UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

FILIPE POSTERAL SILVA

Performance and Coding Efficiency Evaluation of HEVC Parallelization Strategies

Final report presented in partial fulfillment of the
Requirements for the degree in Computer Engineering

Advisor: Prof. Dr. Sergio Bampi
Co-advisor: MSc. Cláudio Machado Diniz

Porto Alegre
2014

CIP – CATALOGING-IN-PUBLICATION

Posteral, Filipe

Performance and Coding Efficiency Evaluation of HEVC Parallelization Strategies / Filipe Posteral Silva. – 2014.

93 f.:il.

Advisor: Prof. Dr. Sergio Bampi ; Co-advisor: MSc. Cláudio Machado Diniz.

Graduation Thesis – Universidade Federal do Rio Grande do Sul. Computer Engineering. Porto Alegre, BR – RS, 2014.

1. Digital video coding. 2. High Efficiency Video Coding 3. Parallel Oriented Tools I. Bampi, Sergio. II. Diniz, Cláudio Machado. III. Performance and Coding Efficiency Evaluation of HEVC Parallelization Strategies

“*The true method of knowledge is experiment*.”

William Blake

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Prof. Dr. Sergio Bampi, who invited me to join his research team and made this work possible. His support, guidance, and advice added considerably to my graduate experience.

A very special thanks goes out to my co-advisor, MSc. Cláudio Machado Diniz, whose expertise, understanding and patience, as well as his pain-staking effort in proof reading the drafts, are greatly appreciated. My gratefulness also goes out to the other members of the research lab - in special MSc. Daniel Palomino, for his valuable suggestions and explanations.

Last but not least, I would like to thank my parents and friends for their unconditional support throughout my degree. In particular the patience and understanding shown by my mom, dad and sister during the past few months. Frank, Raul, Matheus, Bruna and Ingrid: your motivation was very important to the completion of this work.

In conclusion, I recognize that this research would not have been possible without the opportunities offered by both the Universidade Federal do Rio Grande do Sul, in Brazil, and the École Nationale Supérieure de l'Électronique et de ses Applications, in France.

ABSTRACT

For ultra-high definition videos, the requirements of low-latency or real-time processing can exceed the capacity of current single core codecs. The new video compression standard High Efficiency Video Coding (HEVC) incorporates two tools to render it more parallel-friendly: *Tiles* and *Wavefront Parallel Processing* (WPP). This work presents an evaluation of the performance of these normative tools in terms of speedup, coding efficiency, scalability, and parallelization inefficiency. The results were obtained by means of the simulation of a parallel environment based on data produced using the HEVC HM Reference Software. WPP only presented gains when using a number of threads greater than or equal to half the number of CTU columns present in its frames, which is revealed to be a limitation directly related to data dependencies and row scheduling. Using more tiles than threads only produces significant speedups for specific numbers of threads. For HD resolutions, different tiling patterns present intersection points in speedup when compared – cases for which the choice for less partitions incurs more coding efficiency and same gain in performance. In comparison to WPP, tiles achieve higher average performance. Tiles also scale better but present, as a downside, meaningful losses in coding efficiency. For high-definition content and a relatively low number of processing units (30-40% of the number of partitions) – the case of most low-power devices nowadays -, the difference in performance is not so significant, in which case WPP would be preferable since it leads to better compression rates and needs less memory bandwidth.

Keywords: Digital Video Coding, High Efficiency Video Coding Standard, Parallel Tools, Tiles, Wavefront Parallel Processing.

RESUMO

Para vídeos de ultra-alta definição, os requisitos de baixa latência ou de processamento em tempo real podem exceder a capacidade dos atuais codecs *single core*. O novo padrão de codificação de vídeos de alta eficiência (HEVC – *High Efficiency Video Coding standard*) integra duas ferramentas que expandem o seu suporte ao paralelismo: os *Tiles* e o *Wavefront Parallel Processing* (WPP). Este trabalho apresenta uma avaliação do desempenho dessas ferramentas em termos de ganho de performance, eficiência de codificação, escalabilidade e ineficiência de paralelização. Os resultados foram obtidos por meio da simulação de um ambiente paralelo com base em dados produzidos utilizando o software de referência do HEVC (HM Reference Software). O WPP só apresentou ganhos ao utilizar um número de partições superior ou igual à metade do número de colunas de CTUs presentes em seus quadros, uma limitação diretamente relacionada às dependências de dados. A utilização de mais *tiles* do que as *threads* só produz acelerações significativas para números específicos de *threads*. Para resoluções de alta definição, diferentes padrões de particionamento em *tiles* apresentam pontos de interseção quando a aceleração é comparada - casos para os quais a escolha de um menor número de partições gera economias em codificação para um mesmo ganho em desempenho. Em comparação com o WPP, os *tiles* alcançam um desempenho médio superior. *Tiles* também possuem melhor escalabilidade, embora apresentem, como desvantagem, perdas significativas de eficiência de codificação. Para conteúdo de alta definição e um número relativamente baixo de unidades de processamento (30-40% do número de partições) - o caso da maioria dos dispositivos de baixa potência -, a diferença de desempenho não é tão significativa, caso em que o uso do WPP ser preferível, uma vez que garante melhores taxas de compressão e necessita de menor largura de banda de memória.

Palavras-chave: Codificação de Vídeo Digital, Padrão de Codificação de Vídeos de Alta Eficiência, Ferramentas Orientadas ao Paralelismo, Tiles, Wavefront Parallel Processing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS AND ACRONYMS

AI	All-Intra
AMVP	Advanced Motion Vector Prediction
B	Blue
BD-rate	*Bjφntegaard Delta Bit Rate*
CABAC	*Context Adaptive Binary Arithmetic Coding*
Cb	Blue chrominance
Chroma	Chrominance
CODEC	Encoder-Decoder
Cr	Red chrominance
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DBF	Deblocking Filter
DC	Direct Current
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
G	Green
GB	Gigabyte
GoP	Group of Pictures
GPB	Generalized P and B Pictures
HD	High Definition
HE	High Efficiency
HEVC	High-Efficiency Video Coding
HM	HEVC Test Model
IEC	International Electro-technical Commission
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union, Telecommunication Standardization Sector
JCT-VC	Joint Collaborative Team on Video Coding
LB	Low-Delay B
LC	Low Complexity
LP	Low-Delay P

Luma	Luminance
MB	Megabyte
MC	Motion Compensation
ME	Motion Estimation
MP	Main Profile
MPM	Most Probable Mode
MV	Motion Vector
MVD	Motion Vector Difference
OWF	Overlapped Wavefront
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter
R	Red
RA	Random-Access
RGB	Red, Green, and Blue
SAO	Sample Adaptive Offset
TU	Transform Unit
UHD	Ultra-High Definition
VCEG	Video Coding Experts Group
WPP	Wavefront Parallel Processing
Y	Luminance
YCbCr	Luminance, Blue Chrominance and Red Chrominance
Quadtree	Quadratic Tree

CONTENTS

1 INTRODUCTION

Stimulated by the use of applications such as broadcast over cable and satellite, conversational applications over wireless and mobile networks, multimedia streaming services and etc., the growth in production and diffusion of digital video has caused video coding standards to evolve in order to adapt themselves to the ever increasing resolutions and frame rates of digital videos (Vanne et al. 2012).

The newest video coding standard, High Efficiency Video Coding (HEVC) (Sullivan and Ohm 2010), developed by the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Pictures Experts Group (MPEG), was especially designed to support ultra-high definition content with resolutions of 4Kx2K and 8Kx 4K, for example, with a bit rate reduction of about 50% compared to its predecessor, H.264/AVC (ITU-T Rec. 2003).

Because of the huge computational effort needed to encode and decode such ultra-high resolutions, the standard focuses on the use of parallel processing architectures for computational effort reduction. Moreover, it is very unlikely that single-core processor performance will increase to a point where it can decode UHD HEVC video in real time. While International Technology Roadmap for Semiconductors (ITRS 2011) predicts the transistor speed will continue to improve by 25% per technology node, only 15% performance improvement if possible due to energy constraints (Borkar and Chien 2011), forcing cores to operate at low frequency and near threshold voltage. It means that exploiting parallelism is no longer an option but a necessity (Chi et al. 2012).

To address this issue, the HEVC standard focuses on the explicit use of parallelism for video coding and decoding processes by the inclusion of two parallelism-oriented tools into the standard: *Wavefront Parallel Processing* (WPP) and *tiles*.

1.1 Objectives of this work

The objective of this work is to evaluate the performance of the two normative tools for parallel processing proposed by the HEVC coding standard in terms of speedup, coding efficiency, scalability and parallelization efficiency. Although these tools can be used for both parallel encoding and decoding, this work focuses uniquely on the parallel encoding capabilities, since the encoding process is an order-of-magnitude more complex than the decoding process. However, the scalability results presented in this work may also be applied for parallel decoding.

The HEVC HM Reference Software (*HEVC HM Reference Software* (version 14.0)) allows different encoding options to be set through the use of configuration files, while also providing a baseline implementation that can be modified for experimental purposes, for example. This software is used in this work to encode video sequences and to extract data for the analysis.

The result of this analysis shall answer some questions regarding parallelism limitations and optimal application uses in multi-core architectures.

1.2 Text structure

Chapter 2 presents an overview of basic video coding concepts and presents the main tools of the HEVC standard, focusing on the elements that will be important in later sections. Chapter 3 introduces the HEVC parallelization strategies and explains their associated data dependencies. It also presents an analysis of the efficiency of using tile partitioning and the challenges of exploiting parallelism in the video coding process. Chapter 4 describes the methodology adopted for the evaluation of the two parallel tools, also introducing the concepts of parallelism potential and parallelization inefficiency. Chapter 5 presents the experimental results in terms of coding efficiency, speedup, scalability, and parallelization efficiency. Finally, Chapter 6 concludes this work and points future extensions of this work.

2 VIDEO CODING CONCEPTS AND THE HEVC STANDARD

The evolution of technology has made it possible to capture and create video sequences with very high resolutions that produce huge volumes of data. The volume of data is large to the point of rendering processing, storage and transfer impossible in current technology without compression techniques.

2.1 Digital video characteristics and color sampling

A video sequence is a set of frames (static pictures) displayed one after another. Each frame is divided into blocks, constituted by pixels. The pixel is the smallest element of an image, and it is usually represented by samples in three different color components, i.e. ***R**ed*, ***G**reen* and ***B**lue*, when considering a RGB color space. For video compression purposes, video is often represented in the YCbCr color space, in which **Y** is the luminance component; **Cb** and **Cr** are the blue and red chrominance components, respectively. Since the human visual system is more sensitive to luma components, video *codecs* (video encoders-decoders) usually discard some chroma information that do not affect the human visual perception, a process called *color subsampling*.

A raw video sequence captured at 1080p resolution (1920 x 1080 pixels), 24 bits per pixel (8-bits for each component Y, Cb, Cr) and 25 pictures per second (frame rate) would require a data transfer rate of 156 MB/s (25 x 3 x 1920 x 1080). Consequently, a storage unit of 1094 GB (equivalent to 48 single layer Blu-ray discs) would be necessary to store 2 hours of raw (uncompressed) video.

Therefore, there is a clear imbalance between the current data processing capacity, storage and transfer, which makes the representation of uncompressed videos not effective at all. However, with the computing power available today, it is possible to process videos in a way that they can be transmitted and stored using less bandwidth or disk space. Compression and decompression techniques allow a significant reduction in file size with minimal or no effect on visual quality.

2.2 Fundamentals of video coding

Videos contain a very expressive statistical redundancy, in both time and spatial domains. Video compression technologies are designed to reduce and eliminate redundant data to allow the storage and efficient transmission of a digital video file. The main principle lies in the detection of redundant information that can be represented more efficiently.

Lossy compression techniques consist on the elimination of some of the video data available in the input during the compression process at the cost of some loss in video quality. Lossy compression techniques take advantage of the restrictions of the human visual system model to limit the frequencies being represented in an image, quantizing high frequency detail components without a perceptive loss of quality (Szu 2002).

Inside each frame of video the *spatial redundancy* (intra-frame redundancy) (Ghanbari and Institution of Engineering and Technology 2011), i.e. repeated shapes or patterns that can be described by less information, is explored to reduce the amount of bits required for the representation of the same data. Between the different frames of a video sequence there is a *temporal redundancy* (inter-frame redundancy) (Ghanbari and Institution of Engineering and Technology 2011), i.e. elements that are very similar in two different frames.

The main idea is to encode only what changes from one frame to another (and not the whole frame) using prediction techniques. Thus, we assume that the value of a particular pixel of the frame can be predicted from neighboring pixels from the same frame (using *intra*-frame coding) or pixels which belong to other frames (using *inter*-frame coding).

Spatial redundancy expresses the fact that we often find in each image of video sequence one or more regions with similar visual characteristics. We assume that we can predict the values of the pixels of a block from the pixel values of the neighboring blocks (the concept of block partitioning will be explored later). This is called spatial prediction or *intra* prediction.

Temporal redundancy expresses the fact that there is usually no abrupt change in the content displayed between two or more consecutive frames. This redundancy is exploited using motion *estimation* and *compensation* techniques.

Motion estimation (ME): the motion is estimated for a current frame (to be encoded) related to a reference frame (previously encoded) using a best-match algorithm. The motion vector represents the motion of a block from the reference frame to the current frame.

Motion Compensation (MC): the motion compensation is applied to the reference frame to obtain the predicted frame.

However, this estimation is not perfect. In addition to motion information, the difference between the predicted and current frame is also needed for reconstruction. This difference is called residual frame (term also used in intra prediction). Figure 1 shows the residual frame obtained by the subtraction of two successive frames when motion compensation techniques are not used.

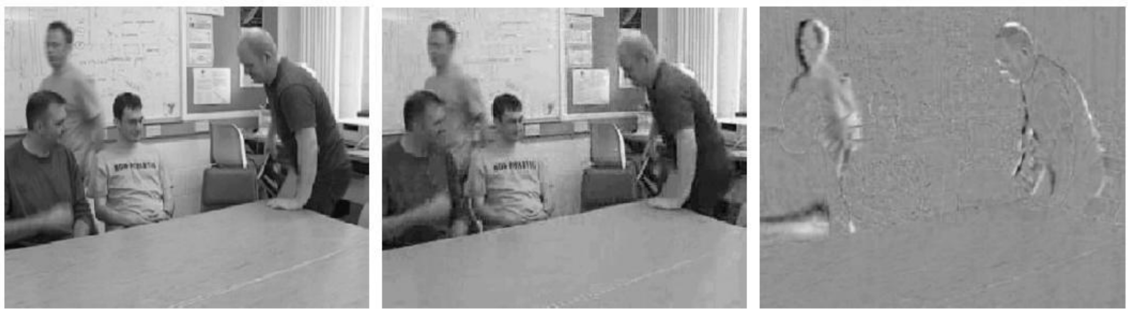


Figure 1: Frame *n* to encode, Frame *n-1* (reconstructed after encoding), and associated residual frame, respectively. Source: (Richardson 2011)

The use of motion compensation techniques changes the result drastically. The energy of the residue is much smaller which means the amount of data needed for representation is reduced.

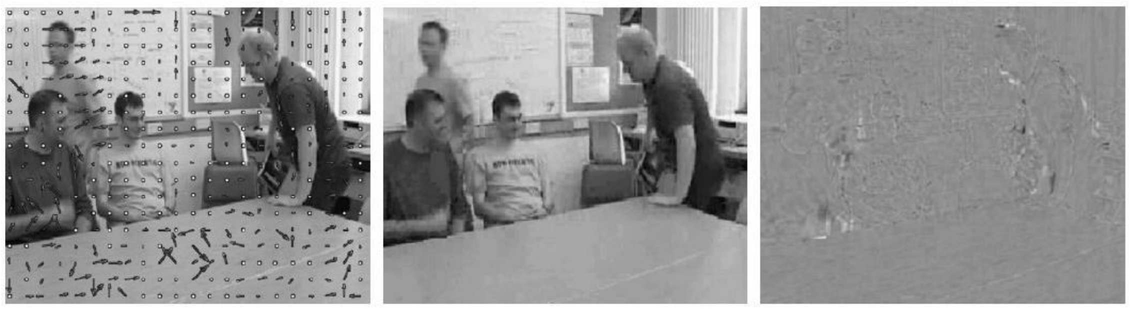


Figure 2: Motion vectors over the frame, frame *n* compensated and associated residual frame, respectively. Source: (Richardson 2011)

The coded representation of a video sequence (the encoded bitstream) contains both image and motion information.

2.3 Structure of a hybrid encoder

The transformation of a raw (uncompressed) video sequence into an encoded bitstream (a coded video sequence) is called *encoding* and is performed by an *encoder*. The inverse transformation is called *decoding*. The *decoder* is needed each time the video has to be displayed. A device capable of performing both functions is called a *codec* (video encoder-decoder).

A hybrid encoder consists of three main functional units:

- a prediction model;
- a transformation/quantization model;
- an entropy encoder.

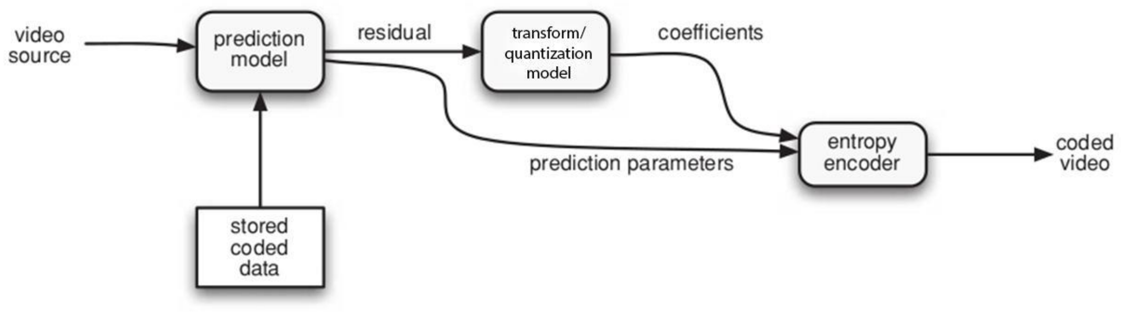


Figure 3: Hybrid video encoder. Source: (Richardson 2011)

The *prediction model* tries to reduce redundancy by exploiting the similarities between neighboring frames temporally or spatially, typically by constructing a prediction of the current frame. After intra or inter prediction, the residue is inferred and encoded.

The *transformation/quantization model* is based on the application of a transformation, a quantization and an arrangement of the coefficients. The purpose of the transform is to convert the data to another domain, where its representation will be more efficient for the quantization. Then, the quantization is applied to reduce the volume of irrelevant information (high-frequency color nuances). The amount of information to be discarded can be controlled by a quantization parameter (QP). The higher is the QP value, the more information is discarded to achieve higher compression rates. The coefficients are then reordered before being processed by the entropy encoder.

The *entropy encoder* produces a compressed bitstream exploiting the statistical redundancy, i.e. the redundancy of data representation. It is typically based on variable length codes or binary arithmetic encoding (Richardson 2011). This type of encoding does not result in losses and further reduces the volume of data required for representation. A compressed sequence is composed of header information, coded prediction parameters and coded residual coefficients.

2.4 Picture types

Not all the video frames are compressed in the same way. In the most recent compression standards, the video stream is previously divided into groups of pictures (GoPs) composed of pictures of different types:

- *I-frames* (Intra-coded picture) are pictures on which only intra prediction is applied. It does not require other frames for encoding and decoding.
- *P-frames* (Predicted picture) are pictures on which inter prediction can be applied (but also intra prediction). Inter prediction uses data from previous frames for encoding and decoding. *P* frames can be more compressible than I-frames.
- *B-frames* (Bi-predictive picture) can use both previous and forward frames for inter prediction to get the highest improvement in compression.

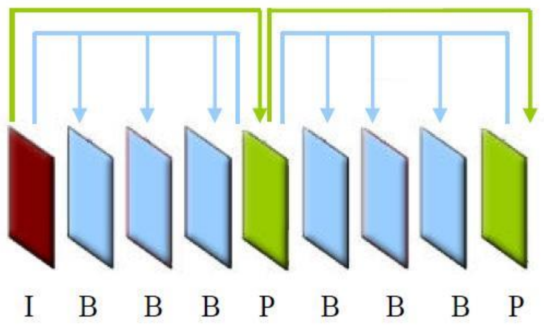


Figure 4: Dependencies between pictures in a GoP.

Figure 4 shows an example of GoP and the dependencies between its pictures. It is important to observe that it is possible to use a spatial prediction in P and B frames. In recent standards, like HEVC, frames are typically divided into multiple processing blocks with different coding types.

2.5 Scope of a video coding standard

The vast amount of existing electronic devices capable of encoding or decoding video sequences needs the creation of rules that enable interoperability between them. A compression standard is primarily a formal set of rules that allows encoders and decoders to work together properly (Motion JPEG, MPEG-4, H.264/AVC and HEVC are some examples of video coding standards). Every decoder respecting the rules of a particular standard is capable of processing a video sequence produced by an encoder that respects the same standard. Encoders and decoders, if separated from each other in time or space, must have their interoperability insured by the standard.

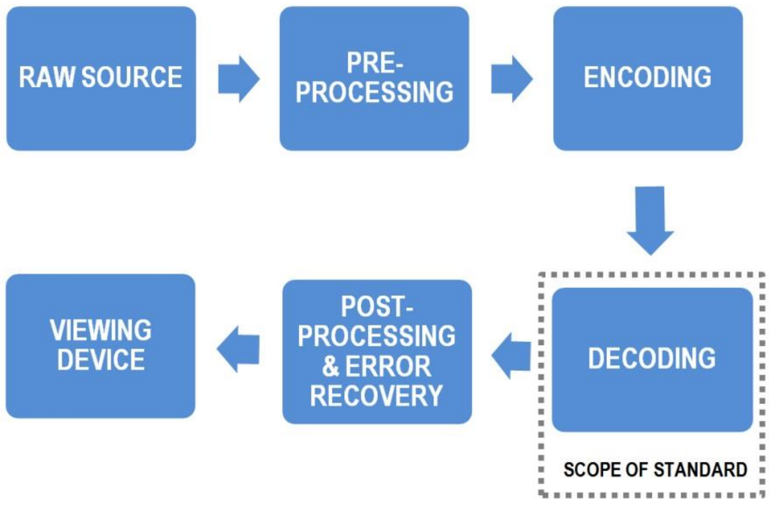


Figure 5: Scope of a video coding standard.

The scope of a video compression standard is limited to decoding, as shown in Figure 5. It defines precisely the way according to which an encoded video stream must be decoded. The implementation of encoding algorithms, on the other hand, is up to developers as long as the produced stream can be processed by a decoder in accordance with the standard. This allows, for example, the creation of encoders that can meet specific needs (e.g. no inter prediction).

2.6 The High Efficiency Video Coding (HEVC) Standard

The High-Efficiency Video Coding (HEVC) is the newest video compression standard proposed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Expert Group (MPEG). The following subsections will provide an overview of the coding tools proposed in the standard.

2.6.1 Structure of an HEVC encoder

HEVC follows the same hybrid video encoder model that was shown in subsection 2.3, including motion estimation/compensation followed by transformation, quantization, and entropy coding. A block diagram of the HEVC encoder is shown in Figure 6. Each frame is divided into blocks of a parameterized size, and the block size is transmitted to the decoder. The first image of a GoP is encoded using only intra prediction. Inter-frame prediction is typically used for most blocks in all other P and B pictures . Inter-frame prediction is the most costly video encoding step in terms of computational effort, being responsible for around 80% of the total encoding time (Monteiro 2012) and usually leading to high contribution it the compression rate (Puri, Chen, and Luthra 2004).

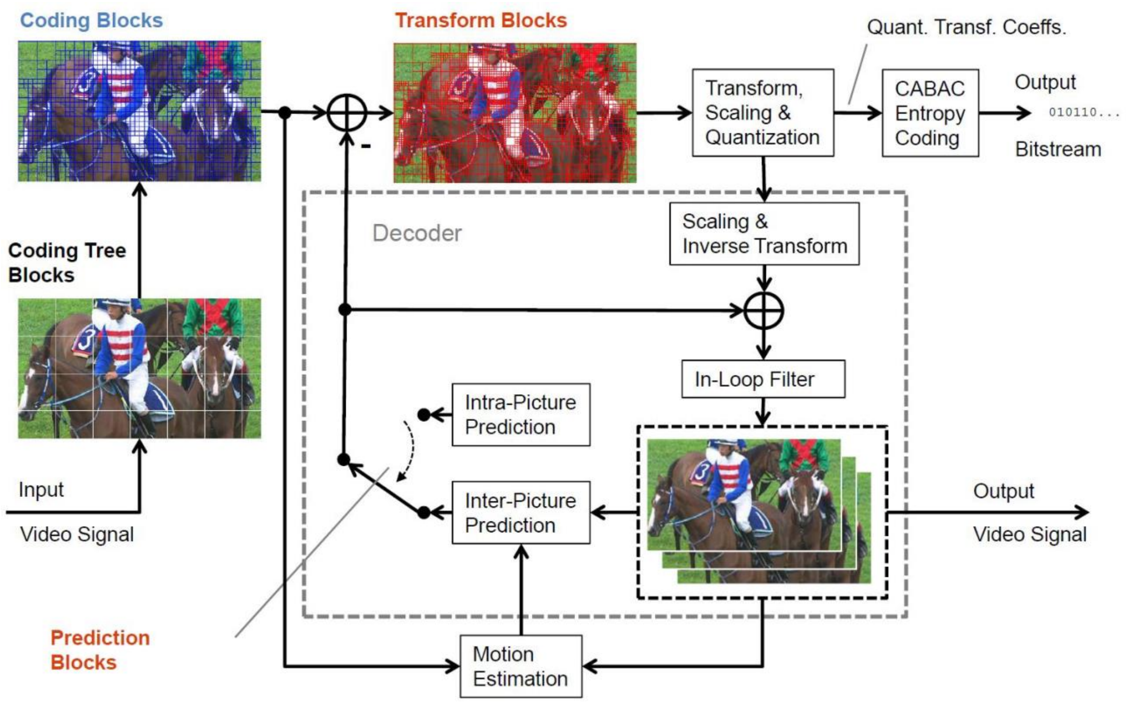


Figure 6: Block diagram of an HEVC encoder. Source: (Bross 2012)

The residue of inter or intra prediction is processed by a linear spatial transformation. The transform coefficients are then scaled, quantized, entropy coded and transmitted with the prediction information. The presence of a simplified decoding path inside the encoder, as in the H.264/AVC standard, ensures that both encoder and decoder generate the same prediction by using the same input.

Thus, an inverse scaling is applied to the quantized transform coefficients. An inverse transform is then applied to the scaled coefficients to reproduce an approximation of the residue. The residue is then added to the prediction, and the result of this addition can be therefore subjected to one or two filters to smooth the artifacts induced by the standard's block partitioning.

The final representation of the frame in the encoder (which is the same as the decoder output) is stored in a frame buffer used for the prediction of the next frame.

2.6.2 Quadtree partitioning

In contrast to previous video coding standards, HEVC employs a flexible quad-tree coding block partitioning structure that enables the use of large and multiple sizes of coding, prediction, and transform blocks (Kim et al. 2013).

The pictures are primarily divided into square regions called *coding tree units* (CTUs) consisting of one luminance and two chrominance *coding tree blocks* (CTBs) and associated syntax elements needed for the parsing stage. The concept of CTU partitioning is of high importance in this work, considering the parallel tools evaluated in further chapters are heavily based on it. CTUs can be recursively partitioned into smaller blocks called Coding Units (CUs) with the partitioning structure based on a *quadtree* representation. It allows the partitioning of the picture into multiple block sizes that adapt easily to the content of images whilst having a reduced signaling cost.

In comparison with the traditional macroblock used in H.264/AVC (16x16 fixed size for luminance), HEVC supports the choice of the size of the CTU according to memory constraints and computing power. Larger than in previous standards, the supported sizes for CTUs are particularly beneficial when encoding HD video content, once the correlation between neighboring pixels increases significantly. Three basic units are also defined in the standard:

Coding unit (CU): is a block with size varying from *8x8* to *64x64* (the maximum size specified by the standard is that of the CTU). The decision between intra or inter coding (base prediction type) is taken at this level. A CTU may contain only one CU or be divided into several CUs; and each CU has its own partitioning into prediction and transform units.

Prediction unit (PU): specifies the type of prediction of a block. A PU is rooted at a CU. HEVC supports varying PU sizes ranging from *4x4* up to *64x64* and they can assume symmetric and asymmetric formats, as shown in Figure 7. The set of possible sizes is defined according to the type of prediction set for the CU:

– Skip : M×M.

– Intra : M×M, M/2×M/2 (if M/2 = 4).

– Inter : M×M, M/2×M, MxM/2, M/2×M/2, M/4×M (L), M/4xM (R), MxM/4 (U) and MxM/4 (D).

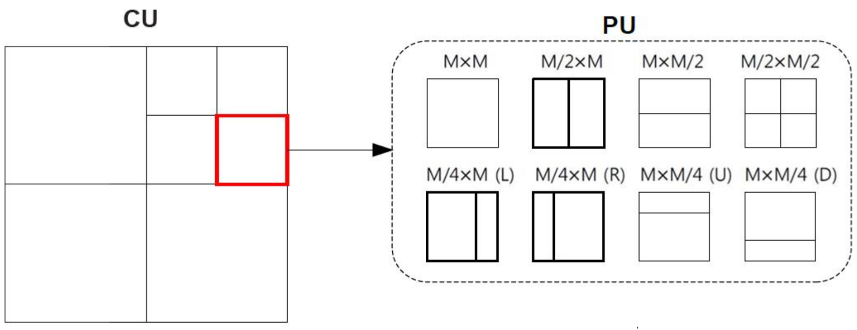


Figure 7: Partitioning modes of a CU into PUs. Source: (Bordes et al. 2012)

Transform unit (TU): the decomposition of a CU in TUs defines the sizes for the transforms and quantization applied to this region of the picture. Three levels of decomposition are possible - forming a *transform tree* (see Figure 8) - and each TU takes sizes ranging from *4x4* to *32x32*. All the TUs are squares and may encompass several PUs.

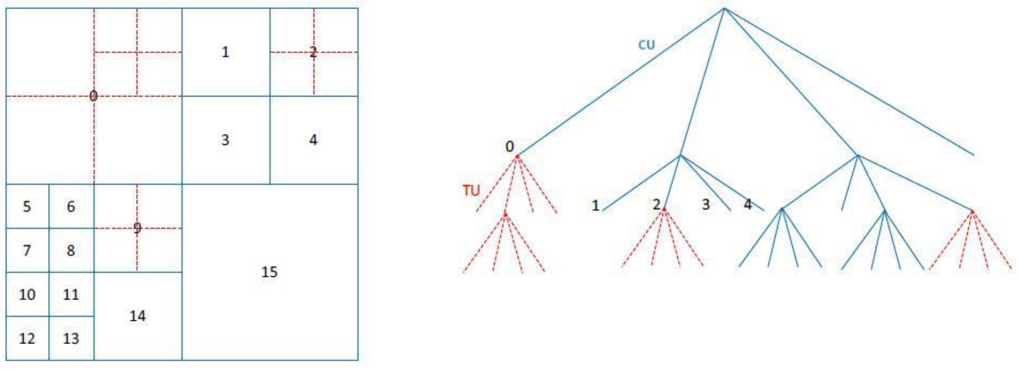


Figure 8: Partitioning of a CTU into CUs and CUs into TUs. Source: (Sullivan et al. 2012)

It is important to note that the partitioning of a CU into PUs and CUs into TUs are independent.

2.6.3 Intra-frame prediction

The HEVC standard performs a directional intra-frame prediction that analyzes the sample information of neighboring PUs' boundaries *already* encoded. HEVC increases the number of prediction modes used in the H.264/AVC standard (see Figure 9 and Figure 10). They include 33 spatial directions, the DC (average of reference pictures) and planar predictors. The availability of prediction modes depends on the size of the PU: 3 predictors for 64x6 PUs, 17 predictors for 4x4 PUs and 35 predictors for other PU sizes. These multiple modes can effectively capture pixel redundancies in the vicinity at the cost of increased computational power and signaling.

According to tests conducted by Sullivan (Sullivan et al. 2012), the bitrate reduction provided by intra-frame coding in HEVC, compared with H.264/AVC is of about 22% and can reach up to 36%.

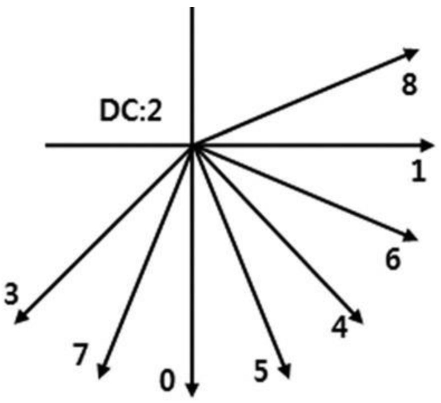


Figure 9: Luma H.264/AVC intra-frame prediction modes. Source: (Sullivan et al. 2012)

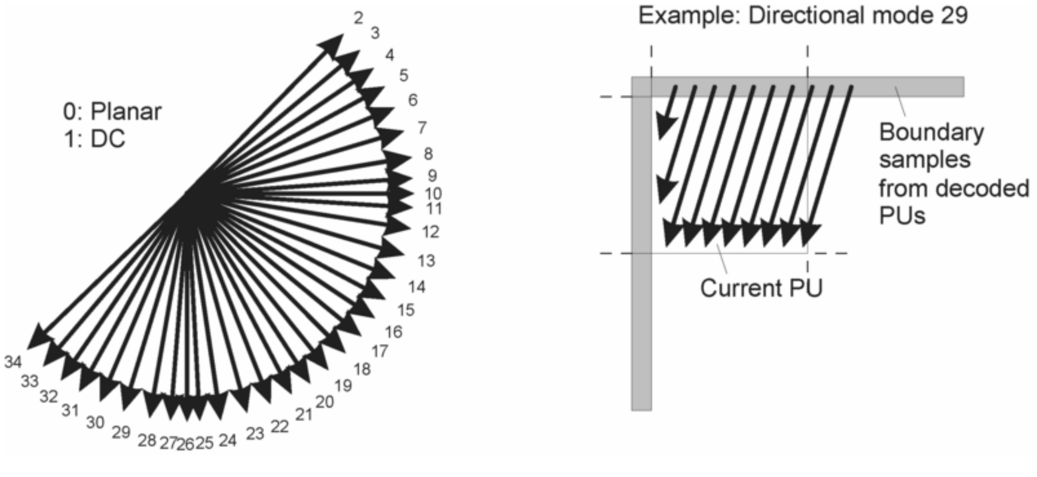


Figure 10: Luma HEVC intra-frame prediction modes Source: (Sullivan et al. 2012)

Considering that the intra-frame prediction process presents a wide range of possibilities to be analyzed, the only way to define the optimal prediction mode among the 35 available is by testing them all, which would require a considerable computational effort. To diminish this, several algorithms propose different techniques which are able to reduce the amount of modes to be analyzed. One example is the *Most Probable Modes* (MPM) strategy, which points the three intra MPM of the current PU based on the intra-prediction modes used by the neighboring PUs. This strategy is valid since the prediction modes tend to be similar between nearby prediction units (Zhao et al. 2011).

2.6.4 Inter-frame prediction

Inter-frame prediction exploits temporal redundancies and is based on two aspects: carrying out the prediction - through motion estimation and compensation - and coding of motion vectors. These two points have been improved in HEVC with a particular emphasis on motion vector coding.

Inter-frame prediction happens at the PU level: the ME process carries out the prediction by comparing the current block to previously encoded and reconstructed ones, from other frames. Through comparison by means of a distortion metric, the best match is then found and a motion vector is generated. Only the residual block, along with the motion vector, is transmitted in the bitstream.

Analyzing the behavior of the MVs generated by the ME process, the HEVC developers perceived a huge correlation between neighboring motion vectors. To exploit this correlation an *Advanced Motion Vector Prediction* (AMVP) step is performed, which makes use of spatial and temporal correlation of neighboring partitions to predict the MV of the current block (Zhao et al. 2011). The AMVP generates a set of candidates used to differentially encode the actual MV, i.e., the best candidate block is encoded in relation to the AMVP predicted vector.

The merge mode allows for the MVs to be inherited from neighboring prediction blocks (Sullivan et al. 2012), and a new motion vector difference (MVD) derivation method is designed to encode MVD efficiently (Correa et al. 2012). It is important to point out that these new tools make use of information coming from the same frame of the CTU being currently processed, which imposes intra-frame data dependencies. These data dependencies will be maintained or not depending on the adopted parallelization tool (presented in section 3).

The MC process accesses the reference pictures stored in the memory and fetches the PU pointed by the coded motion vector to reconstruct the image. It is performed at both the encoder and decoder sides to ensure result consistency.

2.6.5 Transform and quantization

The direct transform is responsible for the transformation of coefficients from the *spatial* to the *frequency* domain. This form of representation is interesting because it helps in the identification and suppression of high frequency details in picture partition being processed.

The transformation of coefficients is very similar to previous standards. The residual CU is partitioned into several square TUs as described in subsection 2.6.2. Possible TU sizes are 4x4, 8x8, 16x16 and 32x32 (Sullivan et al. 2012) which matches the sizes of transform matrices specified by the standard. For simplicity, a single 32x32 transform is specified, since all the other smaller matrices can be derived from it.

The elements of the core transform matrices were derived by the approximation of a *Discrete Cosine Transform* (DCT), in order to make it hardware-friendly and suppress the rescaling step, which was mandatory in H.264/AVC.

A Discrete Sine Transform (DST) is used for the special case of luminance 4x4 intra-frame prediction blocks. As the intra-frame prediction is based on the neighboring pixels in the upper left region and is therefore more accurate in these regions, the residue generated by it will also be less substantial. The DST behaves differently than the DCT in the sense that its basic functions start low and increase, so it is better suited to encode this type of small block since the highest residual magnitude will be located mainly in the lower right corner of each block (Sullivan et al. 2012).

The quantization in HEVC remains the same as in H.264/AVC. The coefficients generated by the DCT are used by the quantization step, where some are discarded to reduce the amount of data. The amount of information to be discarded by the quantization is controlled by the quantization parameter (QP). The larger the QP, the greater will be the impact on visual quality of the encoded video, and the greater will be the amount of information discarded by the quantization step.

2.6.6 Entropy coding and in-loop filtering

The *Context Adaptive Binary Arithmetic Coding* (CABAC) is the only entropy coding method used for video data in the HEVC standard. It represents the syntax elements generated along the encoding process based on three key factors: *(i)* the element to be encoded, *(ii)* the current encoding algorithm step, and *(iii)* the syntax elements already encoded (*context elements*) (Agostini 2007).

Choosing the context of each element directly impacts on the coding efficiency. For this reason, the CABAC algorithm used in the HEVC standard considers the context information of neighboring elements, and also the quadtree depth information. This can be considered an upgrade of the previous CABAC algorithm, since it can now achieve better compression rates with lower computational effort when compared to its previous version (Sullivan et al. 2012).

Each TB is partitioned into 4x4 sub-blocks on which the coding is applied (see Figure 11). The zig-zag scanning order is no longer used (unlike H.264/AVC): the 16 coefficients in each sub-block are processed according to an up-right diagonal, horizontal or vertical scan order.

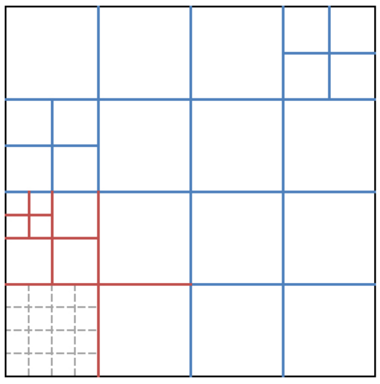


Figure 11: 16x16 TB with 4x4 sub-blocks

After the CU reconstruction, the pictures are filtered by the in-loop filters. A *deblocking filter* (DBF) is always used to reduce the artifacts introduced by the block processing, even if HEVC's larger block sizes partially softens this effect. Moreover, a *Sample-Adaptive Offset* (SAO) *filter* is added. It is applied after the DBF and is based on the classification of CTU pixels according to their intensities, followed by the application of an offset in order to homogenize the image, to remove ringing artifacts and improve its subjective quality (Fu et al. 2012).

3 HEVC PARALLELIZATION TOOLS

In contrast to the development of the previous H.264/AVC standard, where parallelism support was an after-thought, some parallelism tools were incorporated in the HEVC standard. Parallel computing support by video codecs is essential nowadays, since they require high computational effort as video resolutions are constantly increasing (Vanne et al. 2012). Additionally, the diffusion of parallel architectures on consumer multimedia device market encourages the development of solutions that exploit parallel processing. Thus, to address the high computational effort required by the HEVC standard, and to make use of the recent parallel architectures, the JCT-VC group proposed novel parallelism-oriented tools. Using these tools, it is possible to exploit coarse- and fine-grain parallelism during the encoding and decoding process.

In H.264/AVC, the concept of *slices* was introduced for robust video transmission. Because of its characteristics it was also employed for parallelization, despite it not being its initial purpose. In addition to slices, HEVC introduces two high-level structures for parallel processing: *tiles* and the *wavefront parallel processing* (WPP). Both of these tools are based on the subdivision of a picture into multiple *partitions* – which contain an integer number of CTUs - that can be processed in parallel without incurring high coding losses. As it will be seen in the following subsections, tiles and WPP have both advantages and disadvantages according to different application use cases.

3.1 Slices and tiles

The concept of ***slices***, already used in H.264/AVC, remains unchanged except for the fact that *macroblocks* are replaced by CTUs. The main objectives of slices are to enable resynchronization in the case of data loss and to deal with packetized transmission (Fuldseth et al. 2012). However, in HEVC some details were added to the definition to clarify the difference between *slices* and *tiles*.

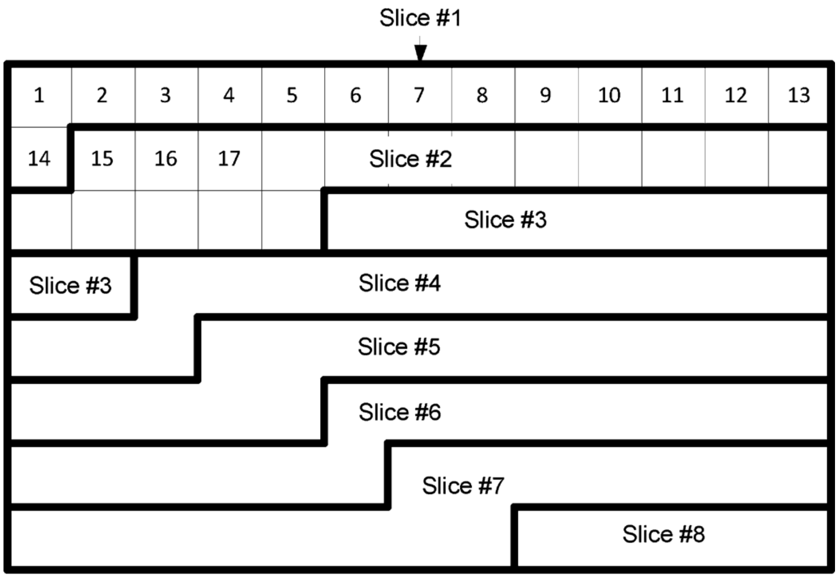


Figure 12: Example illustrating slice-based picture partitioning of CTUs following a raster scan order within the picture. Source: (Misra et al. 2013)

A slice is a self-contained structure that is able to partition a picture in a sequence of CTUs, which are processed in raster scan order. Each picture of a video sequence can be partitioned into one or more slices, as depicted in Figure 12. The concept of *picture type* shown in section 2.4 is also extended to slices: they can be either type I, P or B. When an I slice (also called *intra slice*) is considered, all the CTUs inside its slice use only the intra-frame prediction. For P slices, both intra- and inter-frame prediction can be used, in this later case using only one frame as reference. Finally, B slices can use both prediction types considering two reference frames to perform the inter-prediction.

Slices are self-contained in the sense that their syntax elements can be analyzed from the bitstream and pixel values belonging to the area of the image they represent can be correctly decoded without any information coming from neighboring tiles (Sullivan et al. 2012). This implies that each CTU prediction step is only able to access information that is located inside the slice in which it is placed and therefore that predictions are not performed beyond the slice boundaries.

Even though this independence enables the exploitation of parallelism, some problems can be found when using slices for such purpose, which is the reason why this work has not included experiments involving them. Using many slices to increase parallelism incurs significant coding losses because each defined slice carries an extra header that introduces a non-negligible overhead into the final bit-stream (Misra et al. 2013).

Therefore, while defining more slices would increase parallelism potential (please refer to section 4.5 for the concept of parallelism potential), it would negatively impact coding efficiency as more contexts would be broken by slice boundaries and more headers would be added to the final bitstream (Fuldseth et al. 2012). Besides that, the number of slices in a frame is determined by the encoder and if the decoder relies on slices to obtain real-time performance, it may not achieve this if it receives a video sequence with only one or a few slices per frame (Chi et al. 2012).

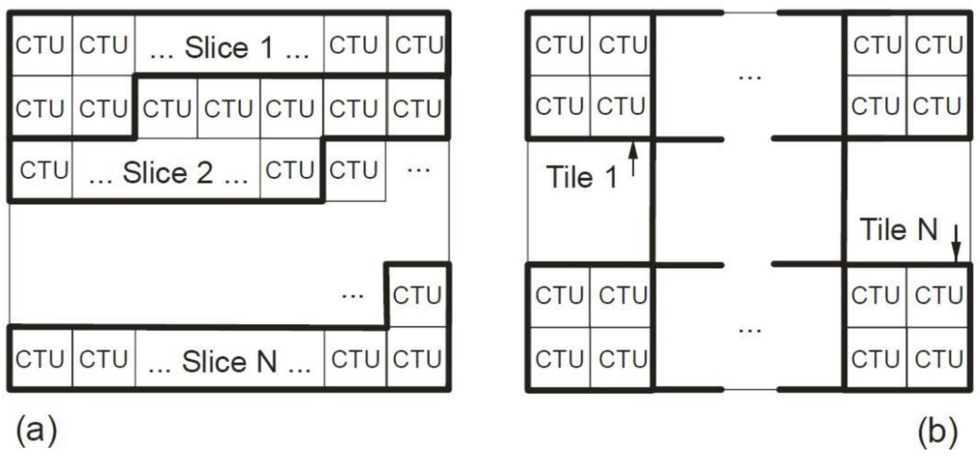
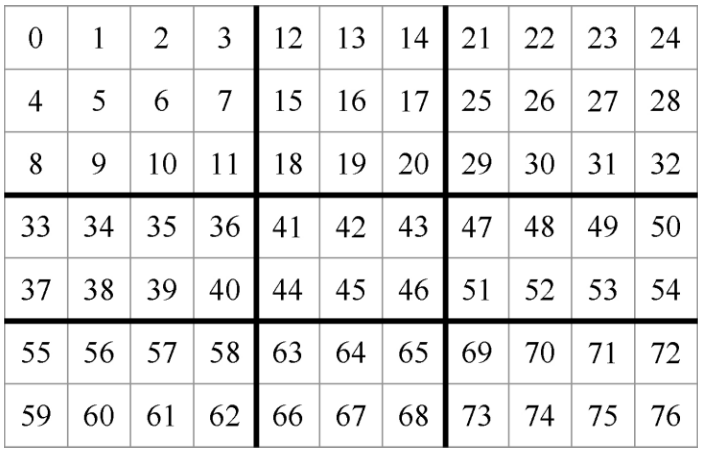


Figure 13: Division of a frame into a) slices and b) tiles. Source:(Sullivan et al. 2012)

Seeking to reduce coding efficiency impact, HEVC also defines ***tiles***, which are autonomous regions of the frame that can be independently processed. A tile consists of a rectangular arranged group of CTUs (Kim et al. 2013) defined by row and column boundaries which can be specified with and without uniform spacing. Tiles provide parallelism at a more *coarse* level of granularity (picture/subpicture), and no sophisticated synchronization of threads is necessary for their use (Sullivan et al. 2012).

The main purpose of tiles is to render the use of parallel processing architectures possible for encoding and decoding rather than provide error resilience. Many tiles can share the same information header if part of the same slice, which improves coding efficiency. Alternatively, a single tile can contain multiple slices. Compared to slices, tiles allow picture partitions that are more likely to have higher correlations (Blumenberg et al. 2013) because of its partitioning flexibility, as they may be spatially more compact than a slice containing the same number of CTUs.



0	1	2	3	12	13	14	21	22	23	24
4	5	6	7	15	16	17	25	26	27	28
8	9	10	11	18	19	20	29	30	31	32
33	34	35	36	41	42	43	47	48	49	50
37	38	39	40	44	45	46	51	52	53	54
55	56	57	58	63	64	65	69	70	71	72
59	60	61	62	66	67	68	73	74	75	76

Figure 14: Picture divided into nine tiles, showing the scanning order of the CTUs. Source:(Chi et al. 2012)

When using tiles, the CTUs within a tile and the tiles within a picture are processed in raster scan order, which complicates the raster scan processing typically performed on a single-core implementation. An example of the above-described partitioning is shown in Figure 14 where a picture is partitioned into three columns and three rows of CTUs. The CTUs within the first (upper-left) tile, depicted as numbered squares 0–11, are scanned in raster scan order. After scanning the first tile, the second tile follows (in tile raster scan order). Specifically, as shown in Figure 14, CTU #12 in the second tile follows CTU #11 in the first tile.

Besides changing the CTU scanning process, tile boundaries denote a break in coding dependencies. Dependencies between tiles are disabled in the same manner as between slices. Specifically, entropy coding and reconstruction dependencies are not allowed across a tile boundary (Misra et al. 2013). This includes motion vector prediction, intra prediction and context selection.

The breaking of coding dependencies at tile boundaries implies that a decoder can process different tiles belonging to the same picture concurrently. To facilitate parallel decoding the locations of tiles are signaled in the bitstream. In HEVC, the bitstream offsets of all but the first tile are explicitly transmitted in the slice header. The location (in the bitstream) of the first tile immediately following the slice header is known to the decoder.

An equally strong benefit of tiles is the reduction of memory bandwidth and on-chip memory requirements. Specifically, with a rectangular partitioning, the size of the line buffers required for motion compensation and in-loop filtering is dramatically reduced (the line buffer width is reduced from the width of the picture to approximately the width of the tile. Thus, for even two tiles, the reduction is nearly 50%). For an encoder with significant memory restrictions, it has the additional advantage of higher coding efficiency, as the reduced memory requirements of tiles enables a larger vertical search range for such an encoder.

In short, it can be said that:

- the concept of slice is more useful for *data transmission*;
- the concept of tile is more useful for *parallel processing*.

Increasing the number of slices or tiles in an image can then improve performance for both encoding and decoding. However, in general, the more slices and tiles one defines, the lower the video coding efficiency. This is because the mutually-independent coding of tiles or slices will prevent the encoder from taking advantage of the spatial/redundancies between the tiles or slices.

3.2 Wavefront Parallel Processing (WPP)

WPP is based on the processing of rows of CTUs in parallel. However, differently from slices and tiles, it preserves all coding dependencies by respecting a shift of at least two CTUs between consecutive rows. This shift is needed because the CTU being currently processed requires the left, top-left, top and top-right CTUs (see Figure 15.a) to be available in order for predictions to operate correctly, since intra-frame and motion vector prediction depend on the information of these blocks. An example of the above-described workload scheduling is shown in Figure 15.b, in which CTUs denoted by S_4 can be processed concurrently since their dependency requirements have been fulfilled.

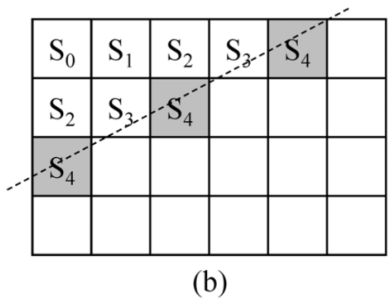
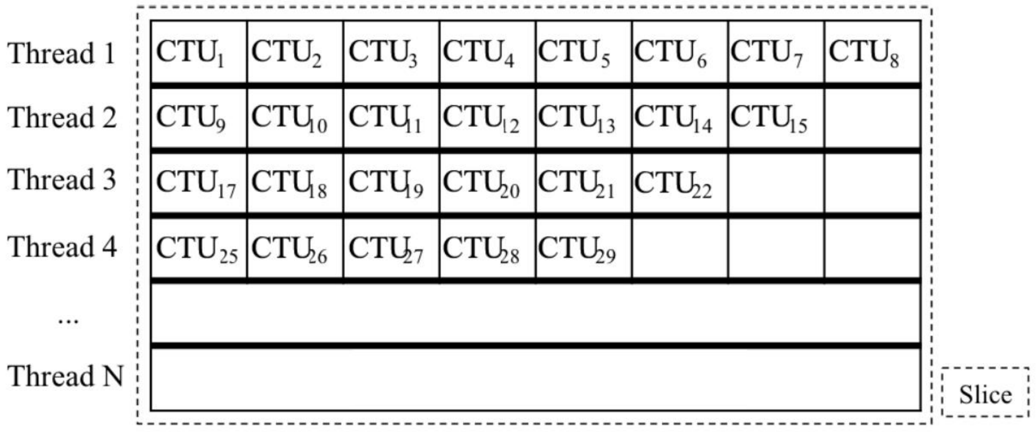


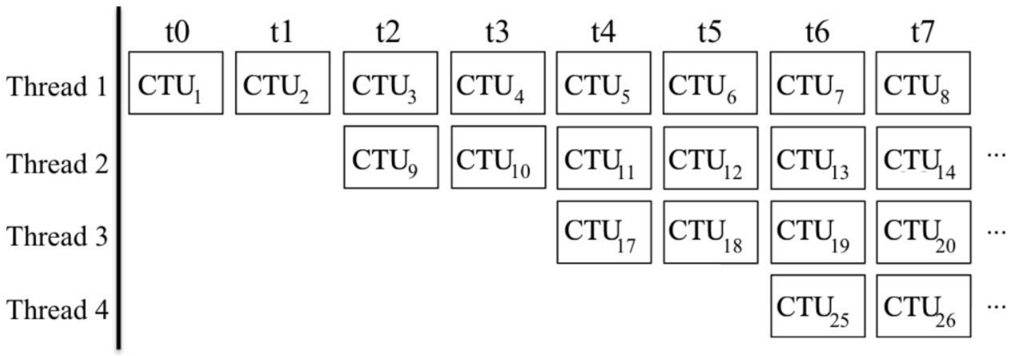
Figure 15: (a) Data dependency on CTU-level (b) CTUs that can be decoded in parallel.

Source: (Jo, Sim, and Jeon 2013)

As shown in Figure 16.b, which shows a timeline of the parallel processing, the second row can only start being processed after at least two CTUs have been processed in the first row, and so on. Additionally, the entropy coding also uses the context from these previously encoded CTUs to benefit from the maximum context information possible on the next thread (Pourazad et al. 2012).


Thread 1
Thread 2
Thread 3
Thread 4
...
Thread N
CTU1 CTU2 CTU3 CTU4 CTU5 CTU6 CTU7 CTU8
CTU9 CTU10 CTU11 CTU12 CTU13 CTU14 CTU15
CTU17 CTU18 CTU19 CTU20 CTU21 CTU22
CTU25 CTU26 CTU27 CTU28 CTU29
Slice

(a)


t0 t1 t2 t3 t4 t5 t6 t7
Thread 1
Thread 2
Thread 3
Thread 4
CTU1 CTU2 CTU3 CTU4 CTU5 CTU6 CTU7 CTU8
CTU9 CTU10 CTU11 CTU12 CTU13 CTU14 ...
CTU17 CTU18 CTU19 CTU20 ...
CTU25 CTU26 ...

(b)

Figure 16: (a) CTU processing of a WPP enabled slice and (b) the Wavefront scheduling per thread.
Source: (Blumenberg 2014)

WPP provides parallelism at a *finer* level of granularity, and some constraints in thread synchronization are set for its use. More concretely, some prediction decisions and adaptations of the context models (refer to subsection 3.3) of the entropy encoder have to be made in the preceding row in order for decoding to start the current row. It means that WPP also requires additional inter-core communication. However, it does not seem to be a burden for current multi-core processor architectures.

Because no coding dependencies are broken at row boundaries (except for the different change of context modeling in the entropy encoder), WPP leads to better visual quality (Bordes et al. 2012) than slices and tiles when comparing sequences with equivalent bitrates. Additionally, it does not change the regular raster scan order.

Being each CTU row processed in parallel, the number of processors that can work in parallel is then up to the number of CTU rows (Figure 16.a) in a frame. As mentioned before, not all the CTU rows can start being processed at the same time because of dependencies, meaning that they also won't finish processing simultaneously. This introduces parallelization (or ramping) inefficiencies (please refer to section 4.6) that become more evident as the number of processing units increases.

Alternative approaches have been proposed to help alleviate this problem, like the *Overlapped Wavefront* (OWF) (Chi et al. 2012), which allows the overlapping of the processing of consecutive frame using wavefronts. It is based on the idea that when a thread finishes a CTU row in the current frame and no more rows are available it can then start processing the next frame instead of being idle (and increasing parallelization inefficiency). However, as OWF is not compatible with the HEVC standard, this work does not consider this alternative in the evaluation.

As it happens with tiles, WPP is generally well suited for the parallelization of the encoder and decoder. It is also suited for both software and hardware implementations and has good speed/coding efficiency trade-off. Although slices and tiles can be together defined in a video frame, for design simplicity the HEVC standard does not support the coexistence of WPP and tiles in the first draft (Sullivan et al. 2012). However, these tools could be combined in the near future (possibly by HEVC extensions) to exploit both fine grain (WPP) and coarse grain (tiles) parallelism.

3.3 Data dependencies

When making use of tiles for parallelism the chosen partitioning scheme creates autonomous partitions which can be independently processed and, in doing so, coding efficiency is compromised in favor of performance speedups. The losses in coding efficiency are explained by the breaking of dependencies and resetting of CABAC contexts along tile boundaries. Since tiles are self-contained, all the tiles inside a frame can be concurrently processed since their CTUs depend solely on the information coming from previously coded frames and the picture region they represent.

Therefore, in terms of synchronization, this data dependency between successive frames determines that a barrier for the group of processing units working on the current frame must be set before the next frame starts being processed. This means that any threads or processes – each one responsible for the workload of a single tile – must stop at this point and cannot proceed until all remaining threads or processes reach this barrier. Consequently, the processing time of a frame in a parallel environment where each tile is assigned to a different processing unit is limited to the time required for encoding or decoding the tile with the most time-demanding workload. This idea will be explored further in subsection 4.6.

Data dependencies concerning the use of WPP are shown in Figure 17. As stated in the previous subsection, some adaptations of the context models of the entropy encoder have to be made in the preceding row in order for encoding to start in the current one.

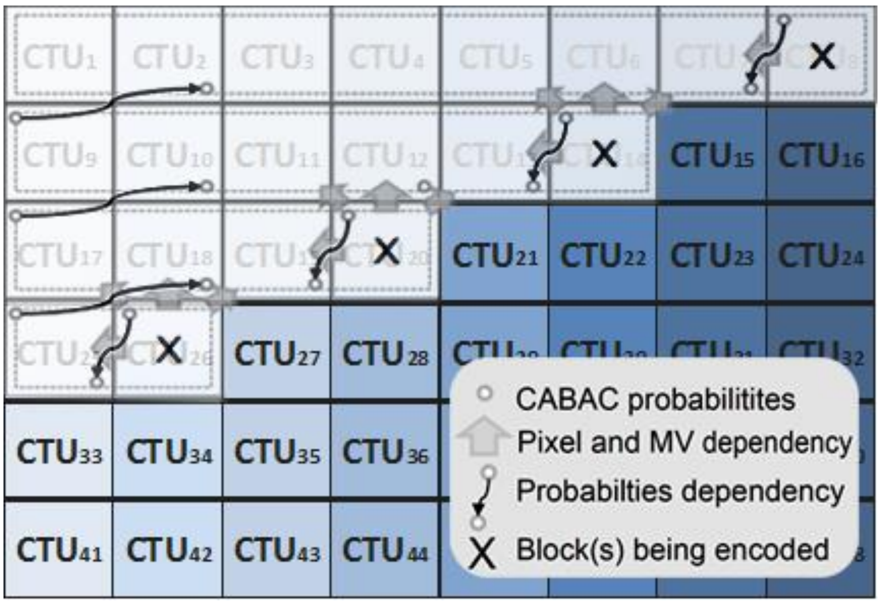


Figure 17: Data dependencies in Wavefront Parallel Processing

More specifically, CABAC encoding is flushed after the last of each CTU row, making each row of CTUs accessible using entry points. If the CTU being encoded is the first one in its row, e.g. CTU #9, then this CTU depends on CABAC probabilities coming from the CTU on its upper-right (CTU #2 in this case). If the CTU being currently encoded is not the first in its row, e.g. CTU #2, then this CTU depends on CABAC probabilities coming from the CTU on its left (CTU #1 in this case). CTU #1, being the first CTU in the frame, is the only one not having any data dependencies. This adaptation complies with data dependencies required when carrying out both intra-frame and motion vector predictions, and therefore there must be a shift of a least two CTUs between the execution of consecutive rows.

In terms of synchronization, this data dependency between rows determines that the row being currently processed must wait until the previous one has finished processing its second CTU. Additionally, synchronization barriers must be set in order to guarantee that the processing of CTUs in different rows start at the same time. This means any thread or process – each one responsible for the workload of a single CTU – must stop after completing its job and cannot proceed until all other threads or processes reach this barrier. Consequently, the processing time of a frame in a wavefront environment is the interval from the start of the processing of first CTU to the end of the last CTU in the frame.

4 EXPERIMENTAL SETUP

The experiments were performed using the HEVC Test Model (HM) 14.0 encoder (refer to subsection 4.1), following the *Common Test Conditions* (Bossen 2012) adopted by HEVC standard developers. A total of ten benchmark video sequences were analyzed: four from Class A (*2560x1600* pixels, with 8-bit and 10-bit depth), three from Class B (*1920x1080* pixels), one from Class C (*832x480* pixels), one from Class D (*416x240* pixels) and one from Class E (*1280x720* pixels), as shown in Table 1. A greater number of video sequences from classes A and B was selected since the exploitation of parallelism is more important in high resolution video processing.

Class	Sequence	Resolution	CTUs per frame
A	*NebutaFestival*	2560x1600	40x25
	PeopleOnStreet	2560x1600	40x25
	SteamLocomotiveTrain	2560x1600	40x25
	Traffic	2560x1600	40x25
B	*BasketballDrive*	1920x1080	30x17
	BQTerrace	1920x1080	30x17
	ParkScene	1920x1080	30x17
C	*BasketballDrillText*	832x480	13x8
D	*BQSquare*	416x240	7x4
E	*FourPeople*	1280x720	20x12

Table 1: Video sequences used in the analysis

From the HEVC common conditions, the random-access (RA) (refer to section 4.2) settings were selected for the encoder, which achieves the highest compression efficiency and includes, among others, CABAC entropy coding and SAO filter (Bossen 2012).

For the comparison of coding losses the *Bjφntegaard Delta Bit Rate* (BD-rate) was computed using a weighted average for *luma* and *chroma* components (0.75Y + 0.125U + 0.25V) for all the videos. BD-rate is a metric that represents the bit-rate percentage difference in logarithmic scale between two videos for the same video quality (BJONTEGAARD 2001) measured with peak signal-to-noise ratio (PSNR) objective video quality metric. As a baseline, a configuration without picture partitions was used (i.e., one slice and one tile per picture, no WPP). All the sequences used in this work were encoded with four *quantization parameters* (QP) values (22, 27, 32, and 37) and fixed CTU width and height of 64 pixels.

In the experiments, we pin each decoding thread to only one core to reduce the influence of the OS scheduling policy and to improve the reproducibility of the results. The experiments are performed for all sequences that have been listed in Table 1 and using 1 to *n* encoding threads, where *n* is parameterized according to the different evaluations in the analysis.

The methodology adopted in this work for performance and coding efficiency evaluation is explained in this section.

4.1 The HEVC Test Model (HM)

The HEVC reference software codec is called *HEVC Test Model* (*HM*) (*HEVC HM Reference Software* (version 14.0)). In earlier HM versions, the coding tools have been separately specified for *Low Complexity* (*LC*) and *High Efficiency* (*HE*) operations in order to examine the different tradeoffs between coding efficiency and coding complexity (McCann et al. 2012). HM 5.0 introduced a separate HE10 configuration for 10-bit operation mode besides HE and LC modes. Since the launching of HM 6.0, the tools of HM have been divided between MP (Main Profile) and HE10 (High Efficiency 10-bit).

Currently, HM 14.0 is the latest version of HM and two versions of each coding profiles are provided: *Main* and *Main10*. The coding tools for the Main configuration correspond to those supported in the Main profile of the HEVC specification (Kim et al. 2013). The coding tools for the Main 10 configuration correspond to the Main 10 profile in the HEVC specification with the bit depth for both luma and chroma set to 10 bits.

HM testing is recommended to be accomplished according to common test conditions (Bossen 2012) which include four predefined coding profiles: *all-intra* (*AI*), *random-access* (*RA*), *low-delay P* (*LP*), and *low-delay B* (*LB*). In the context of this work, the random-access coding profile was select for the tests.

4.2 Random-access coding configuration

As stated previously, random-access settings were chosen for the experiments performed in this work because they achieve the highest compression efficiency and includes a vast amount of coding tools.

In this configuration, encoding order is different from the display order. Encoding order is instead based on a hierarchical B structure. Figure 18 shows a graphical representation of a random-access configuration, where the number associated with each picture represents the encoding order.

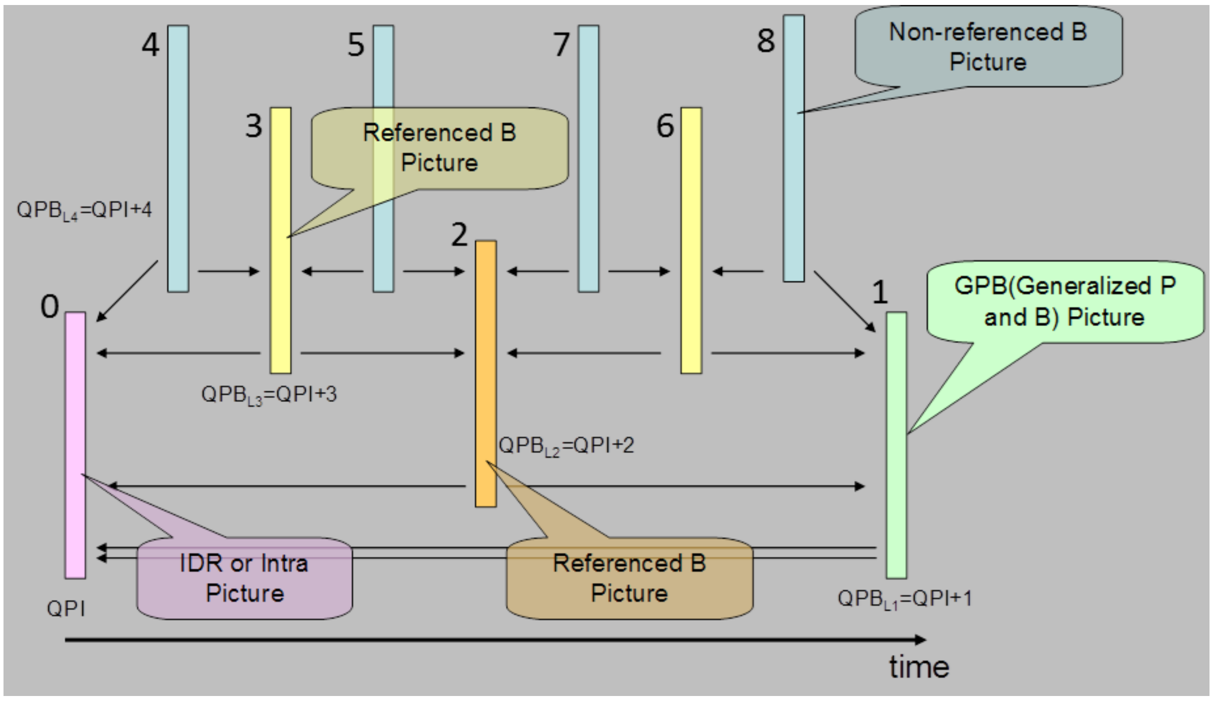


Figure 18: Graphical presentation of the random-access configuration. Source: (Kim et al. 2013)

Intra-pictures are encoded at approximately one second intervals. The first picture of a video sequence is encoded as an IDR (Intra Decoder Refresh) picture and the other intra pictures are encoded as non-IDR intra pictures. IDR pictures are used to reduce error propagation impact (Sullivan et al. 2012) and to enable random access points, as they do not have dependencies with other pictures. The pictures located between successive intra pictures (in display order) are encoded as B-pictures. The generalized P and B picture (GPB) is used as the lowest temporal layer that can refer to I or GPB pictures for inter prediction. The second and third temporal layers consist of referenced B pictures, while the highest temporal layer contains non-referenced B pictures only. The QP of each inter-coded picture is derived by adding an offset to the QP of the intra-coded picture depending on the temporal layer. The reference picture list combination is used for management and entropy coding of the reference picture index. The encoding process for frames with longer reference pictures lists will also take longer time, as shown in the experimental results in chapter 5.

4.3 Encoding using Tiles

For the evaluation of tiles, a test scenario consisting of three tiling patterns was adopted: *2x2*, *3x3*, and *4x4* tiles, totalizing four, nine, and sixteen partitions per picture, respectively. These tiling patterns are encoded using the uniform-spaced tiling approach. The number of partitions produced by each configuration reflects well the number of processing units present in current multi-core systems. The uniform-spaced option can be activated by enabling the *UniformSpacingIdc* parameter in the encoding configuration file. By doing so, the column and row boundaries are distributed uniformly in *NumTileColumnMinus1* + 1 columns and *NumTileRowsMinus1 + 1* rows. All the results regarding coding efficiency are presented in terms of BD-rate (BJONTEGAARD 2001) percentage increase (meaning that a smaller increase in percentage represents higher gain in coding efficiency) in comparison to the baseline, partition-free configuration.

The results were obtained by means of a parallel environment simulation able to measure the time required to encode each tile based on the encoding time of each CTU belong to it. All the video sequences shown in Table 1 are encoded for different QPs and partitioning schemes, generating performance- and PSNR-related data that was then used for simulations along with the desired number of processing units. The simulated parallel environment is capable of processing each tile in parallel (each tile has to be processed by a single processing unit) and the adopted tile scheduling algorithm respects the conditions presented in subsection 3.3, which means that a frame can only start being encoded once all tiles from reference frames are completely encoded.

4.4 Encoding using Wavefront Parallel Processing (WPP)

For the evaluation of WPP, one partitioning scheme is defined for each video sequence. As explained in subsection 3.2, the number of picture partitions when using WPP is fixed to one partition per CTU row. It results in 25, 17, 12, 8 and 4 picture partitions for 1600p (Class A), 1080p (Class B), 720p (Class E), 480p (Class C) and WQVGA (Class D) resolutions, respectively. All video sequences were encoded with the option *WaveFrontSynchro* enabled, which produces a number of substreams equivalent to the number of CTU rows in each frame.

In the wavefront encoder, each encoding thread processes a CTU row of the picture at a time. WPP dependencies are maintained between the encoding threads by the use of synchronization barriers. Each encoding thread checks the progress of the thread processing the previous row before encoding the next CTU in its row. To maintain WPP dependencies, the thread processing the previous row must have processed the top-right CTU to the current CTU. In other words, at any time the thread processing the previous row must have processed *two* CTUs in advance than the thread processing the current row. When the number of CTU rows is larger than the number of processing units, then a new row must wait until one of the processing units is available.

As mentioned in subsection 3.3, some adaptations of the context models of entropy encoder are adopted when using WPP. The following operations are performed by the HM encoder:

- When starting the encoding of the first CTU in a CTU row, the following process is applied:
    - if the last CU of the second CTU of the row above is available, the CABAC probabilities are set to the values stored in the buffer.
    - if not, the CABAC probabilities are reset to the default values.
- When the encoding of the second CTU in a CTU row is finished, the CABAC probabilities are stored in a buffer.
- If the encoding of the last CTU in a CTU row is finished and the end of a slice has not been reached, CABAC is flushed and a byte alignment is performed.

Resetting the CABAC probabilities to the default values incurs losses in coding efficiency, which will be verified in subsection 5.1.

4.5 Parallelism Potential

In this work, an estimate of the theoretical processing time is computed and used for evaluation. This estimate will be referred to as *parallelism potential* and is measured in seconds. For simplicity, the ratio between the sequential execution time (encoding time for the baseline configuration) and parallelism potential will be called *speedup*, as shown in Eq. 1. Therefore,

$$Speedup = \frac{S_T}{P_T} \qquad (1)$$

. In (1), S_T is the sequential execution time, and P_T is the parallelism potential.

When using tiles, parallelism potential can be represented by the sum shown in Eq. (2).

$$P_{T_{Tiles}} = \sum_{i=1}^{n} t_{Tiles_i} \qquad (2)$$

In (2), n is the number of frames of the video sequence, and $t_{Tiles_i}$is the time needed to compute the frame i of the sequence. The value of $t_{Tiles_i}$is equal to the required processing time of the tile which have the most time-demanding workload in frame i.

When using WPP, the sum shown in Eq.2 representing parallelism potential is analogous, except for the fact that t_{WPP_i} - i.e., the time needed to compute the frame i of the sequence when using WPP – is defined by the time interval from the start of the processing of the first CTU to the end of the last CTU in frame i.

4.6 Parallelization Inefficiency

Directly related to workload unbalance and/or excess of computational resources to exploit parallelism (more processing units than available tasks), *parallelization inefficiency* is used in this work as an estimate of the amount of time in the encoding of a sequence during which a least *one* processing unit is *idle* while there is still workload to process. A processing unit is idle only when no workload is assigned to it. Parallelization inefficiency is measure in seconds and its definition is based on the concept of *dependency stalls*.

Dependency stalls originate from the variability in CTU execution times. WPP execution must respect the dependencies to the top-right CTU, which results in dependency stalls when the difference in CTU execution times between rows becomes too large. With higher resolutions there is more tolerance to these execution time differences and, therefore, have better scalability.

WPP has in addition to the dependency stalls also the *ramping stalls*, which occur at the beginning and end of encoding a frame when not all encoding threads can be active. Although having no dependencies between one another in the same frame, tiles also do not scale ideally. This also originates from the load balancing problem, in which not every encoding thread gets the same amount of work to process. For instance in 1600p with 4x4 tiles: this does not divide evenly when there are, e.g., four encoding threads. Furthermore, being composed by CTUs, tiles also vary in execution time.

5 EXPERIMENTAL RESULTS AND EVALUATION

5.1 Coding efficiency

The parallelization approaches proposed by the HEVC standard rely on the creation of picture partitions. Coding losses may appear due to breaking dependencies for prediction, CABAC context modeling, and/or slice header overhead. In general, more picture partitions leads to higher compression losses.

To better understand the impact of picture partitions (introduced by parallel tools) on coding efficiency, Figure 19 shows the BD-rate increases caused by the use of three different tiling patterns (*2x2*, *3x3* and *4x4*) and WPPFigure 19. This test scenario considers video sequences from classes A, B, C, D and E shown in Table 1. In this evaluation, the video sequences encoded using uniform tile partitioning and WPP are compared to baseline sequences (one slice and one tile per picture, no WPP).

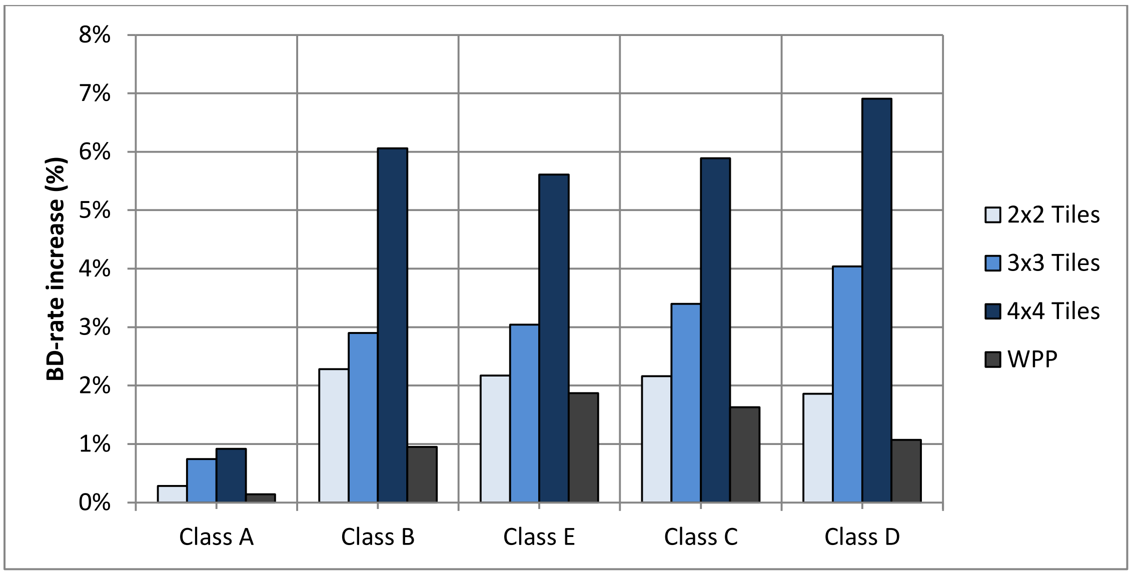


Figure 19: Coding efficiency impact (compared with baseline) for different parallelization methods and video sequence classes

For all video classes analyzed the impact on coding efficiency increases with the number of tiles. This happens because the number of breaks in coding contexts also increases with the number of tiles. The BD-rate increase when using WPP is the lowest compared with using tiles. It confirms that maintaining and respecting prediction dependencies between CTUs reflects directly on coding efficiency. Because WPP allows crossing partitions for both entropy encoding and prediction, the small remaining losses are due to the created entry points which change CABAC context modeling.

5.2 Results of encoding using tiles

In the context of encoding using tiles, two main investigations were performed. The first one, presented in subsection 5.2.2, aims to analyze coding performance for an incremental raise in computational resources - i.e. working threads - for different tiling patterns. The second one, presented in section 5.2.4, aims at the comparison of tiling patterns when the number of working threads matched the number of partitions.

However, it is interesting to analyze some results regarding tile workload distribution beforehand.

5.2.1 Workload distribution among tiles

Figure 20 shows the time required to encode each tile in the first 30 frames of the *NebutaFestival* sequence, comparing the baseline with the 3x3 tile partitions configuration. Both charts present a very similar pattern, reaching maximum execution time for frames 10 and 18. This is a behavior shared by all video sequences listed on Table 1 and can be explained by Figure 18 in subsection 4.2. What happens is that these frames, in encoding order, have considerably longer frame reference lists in comparison to others, which renders their predictions more computationally demanding.

The white bars in Figure 20.b represent the difference in workload between the higher and lower time-demanding tiles in each frame – a measure of parallelization inefficiency -, which will be referred to as *maximum workload variation*. Figure 21 shows the values of average maximum work variation measured for all video sequences in Table 1 and the three different tiling patterns.

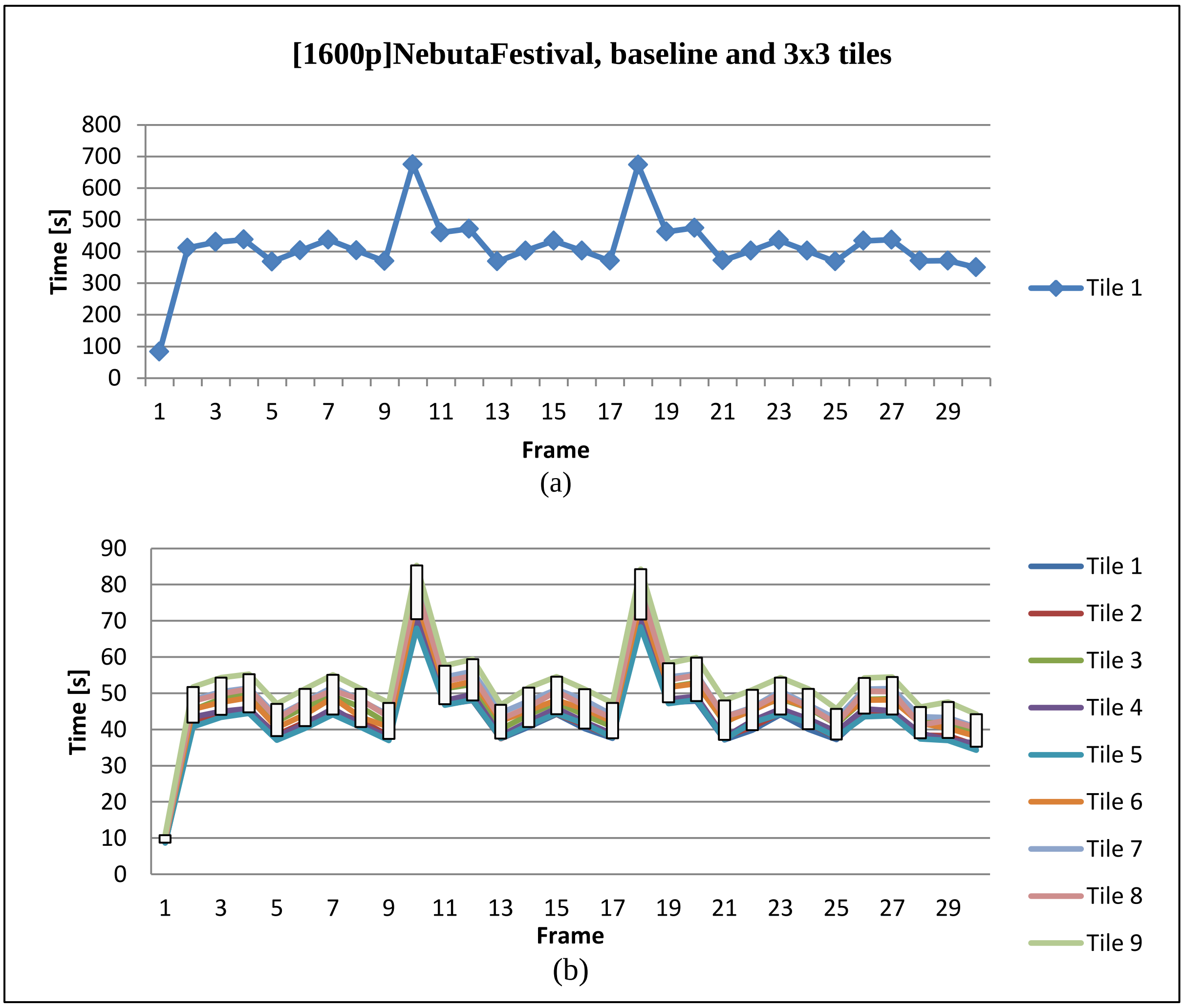


Figure 20: encoding time for each tile and workload difference, (a) baseline and (b) 3x3 tiles configuration

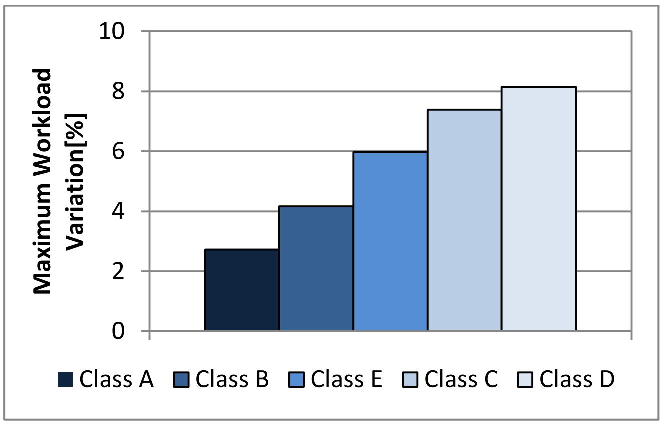


Figure 21: maximum workload variation between tiles as resolution changes

The average maximum workload difference tends to increase as the resolution decreases. The reduction in the number of CTU rows and columns in a frame causes the number of CTUs in each tile to also get smaller. Neighbor CTUs in lower resolutions tend to have a much lower correlation and thus vary in processing time.

5.2.2 Incremental speedup

Based on the experimental setup presented in subsection 4.3, the initial experiments performed have as main objectives the evaluation of performance gains for fixed tiling patterns and variable number of processing units always lower or equal to the number of partitions. Therefore, the number of processing units ranges from one to the total number of tiles in each frame of the sequence. Concretely, for *2x2* tiles, the number of working processing units will vary from 1 to 4. Since tile partitioning is defined be the encoder, the situation where the number of threads is lower than the number of partition is not possible but very likely to happen in the decoding process (to which the scalability results can be also applied to).

The results shown in subsection 5.2.4 show that the average gains of a video sequence can be generalized to those of its class. Therefore, one video sequence of each class was selected for this fine-grain analysis, namely *NebutaFestival* (Class A), *BasketballDrive* (Class B), *BasketballDrillText* (Class C), *BQSquare* (Class D) and *FourPeople* (Class E).

Figure 22 and Figure 23 present the results in two different ways: grouped by tiling pattern and by video sequence, respectively. Figure 22 shows that the average speedup increases with the number of tiles.

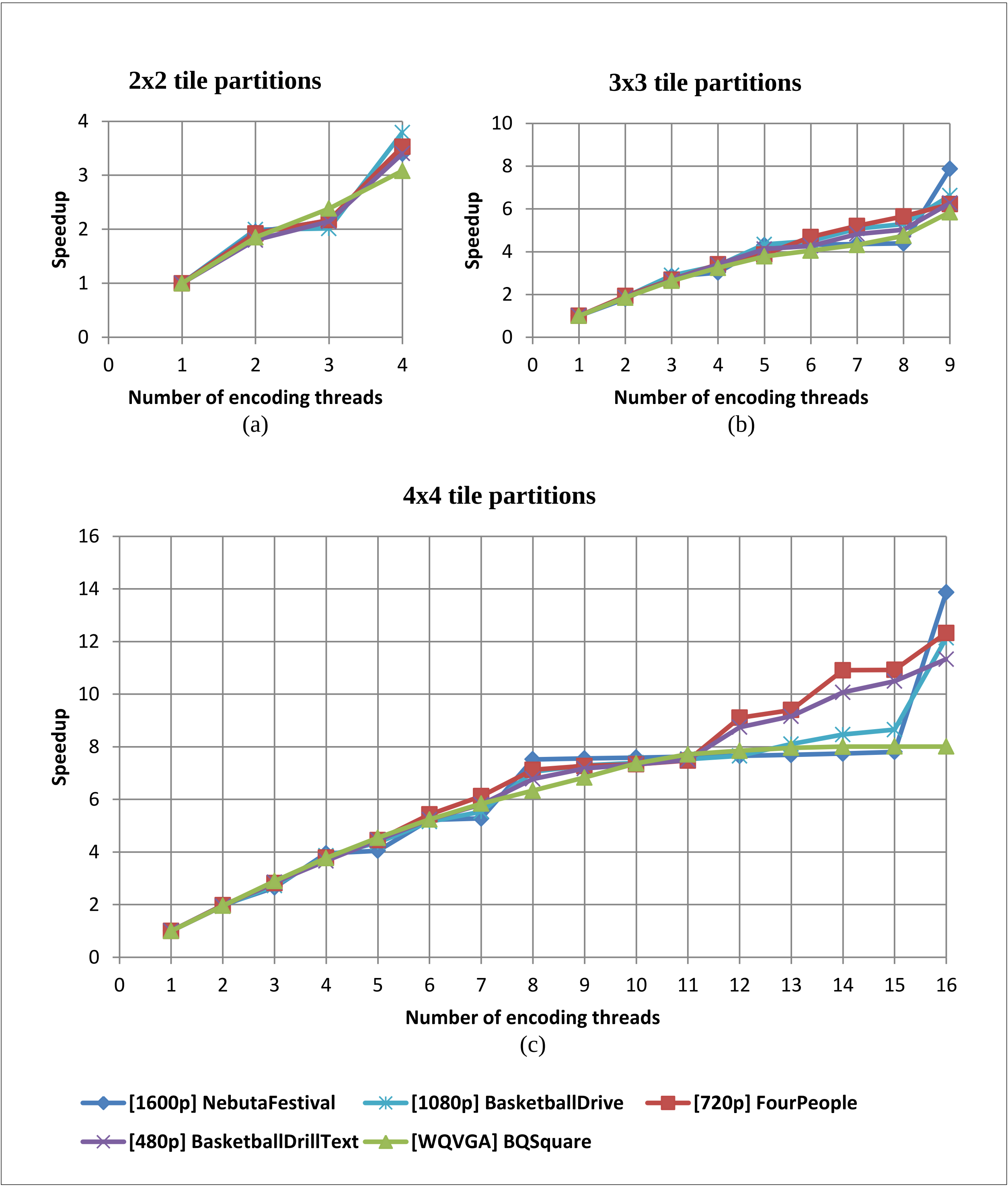


Figure 22: Speedup for different tiles partitions and video sequences

Moreover, there seems to be very little variation in speedups for a number of threads varying from 1 to around one third of the total number of partitions. It means that devices with fewer cores might benefit from a tiling pattern with less partitions and thus resulting in higher bitrate savings.

In Figure 23.a, the existence of several points of intersection in curves representing different tiling patterns indicates interesting possibilities. Using a number of threads ranging from 9 to 15 produces basically the same results for either *3x3* or *4x4* tiles. It means that the adoption of a tiling pattern associated with lower coding efficiency losses might be the best option. As previously mentioned, fewer tiles produce a lower increase in BD-rate, meaning that the *3x3* tiling pattern is preferable over the *4x4*, in this context, in opposition to the conception that a setting with more partitions and processing units will always perform better.

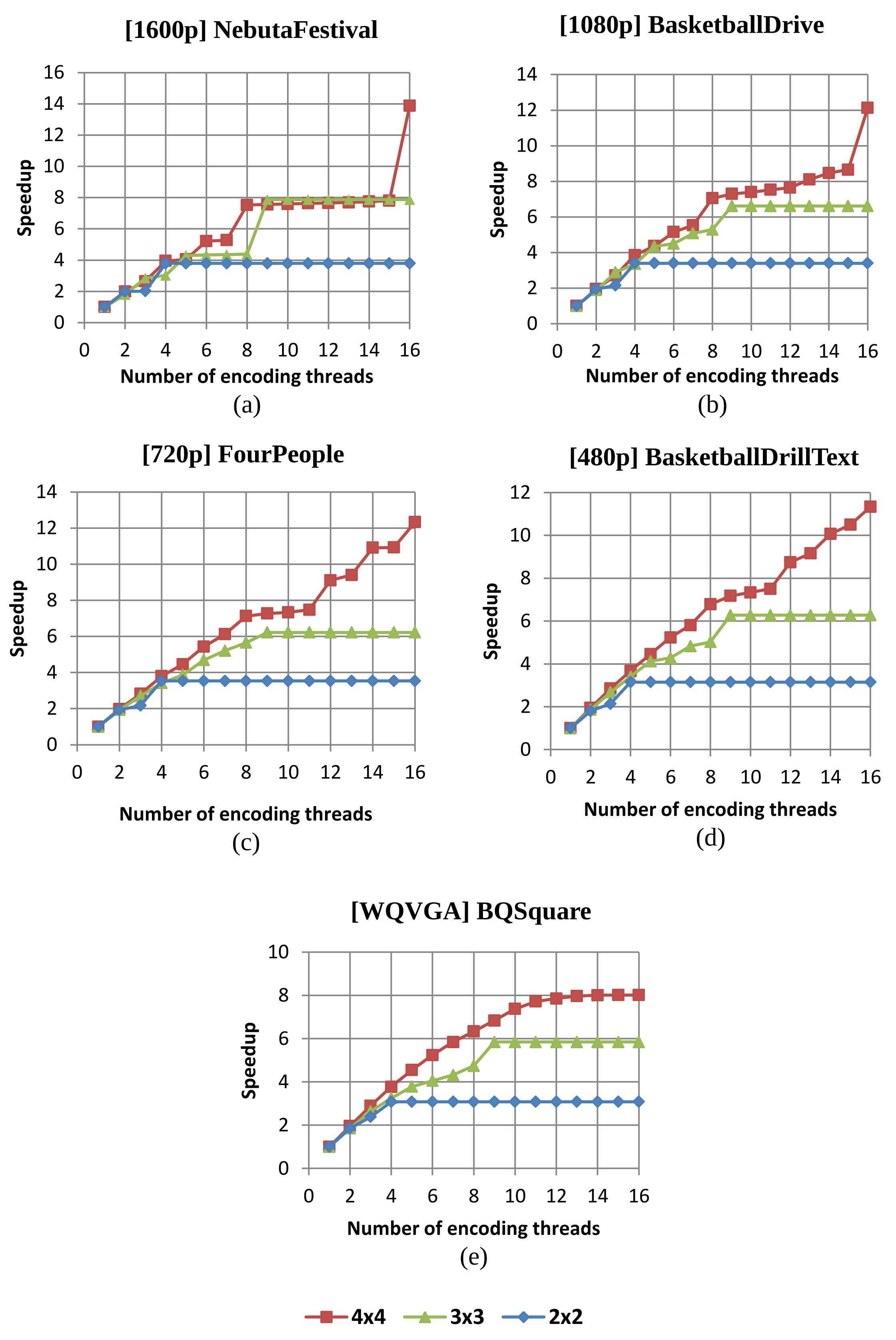


Figure 23: Incremental speedups for (a) 1600p (Class A), (b) 1080p (Class B), (c) 720p (Class E), (d) 480p (Class C) and (e) WQVGA (Class D) video sequences using different tiling patterns

Especially for Class A sequences (Figure 23.a), there seems to be intervals in the curves - e.g. number of threads varying from 8 to 15, *4x4* tiles - during which the incremental speedup reaches a semi-plateau. This behavior gets more and more subtle as the resolutions decrease. This behavior is less perceptible for Class D sequence (Figure 23.e). The following subsection explains the reason for this behavior.

5.2.3 Tile scheduling

In order to explain the behavior of the curves shown in Figure 23, it is very important to thoroughly analyze the thread scheduling algorithm in action when tiles are used. Figure 24 shows a *512x384* pixel frame of a hypothetical video sequence encoded with 3x3 uniformly spaced tiles. Because the CTU size is set to 64 pixels in height and width, it is not possible to divide the CTUs evenly among the 9 tiles. This is the reason why tiles 3, 6 and 9 have less CTUs than others.

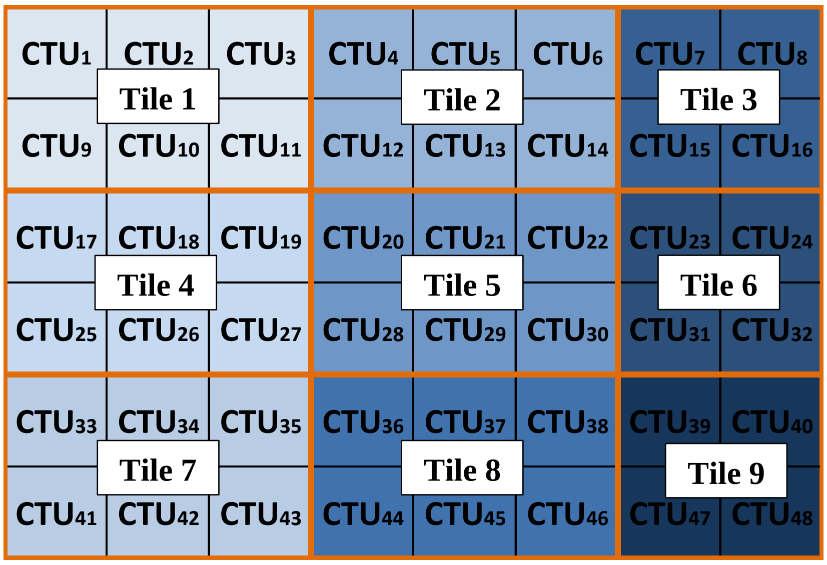


Figure 24: 3x3 tile partitioning scheme for 8x6 CTU/frame sequence

As mentioned in subsection 4.6, the variability in CTU execution times, as well as the differences in number of CTUs in each tile, generates variations in the execution times of tiles. Figure 25 shows tile scheduling for *3x3* tiles, 3 processing units and unevenly balanced workloads. Tiles are assigned to threads as soon as they finish processing their previous task.

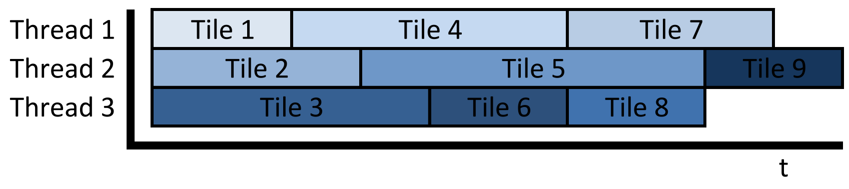


Figure 25: Tile scheduling for 3x3 partitions and 3 processing threads, unbalanced tile workloads

In a hypothetical setting with the same tiling pattern and number of processing units, but where the workloads of all tiles, and thus their execution times, are the same, the scheduling timeline would be correctly represented by Figure 26.

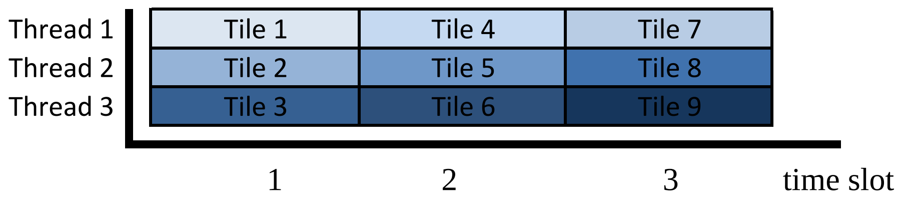


Figure 26: Tile scheduling for 3x3 partitions and 3 processing threads with balanced tile workloads

As it happens, because tile execution times are the same, the concept of *time slot* can be used to help explain curves shown in Figure 23. In this context, a time slot will be represented by the period of time assigned to the execution of a column of tiles. In Figure 26, time slot 1 represents the amount of time required for the concurrent execution of tiles 1, 2 and 3. The sum of time slots 1, 2 and 3 represents the amount of time required to process the whole frame.

Figure 27 shows how the scheduling of this frame would work for a number of threads varying from 1 to 9, which is the total number of partitions in its tiling pattern. The cases where the columns are colored in red are the ones where there is a reduction in time slots required for the processing of the frame compared to the scheduling directly above them. Therefore, noticeable gains in speedup are expected to happen when using 2 (Figure 27.b), 3 (Figure 27.c), 5 (Figure 27.e) and 9 (Figure 27.i) processing units for a *3x3* tile partitioning scheme.

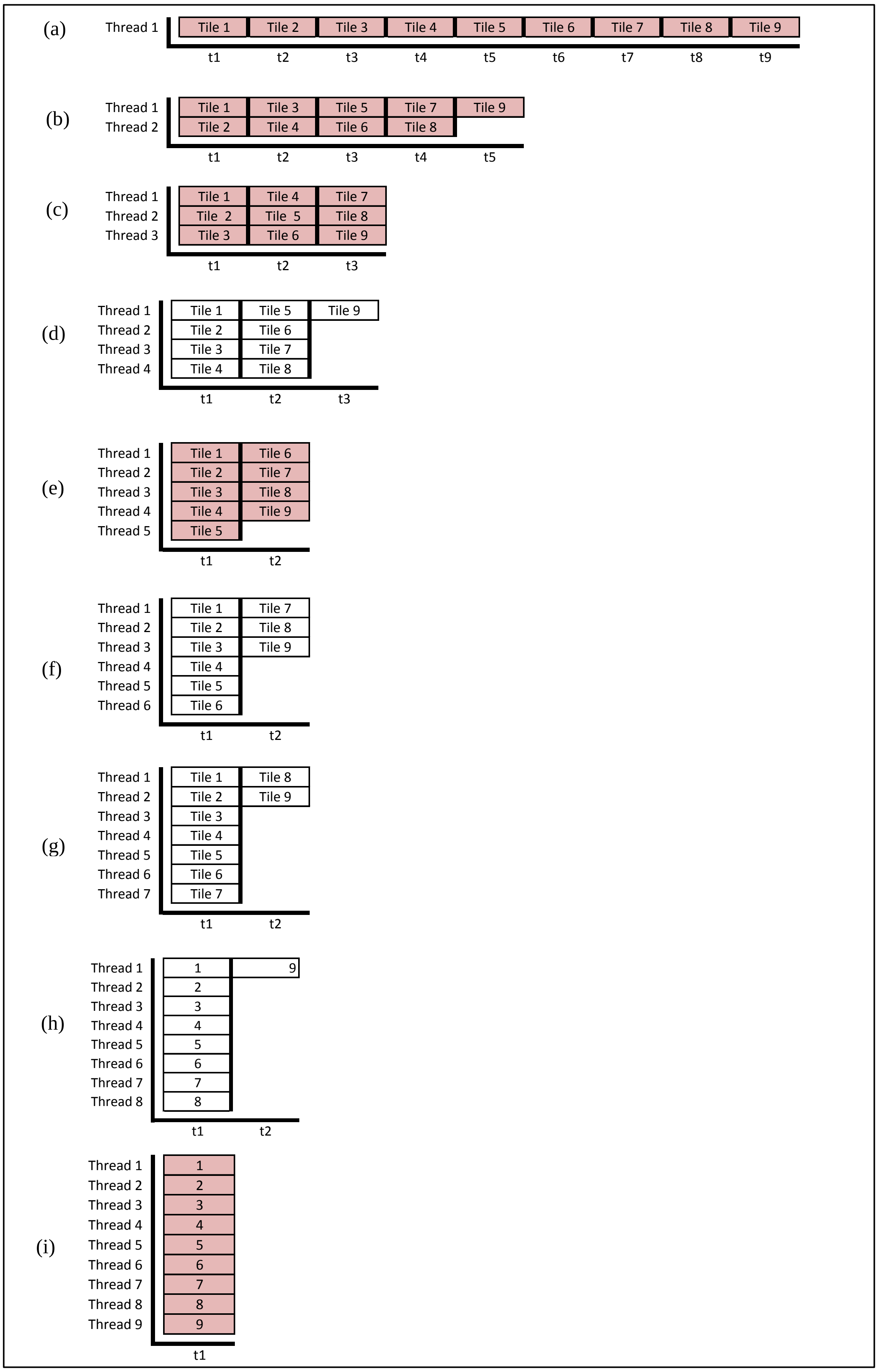


Figure 27: Tile scheduling for (a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6 (g) 7 (h) 8 and (i) 9 processing threads, balanced tile workloads

The following algorithm written in pseudo-code identifies the values in processing units that result in larger incremental speedups:

```
$previous = nbTiles$
$for(\ i = 2\ ;\ i \leq nbTiles;\ i++)$
{
    $value = \ ceiling(\frac{nbTiles}{i})$
    $if\ \boldsymbol{value}\ != \ \boldsymbol{previous}$
            $add\ \boldsymbol{i}\ to\ the\ list\ of\ points$
    $previous = value$
}
```

In this function, *nbTiles* is the total number of tiles in each frame and *ceiling* is the function that maps a real number to the smallest following integer.

For the example analyzed, values 2, 3, 5 and 9 present this characteristic:

$$i = 2 \qquad \left\lceil \frac{9}{2} \right\rceil = \mathbf{5}$$

$$i = 3 \qquad \left\lceil \frac{9}{3} \right\rceil = \mathbf{3}$$

$$\vdots$$

$$i = 5 \qquad \left\lceil \frac{9}{5} \right\rceil = \mathbf{2}$$

$$\vdots$$

$$i = 9 \qquad \left\lceil \frac{9}{9} \right\rceil = \mathbf{1}$$

When applying the algorithm for the other tiling patterns adopted in this work, we obtain the results shown in Table 2. Values for other *nxn* tiling patterns can also be calculated.

Tiling pattern	**Values**
2x2	2, 4
3x3	2, 3, 5, 9
4x4	2, 3, 4, 6, 8, 16

Table 2: Number of processing units for which prominent speedups are expected in each tiling pattern

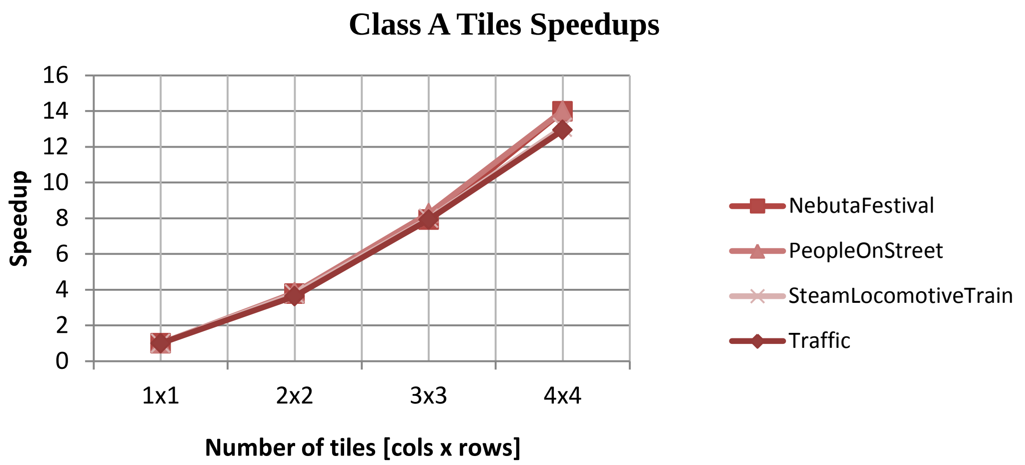
Even though tile workloads are very frequently unevenly balanced and the concept of time slots can be no longer applied, Figure 23.a shows that prominent speed-ups are indeed obtained for the values presented in Table 2 for a Class A sequence, for example. This happens due a low variability in execution times between tiles (because of the increased number of CTUs per tile). As for the other video classes, it is safe to assume that as the resolution decreases (and thus tiles decrease in number of CTUs), tile execution times no longer show a low variability, alleviating the effect of the scheduling. With a small number of CTUs in each tile, it is expected that the workload difference between different tiles will increase since neighboring CTUs tend to be very similar in terms of processing requirements. In smaller resolutions this effect is even more notorious because highly correlated regions are also smaller in number of CTUs.

In Figure 23.a, as stated before, the use of a number of threads ranging from 9 to 15 produces basically the same results for either *3x3* or *4x4* tiles. Because the 3x3 tiling pattern incurs less coding efficiency losses, it is the best option.

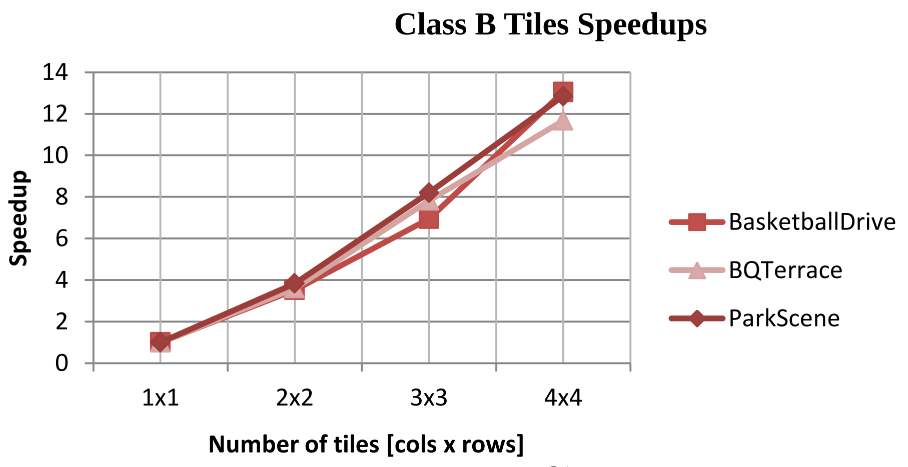
It is interesting to point out how the curve representing the incremental speedup for a Class D sequence (Figure 23.e) with *4x4* tiles saturates gradually, differently from others. This video sequence has only *7x4* CTUs in each frame, which means its 16 tiles contain each one only 1 or 2 CTUs. In this case, if a single CTU (that can represent 100% or 50% of the tile) needs much more time to be processed than the rest, then it will compromise the scheduling. In this case the incremental speedups have progressively smaller slopes and parallelization for tiles reaches a limit. .

5.2.4 Number of tiles matching the number of encoding threads

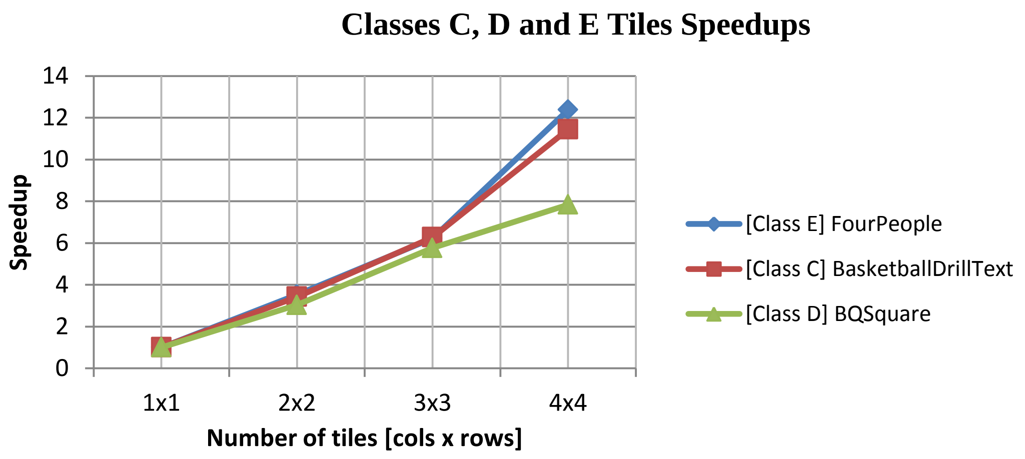
This experiment considers the speedup obtained for each tiling pattern when the number of working processing units is equal to the number of tiles used. As can be seen in Figure 28, higher speedups are obtained for higher resolutions: while Class A sequences present an average speedup of about 14 times for *4x4* tiles, the same number of threads and tiling pattern offers average speedup gains of 12, 11 and 8 times for Classes B, C and D, respectively. This happens due to the effect mentioned in the previous sections. For Classes A and B, for which more than one sequence was encoded, it is important to point out that speedup does not vary significantly between sequences; this variation increases slightly with the number of threads/partitions.



(a)



(b)



(c)

Figure 28: Tile partitioning speedups for (a) Class A, (b) B, (c) C, D and E video sequences.

In theory, the small differences in performance gain between video sequences of the same resolution and same tiling pattern could be explained in terms of workload unbalance. That is, video sequences with better workload division should provide larger speedups. According to this, *NebutaFestival* for example, should exhibit a lower average inter-tile workload difference (*1.334s*) than *Traffic* (*2.365s*) for the same partitioning scheme. These values represent the average differences in time processing for the tiles with the highest and lowest time-demanding workloads in each frame.

5.3 Results of encoding using WPP

Similarly to tiles, two main investigations were also performed for WPP. Firstly, a fine-grained analysis was carried out. One sequence per video class was selected (the same used in subsection 5.2.2) and the encoding using WPP was executed for a number of processing units ranging from 1 to 25 (the number of CTU rows in a Class A sequence). The objective of this first analysis is to show the evolution of incremental speedup when using WPP, as well as its parallelization limits.

The second analysis encompasses all video sequences presented in Table 1. It has main objective an intra-class comparison of performance gains for Classes A and B.

5.3.1 Fine-grained analysis of incremental speedup

Figure 29 presents the results for the first experiment performed.

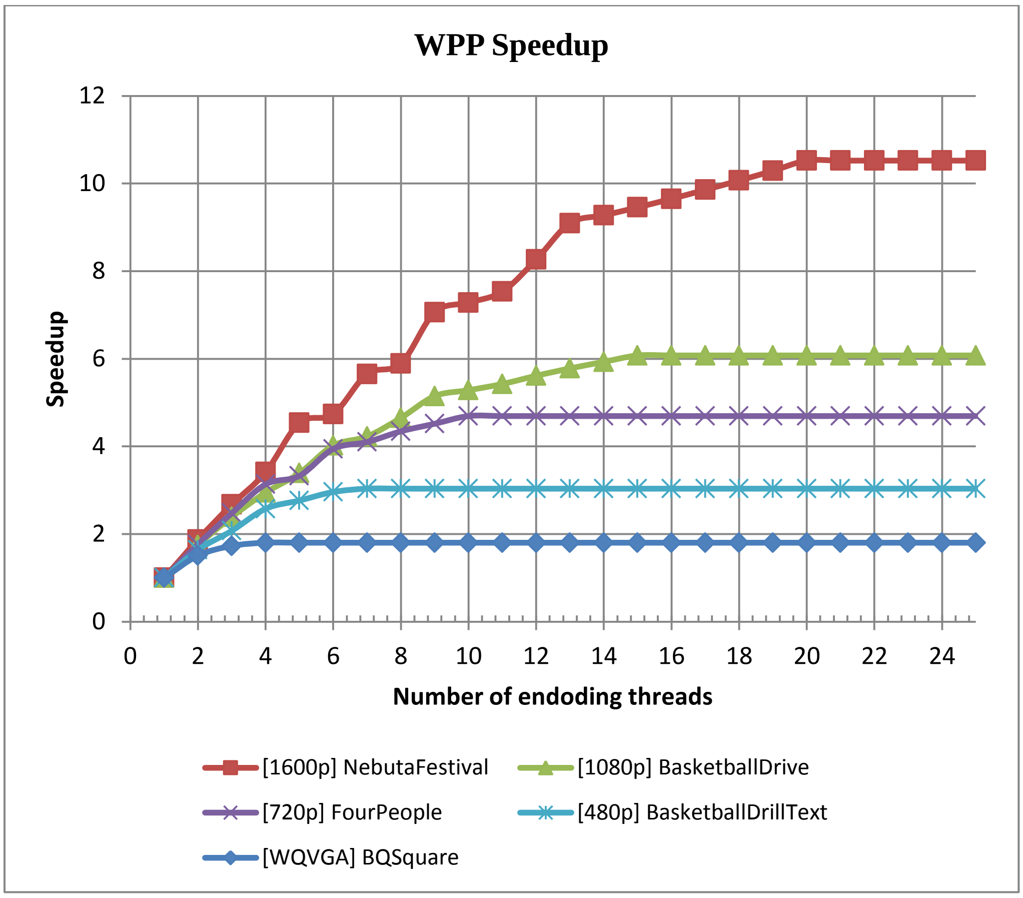


Figure 29: Speedup of encoding using WPP for different resolutions

The gain in speedup when using WPP is directly proportional to the number of CTU rows in a frame of each sequence: 25, 17, 8, 4 and 12 for Classes A, B, C, D and E, respectively. It is clear, however, that WPP does not scale linearly because of its inherent parallelism ramp-up and ramp-down. The curves reach plateaus *before* the number of processing units equalizes the aforementioned values. In the *NebutaFestival* sequence results, for example, there are no more gains in speedup as soon as the number of encoding threads reaches 20. The reasons behind these limitations are discussed in the following subsection.

5.3.2 Wavefront parallelization limits

The initial thought when considering WPP is that the number of CTU rows in each frame of the video sequence is a natural limit to parallelization. However, the results obtained through the encoding tests and calculation of parallelism potential show another important limitation to acceleration when it comes to the wavefront strategy.

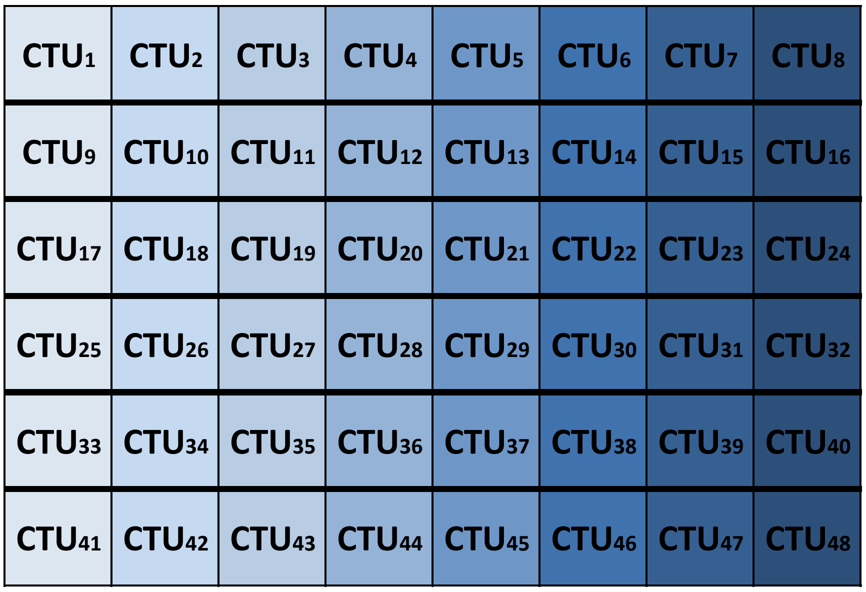

CTU1 CTU2 CTU3 CTU4 CTU5 CTU6 CTU7 CTU8
CTU9 CTU10 CTU11 CTU12 CTU13 CTU14 CTU15 CTU16
CTU17 CTU18 CTU19 CTU20 CTU21 CTU22 CTU23 CTU24
CTU25 CTU26 CTU27 CTU28 CTU29 CTU30 CTU31 CTU32
CTU33 CTU34 CTU35 CTU36 CTU37 CTU38 CTU39 CTU40
CTU41 CTU42 CTU43 CTU44 CTU45 CTU46 CTU47 CTU48

Figure 30: A video frame divided into 8x6 CTUs

For this investigation, the same *512x384* pixel frame (Figure 30) of a hypothetical video sequence seen in subsection 5.2.3 is encoded using the wavefront approach. Six substreams, each one containing one CTU row are created and processed in accordance to the data requirements discussed in subsection 3.3.

When the number of encoding tiles is 3, the necessary synchronization barriers will determine a scheduling timeline for the processing of this frame than is represented by Figure 31.

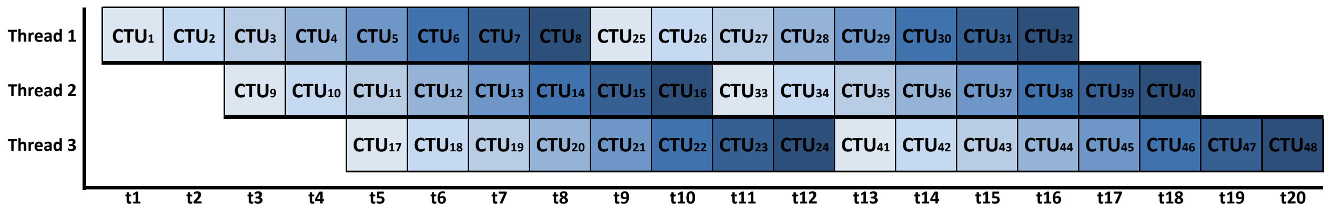

Thread 1
CTU1 CTU2 CTU3 CTU4 CTU5 CTU6 CTU7 CTU8 CTU25 CTU26 CTU27 CTU28 CTU29 CTU30 CTU31 CTU32
Thread 2
CTU9 CTU10 CTU11 CTU12 CTU13 CTU14 CTU15 CTU16 CTU33 CTU34 CTU35 CTU36 CTU37 CTU38 CTU39 CTU40
Thread 3
CTU17 CTU18 CTU19 CTU20 CTU21 CTU22 CTU23 CTU24 CTU41 CTU42 CTU43 CTU44 CTU45 CTU46 CTU47 CTU48
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19 t20

Figure 31: WPP scheduling when number of threads is smaller than CTUCols/2.

As shown in Figure 31, this frame takes 20 times slots to be processed. In the context of WPP, time slots will be called *synchronization barriers*. Increasing the number of encoding threads to 4 (Figure 32.a), the number of required synchronization barriers is reduced to 18, resulting in performance gain. However, for a number of encoding threads ranging from 4 to 6, (the number of CTU rows in the frame), the CTU scheduling does not shrink its timeline any more than seen in Figure 32.a, still only finishing frame processing after 18 synchronization barriers.

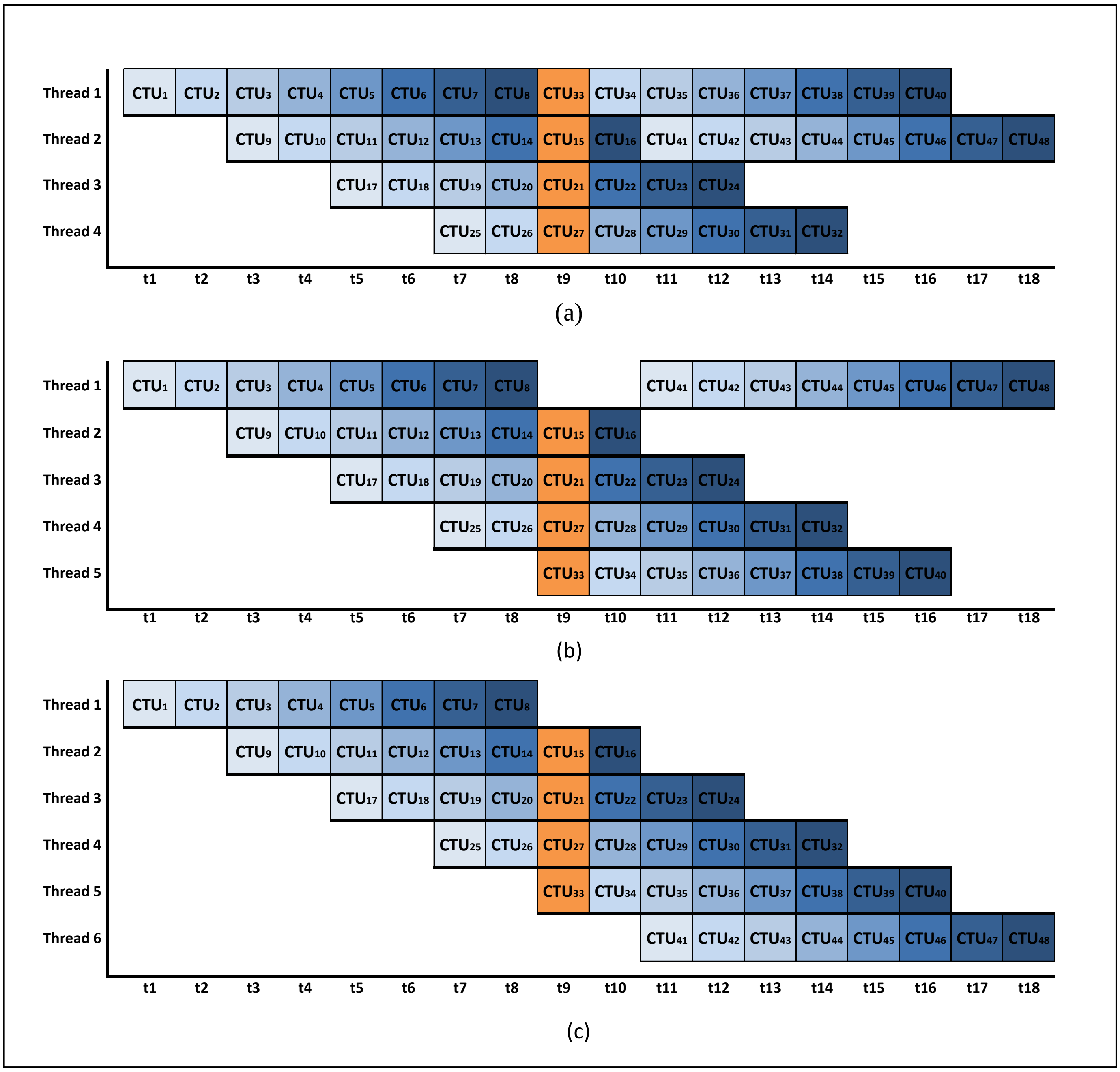


Figure 32: WPP row scheduling for (a) 4; (b) 5; and (c) 6 threads. The number of threads is greater or equal to *CTUCols/2*.

From this example, we can infer that the total time required to encode a video sequence using WPP can be expressed mathematically by the Eq. 3.

$$T_{WPP} = \sum_{i=1}^{n} t_{WPP_i} \qquad (3)$$

In Eq.3, n is the number of frames in the sequence, and $t_{WPP_i}$the time needed to compute the frame i of the sequence, which can in turn be expressed by Eq. 4.

$$t_{WPP} = \sum_{i=1}^{m} t_i \qquad (4)$$

In Eq. 4, m is the number of synchronization barriers needed to encode one frame of the video sequence.

For the configuration shown in Figure 31, for example (*8x6* CTUs and 3 threads), n = 20 synchronization barriers are needed. In turn, all three configurations shown in Figure 32 need n = 18.

Therefore, the total WPP computing time for a single frame in this case is shown in Eq. 5.

$$t_{WPP} = \sum_{i=1}^{n} t_i = \sum_{i=1}^{18} t_i \qquad (5)$$

where $t_1 = \max(t_{CTU_1}) = t_{CTU_1}$

(t_{CTU_i} being the time required for the computation of CTU_i) and

$$t_2 = \max(t_{CTU_2}) = t_{CTU_2}$$

$$t_3 = \max(t_{CTU_3},\ t_{CTU_9})$$

$$t_4 = \max(t_{CTU_4},\ t_{CTU_{10}})$$

$$\vdots$$

$$t_9 = \max(t_{CTU_{33}},\ t_{CTU_{15}},\ t_{CTU_{21}},\ t_{CTU_{27}})$$

$$\vdots$$

$$t_{48} = \max(t_{CTU_{48}}) = t_{CTU_{48}}$$

As the highlighted columns in Figure 32 show, the value of t_9 is the same for the three different configurations presented.

$$a)\ t_9 = \max(t_{CTU_{33}},\ t_{CTU_{15}},\ t_{CTU_{21}},\ t_{CTU_{27}})$$

$$b)\ t_9 = \max(\ t_{CTU_{15}},\ t_{CTU_{21}},\ t_{CTU_{27}}, t_{CTU_{33}})$$

$$c)\ t_9 = \max(\ t_{CTU_{15}},\ t_{CTU_{21}},\ t_{CTU_{27}}, t_{CTU_{33}})$$

This is the case for every t_i in the frame and therefore the value of T_{WPP} does not change anymore after a number of threads is higher than 4. In fact, the same happens for *every* video sequence when using a number of threads greater than or equal to the ceiling of half the number of CTU columns present in its frames. Thus, the limitation for the parallelization using WPP can be expressed as shown in Eq. 6.

$$T_{WPP_m} = T_{WPP_{m+1}} \text{ if } m \geq \left\lceil \frac{CTUCols}{2} \right\rceil \qquad (6)$$

In Eq. 6, m is the number of encoding threads. This work considers CTUCols to be always greater than or equal to CTU Rows.

Figure 33 shows the number of parallel CTUs (independent tasks, assuming the number of processing units is equal or greater than *CTUCols/2*) of different video classes being encoded at a time for threads with wavefront dependencies. This type of dependencies generates a variable number of parallel tasks with a ramp pattern. The pipeline can be filled until the number of parallel tasks reaches *CTUCols/2*, which happens in $(\lceil CTUCols/2 \rceil - 1) * 2$ time slots. It remains full (and thus providing maximum speedup) for a period of time equal to $(CTURows - \lceil CTUCols/2 \rceil) * 2$ time slots, and then empties out for another $(\lceil CTUCols/2 \rceil - 1) * 2$ time slots. Therefore,

$$T_{WPP} = (\left\lceil \frac{CTUCols}{2} \right\rceil - 1) * 4 + (CTURows - \left\lceil \frac{CTUCols}{2} \right\rceil) * 2$$

$$T_{WPP} = 2 * CTURows + 2 * \left\lceil \frac{CTUCols}{2} \right\rceil - 4$$

where *T_{WPP}* is measured in time slots or synchronization barriers.

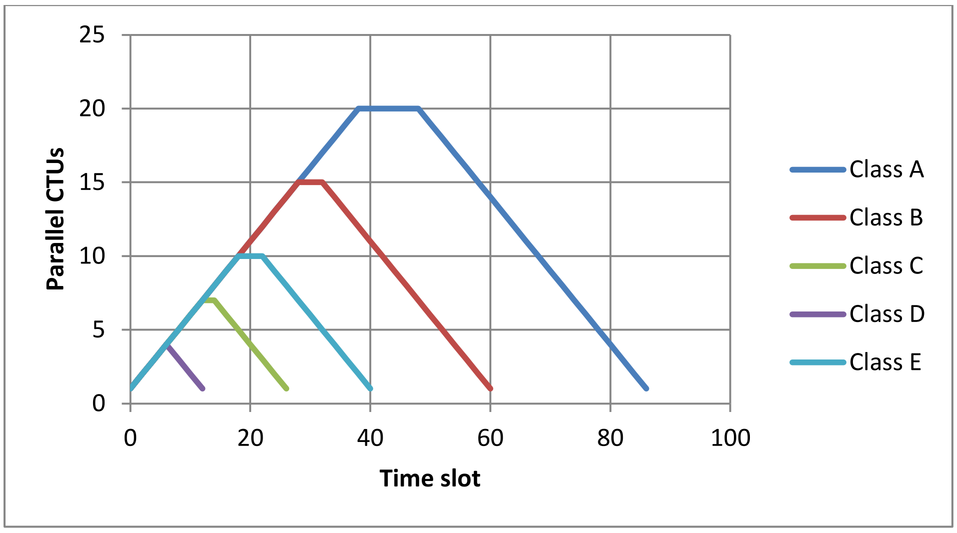


Figure 33: maximum parallelism when using WPP

5.3.3 Speedups for 1, 4, 9 and 16 processing units

This experiment considers the speedups obtained for each sequence when the number of working processing units is equal to 1, 4, 9 or 16. Beyond providing information on an intra-class level, it also allows a direct comparison with the results from subsection 5.2.4.

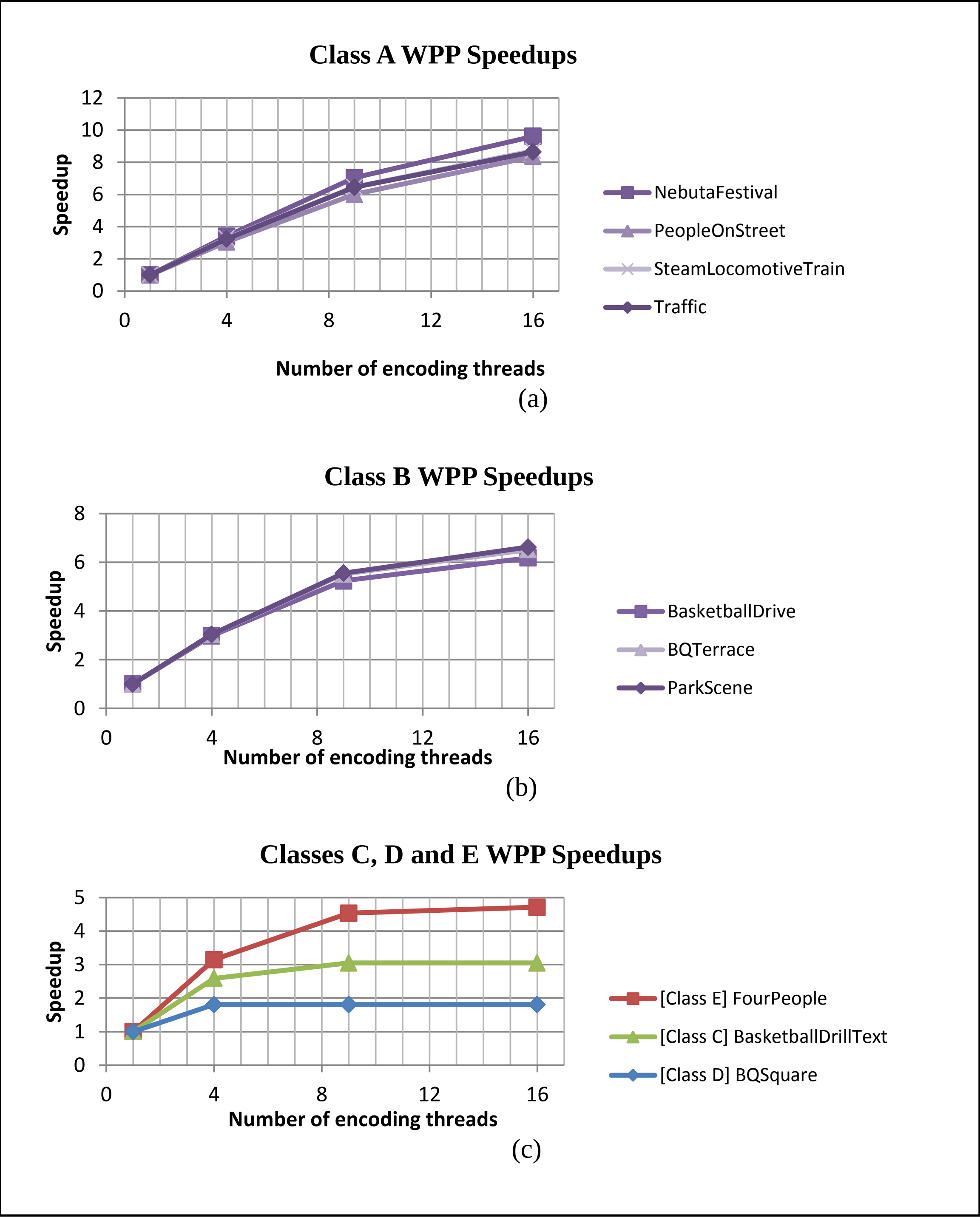


Figure 34: WPP speedups for (a) Class A, (b) B, (c) C, D and E video sequences

As can be seen in Figure 34, higher speedups are obtained for higher resolutions: while Class A sequences present an average speedup of about 9 times for 16 threads, the same number of threads offers average gains of 6.5, 3 and 2 times for Classes B, C and D, respectively.

5.3.4 Wavefront parallelization inefficiencies

In the context of WPP, parallelization inefficiency is measured by the time during which at least one thread is idle due to data dependencies. Figure 35 presents stacked area charts which help identifying parallelization inefficiency in the total encoding time.

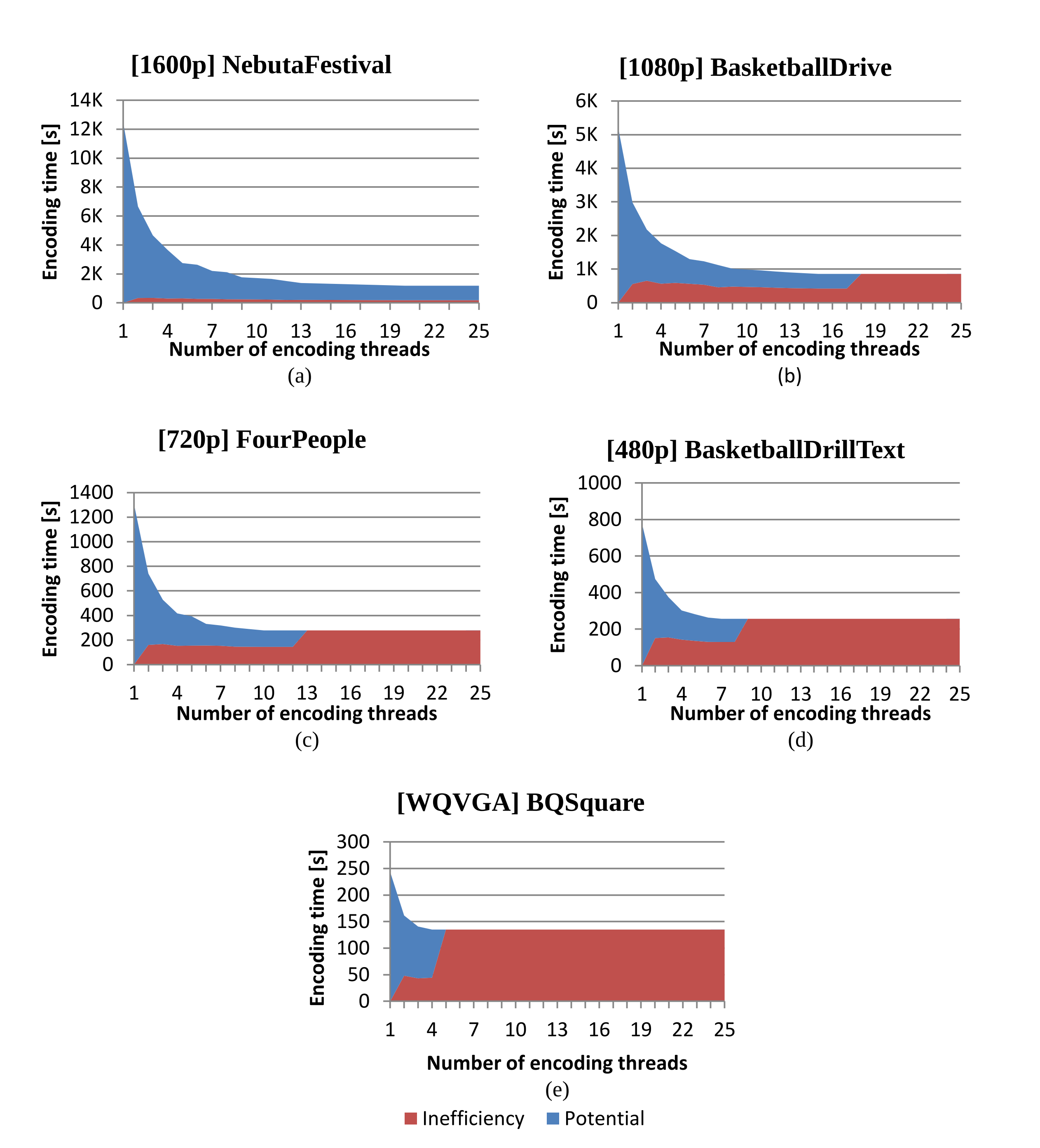


Figure 35: Parallelism potential and parallelization inefficiency in WPP.

The plots in Figure 35 show that the reduction in encoding time using WPP is at first exponential, then linear, and finally non-existent. To explore the exponential reduction, about 30-40% of the total number of CTU rows in number of threads can be used, producing enough acceleration for most applications. For a mobile phone, for example, with a few CPUs, WPP could be more interesting because the BD-rate increase is lower (increased code efficiency) and the gain is similar to that of tiles.

Parallelization inefficiency is considerably lower on higher resolutions because the pipeline remains full for a longer time, as seen in subsection 5.3.2. It also varies very little as the number of processing units increase, increasing only when there are more units than CTU rows.

5.4 Comparison between Tiles and WPP

Tiles can have many configurations and the amount of parallelism is not fixed, except for a lower limit. The parallelism in WPP, however, is fixed to one substream per row and, therefore, scales naturally with resolution. For this reason, two different forms of comparison are presented.

The first one (subsection 5.4.1) compares WPP to the 2x2, 3x3 and 4x4 tiling patterns for an increasing number of encoding threads. The second one (subsection 5.4.2) investigates the results of using, against WPP, a number of tiles that matches the number of encoding threads.

5.4.1 Incremental speedup

To enhance readability, only one video sequence of each class was selected for comparisons, namely *NebutaFestival* (Class A), *BasketballDrive* (Class B), *BasketballDrillText* (Class C), *BQSquare* (Class D) and *FourPeople* (Class E).

The values of the horizontal axis ranges from 1 to the maximum number of partitions in the frame following the test scenario restrictions: 16 (4x4 tiles) or the number of WPP substreams, when the number of the CTU rows is higher than 16. The latter happens only for video sequences belonging to Class A and B, with 25 and 17 CTU rows per picture respectively.

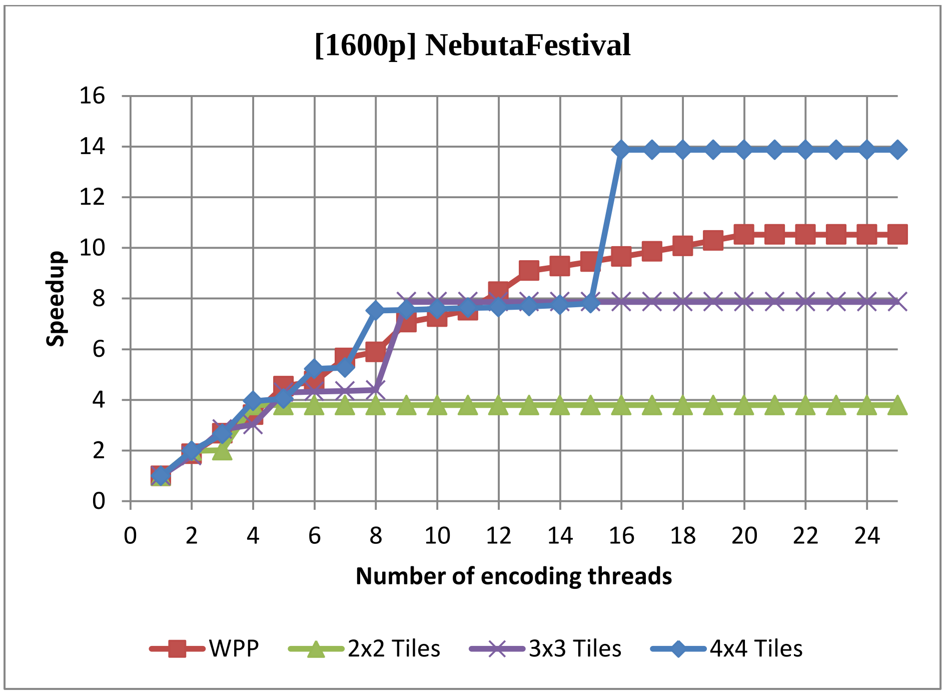


Figure 36: WPP x Tiles comparison, Class A video sequence

Figure 36 shows that WPP can outperform 4x4 tiles for a certain set of number of threads available. It happens when increasing the number of processing units incurs a greater boost in performance for WPP than for tiles. As seen in previous sections, the addition of a new processing unit means, for WPP, that the number of rows being processed in parallel increases by one. Tile-wise, however, if the number of tiles is greater than the number of processing units, increasing the number of processing units does *not* always produce a reasonable boost in performance, since the scheduling algorithm does not always reduce the number of encoding time slots (please refer to subsection 5.2.3). For 12 encoding threads, for example, which is both less than 20 (wavefront parallelization limit for Class A sequences, refer to subsection 5.3.2) and between 8 and 16 (two points of significant gain in performance for a 4x4 tiling pattern), WPP performs better than *all* tiling patterns.

For a number of threads equal to 16 (after which tiles do not present gain in performance), WPP ranks second place with a speedup about 30% lower in comparison to 4x4 tiles.

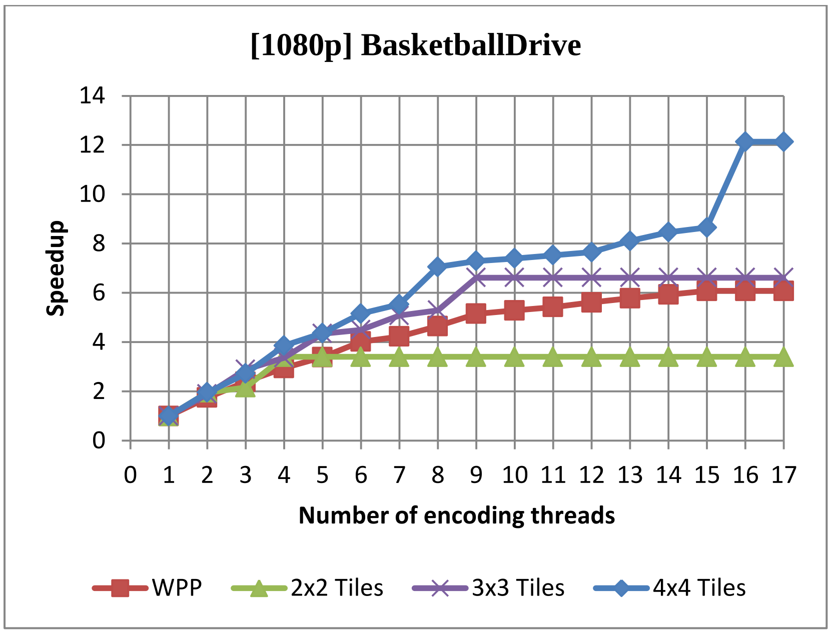


Figure 37: WPP x Tiles, Class B video sequence

However, it can be seen on Figure 37,Figure 38, Figure 39Figure 40 that WPP performs increasingly worse for smaller resolutions, with speedups around 45% (Class B), 62% (Class E), and 72% (Class C) lower in comparison to 4x4 tiles.

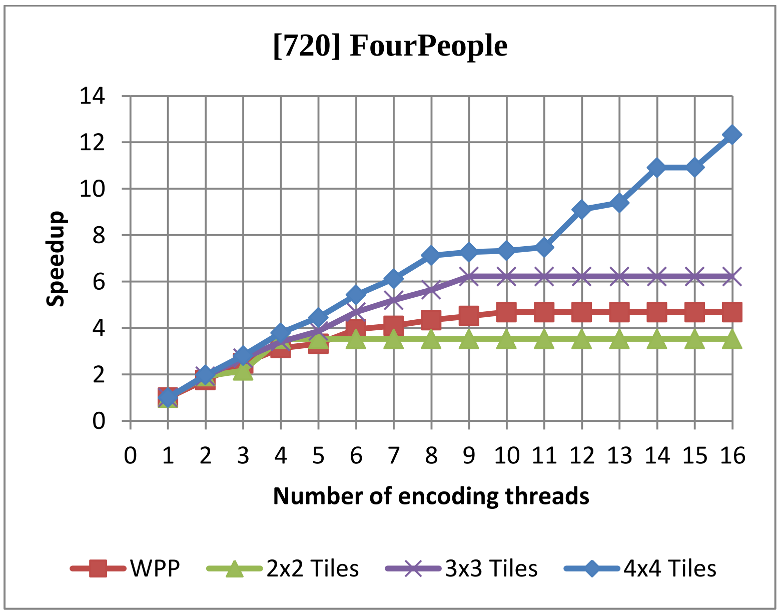


Figure 38: WPP x Tiles, Class E video sequence

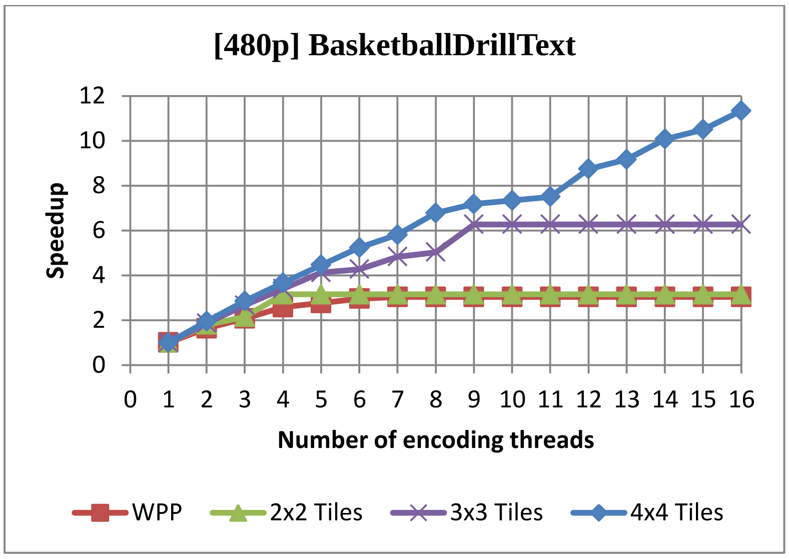


Figure 39: WPP x Tiles, Class C video sequence

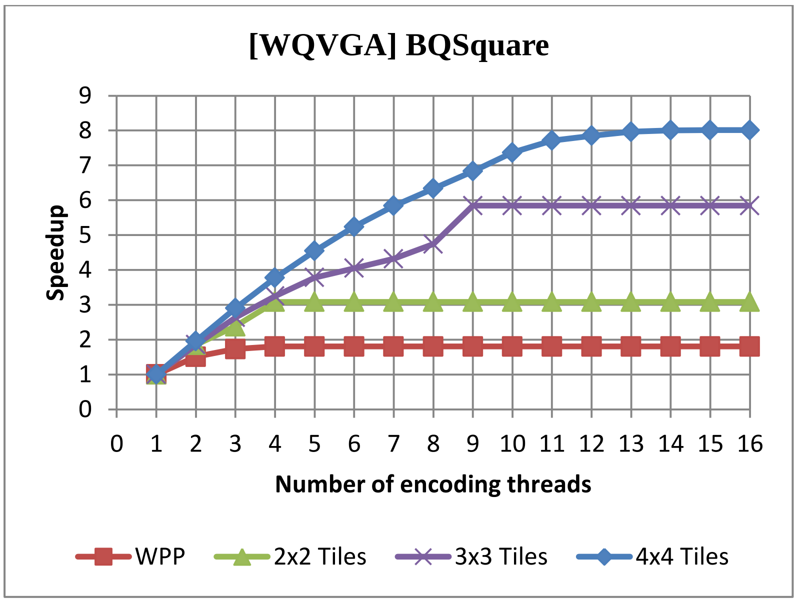


Figure 40: WPP x Tiles, Class D video sequence

For D video sequences WPP performs even worse than 2x2 tiles. Although both strategies work on the same number of partitions, the rampant inefficiencies of WPP make it about 28% worse in performance for a number of processing units equal or greater to 4. For a number of threads equal to 16 (after which tiles do not present gain in performance), WPP presents a speedup about 78% inferior in comparison to 4x4 tiles, since the small number of CTU rows significantly limits parallelization.

5.4.2 Speedups for 1, 4, 9 and 16 encoding threads

Figure 41 shows a comparison of the performance of tiles and WPP for different classes. In this experiment, the speedup values are an average for each class. While WPP maintains the same number of partitions throughout the experiment in each class (since its partitioning scheme does not allow changes), the tiling pattern is modified to adapt to the number of encoding threads used. For a Class A sequence, for example, the number of wavefront substreams is 25 for 1, 4, 9, or 16 encoding threads, while the tiling pattern changes from 1x1 (baseline), to *2x2*, *3x3* or *4x4*. This adaptation is adopted because tiles present maximum performance when the number of threads is equal or greater than the number of partitions.

Following the same methodology, Figure 42 shows a parallelization inefficiency comparison between the two parallel tools. Parallelization inefficiency is presented as a percentage of the total encoding time during which at least one processing unit is idle.

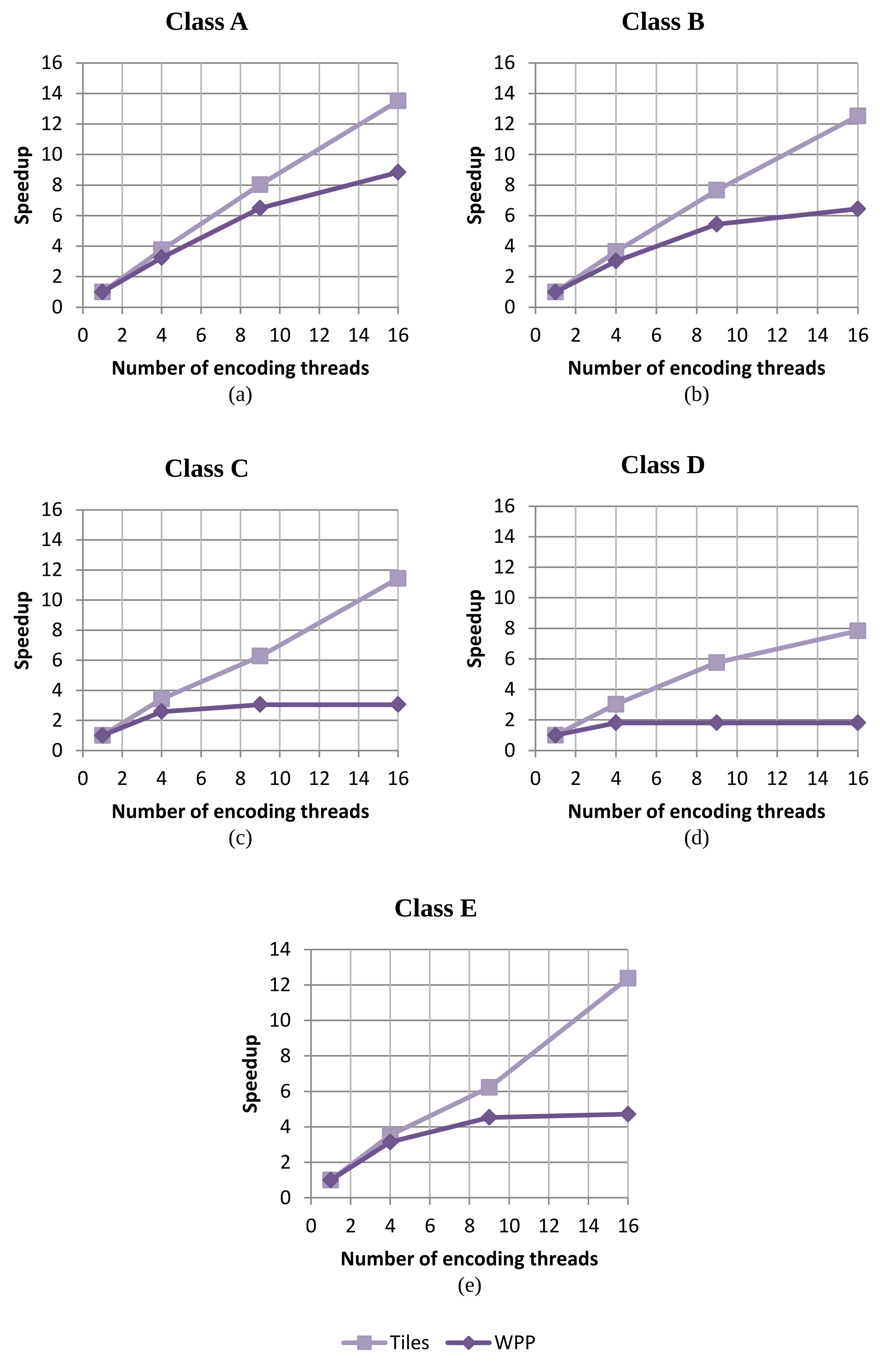


Figure 41: WPP x Tiles for 4, 9 and 16 encoding threads for Classes (a) A (b) B (c) C (d) D and (e) E

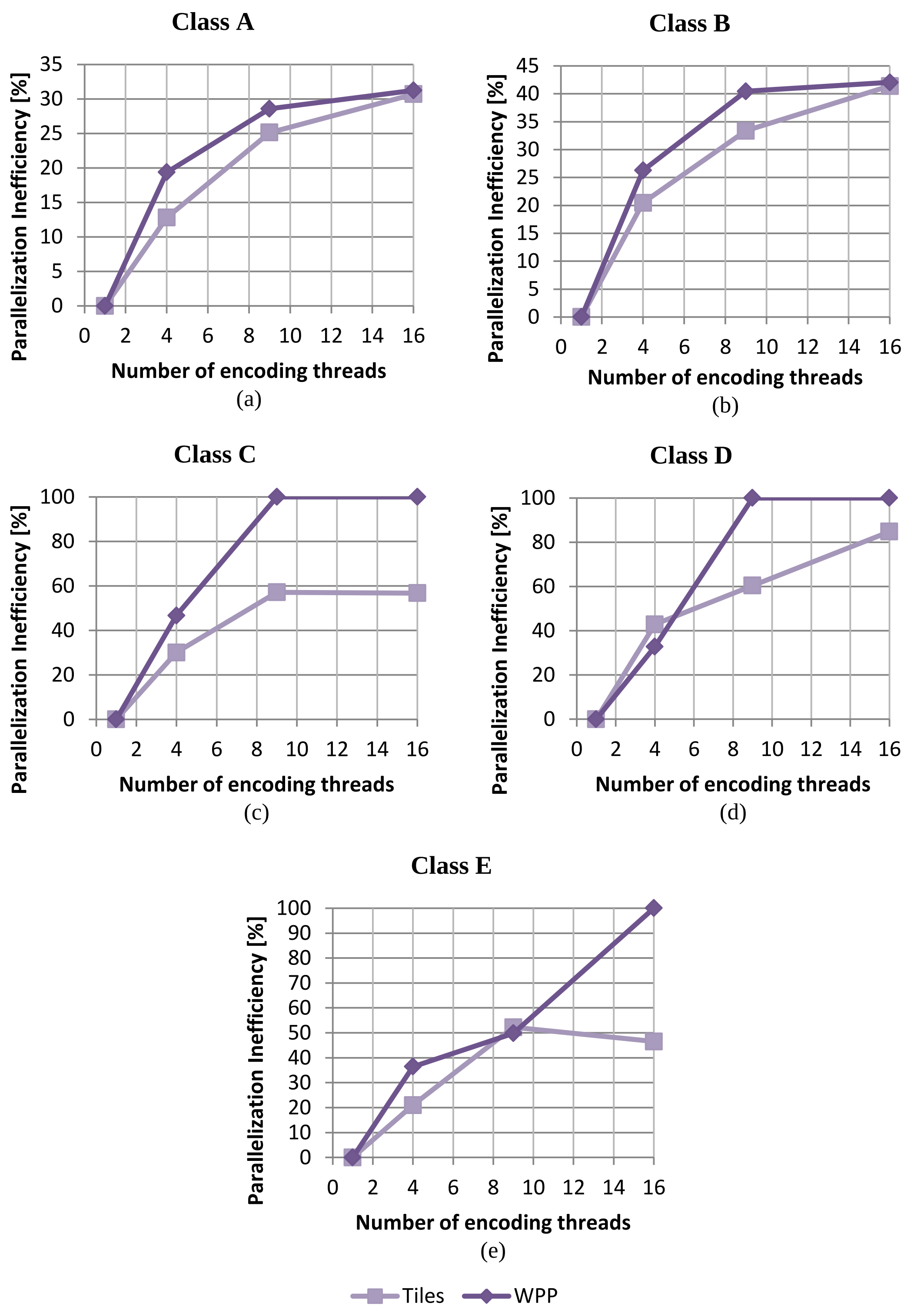


Figure 42: Parallelization inefficiency in WPP and tiles for 4, 9 and 16 encoding threads for Classes (a) A (b) B (c) C (d) D and (e) E

Tiles scale well (almost linearly) for all classes and offer higher speedups than WPP. WPP scales worse because of the ramping inefficiencies and parallezation limits. However, WPP offers better coding efficiency than tiles. For higher resolutions (see Figure 41.a and b) the differences in speedup are smaller because of the higher number of wavefront substreams.

As expected, because of the ramping stalls, WPP present higher parallelization inefficiency than tiles (see Figure 42). As seen in Figure 42.c), d) and e), the inefficiency goes up to 100% once WPP reaches its saturation point.

5.4.3 Considerations

- Multiple WPP encoding threads do not require additional line buffers as the single line buffer set can be shared by any number of WPP encoding threads. Because of the wavefront order, WPP encoding threads will always use a different part of the line buffer. For each Tiles encoding thread, however, separate line buffers need to be allocated.
- Tiles can have many configurations - including complex combinations with slices - and the amount of parallelism is not fixed, except for a lower limit. The parallelism in WPP is fixed to one substream per row and, therefore, scales naturally with resolution, which ensures that codec implementations have sufficient parallelism.
- While WPP presents good results for many use cases, it struggles to get full utilization of threads when a high number of encoding threads is used - e.g. 15 for 1080p (Class B) sequences where the last row cannot start being processed until the first is finished.
- In a computer architecture sense, tiles seem better when compared to WPP. This is because:
    - WPP requires synchronization much more often than tiles.
    - Each tile or wavefront can be processed on a separate core. Tiling patterns can be adapted in a way that the distribution of workload is even among cores; or the most complex tile can be dispatched to the most powerful processor.

6 CONCLUSION

Targeting high-level parallel processing capabilities, the HEVC coding standard includes two different parallelization strategies in its toolset: *WPP*, and *Tiles*. Both techniques share in common the idea of creating picture partitions that can be processed in parallel by multiple processing units in a parallel system, but present advantages and disadvantages for different applications and settings.

This work presented an evaluation of the performance of these normative tools for parallel processing proposed by the HEVC coding standard in terms of speedup, coding efficiency, scalability, and parallelization inefficiency.

The results were obtained by means of the simulation of a parallel environment using as input the required encoding times of all CTUs in parallel-enabled video sequences encoded sequentially. The HEVC HM Reference Software provided a baseline implementation that was modified in order to provide the data required for the simulation, as well as to produce HEVC bitstreams containing tile or wavefront partitions.

Being analyzed individually, WPP and tiles provided interesting information regarding their parallelization limits. WPP only presented gains when using a number of threads greater than or equal to the ceiling of half the number of CTU columns present in its frames, which was later revealed to be a limitation directly related to data dependencies and row scheduling. When using tiles for a lower number of processing units then partitions, tile scheduling also affects results – in some cases even producing higher speedups for simpler tiling patterns.

The comparison of WPP to tiles showed that the tiles approach achieves higher average performance than WPP. Tiles also scale better but present, as a downside, meaningful losses in coding efficiency since all partitions are self-contained. For high-definition content and a relatively low number of processing units (30-40% of the number of partitions) – the case for most low-power devices nowadays -, the difference in performance is not so significant, in which case WPP would be preferable since it leads to better compression rates and needs less memory bandwidth.

As future work, the classes implemented for the simulation of the parallel environment can be adapted to estimate parallelism potential and parallelization inefficiency when using wavefronts inside tile partitions. This way, it is possible measure the gains of both levels of parallelism: coarse grain, represented by the tile partition itself; and fine grain, represented by the WPP rows contained inside each tile partition.

7 REFERENCES

Agostini, Luciano Volcan. 2007. **“Desenvolvimeto de Arquiteturas de Alto Desempenho Dedicadas À Compressão de Vídeo Segundo O Padrão H.264/AVC”**. Porto Alegre: Universidade Federal do Rio Grande do Sul.

BJONTEGAARD, G. 2001. **“Calculation of Average PSNR Differences between RD-Curves.”** *VCEG*.

Blumenberg, Cauane. 2014. **“Adaptive Tiling Algorithm Based on Highly Correlated Picture Regions for the HEVC Standard”**. Porto Alegre, Brazil: Universidade Federal do Rio Grande do Sul.

Blumenberg, Cauane, Daniel Palomino, Sergio Bampi, and Bruno Zatt. 2013. **“Adaptive Content-Based Tile Partitioning Algorithm for the HEVC Standard.”** In *Picture Coding Symposium (PCS), 2013*, 185–88. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6737714.

Bordes, Philippe, Gordon Clare, Félix Henry, Mickaël Raulet, and Jérôme Viéron. 2012. **“An Overview of the Emerging HEVC Standard.”** In *International Symposium on Signal, Image, Video and Communications, ISIVC],(July 2012)*. http://www.live-production.tv/system/files/The_HEVC_Standard.pdf.

Borkar, S, and A A Chien. 2011. **“The Future of Microprocessors.”** *Commun. ACM*, May.

Bossen, Frank. 2012. **“Common Test Conditions and Software Reference Configurations”**. ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC).

Bross, Benjamin. 2012. **“Relax, It’s Only HEVC”**. Geneva (CH), November 27.

Chi, Chi Ching, Mauricio Alvarez-Mesa, Ben Juurlink, Gordon Clare, Félix Henry, Stéphane Pateux, and Thomas Schierl. 2012. **“Parallel Scalability and Efficiency of HEVC Parallelization Approaches.”** *IEEE Transactions on Circuits and Systems for Video Technology* 22 (12): 1827–38. doi:10.1109/TCSVT.2012.2223056.

Correa, Guilherme, Pedro Assuncao, Luciano Agostini, and Luis A. da Silva Cruz. 2012. **“Performance and Computational Complexity Assessment of High-Efficiency Video Encoders.”** *IEEE Transactions on Circuits and Systems for Video Technology* 22 (12): 1899–1909. doi:10.1109/TCSVT.2012.2223411.

Fu, Chih-Ming, Elena Alshina, Alexander Alshin, Yu-Wen Huang, Ching-Yeh Chen, Chia-Yang Tsai, Chih-Wei Hsu, Shaw-Min Lei, Jeong-Hoon Park, and Woo-Jin Han. 2012. **“Sample Adaptive Offset in the HEVC Standard.”** *IEEE Transactions on Circuits and Systems for Video Technology* 22 (12): 1755–64. doi:10.1109/TCSVT.2012.2221529.

Fuldseth, Arild, Michael Horowitz, Shilin Xu, Kiran Misra, Andrew Segall, and Minhua Zhou. 2012. **“Tiles for Managing Computational Complexity of Video Encoding and Decoding.”** In *Picture Coding Symposium (PCS), 2012*, 389–92. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6213371.

Ghanbari, M, and Institution of Engineering and Technology. 2011. ***Standard Codecs Image Compression to Advanced Video Coding***. London: Institution of Engineering and Technology. http://dx.doi.org/10.1049/PBTE054E.

HEVC HM Reference Software (version 14.0). https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-14.0/.

ITRS. 2011. **"International Technology Roadmap for Semiconductors 2011 Update System Drivers."**

ITU-T Rec. 2003. **"H.264: Advanced Video Coding for Generic Audiovisual Services."** http://129.97.134.72:8888/T-REC-H.264-200901-I!Cor1!PDF-E.pdf.

Jo, Hyunho, Donggyu Sim, and Byeungwoo Jeon. 2013. **"Hybrid Parallelization for HEVC Decoder."** In *Image and Signal Processing (CISP), 2013 6th International Congress on*, 1:170–75. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6743980.

Kim, Il-Koo, Ken McCann, Kazuo Sugimoto, Benjamin Bross, and Han Woo-Jin. 2013. **"High Efficiency Video Coding (HEVC) Test Model 10 (HM10) Encoder Description."**

McCann, K., Benjamin Bross, Il-Koo Kim, K. Sugimoto, and Woo-Jin Han. 2012. **"HM6: High Efficiency Video Coding (HEVC) Test Model 6 Encoder Description"**. ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC).

Misra, Kiran, Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, and Minhua Zhou. 2013. **"An Overview of Tiles in HEVC."** *IEEE Journal of Selected Topics in Signal Processing* 7 (6): 969–77. doi:10.1109/JSTSP.2013.2271451.

Monteiro, Eduarda. 2012. **"Implementação E Análise de Algoritmos Para Estimação de Movimento Em Processadores Paralelos Tipo GPU (Graphics Processing Units)"**. Porto Alegre: Universidade Federal do Rio Grande do Sul.

Pourazad, Mahsa, Colin Doutre, Maryam Azimi, and Panos Nasiopoulos. 2012. **"HEVC: The New Gold Standard for Video Compression: How Does HEVC Compare with H.264/AVC?"** *IEEE Consumer Electronics Magazine* 1 (3): 36–46. doi:10.1109/MCE.2012.2192754.

Puri, Atul, Xuemin Chen, and Ajay Luthra. 2004. **"Video Coding Using the H.264/MPEG-4 AVC Compression Standard."** *Signal Processing: Image Communication* 19 (9): 793–849. doi:10.1016/j.image.2004.06.003.

Richardson, Ian E. 2011. *The H.264 Advanced Video Compression Standard*. 2nd ed. John Wiley & Sons.

Sullivan, Gary J., and Jens-Rainer Ohm. 2010. **"Recent Developments in Standardization of High Efficiency Video Coding (HEVC)."** In *SPIE Optical Engineering+ Applications*, 77980V–77980V. International Society for Optics and Photonics. http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=754649.

Sullivan, Gary J., Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. **"Overview of the High Efficiency Video Coding (HEVC) Standard."** *IEEE Transactions on Circuits and Systems for Video Technology* 22 (12): 1649–68. doi:10.1109/TCSVT.2012.2221191.

Szu, Harold. 2002. **"Video Compression by Means of Singularity Maps of Human Vision System."** In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, 3:2120–25. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1007469.

Vanne, Jarno, Marko Viitanen, Timo D. Hamalainen, and Antti Hallapuro. 2012. **"Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video**

Codecs.” *IEEE Transactions on Circuits and Systems for Video Technology* 22 (12): 1885–98. doi:10.1109/TCSVT.2012.2223013.

Zhao, Liang, Li Zhang, Siwei Ma, and Debin Zhao. 2011. **“Fast Mode Decision Algorithm for Intra Prediction in HEVC.”** In *Visual Communications and Image Processing (VCIP), 2011 IEEE*, 1–4. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6115979.

APPENDIX A: TG1 PAPER

Overview of Parallel Processing Tools in the new High Efficiency Video Coding Standard (HEVC)

Filipe Posteral Silva, Cláudio Machado Diniz, Sergio Bampi

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
{fpsilva,cmdiniz,bampi}@inf.ufrgs.br

***Abstract**. For ultra-high definition videos with resolutions of 4Kx2K and 8Kx 4K, the requirements of low-latency or real-time processing can exceed the capacity of current single core codecs. The new video compression standard High Efficiency Video Coding (HEVC) incorporates some tools to render it more parallel-friendly, differently from its predecessor, H.264/AVC, which does not contain tools for parallel processing from its conception. The aim of this paper is to provide an overview of the new standard, with a special emphasis on the two main tools proposed for parallel processing: Tiles and Wavefront Parallel Processing.*

1 Introduction

Stimulated by the use of applications such as broadcast over cable and satellite, conversational applications over wireless and mobile networks, multimedia streaming services and etc., the growth in production and diffusion of digital video has caused video coding standards to evolve in order to adapt themselves to the ever increasing resolutions and frame rates of digital videos.

The newest video coding standard, High Efficiency Video Coding (HEVC), developed by the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T and ISO/IEC Moving Pictures Experts Group, was especially designed to support ultra-high definition content with resolutions of 4Kx2K and 8Kx 4K, for example, with a bit rate reduction of about 50% compared to its predecessor, H.264/AVC.

Because of the huge computational effort needed to encode and decode such ultra-high resolutions, the standard focuses on the use of parallel processing architectures for computational effort reduction. It is very unlikely that single-core processor performance will increase to a point where it can decode UHD HEVC video in real time, which means that parallelism is no longer an option but a necessity [1]. To address this issue, HEVC includes Wavefront Parallel Processing (WPP) for the parallel entropy coding and also tile and slice segments for picture partitioning for data level parallelism (DLP).

This paper provides an overview of the standard, putting more emphasis on the modules and concepts which will be explored by the parallel tools mentioned above. The paper is organized as follows. In Section 2, the principles of video compression are explained. In Section 3, an overview of the HEVC standard is presented. The proposed parallel methods and tools are presented in Section 4. Finally, we summarize and conclude the paper in Section 5.

2 Video compression

The evolution of technology has made it possible to capture and create video sequences with resolutions that produce huge volume of data. This huge volume of data is becoming large to the point of rendering processing, storage and transfer impossible in current technology without compression.

A video sequence is a set of frames (pictures) displayed one after another. Each frame is divided into blocks, constituted by pixels. The pixel is the smallest element of an image, and it is usually represented by samples in three different color components, i.e. red (R), green (G) and blue (B), when considering a RGB color space. For video compression purposes, video is often represented in the YCbCr color space, in which Y is luminance component; Cb and Cr are the blue and red chrominance components, respectively.

A raw video sequence captured at 1080p resolution (1920 x 1080 pixels), 24 bits per pixel (8-bits for each component Y, Cb, Cr) and 25 pictures per second (frame rate) would require a data transfer rate of 156 MB/s (25 x 3 x 1920 x 1080). A storage unit of 1094 GB (equivalent to 48 single layer Blu-ray discs) would be necessary to store 2 hours of video.

Therefore, there is a clear imbalance between the current data processing capacity, storage and transfer, which makes the representation of uncompressed videos not effective at all. However, with the computing power available today, it is possible to process videos in a way that they can be transmitted and stored by using less bandwidth or disk space. Compression and decompression techniques allow a significant reduction in file size with minimal or no effect on visual quality.

2.1 Fundamentals of video compression

Videos contain a very expressive statistical redundancy, in both time and spatial domains. Video compression technologies are designed to reduce and eliminate redundant data to allow the storage and efficient transmission of a digital video file. The main principle lies in the detection of redundant information that can be represented more efficiently.

Lossy compression techniques consist on the elimination of some of the video data available in the input during the compression process at the price of some loss in video quality. Lossy compression techniques take advantage of the restrictions of the human visual system model to limit the frequencies being represented in an image, quantizing high frequency detail components without a perceptive loss of quality.

Inside each frame of video the *spatial redundancy,* i.e. repeated shapes or patterns that can be described by few information, is explored to reduce the amount of information to represent the same data. Among the different frames of video there is the *temporal redundancy,* i.e. elements that are very similar in two different frames.

The main idea is to encode only what changes from one frame to another (and not the whole frame) using prediction techniques. Thus, we assume that the value of a particular pixel of the frame can be predicted from neighboring pixels from the same frame (using intra-frame coding) or other frame pixels (using inter-frame coding).

Spatial redundancy expresses the fact that we often find in each image of video sequence one or more regions with similar visual characteristics. We assume that we can deduce the values of the pixels of a block from the pixel values of the neighboring blocks (the concept of block partitioning will be explored later). This is called spatial prediction or intra prediction.

Temporal redundancy expresses the fact that there's usually no abrupt change in the content displayed between two or more consecutive frames. This redundancy is exploited using motion *estimation* and *compensation* techniques.

Motion estimation (ME): the motion is estimated for a current frame (to be encoded) related to a reference frame (previously encoded). The motion vector represents the movement of a block from the reference frame to the current frame.

Motion Compensation (MC): the motion compensation is applied to the reference frame to obtain the predicted frame.

However, this estimation is not perfect. In addition to motion information, the difference between the predicted and current frame is also needed for reconstruction. This difference is called residual frame (this term is also used in intra prediction). Figure 1 shows the residual frame obtained by the subtraction of two successive frames when motion compensation techniques are not used.

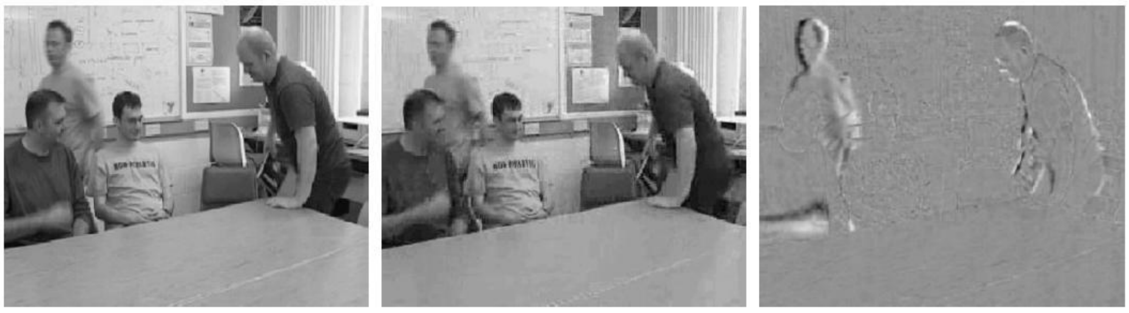


Figure 43: Frame *n* to encode, Frame *n-1* (reconstructed after encoding), and associated residual frame, respectively. Source: [7]

The use of motion compensation techniques changes the result drastically. The energy of the residue is much smaller which means the amount of data needed for representation is equally smaller.

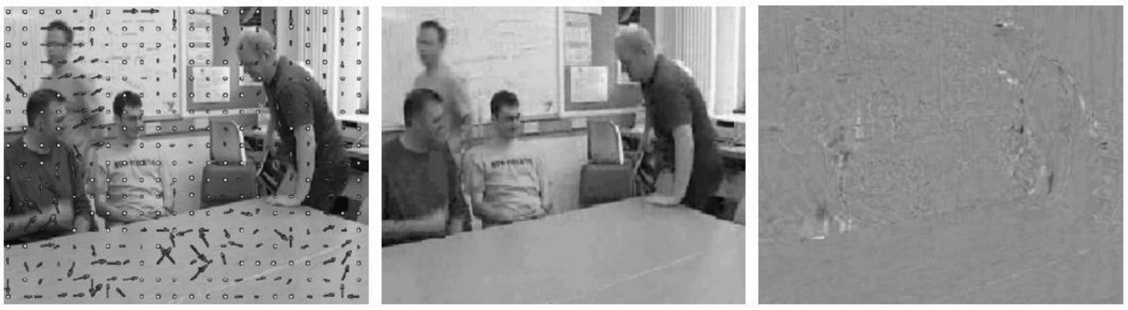


Figure 44: Motion vectors over the frame, frame *n* compensated and associated residual frame, respectively. Source: [7]

The coded representation of a video sequence (the encoded bitstream) contains both image and motion information.

2.2 Structure of a hybrid encoder

The transformation of a raw (uncompressed) video sequence into an encoded bitstream (a coded video sequence) is called *encoding* and is performed by an *encoder*. The inverse transformation is called *decoding*. The *decoder* is needed each time the video has to be displayed. A device capable of performing both functions is called a *codec* (coder/decoder).

A hybrid encoder consists of three main functional units:

- a prediction model;
- a transformation/quantization model;
- an entropy encoder.

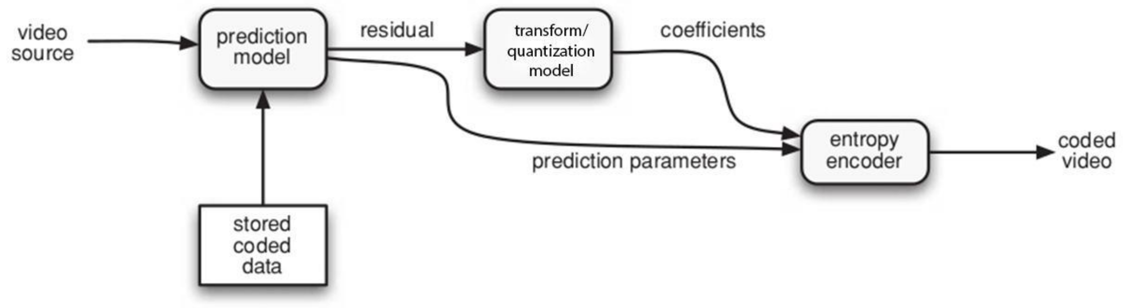


Figure 45: Hybrid video encoder. Source: [7]

The *prediction model* tries to reduce redundancy by exploiting the similarities between neighboring frames temporally or spatially, typically by constructing a prediction of the current frame. After intra or inter prediction, the residue is inferred and encoded.

The *transformation/quantization model* is based on the application of a transformation, a quantization and an arrangement of the coefficients. The purpose of the transform is to convert the data to another domain, where its representation will be more efficient for the quantization. Then, the quantization is applied accordingly to reduce the volume of data of irrelevant information. The coefficients are reordered to be delivered for entropy encoder.

The *entropy encoder* produces a compressed bitstream exploring the statistical redundancy, i.e. the redundancy of data representation. Tipically is based on variable length codes or binary arithmetic encoding [7]. This type of encoding does not result in losses and further reduces the amount of representation data. A compressed sequence is composed of header information, coded prediction parameters and coded residual coefficients.

2.3 Picture types

Video frames are not all compressed in the same way. In the most recent compression standards, the video stream is previously divided into groups of pictures (GoPs) composed of pictures of different types:

I-frames (Intra-coded picture) are pictures that only intra prediction is applied. It does not require other frames for encoding and decoding.

P-frames (Predicted picture) are pictures that inter prediction can be applied (but also intra prediction). Inter prediction uses data from previous frames for encoding and decoding. These can be more compressible than I-frames.

B-frames (Bi-predictive picture) can use both previous and forward frames for inter prediction to get the highest improvement in compression.

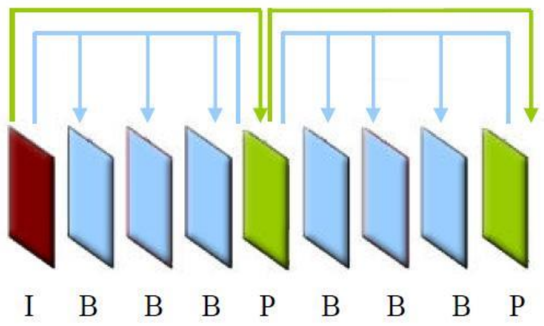


Figure 46: GoP example

Figure 4 shows an example of GoP and the dependencies between its pictures. It is important to note that it is possible to use a spatial prediction in P and B frames. In recent standards, like HEVC, frames are typically divided into multiple processing blocks with different coding types.

2.4 Scope of video compression standard

The vast amount of existing electronic devices capable of encoding or decoding video sequences needs the creation of rules that enable interoperability between them. A compression standard is primarily a formal set of rules that allows encoders and decoders to work together properly (Motion JPEG, MPEG-4, H.264/AVC and HEVC are some examples of video coding standards). Every decoder respecting the rules of a particular standard is capable of processing a video sequence produced by an encoder that respects the same standard. Encoders and decoders being separated from each other in time or space, the standard must ensure interoperability between bitstreams in transit from the first to the second.

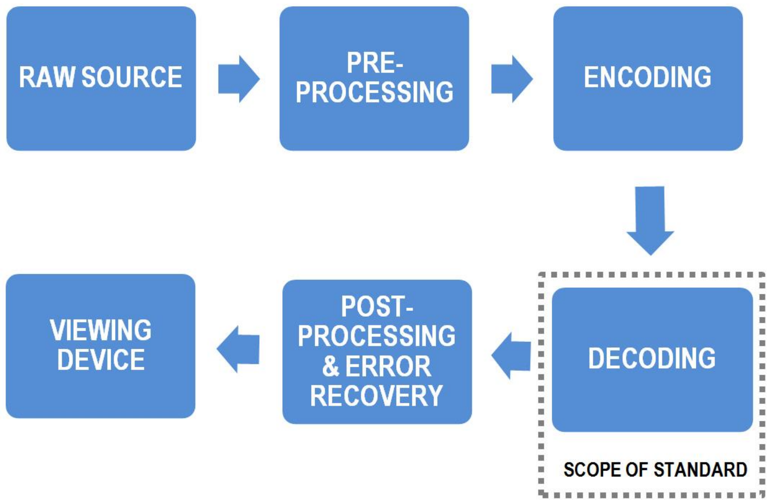


Figure 47: Scope of a video compression standard

The scope of a video compression standard is limited to decoding, as shown in Figure 5. It defines precisely the way according to which a coded video stream must be decoded. On the other hand, the implementation of encoding algorithms is up to developers as long as the produced stream can be decoded by a decoder in accordance with the standard. This allows, for example, the creation of encoders that can meet specific needs (e.g. no inter prediction).

3 The HEVC standard

The High-Efficiency Video Coding (HEVC) is the newest video compression standard proposed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving

Picture Expert Group (MPEG). The following subsections will provide an overview of the coding tools proposed in the standard.

3.1 Structure of an HEVC encoder

HEVC follows the same hybrid video encoder model that we show in section 2.2, including motion estimation/compensation followed by transforms, quantization, and entropy coding. A block diagram of HEVC encoder is shown in Figure 6. Each frame is divided into blocks of a parameterized size, and the block size is transmitted to the decoder. The first image of a GoP is coded using only intra prediction. Inter-frame prediction is typically used for most blocks in all other P and B pictures.

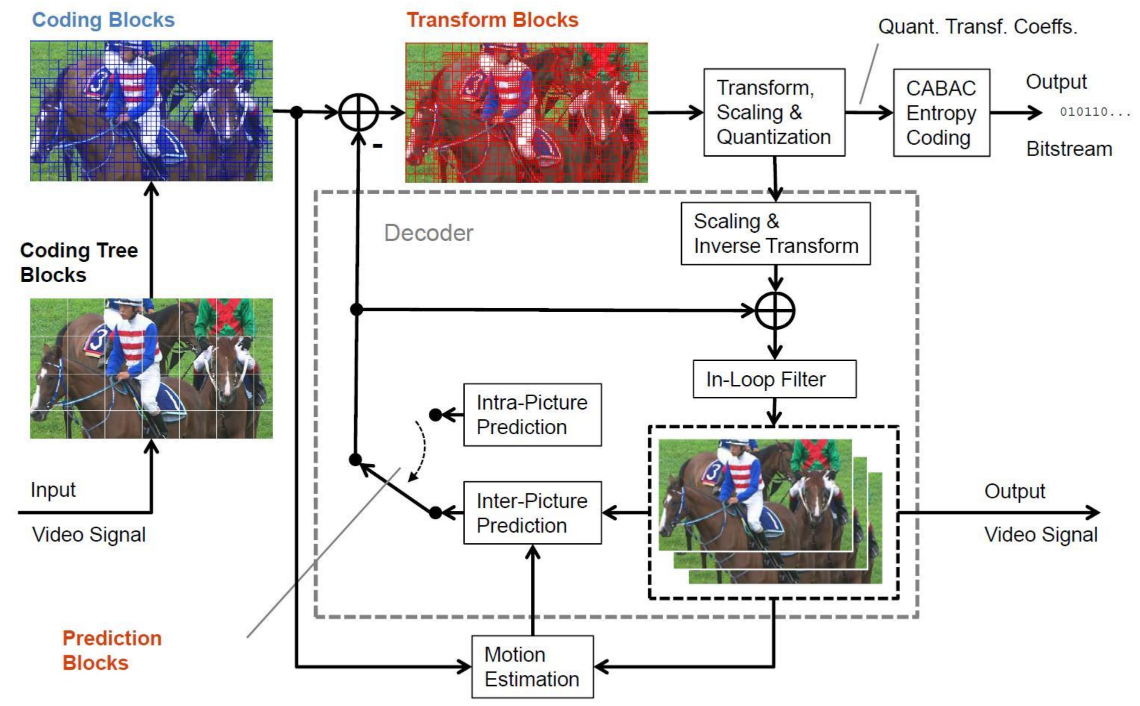


Figure 48: Block diagram of an HEVC encoder. Source: [6]

The residue of the inter or intra prediction is processed by a linear spatial transformation. The transform coefficients are then scaled, quantized, entropy coded and transmitted with the prediction information. The presence of a simplified decoding path inside the encoder, as in the H.264/AVC standard, ensures that both encoder and decoder generate the same prediction by using the same input data.

Thus, the quantized transform coefficients undergo an inverse scaling and then an inverse transform to reproduce an approximation of the residue. The residue is then added to the prediction, and the result of this addition can be therefore subjected to one or two filters to smooth the artifacts induced by the standard's block partitioning.

The final representation of the frame (which is a duplicate of the decoder output) is stored in a frame buffer used for the prediction of the next frame.

3.2 Quadtree partitioning

The pictures are divided into square regions called *coding tree units* (CTUs) consisting of one luminance and two chrominance *coding tree blocks* (CTBs) and an associated syntax. CTUs can be recursively partitioned (until a depth defined by the standard) into smaller blocks called Coding Units (CUs) with the partitioning structure based on a *quadtree* representation. It allows picture partitioning into multiple block sizes that adapt easily to the content of images whilst having a reduced signaling cost. The concept of CTU is very similar to the one of macroblock in the H.264 standard.

In comparison with the traditional macroblock (16x16 fixed size for luminance), HEVC supports the choice of the size of the CTB according to memory constraints and computing power. Larger than in previous standards, the supported sizes for CTBs are particularly beneficial when encoding HD video content, once the correlation between neighboring pixels increases significantly. Three basic units are also defined in the standard:

Coding unit (CU): is a block with size varying from 8x8 to 64x64 (the maximum size specified by the standard is that of the CTU). The decision between intra or inter coding (base prediction type) is taken at this level. A CTU may contain only one CU or be divided into several CUs; and each CU has its own partitioning into prediction and transform units.

Prediction unit (PU): specifies the type of prediction of a block. A PU is rooted at a CU. HEVC supports varying PU sizes ranging from 4x4 up to 64x64 and they can assume symmetric and asymmetric formats. The set of possible sizes is defined according to the type of prediction set for the CU:

– Skip : M×M.

– Intra : M×M, M/2×M/2 (if M/2 = 4).

– Inter : M×M, M/2×M, MxM/2, M/2×M/2, M/4×M (L), M/4xM (R), MxM/4 (U) and MxM/4 (D).

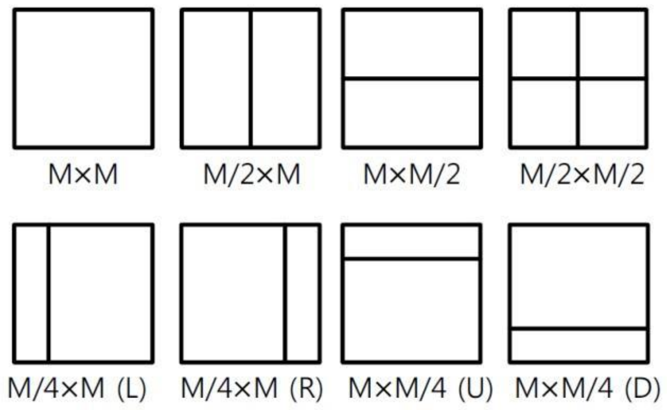


Figure 49: Partitioning modes of a CU into PUs

Transform unit (TU): the decomposition of a CU in TUs defines the sizes for the transforms and quantization applied to this region of the picture. Three levels of decomposition are possible (forming a *transform tree*), and each TU takes sizes ranging from 4x4 to 32x32. All the TUs are squares and may encompass several PUs.

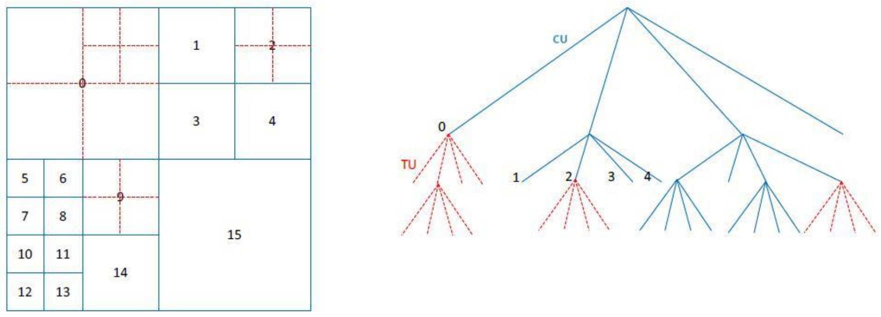


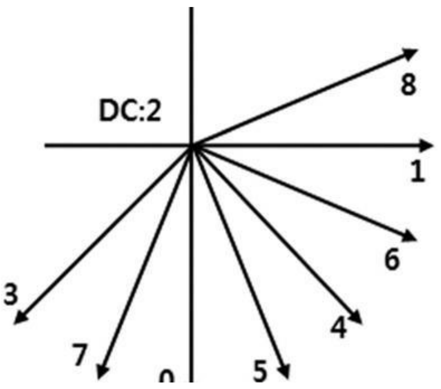
Figure 50: Partitioning of a CTU into CUs and CUs into TUs. Source: [2]

It is important to note that the partitioning of a CU into PUs and TUs are independent.

3.3 Intra-frame prediction

Intra-frame prediction exploits spatial redundancies and is based on an increase in the number of prediction modes compared with the H.264/AVC standard (Figures 9 and 10). They include 33 spatial directions, the DC (average of reference pictures) and planar predictors. The availability of prediction modes depends on the size of the PU: 3 predictors for 64x6 PUs, 17 for 4x4 PUs and 35 for other PU sizes. These multiple modes can effectively capture pixel redundancies in the vicinity at the cost of increased computational power and signaling.

According to tests conducted in [2], the bitrate reduction provided by intra-frame coding in HEVC, compared with H.264/AVC is of about 22% and can reach up to 36%.



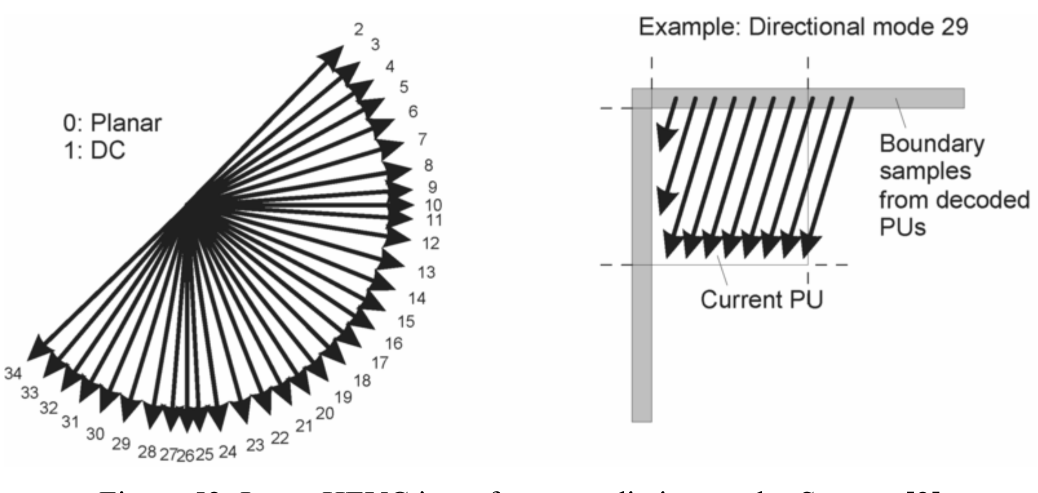


Figure 52: Luma HEVC intra-frame prediction modes Source: [2]

3.4 Inter-frame prediction

Inter-frame prediction exploits temporal redundancies and is based on two aspects: carrying out the prediction - through motion estimation and compensation - and coding of motion vectors. These two points have been improved in HEVC with a particular emphasis on motion vector coding. Inter-frame prediction is responsible for the largest percentage of bitrate reduction but at the cost of high computational complexity.

Inter-frame prediction happens at the PU level: the ME process carries out the prediction by comparing the current block to previously encoded and reconstructed ones, from other frames. Through comparison (by the use of a distortion metric), the best match is then found and a motion vector is generated. Only the residual block, along with the motion vector, is transmitted in the bitstream.

The MC process accesses the reference pictures stored in the memory and fetches the PU pointed by the coded motion vector to reconstruct the image. It is performed at both the encoder and decoder sides to ensure result consistency.

3.5 Transform and quantization

The transformation of coefficients is very similar to previous standards. The residual CB is partitioned into several square TBs as described in section 3.2. Possible TB sizes are 4x4, 8x8, 16x16 and 32x32 [2] which matches the sizes of transform matrices specified by the standard. For simplicity, a single 32x32 transform is specified, since all the other smaller matrices can be derived from it.

The elements of the core transform matrices were derived by the approximation of a Discrete Cosine Transform (DCT), in order to make it hardware-friendly and suppress the rescaling step, which was mandatory in H.264/AVC.

A Discrete Sine Transform (DST) is used for the special case of luminance 4x4 intra-frame prediction blocks. As the intra-frame prediction is based on the neighboring pixels in the upper left region and is therefore more accurate in these regions, the residue generated by it will also be less substantial. The DST behaves differently than the DCT in the sense that its basic functions start low and increase, so it is better suited to encode this type of small block since the highest residual magnitude will be located mainly in the lower right corner of each block.

The quantization in HEVC remains the same as in H.264/AVC.

3.6 Entropy coding and in-loop filtering

Context Adaptive Binary Arithmetic Coding (CABAC) is the only entropy coding method used for video data in the HEVC standard. Each TB is partitioned into 4x4 sub-blocks on which the coding is applied (see Figure 11). The zig-zag reading order is no longer used (unlike H.264/AVC): the 16 coefficients in each sub-block are processed according to an up-right diagonal, horizontal or vertical scan order.

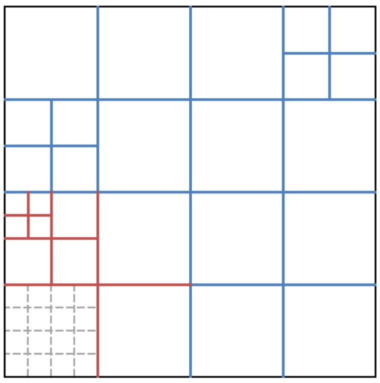


Figure 53: 16x16 TB with 4x4 sub-blocks. Source: [6]

After the CU reconstruction, the pictures are filtered by the in-loop filters. A *deblocking filter* (DBF) is always used to reduce the artifacts introduced by the block processing, even if HEVC's larger block sizes partially softens this effect. Moreover, a *Sample-Adaptive Offset* (SAO) *filter* is added. It is applied after the DBF and is based on the classification of CTU pixels according to their intensities, followed by the application of an offset in order to homogenize the image, to remove ringing artifacts [3] and improve its subjective quality [3].

4 Parallel tools in HEVC

In H.264/AVC, the slices were introduced for robust video transmission. Because of its characteristic, it was also employed for parallelization, but it was not the initial purpose. In addition to slices, HEVC introduces two high-level tools specific for parallel processing: *tiles* and *wavefront parallel processing* (WPP). Both of these tools are based on the subdivision of a picture into multiple *partitions* – which contain an integer number of CTBs - that can be processed in parallel without incurring high coding losses. As it will be seen, tiles and WPP have advantages and disadvantages according to different application use cases.

7.1 Slices and tiles

The concept of *slice*, already used in H.264/AVC, remains unchanged except for the fact that macroblocks are now CTUs. However, some details were added to the definition to clarify the difference between *slices* and *tiles*.

Slices are autonomous in the sense that their syntax elements can be analyzed from the bitstream and pixel values belonging to the area of the image they represent can be correctly decoded without further information. This implies that predictions are not performed beyond the boundaries of the slices. These factors define well their main goal: to ensure resynchronization after data loss, and are often limited in size, which makes slices very useful in the context of data transmission (packetization). The concept of image type is also extended to slices: they can be either I, P or B.

Some problems can be found when using slices for parallelism, which is the reason why this work has not given much attention. Most importantly, using many slices to increase parallelism incurs significant coding losses. Besides that, the number of slices in a frame is determined by the encoder and if the decoder relies on slices to obtain real-time performance, it may not achieve this if it receives a video sequence with only one or a few slices per frame [1].

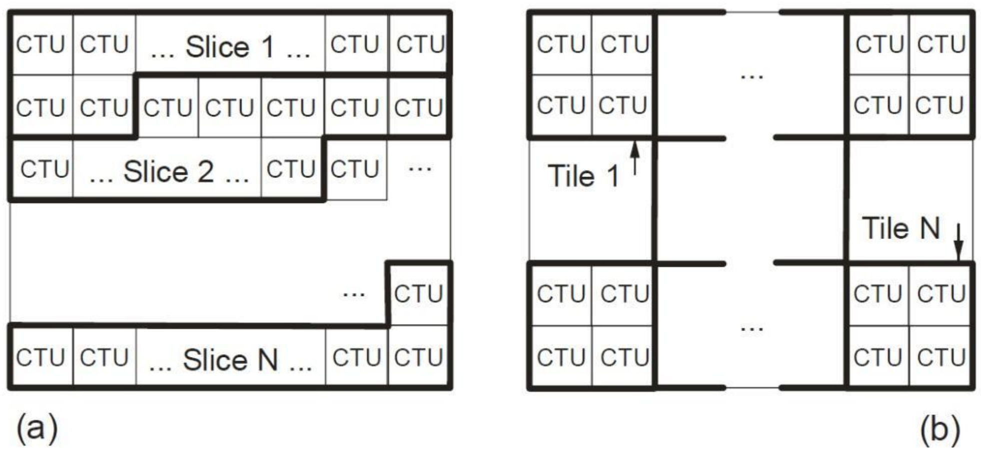
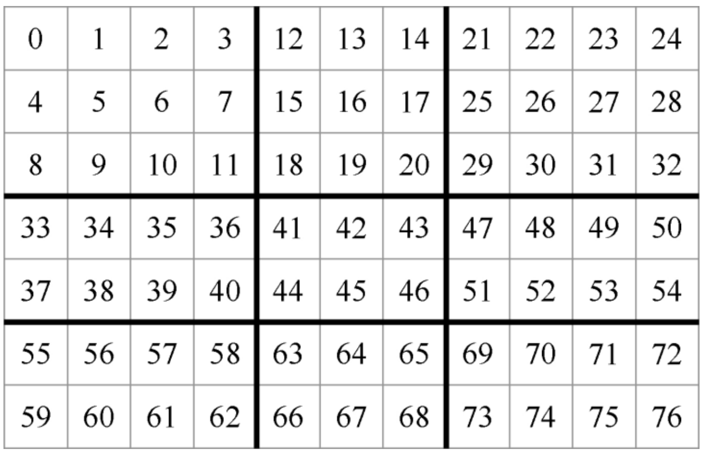


Figure 54: Division of a frame into a) slices and b) tiles. Source: [2]

HEVC also defines *tiles*, which are autonomous regions of the frame that can be independently decoded. A tile consists of a rectangular arranged group of CTUs. Tiles provide parallelism at a more coarse level of granularity (picture/subpicture), and no *sophisticated* synchronization of threads is necessary for their use [2].

The main purpose of tiles is to render the use of parallel processing architectures possible for encoding and decoding rather than provide error resilience. Many tiles can share the same information header if part of the same slice. Alternatively, a single tile can contain multiple slices. Compared to slices, tiles allow picture partitions that are more likely to have higher correlations [4].



0	1	2	3	12	13	14	21	22	23	24
4	5	6	7	15	16	17	25	26	27	28
8	9	10	11	18	19	20	29	30	31	32
33	34	35	36	41	42	43	47	48	49	50
37	38	39	40	44	45	46	51	52	53	54
55	56	57	58	63	64	65	69	70	71	72
59	60	61	62	66	67	68	73	74	75	76

Figure 55: Picture divided into nine tiles, showing the scanning order of the CTBs. Source: [1]

When using Tiles, the regular CTB scan order is changed to a tiles scan order (Figure 13). This complicates the raster scan processing typically performed on single-core implementation.

Therefore, it can be said that:

- the concept of slice is more useful for *data transmission*;
- the concept of tile is more useful for *parallel processing*.

Increasing the number of slices or tiles in an image can then improve coding performance. However, in general, the more slices and tiles one defines, the lower the video coding efficiency. This is because the mutually-independent coding of tiles or slices will prevent the encoder from taking advantage of the spatial/redundancies between the tiles or slices.

4.2 Wavefront Parallel Processing

WPP is based on the processing of rows of CTUs in parallel. In pixel decoding, it preserves all coding dependencies by respecting a shift of at least two CTUs between consecutive rows. This shift is needed because the CTU being currently processed requires the left, top-left, top and top-right CTUs (Figure 14) to be available in order for predictions to operate correctly, since many coding tools need this blocks as input (e.g. intra prediction, motion vector prediction, etc.). This means that the second row can only begin to be processed after at least two CTUs have been processed in the first row, and so on. More concretely, some prediction decisions and adaptation of the context models of entropy coder have to be made in the preceding row in other for decoding to start in the current one. Therefore, WPP also requires additional inter-core communication, which does not happen to be a burden for current multi-core processor architectures.

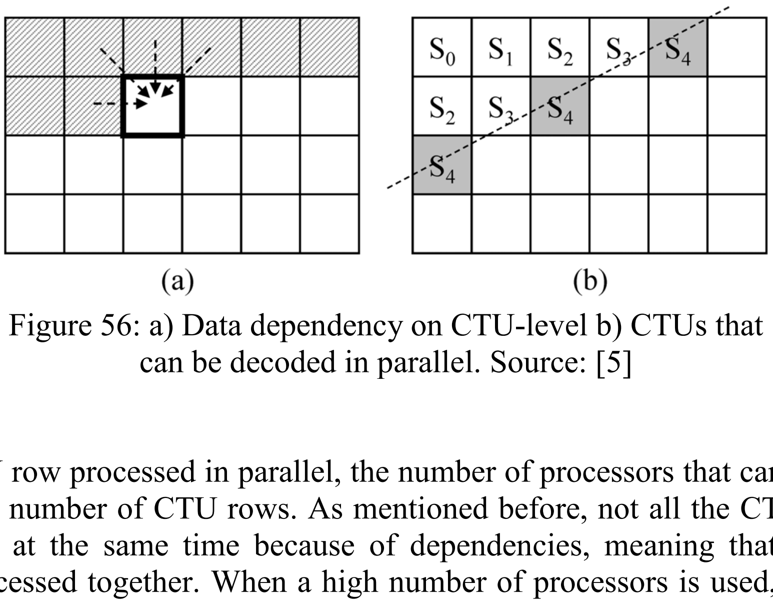


Figure 56: a) Data dependency on CTU-level b) CTUs that can be decoded in parallel. Source: [5]

Being each CTU row processed in parallel, the number of processors that can work in parallel is then up to the number of CTU rows. As mentioned before, not all the CTU rows can start being processed at the same time because of dependencies, meaning that they also won't finish being processed together. When a high number of processors is used, this inefficiency becomes more evident. Overlapping the execution of consecutive pictures can help alleviate this problem, as shows the results obtained through the use of the *Overlapped Wavefront (OWF)* proposed and tested in [1].

Because no coding dependencies are broken at row boundaries, WPP sets itself apart from slices and tiles. Additionally, it does not change the regular raster scan order. Compared to slices, WPP provides parallelism at a more fine level of granularity.

WPP can also be used in the in-loop filtering process. In DBF, the CTU being currently processed has no dependency with its 8-neighboring CTUs since horizontal and vertical filtering are conducted only on its edges in picture-level. This means that all CTUs in a frame can be horizontally and vertically filtered in parallel. The HEVC SAO can be optionally used on CTU-level, where a CTU can select one of options among band offset (BO), edge offset (EO), and no filtering of SAO. The BO of SAO does not have data dependency on CTU-level as it adds the offsets to reconstructed pixels *without* referencing neighboring pixels. There is, however, data dependency between DBF and SAO, so these two processes cannot be conducted in parallel [5].

As it happens with tiles, WPP is generally well suited for the parallelization of the encoder and decoder. It is also suited for both software and hardware implementations and has good speed-coding efficiency trade-off. In fact, even though providing better coding efficiency than tiles [2] (and avoiding the visual artifacts incurred by the latter), the experiments conducted in [1] have shown that WPP achieves slightly lower performance than tiles.

5 Conclusion

The new HEVC video coding standard aims not only at a higher compression ratio than current coding standards, but is also designed with high-level parallel processing capabilities. Different parallelization strategies have been proposed and are included in the standard. The main proposals are slices, WPP and tiles. All these techniques have in common the idea of creating picture partitions that can be processed in parallel by multiple threads/cores in a parallel system. However, they differ in terms of data dependencies, rate-distortion performance and parallel scalability.

The main goal of this paper was to provide an overview of HEVC and the study of its parallel tools. We reviewed the basic concepts of video compression and discussed about the modules where parallelization could play a significant role in performance improvement. All concepts necessary for the understanding of how these parallel tools operate were briefly explained.

Further work consists of an analysis of the implementation of the proposed parallel tools in the HEVC test model reference software and an evaluation of their coding efficiency and performance. Being mainly applied to pixel decoding and parallel-friendly DBF and SAO, there is still a lack of better analysis of WPP in other modules of both encoding and decoding parts of HEVC, which could represent an interesting field for further research.

6 References

[1] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827–1838, Dec. 2012.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample Adaptive Offset in the HEVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[4] C. Blumenberg, D. Palomino, S. Bampi, and B. Zatt, "Adaptive Content-Based Tile Partitioning Algorithm for the HEVC Standard," in *Picture Coding Symposium (PCS), 2013*, 2013, pp. 185–188.

[5] H. Jo, D. Sim, and B. Jeon, “Hybrid Parallelization for HEVC Decoder,” in *Image and Signal Processing (CISP), 2013 6th International Congress on*, 2013, vol. 1, pp. 170–175.

[6] B. Bross, “Relax, it’s only HEVC,” Geneva (CH), 27-Nov-2012.

[7] E. R. Ian, *The H.264 Advanced Video Compression Standard*, 2nd ed. John Wiley & Sons,