solution propposed in B aims to minimize the BD-rate increase.

In Table 6.5 the results for each work are summarized for a better comparison. The column *Our* contains the lowest BD-rate values for each sequence (bold values in table 6.4), while the column *Our(best PSU)* contains the BD-rate obtained when using the estimation method that guarantees the best PSU value (italic values in table 6.4).

The values between square brackets represent the difference between the BD-rate of using same-sized tiles and the listed value. This is used to compare with A, as their authors presented the BD-rate values this way. A positive value means a lower BD-rate increase compared to the same-sized tile configuration. Additionally, B only presents the average PSU values. Therefore only the average ATS value is compared.

Note that the sequence *Jockey* is not listed, as it was not tested by any of the referenced works and, therefore, cannot be used for comparison.

Sequence	**BD-rate (%)**				**ATS (%)**		
	Our	Our (best PSU)	A[8]	B[4]	Our	A	B
BasketballDrive	1.0[-0.2]	1.1[-0.3]	[-0.01]	0.6	14.25	9.82	-
BQTerrace	0.3[0.0]	0.4[-0.1]	[-0.13]	0.2	0.84	9.12	-
Cactus	0.2[0.1]	0.7[-0.4]	[-0.09]	0.25	5.28	-0.48	-
Kimono	1.1[-0.1]	1.1[-0.1]	[-0.03]	0.7	4.68	9.89	-
ParkScene	0.4[-0.1]	0.4[-0.1]	[0.02]	0.25	4.57	2.16	-
Average	0.6[-0.1]	0.7[-0.2]	[-0.05]	0.4	5.9	6.1	-80.0

Table 6.5: Comparison of results with other works. Sources: [4] and [8].

Even though the goal of this work is not minimizing the impact in coding efficiency, comparing our results with the results of B the differences are small. The highest values of BD-rate increase from our work and work A is 1.1% and 0.7%, respectively, which represents a difference of 0.4%. The average results show an even smaller difference. From this we can conclude that even by aiming for the optimal tiling partition, to maximize the parallel speed-up, the high impact on the coding efficiency is not large. Comparing the values in square brackets with A, our results are slightly worse, with an average difference of 0.5%.

In terms of ATS, results show that, on average, the solution proposed in A is slightly more efficient (considering the given five sequences), obtaining an ATS of 6.1%. While our average ATS value is inferior (5.9%), the difference is small. Additionally, our solution obtains better ATS results for three sequences (*BasketballDrive*, *Cactus* and *ParkScene*). Therefore, we can conclude that our results are acceptable. On the other hand, the average ATS value of B is very low.

7 Conclusions and Future Work

This work proposed an approach to solve the load-balancing problem, that is caused by the usage of parallel tools of HEVC such as tiles. The encoding complexity is not uniform within a video frame, as it depends on its content. Because of that, some regions of the video frame take longer to process. Therefore, to reduce the impact of the load unbalancing problem, the geometry of the tiles can be dynamically adjusted.

The proposed solution is based on two steps: encoding complexity estimation and tile geometry adjustment. As an initial attempt to design an estimation model, the Spatial Index and Temporal Index (SI and TI) were studied, with unsatisfatory results. The final estimation model defined four different methods to estimate the complexities of the CTUs within a video frame, based on previously encoded frames. The proposed methods use, as base, encoding times or the amount of bits used to encode co-located CTUs from previous frames.

The tile boundaries are then automatically adjusted based on the estimated complexities. The adjustment is done by testing various candidate tile partitions and choosing the one that leads to the best estimated workload distribution.

The proposed method achieved improvements over the same-sized tiles. Results have shown that dynamicaly adjusting the tile geometry, using the best estimation method for a given sequence, it is possible to achieve an average ATS of 6.1% when compared to the usage of same-sized tiles. This comes at a cost of an average BD-rate increases between 0.82% and 1.67%, compared to the 0.7% increase when using uniform tiles, which is perfectly acceptable.

Comparisons with other similar works show that the obtained results are viable. Our proposed method managed to obtain better values in parallel efficiency in the majority of the tested video sequences, while not having a significant impact on the coding efficiency.

While it is effective for load balancing, the method proposed in this work only works for a 2x2 tile configuration (4 tiles). For future work, it could be adapted to work with a variable tile configuration, such as 2x3 (6 tiles) or 3x3 (9 tiles) configurations.

As the results have shown, the most efficient complexity estimation method varies for each sequence. Therefore, as future work, finding a way to select the best estimation method for a given video file.

Another improvement would be either develop new complexity estimation methods, or adapt the ones proposed in this work, to support other HEVC encoding configurations such as Random Access.

To conclude this work, it is important to mention that performing the experiments on an actual parallel evironment would lead to more accurate results. However, the estimated parallel results presented in this work provide a close representation of reality.

8 Bibliography

[1] B.G Haskell and A. Puri. *The MPEG Representation of Digital Media.* Springer, 2012.

[2] Gary J Sullivan, Jens Rainer Ohm, Woo Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[3] Frank Bossen, Benjamin Bross, Student Member, S Karsten, and David Flynn. HEVC Complexity and Implementation Analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1685–1696, 2013.

[4] Cauane Blumenberg, Bampi, and Bruno Sergio Zatt. Adaptive Tiling Algorithm Based on Highly Correlated Picture Regions for the HEVC Standard. Master's thesis, 2014.

[5] Qin Yu, Liang Zhao, and Siwei Ma. Parallel AMVP candidate list construction for HEVC. *2012 IEEE Visual Communications and Image Processing, VCIP 2012*, pages 2–7, 2012.

[6] Kiran Misra, Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, and Minhua Zhou. An overview of tiles in HEVC. *IEEE Journal on Selected Topics in Signal Processing*, 7(6):969–977, 2013.

[7] Chi Ching Chi, Mauricio Alvarez-Mesa, Ben Juurlink, Gordon Clare, Félix Henry, Stéphane Pateux, and Thomas Schierl. Parallel scalability and efficiency of HEVC parallelization approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1827–1838, 2012.

[8] Yong-Jo Ahna, Tae-Jin Hwang, Dong-Gyu Sim, and Woo-Jin Han. COMPLEXITY MODEL BASED LOAD-BALANCING ALGORITHM FOR PARALLEL TOOLS OF HEVC. *Visual Communications and Image Processing (VCIP)*, 2013.

[9] Sung Ho Bae, Jaeil Kim, Munchurl Kim, Sukhee Cho, and Jin Soo Choi. Assessments of subjective video quality on HEVC-encoded 4K-UHD video for beyond-HDTV broadcasting services. *IEEE Transactions on Broadcasting*, 59(2):209–222, 2013.

[10] Muhammad Usman, Karim Khan, Muhammad Shafique, and Jörg Henkel. An Adaptive Complexity Reduction Scheme With Fast Prediction Unit. *Image Processing (ICIP), 20th IEEE International Conference on Image Processing*, pages 1578–1582, 2013.

[11] Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[12] Thaísa L. Da Silva, Luciano V. Agostini, and Luis A. Da Silva Cruz. Fast HEVC intra prediction mode decision based on EDGE direction information. *European Signal Processing Conference*, (April 2010):1214–1218, 2012.

[13] Eduarda Monteiro, Bruno Vizzotto, Cláudio Diniz, Marilena Maule, Bruno Zatt, and Sergio Bampi. Parallelization of Full Search Motion Estimation Algorithm for Parallel and Distributed Platforms. *International Journal of Parallel Programming*, 42(2):1–26, August 2012.

[14] Nidhi Parmar and Myung Hoon Sunwoo. Enhanced Test Zone Search Motion Estimation Algorithm for HEVC. *SoC Design Conference (ISOCC)*, pages 260–261, 2014.

[15] Rickard Sjoberg, Ying Chen, Akira Fujibayashi, Miska M. Hannuksela, Jonatan Samuelsson, Thiow Keng Tan, Ye Kui Wang, and Stephan Wenger. Overview of HEVC high-level syntax and reference picture management. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1858–1870, 2012.

[16] RyeongHee Gweon and Yung Lyul Lee. N-level quantization in HEVC. *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB*, pages 1–5, 2012.

[17] Vivienne Sze and Madhukar Budagavi. High throughput CABAC entropy coding in HEVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1778–1791, 2012.

[18] Seunghyun Cho, Hyunmi Kim, Hui Yong Kim, and Munchurl Kim. Efficient In-loop Filtering across Tile Boundaries for Multi-Core HEVC Hardware Decoders with 4K/8K-UHD Video Applications. *IEEE Transactions on Multimedia*, 9210(c):1–1, 2015.

[19] Arild Fuldseth, Michael Horowitz, Shilin Xu, Kiran Misra, Andrew Segall, and Minhua Zhou. Tiles for managing computational complexity of video encoding and decoding. *2012 Picture Coding Symposium, PCS 2012, Proceedings*, pages 389–392, 2012.

[20] Yj Ahn, Tj Hwang, Dg Sim, and Wj Han. Implementation of fast HEVC encoder based on SIMD and data-level parallelism. *EURASIP Journal on Image and Video Processing*, 2014(1):1–19, 2014.

[21] Muhammad Usman Karim Khan, Muhammad Shafique, and Jorg Henkel. Software architecture of High Efficiency Video Coding for many-core systems with power-efficient workload balancing. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, pages 1–6, 2014.

[22] Huang Li. Gitl HEVC Analyzer. `https://github.com/lheric/GitlHEVCAnalyzer`. Accessed: 2016-03-04.

[23] ITU-T Study Group 9. ITU-T Rec. P.910 (04/2008) Subjective video quality assessment methods for multimedia applications. *SERIES P: TELEPHONE TRANSMISSION QUALITY, TELEPHONE INSTALLATIONS, LOCAL LINE NETWORKS, Audiovisual quality in multimedia services*, pages 1–42, 2009.

[24] M. Nacari. Spatial index (SI) and temporal index (TI) calculator, version 1.0, 2013.

[25] Frank Bossen. Common test conditions and software reference configurations. *Joint Collaborative Team on Video Coding (JCT-VC), JCTVC-F900*, 2011.

[26] Gisle Bjontegaard. Calculation of average PSNR differences between RD-curves. *Video Coding Experts Group (VCEG)*, 2001.

Appendix A

Tested Video Sequences

Video sequences used for experiments in this work.

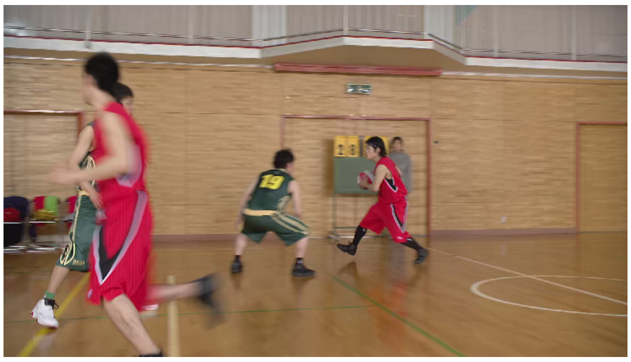

Figure A.1: Name: *Basketballdrive*; Resolution: 1920x1080; FPS: 50; Frame Count: 500

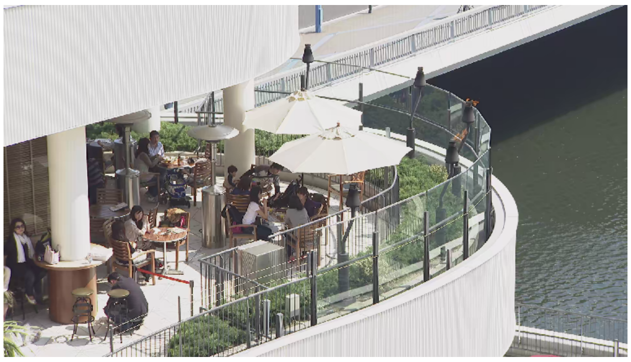

Figure A.2: Name: *BQTerrace*; Resolution: 1920x1080; FPS: 60; Frame Count: 600

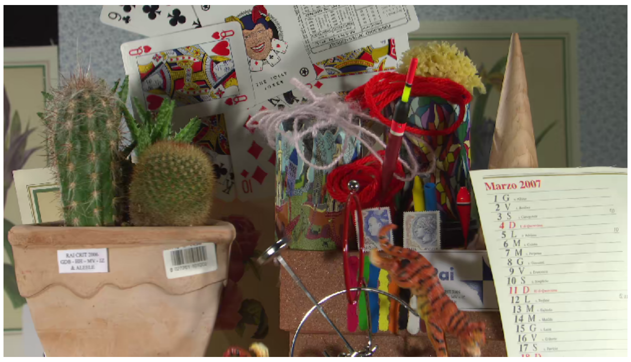
Figure A.3: Name: *Cactus*; Resolution: 1920x1080; FPS: 50; Frame Count: 500

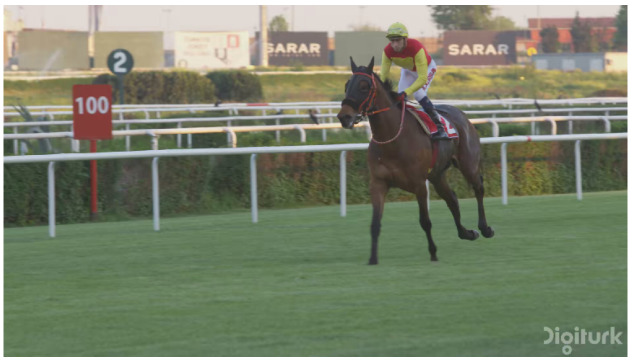
Figure A.4: Name: *Jockey*; Resolution: 1920x1080; FPS: 120; Frame Count: 600

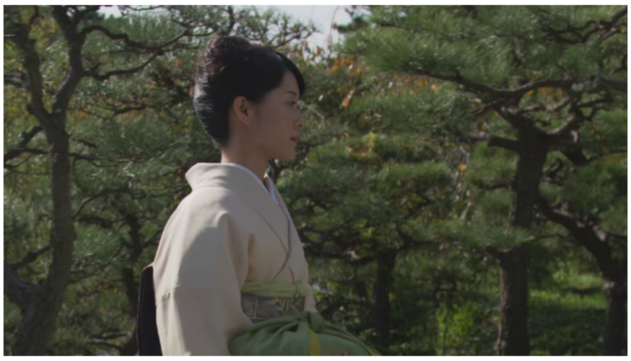
Figure A.5: Name: *Kimono*; Resolution: 1920x1080; FPS: 24; Frame Count: 240

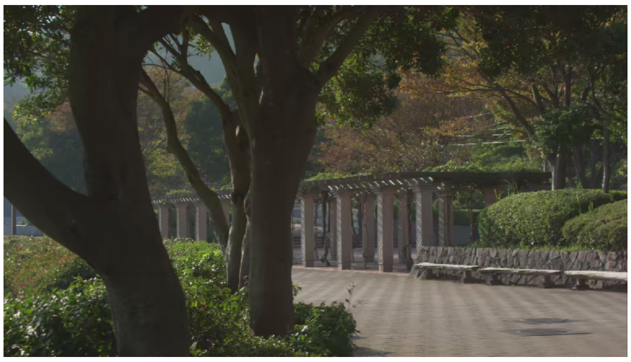
Figure A.6: Name: *ParkScene*; Resolution: 1920x1080; FPS: 24; Frame Count: 240

Appendix B

Graphics

Graphics to complement the results of chapter 6.

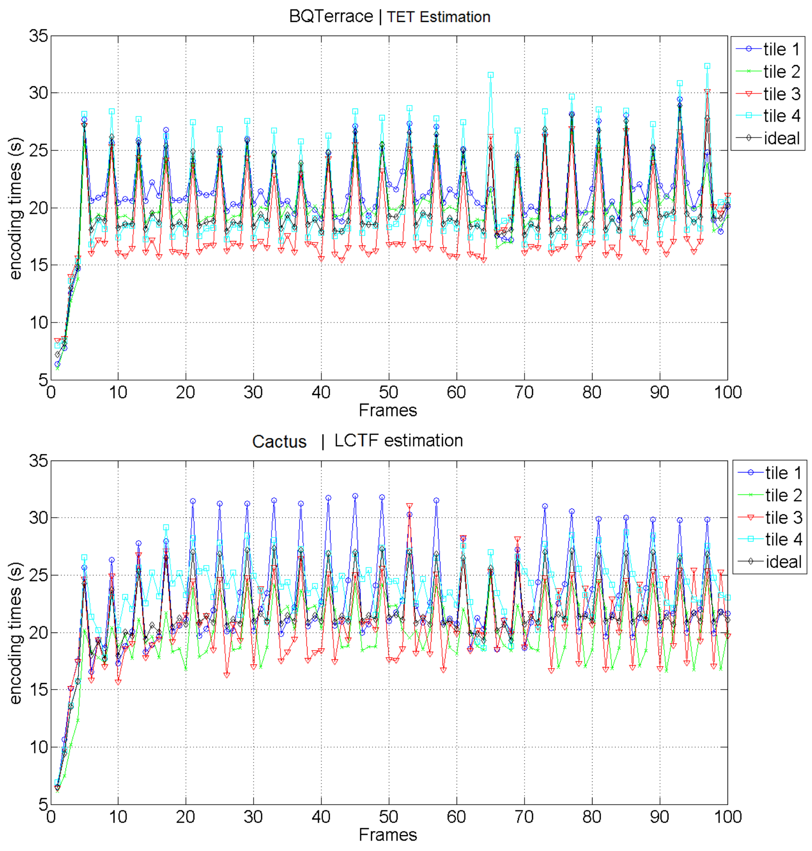

Figure B.1: Encoding times per tile for the first 100 frames.

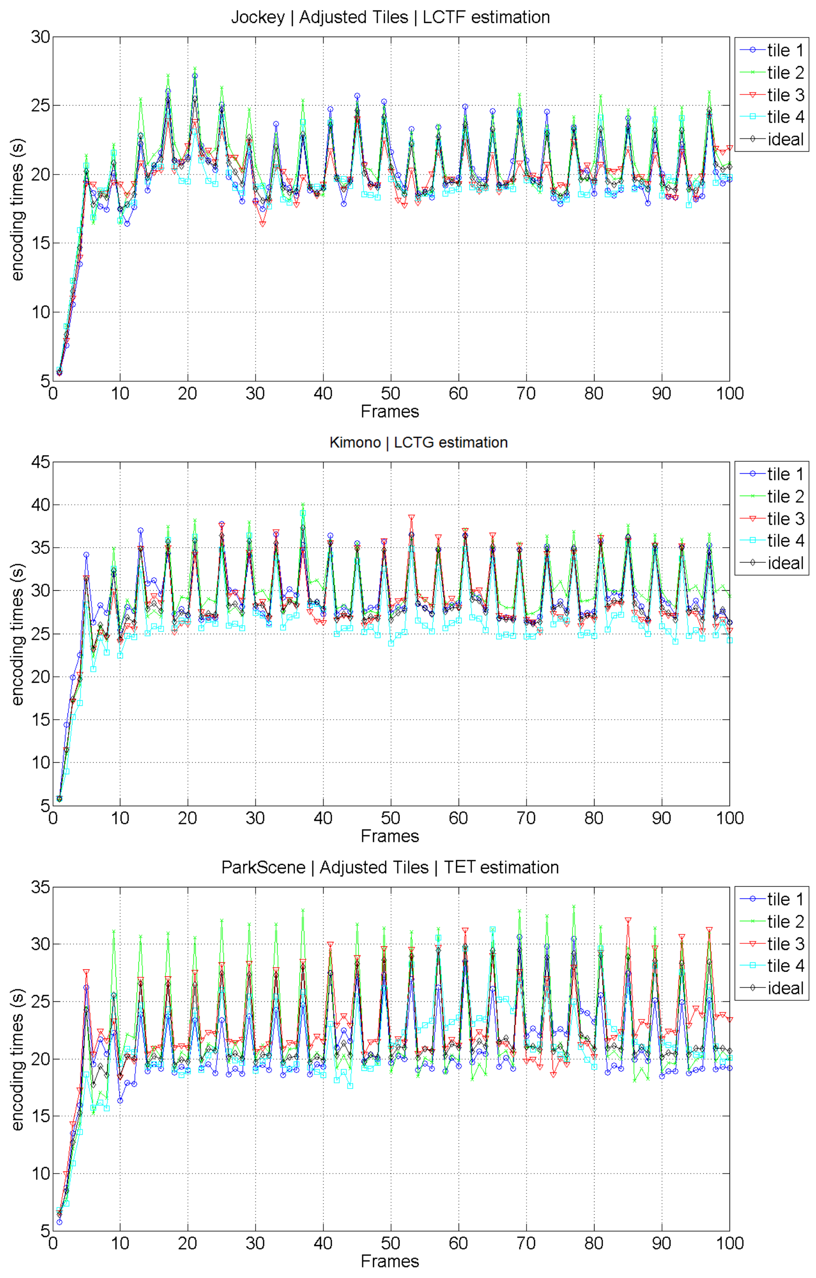

Figure B.2: Encoding times per tile for the first 100 frames.

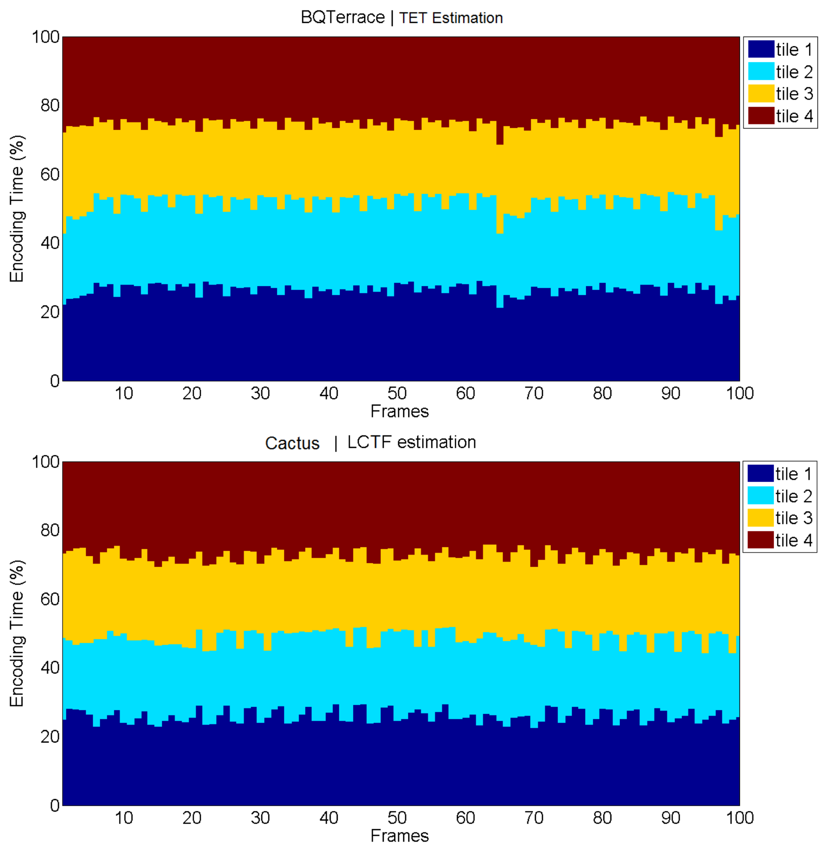

Figure B.3: Encoding complexity distribution per tile

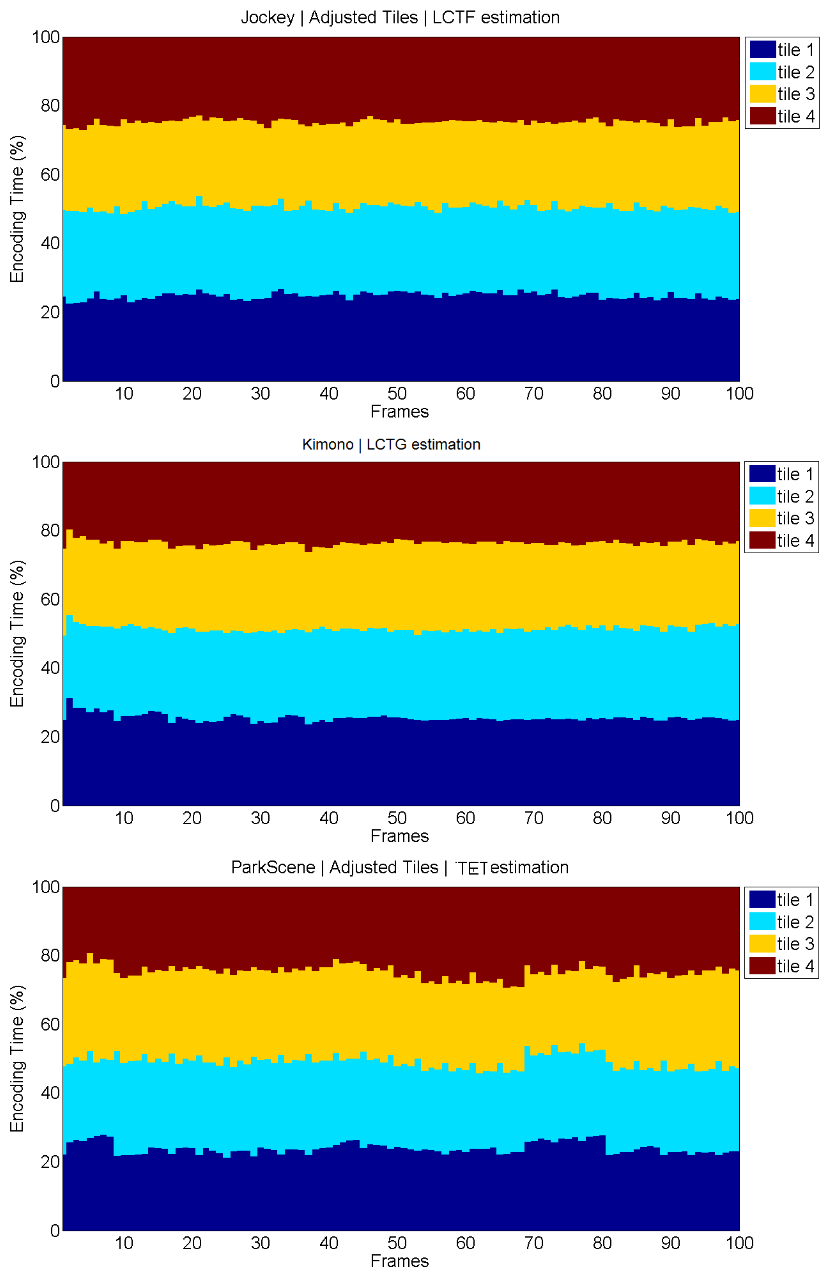

Figure B.4: Encoding complexity distribution per tile