

## TABLE OF CONTENTS

	Page
INTRODUCTION .....	1
CHAPTER 1 HEVC ENCODER OVERVIEW .....	7
1.1 Encoder overview .....	7
1.1.1 Frame partitioning .....	8
1.1.2 Block partitioning .....	10
1.1.3 Block prediction .....	12
1.1.4 In-loop filter, quantization and entropy coding .....	14
1.2 Inter-prediction in HEVC .....	15
1.2.1 Motion vector representation .....	16
1.2.2 Subsample interpolation .....	18
1.2.3 Rate-constrained motion estimation .....	18
1.2.3.1 Fullsearch .....	21
1.2.3.2 Test Zone Search (TZS) a suboptimal search .....	22
1.3 Rate-distortion optimization (RDO) .....	25
1.4 Parallel tools in HEVC .....	29
1.5 Summarize .....	33
CHAPTER 2 LITERATURE REVIEW ON HEVC PARALLEL PROCESSING METHODS .....	35
2.1 Coarse-grained parallelization .....	36
2.2 Fine-grained parallelization .....	40
2.2.1 Intra-prediction, in-loop filter, transformation, entropy coder .....	40
2.2.2 Parallel rate-constrained motion estimation .....	41
2.3 Discussion .....	46
2.3.1 Reduced rate-distortion performance .....	47
2.3.2 Bandwidth limitation .....	48
CHAPTER 3 PROPOSED PARALLEL ENCODING FRAMEWORK .....	51
3.1 Dependency analysis .....	51
3.2 Two stage multiple predictor rate-constrained motion estimation (MP- RCME) .....	54
3.2.1 Parallel prediction using MP-RCME .....	55
3.2.2 Best parameter selection .....	56
3.2.3 Offloading MP-RCME to the GPU .....	57
3.3 MP-RCME with postponed fractional calculation .....	59
3.4 Multiple temporal predictor rate-constrained motion estimation (MTP- RCME) .....	62
3.5 Suboptimal parallel RCME on the GPU .....	63
3.5.1 GPU architecture considerations .....	63

3.5.2	Nested diamond search (NDS)	64
3.6	Parallel encoding framework for massively parallel architectures	69
CHAPTER 4 EXPERIMENTAL RESULTS		71
4.1	Setup	71
4.2	Results	75
4.3	Discussion	79
CONCLUSION AND RECOMMENDATIONS		81
APPENDIX I GPU PROGRAMMING WITH OPENCL		83
APPENDIX II PERFORMANCE MEASURES		89
APPENDIX III COMMON TEST SEQUENCES		91
LIST OF REFERENCES		92

## LIST OF TABLES

		Page
Table 1.1	Luma interpolation filter coefficients .....	18
Table 1.2	HEVC high-level parallel tools comparison .....	33
Table 4.1	Hardware specification .....	71
Table 4.2	HEVC encoder configuration.....	72
Table 4.3	MVP prediction methods .....	73
Table 4.4	Comparison of proposed MTP-RCME method with the state-of-the-art methods in terms of RD performance loss .....	75
Table 4.5	Comparison of proposed NDS method with the state-of-the-art methods in terms of RD performance loss .....	76
Table 4.6	TR comparison of FS methods with HM-TZS .....	77
Table 4.7	TR comparison of NDS proposed methods with HM-TZS .....	77
Table 4.8	The BD-Rate improvement for performing the fractional refinement on the CPU compared to performing on the GPU .....	78
Table 4.9	The decrease of TR when performing fractional refinement in the CPU compared to the GPU .....	78



## LIST OF FIGURES

	Page
Figure 1.1      Simplified hybrid video encoder .....	8
Figure 1.2      HEVC block diagram .....	9
Figure 1.3      Frame partitioned by tiles and slices .....	10
Figure 1.4      Partitioning of a $64 \times 64$ CTU into smaller CU .....	11
Figure 1.5      Frame partitioning with the quadtree coding blocks .....	12
Figure 1.6      The possible modes of partitioning a CB into a PBs .....	13
Figure 1.7      MVPs derivation positions from the neighbors and the co-located blocks.....	17
Figure 1.8      Integer and fractional sample positions for luma interpolation .....	19
Figure 1.9      PU motion estimation.....	20
Figure 1.10      TZS initial search patterns.....	23
Figure 1.11      Raster scan in TZS .....	24
Figure 1.12      TZS algorithm flowchart .....	24
Figure 1.13      RDO processing order in CTU .....	27
Figure 1.14      RDO block diagram .....	28
Figure 1.15      GOP parallelization .....	29
Figure 1.16      Frame parallelization .....	30
Figure 1.17      Wavefront parallel processing .....	32
Figure 2.1      Parallel video coding literature categorization .....	35
Figure 2.2      Overlapped wavefront .....	36
Figure 2.3      Inter-frame wavefront.....	37
Figure 2.4      Variable block bottom-up SAD calculation.....	43

Figure 2.5	Using GPU for distortion pre-calculation.....	46
Figure 3.1	RDO hard dependencies between CUs .....	53
Figure 3.2	Inter-prediction soft dependencies between CUs.....	54
Figure 3.3	Proposed two stage MP-RCME framework .....	58
Figure 3.4	MP-RCME with fractional refinement in the GPU .....	60
Figure 3.5	MP-RCME with fractional refinement in the CPU.....	61
Figure 3.6	Fixed search pattern with 64 MV positions.....	65
Figure 3.7	Execution path of PUs in NDS .....	66
Figure 3.8	Predefined PU information required in NDS .....	67
Figure 3.9	Workitem (WI) and workgroup (WG) mapping.....	67
Figure 3.10	NDS GPU kernel algorithm .....	68
Figure 3.11	HEVC parallel encoding framework .....	69

## LIST OF ABBREVIATIONS AND ACRONYMS

AMP	Asymmetric Motion Partition
AMVP	Advanced Motion Vector Prediction
BD-Rate	Bjøntegaard-Delta Rate
BMA	Block-Matching Algorithm
CABAC	Context-Adaptive Binary Arithmetic Coding
CB	Coding Block
CPU	Central Processing Unit
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
DOP	Degree of Parallelism
DPB	Decoded Picture Buffer
ESA	Exhaustive Search Algorithm
FME	Fractional Motion Estimation
FS	Full Search
GPU	Graphics Processing Unit
HD	High Definition
HEVC	High Efficiency Video Coding

HM	HEVC test Model
IME	Integer Motion Estimation
JCT-VC	Joint Collaborative Team on Video Coding
LD	Low Delay
MB	Macroblock
MC	Motion Compensation
ME	Motion Estimation
MPEG	Moving Picture Experts Group
MP-RCME	Multiple Predictor Rate-Constrained Motion Estimation
MTP	Multiple Temporal Predictor
MTP-RCME	Multiple Temporal Predictor Rate-Constrained Motion Estimation
MV	Motion Vector
MVP	Motion Vector Predictor
NDS	Nested Diamond Search
OpenCL	Open Computing Language
PB	Prediction Block
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter



RCME	Rate-Constrained Motion Estimation
RD	Rate Distortion
RDO	Rate Distortion Optimization
RQT	Residual Quad Tree
SAD	Sum of the Absolute Differences
SAO	Sample Adaptive Offset
SATD	Sum of the Absolute Transformed Differences
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
Sub-CU	Sub Coding Unit
TB	Transform Block
TR	Time Reduction
TU	Transform Unit
TZS	Test Zone Search
WPP	Wavefront Parallel Processing



## INTRODUCTION

### **Problem statement and motivation**

Current advances in display manufacturing will increase the usage of high-quality video like high-definition (HD) and ultra-HD (4K UHD and 8K UHD). Video stream delivery at these resolutions demands new and improved video coding technology beyond the capability of the popular H.264/AVC standard (Wiegand *et al.*, 2003).

To address the demand for new video codecs, the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group (MPEG) made a collaborative group entitled the Joint Collaborative Team on Video Coding (JCT-VC) in 2010. The goal was to design a new video coding standard according to the new requirements. The outcome of this group is HEVC/H.265 (Bross *et al.*, 2013) that can reduce the bit rate by half relative to H.264/AVC for the same perceived quality (Sullivan *et al.*, 2012).

HEVC is based on the same hybrid architecture as its preceding standard. However, numerous improvements have been made in the frame splitting, inter and intra-prediction modes, transformation, in-loop filtering and entropy coding of the new design. The coding improvement of HEVC is obtained at the expense of higher computational complexity in the encoder structure. This means that coding a video sequence for real-time applications needs more powerful hardware. In addition, a single processor with existing technologies is not able to perform enough computations to meet the demand.

Parallel processing techniques have been used in many computationally intense problems. In addition, it is apparent that hardware manufacturers intent to build multi-core processors to provide more computation capability. Accordingly, using a powerful graphics processing unit (GPU) along with a multi-core central processing unit (CPU) in smartphones and personal computers is common. The aforementioned reasons and the high complexity of HEVC are

the main reasons for using parallel processing in HEVC. Yet, several difficulties exist for parallelization of an HEVC encoder. In subsequent sections, we explain these complications and aim to increase parallelization of the most complex part of the encoder.

Using parallel processing techniques in HEVC for highly complex processes can be beneficial. Accordingly, the need to take advantage of parallel processing architectures was recognized during the development of HEVC, so it contains features that are friendly to parallel implementation. The novel tools for supporting high-level parallelism include tiles and wavefront parallel processing (WPP). However, in the future sections, it is demonstrated that these tools only provide a low degree of parallelism. Conversely, algorithms need to have a high degree of parallelism in order to exploit GPU and many-core architecture capabilities.

Furthermore, using the high-level parallelization tools will reduce the coding performance of the encoder since they restrict some features of the encoder. These restrictions impact the encoded video quality and the compression ratio. To cope with the aforementioned problems, researches aim at proposing algorithms that provide a fine degree of parallelism while preserving the performance of HEVC. According to many researches, inter-prediction is the most complex step in the encoding process (Bossen *et al.*, 2012),(Vanne *et al.*, 2012). Among the HEVC encoding steps, inter-prediction shows a great potential for providing a high degree of parallelization because of its high computational complexity. Furthermore, it is well suited for implementations on heterogeneous architectures because it can be considered as a data parallelism problem. Therefore, we focus on the parallelization of the inter-prediction and particularly of the rate-constrained motion estimation (RCME) algorithm in HEVC.

In this research, different aspects of inter-prediction parallelization will be investigated and existing issues will be addressed using novel approaches. A major problem on the way to parallelization is the unavailability of motion vector predictor (MVP) that prevents the RCME to be performed in parallel. Furthermore, inter-prediction highly contributes to the improved

performance of HEVC. Thus, a slight performance degradation in the coding algorithm will result in a high rate-distortion (RD) performance degradation. Besides, the implementation aspect of a parallel algorithm has to be considered. In addition to the HEVC encoder, the proposed techniques and methods can be used in transcoding and transrating systems. The results of this research will help designing new generations of encoders, transcoders and transraters that can be employed in both real-time and non-real-time applications.

## Objectives

Considering the aforementioned problem, we defined the main objective of this research as providing an efficient parallel framework with a high degree of parallelism for the rate-constrained motion estimation step of HEVC. This results in an appropriate parallel processing technique for heterogeneous system to speedup the HEVC encoding. To accomplish that goal, we determined the specific objectives and sub-objectives as follows:

- Develop an efficient HEVC parallel framework for many-core architectures with:
  - Execution model for HEVC process with separated steps.
  - Low memory transfer requirements.
  - Parallel rate-constrained motion estimation.
  - Fine-grained parallelization.
- Develop a parallel HEVC encoder software for heterogeneous hardware.

These objectives will be achieved by the approaches and ideas in Chapter 3. To achieve an efficient HEVC parallel framework for many-core hardware, we addressed four sub-objectives. First, to offload calculations on the GPU, the calculation process should be modified. However, the HEVC steps are very dependent of each other; the execution model with separated steps is

a required objective. Furthermore, one major drawback of using a co-processor is due to cost of transferring the raw data to co-processor and transferring the results back to the processor. This objective is addressed by reducing the transferred data and introducing the multi-predictor candidates list.

A high degree of parallelism is an essential objective for efficiency of many-core architectures. We have proposed the multi-predictor rate-constrained ME in order to provide such parallelization. Furthermore, the multiple temporal predictor method will address the RD performance loss caused by removing dependencies of motion estimation. Moreover, a nested diamond search method specifically designed to perform motion estimation in GPU is introduced to improve the implementation for heterogeneous hardware. This work has led to the publication of two conference papers in prestigious international conferences (Hojati *et al.*, 2017a,b).

Since the high complexity encoder is usually used by video providers, the industrial objective is addressed by developing an HEVC encoder for heterogeneous hardware. The results of this project will help video streaming providers to deliver high-quality video content with increased speed. Reducing the encoding time allows video providers to produce and deploy high quality and real-time applications, which enables them to extend the applications of HEVC. As a result, we expect, besides the academic contribution achievements of this project, to benefit video industries for entertainment and interactive video applications as the main areas of video coding.

## **Thesis Structure**

In this thesis, after a brief survey of the HEVC architecture in Chapter 1, we review the literature of the parallel HEVC in Chapter 2. Furthermore, we especially investigate publications on the parallel motion estimation field. In Chapter 3, the proposed methods are explained which consist of multiple predictor rate-constrained motion estimation (MP-

RCME), multiple temporal predictor rate-constrained motion estimation (MTP-RCME) and nested diamond pattern (NDS). Afterwards, we will present our experimental results. Finally, we conclude our work and provide recommendations.





## CHAPTER 1

### HEVC ENCODER OVERVIEW

In this chapter, we briefly explain the high efficiency video coding (HEVC) standard with focus on parts that are important for our research. In the first section, the HEVC encoding process is illustrated with a brief description of block structures and other key features of HEVC. Afterward, the inter-prediction as the most time-consuming part of the encoder will be explained in more details. Then, the rate distortion optimization (RDO) is described. Finally, we describe the existing parallel tools in HEVC followed by an analysis of their strengths and weaknesses. Moreover, a complete description of the HEVC standard is available in (Sullivan *et al.*, 2012), (Sze *et al.*, 2014) and (Wien, 2015).

#### 1.1 Encoder overview

Similar to H.264/MPEG-4 AVC, HEVC adopted a hybrid video coding architecture with several enhancements in each part (Katsigiannis *et al.*, 2013). The HEVC coding process is based on removing redundant information in several stages. Figure 1.1 shows a simplified cascade schematic of the HEVC encoder architecture. Each picture is partitioned into rectangular block-shaped regions and the pixels of this block are coded by prediction mode and residual signal. There are two possible prediction modes in HEVC, intra-prediction and inter-prediction. Intra-prediction uses spatial information within the same frame to build the current block content. On the other hand, inter-prediction will find the best match for the current block in previously coded frames. Regardless of the prediction type, the difference between the original block and its predicted block will be transformed by a linear spatial transform. Subsequently, the resulting transform coefficients will be scaled and quantized. Finally, this residual data plus prediction information are coded by the entropy coder to form the encoded bit stream.

However, as it is depicted in fig. 1.2, a real encoder architecture is much more complex. The encoder also performs the decoding process to generate the exact same results as on the decoder



Figure 1.1 Simplified hybrid video encoder

side. Inverse scaling and inverse transform of the encoded data will reproduce the image and then the residual signal can be calculated. Furthermore, a loop filter is used to smooth out the artifacts induced by the block-wise processing of HEVC. The reconstructed picture is stored in a buffer in order to be used for subsequent predictions. This architecture prevents any drifting or aggregation error in the coding progress because the reconstructed picture is identical to the picture on the decoder side.

Furthermore, there are several decisions in each step that have to be made by the encoder in order to provide best rate-distortion performance. For instance, split blocks and prediction parameters have a great influence on rate-distortion. This process is performed by an exhaustive execution of RDO by evaluating all possible choices and selecting the best as the final choice.

In the following sections, we will explain the HEVC features more precisely.

### 1.1.1 Frame partitioning

The high-level segmentation of a picture in HEVC is achieved based on the slice concept. Using the slices, the frame can be partitioned in such a way that each slice is independently decodable from other slices (Sullivan *et al.*, 2012). A slice may consist of a complete picture or parts of it. Each slice contains an integer number of consecutive coding tree units (CTUs). The main advantage of slices in HEVC can be mentioned as:

1. **Error Robustness:** Partitioning the picture into smaller independent parts allows to gain error robustness. Therefore, in case of data losses, the decoder is able to discard the erroneous stream parts and start decoding from a correct block. Furthermore, the slices are sent in separate network packets, thus, the loss of a transport packet results in a loss of only one slice.

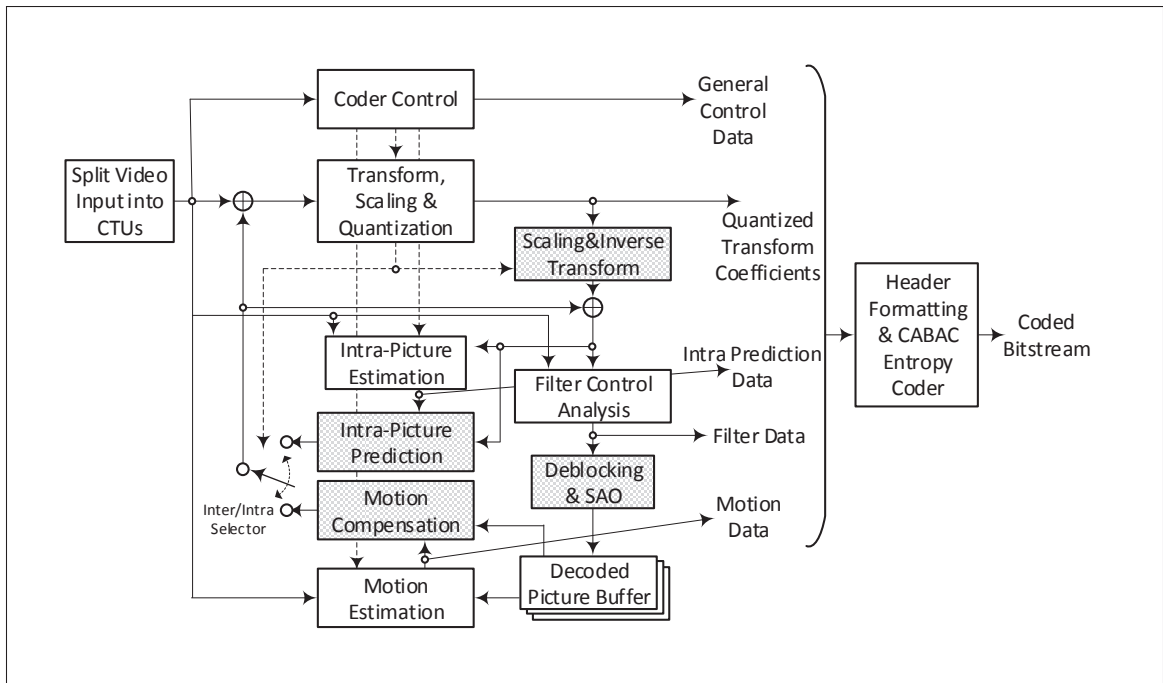


Figure 1.2 Typical HEVC video encoder with embedded decoder elements shaded in light gray. Adapted from (Sullivan *et al.*, 2012, p. 1651)

2. Parallel Processing: To partition the picture into units that can be processed in parallel since there is no inter-dependencies between slices.

Coding units (CUs) in a slice are processed in raster scan order such that each slice of a picture is independently decodable. This is achieved by terminating the context-adaptive binary arithmetic coding (CABAC) bitstream at the end of each slice and by breaking CTU dependencies across slice boundaries. This prevents the encoder from using the spatial information outside the slice boundaries. Thus, the coding efficiency usually decreases quite substantially when increasing the number of slices used for a picture.

Additionally, each slice can be coded using different coding types among three slice types. The first type is I slice in which all CUs of the slice are coded using only intra-prediction mode. The second type is P slice. Inside a P slice, in addition to the coding types of I slice, some CUs can be coded using inter-prediction with one motion-compensated prediction signal per

prediction block (PB). The last slice type is called B slice that is similar to P slices with two predictions per PB.

The tile is a partitioning mechanism similar to slices, which is based on a flexible subdivision of the picture into rectangular regions of CTU. In addition, coding dependencies between CTUs of different tiles are prohibited. Contrary to slices tiles provide better support for parallel processing rather than error resilience (Zhou *et al.*, 2012). Although, non-uniform tiles are allowed in HEVC, typically each tile consists of an approximately equal numbers of CTUs (Misra *et al.*, 2013). In fig. 1.3, tiles and slices partitioning is illustrated.

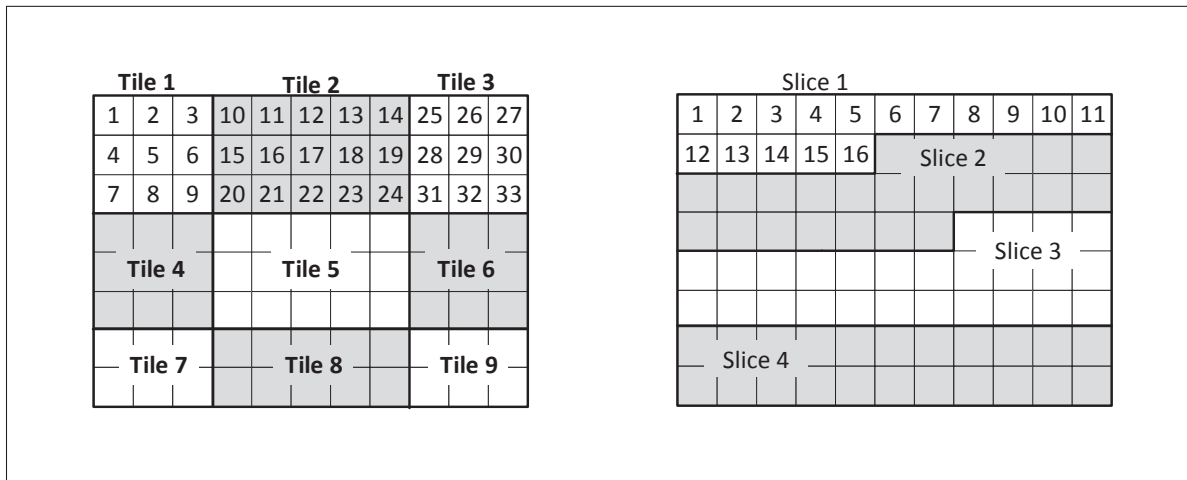


Figure 1.3 Frame partitioned by Tiles (left) and Slices (right). Adapted from (Misra *et al.*, 2013)

Furthermore, using slices and tiles simultaneously is permitted but tiles must include complete slices or slices must include complete tiles.

### 1.1.2 Block partitioning

In contrast to the fixed Macroblock (MB) size ( $16 \times 16$ ) in H.264/AVC, HEVC uses a more adaptive quadtree structure called CTU. The quadtree structure consists of blocks and units with a maximum size of  $64 \times 64$ . A block includes a rectangular area of picture samples and

a unit is formed by a luma block and two chroma blocks with related syntax information. For instance, a CTU is formed by a luma coding tree block (CTB) and two chroma CTBs with syntax determining further subdivisions. As a result of subdivisions, new units called CUs are generated. The encoder can decide whether a CTB divides into one CU or more. In fig. 1.4 an example of a coding tree is depicted.

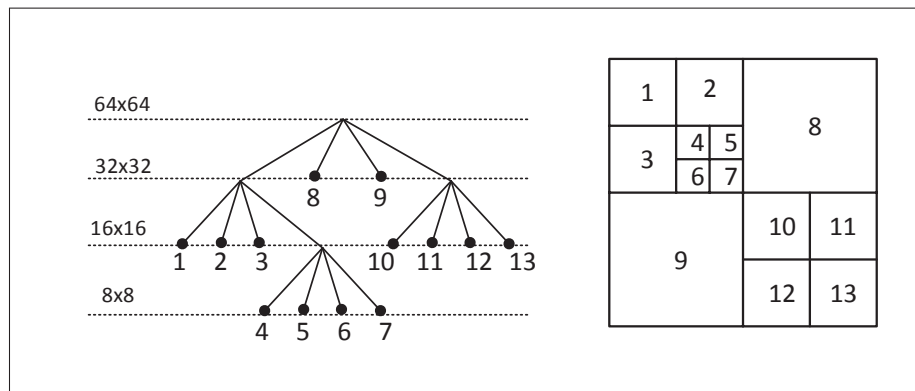


Figure 1.4 Partitioning of a  $64 \times 64$  CTU into CU of  $8 \times 8$  to  $32 \times 32$  luma samples. The partitioning described by a quadtree (left). The numbers indicate the coding order of the CU

The CU can be divided into prediction units (PUs) and transform units (TUs) that perform prediction and transformation respectively. CUs, PUs and TUs consist of associated luma and chroma blocks called coding blocks (CBs), PBs and transform blocks (TBs) respectively.

It is clear that HEVC splitting is more adaptive relative to the approach used in H.264/AVC and is notably useful for high resolution videos. Figure 1.5 shows an example of dividing a picture into CBs and TBs using quadtree structure. The quadtree that is used for dividing a CB into TBs has its root at the CB level and is called residual quad tree (RQT) since it is built over the residual data.

As it can be seen, there are many possible ways to split a picture into units and blocks. The encoder should perform heavy computations to use the full capabilities of the syntax. This is mostly because the encoder should choose the most efficient coding tree structure and the

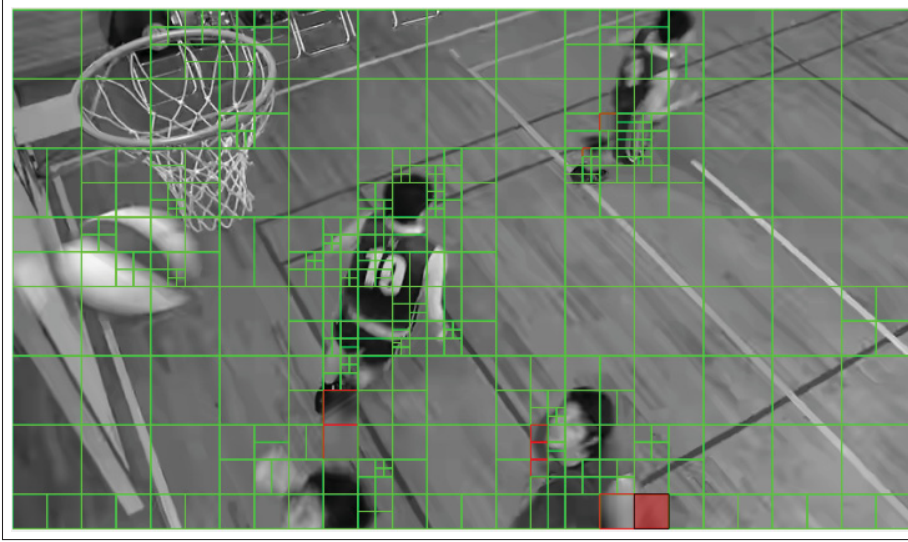


Figure 1.5 Frame partitioning with the quadtree coding blocks

best way for the subdivision of a CB into PBs and TBs. The RDO process considers all the encoding possibilities and compares them with regard to bit rate and picture quality. Since CB is the root for PB partitioning and RQT configuration, it could be generally deduced that the computational complexity of RDO increases monotonically with the depth of CB splitting (Ma *et al.*, 2013). It is obvious that CB depth can be limited to reduce the complexity but it will decrease the coding efficiency because small CBs are efficient for regions of the picture with complex details while large CBs provide better results for large homogeneous areas.

### 1.1.3 Block prediction

In HEVC, each coding block is the root for prediction blocks. The possible modes of splitting a CB into PBs are shown in fig. 1.6. Furthermore, intra-prediction mode can use  $N \times N$  and  $2N \times 2N$  partitioning, while inter-prediction can use all partitionings. Here we explain intra-prediction and inter-prediction briefly.

#### • Intra-prediction

Intra-prediction reduces the picture spatial redundancy. The available samples on the edge of a block will be used to predict samples inside the block. There are 33 directional modes, 1

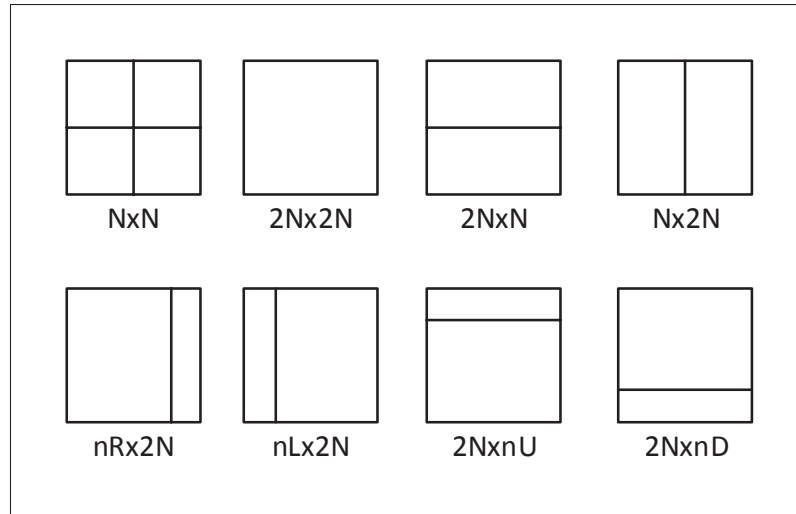


Figure 1.6 The possible modes of partitioning a CB into a PB. Intra CBs can apply only the  $N \times N$  and  $2N \times 2N$

planar and 1 DC prediction modes in HEVC. The selected modes are encoded by deriving most probable modes according to previously coded adjacent blocks.

#### • Inter-prediction

In order to remove temporal redundancy between the frames of a video, inter-prediction is used in HEVC. Typically, in video sequences, only small differences between consecutive scene content are observed and these differences are usually due to the movement of objects. Thus, extensive parts of the scene can be efficiently represented by motion vectors and a prediction error. The motion vectors determine what picture regions should be used from the reference picture to predict the content of the current picture. The prediction error compensates the part of the content that could not be obtained by the applied motion model. In HEVC, the process of finding the best match with respect to rate-distortion cost is RCME. The goal of RCME is to find the motion vector that jointly requires the minimum number of bits and produce minimum distortion. Since the inter-prediction concept is crucial to this research, we explain this part with more details in Section 1.2.

#### 1.1.4 In-loop filter, quantization and entropy coding

As illustrated in fig. 1.2, there are other important parts in the HEVC encoder which we will explain briefly.

##### • In-loop filtering

Since hybrid video coding is a block-based process, appearance of visual artifacts in block boundaries is inevitable. Loop filtering is a method to increase the reconstruction quality of the picture especially at block edges. Since the filter is located in the loop, the enhancement will affect the quality of the output pictures and reference pictures, which are accessible for prediction of next pictures.

In HEVC, two sequential in-loop filters are considered. The first step is a deblocking filter that is applied to prediction blocks and transform blocks' edges. The second filter is a sample adaptive offset (SAO) which can be calculated by sample value differences of a local area, or by classifying all pixels of a region into multiple groups.

##### • Transformation and quantization

The prediction step will find the best content prediction of a block. However, the difference between actual content and predicted value can generate significant distortion. This difference is called residual signal and this residual has to be transmitted to the decoder in order to allow the proper reconstruction of the original signal.

In HEVC, the residual signal of a picture is divided into square blocks called transform block, each residual block is then input to a two-dimensional forward transform. The resulting transform coefficients (coeff) are then processed by a quantizer (which performs division by a quantization step size  $Q_{step}$  and then rounding) to get quantized transform coefficients. Using this process, the number of bits required for residual coding will be significantly reduced.

##### • Entropy coding

Entropy coding is a lossless compression method that uses the statistical information to achieve compression. For example, when compressing a picture, frequently used colors are each



represented by a few bits, while infrequently used colors are each represented by more bits. CABAC is used for entropy coding in HEVC. Entropy coding is the last stage of the video encoding process, after the video signal has been condensed to a sequence of syntax elements. Syntax elements include all required information to reconstruct the video sequence, for instance quadtree structures, prediction information, etc. The simplified design of CABAC includes the key elements of binarization, context modeling, and binary arithmetic coding. In the binarization step, syntax elements will be mapped to binary symbols (bins). Context modeling estimates the probability of each bin based on specific context from previously coded bins. Lastly, binary arithmetic coding compresses the bins to bits considering the estimated probability. The context modeling improves the compression ratio by building a statistical model of previously coded bins, however, it forces a dependency on the encoding process. Without knowing the state of the context, CABAC is not able to produce the correct compressed bit stream.

## **1.2 Inter-prediction in HEVC**

The inter prediction is performed on the PB and tries to find the best possible prediction in an available reference picture. The corresponding PU includes the information of how inter prediction is performed. Inter prediction is also called motion compensated prediction since moved areas of the reference pictures are used for the prediction of the current PB. The resulting displacement between the region in the reference picture and the current PB is inferred as the motion vector. Since the encoded motion vectors are usually determined by the application of a rate-distortion criterion, these vectors do not essentially represent the true motion of the region but the most efficient representation in terms of rate-distortion. The process of finding best motion vector is called RCME.

Furthermore, in order to achieve better prediction, motion vectors are applied in quarter-sample accuracy for luma inter prediction. The same motion vectors are applied for chroma components. The required subsamples are generated from integer pixels using interpolation filters.

In this section, first the motion vector representation is explained to illustrate the number of bits required for signaling the motion vector. It can be seen that the data dependency in rate-constrained motion estimation is due to motion vector representation. Next, subsample interpolation filters are briefly described. Finally, the rate-constrained motion estimation is explained.

### 1.2.1 Motion vector representation

The related motion vectors for motion compensation can be signaled in two ways. In the first method, the motion information can be encoded, where the applicable motion vector is made from a motion vector predictor (e.g., adjacent block motion vector) and a motion vector difference. Using a predictor and the difference allows higher compression. The motion vector predictor is chosen from candidates obtained from spatial and temporal neighborhoods of the current block. This representation of motion vector is called advanced motion vector prediction (AMVP). The motion vector difference can be expressed as follows:

$$\mathbf{mvd} = \mathbf{mvp} - \mathbf{mv} \quad (1.1)$$

where,  $\mathbf{mv}$  is the motion vector,  $\mathbf{mvp}$  is the motion vector predictor and  $\mathbf{mvd}$  is the difference between the two vectors. The  $\mathbf{mv}$  is the final result of the RCME.

The motion vector predictor is derived from the other PUs by the MVP derivation process. The result of the MVP derivation process is a vector set consisting of two motion vectors  $\mathbf{mvp}_A$  and  $\mathbf{mvp}_B$ . The encoder selects one of these two and it is used as  $\mathbf{mvp}$  in Eq. (1.1).

The derivation process starts with building a list of neighboring and temporal motion vectors. This list is generated from the candidates that are illustrated in fig. 1.7 where A0, A1, B0, B1, B2 are adjacent PU blocks and C1 and C0 are co-located and adjacent to co-located PU blocks in the previous frame.

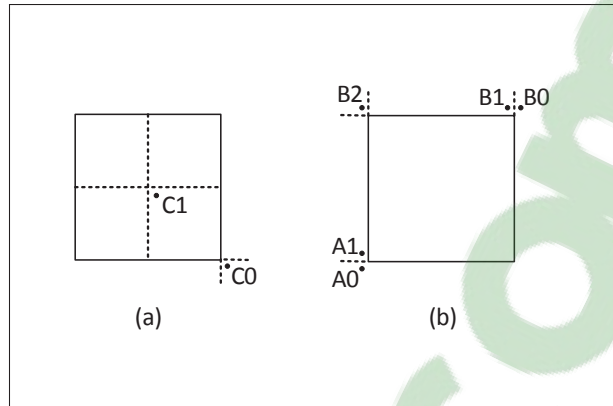


Figure 1.7 MVPs derivation positions  
from (a) temporal MVPs positions (b)  
spatial MVPs positions

In order to derive a list of motion vector predictors for AMVP,  $mvp_A$  is derived from A0 and A1 locations and  $mvp_B$  from B0, B1 and B2 locations, respectively in that order. A candidate is used if inter-prediction is used for that PU. If fewer than two candidates are derived, a temporal candidate will be used. Finally, if  $mvp_A$  or  $mvp_B$  are still not available, they will be filled with motion vector (0,0). For instance, when all the PUs in the aforementioned locations are encoded using intra-prediction there is no candidate available. Furthermore, we use the following notation to perform this process:

$$\{mvp_A, mvp_B\} = \text{Derive}(PU_{A0}, PU_{A1}, PU_{B0}, PU_{B1}, PU_{B2}, PU_{C0}, PU_{C1}) \quad (1.2)$$

Second method for signaling motion information is called merge mode prediction. In this mode, the motion information can be inferred by selection from a set of neighboring candidates, without encoding a motion vector difference. After building the merge MVP candidates, there is no need for motion estimation and the encoder just evaluates the RD cost of the current block, using the exact prediction information of the merge MVPs.

### 1.2.2 Subsample interpolation

To improve the motion estimation accuracy, the encoder can conduct fractional motion estimation performed on interpolated samples. In that case, quarter-sample precision is used for the MVs, and 7-tap or 8-tap filters are used for interpolation of fractional-sample positions (Sullivan *et al.*, 2012). The interpolated values for luma components can be calculated using the coefficients presented in table 1.1.

Table 1.1 Luma interpolation filter coefficients

Phase	Filter coefficients
1/4	$[-1, 4, -10, 58, 17, -5, 1]/64$
1/2	$[-1, 4, 11, 40, 40, -11, 4, -1]/64$

In fig. 1.8, integer and fractional sample positions are shown, where fractional positions are denoted by lowercase symbols. It is evident that the calculation of interpolated samples is highly complex for large regions. As a result, fractional motion estimation on subsamples is performed for a small region around the best integer motion vector.

### 1.2.3 Rate-constrained motion estimation

Rate-constrained motion estimation is a process to estimate the best prediction parameters based jointly on distortion and rate. The HEVC standard is not explicitly determining how to perform the motion estimation. Figure 1.9 shows the motion estimation of the current block based on a reference frame.

Therefore, the motion estimation algorithm in the HEVC test Model implementation (HM1) is formulated in this section. Since the calculation of the exact distortion and rate would be extremely time-consuming for all motion vectors, an estimation of the cost is used.

In this estimated cost, the sum of the absolute differences (SAD) is used as distortion measure for integer precision motion vectors. Also, the sum of the absolute transformed differences

A <sub>-1,-1</sub>				A <sub>0,-1</sub>	a <sub>0,-1</sub>	b <sub>0,-1</sub>	c <sub>0,-1</sub>	A <sub>0,-1</sub>				A <sub>0,-1</sub>
A <sub>-1,0</sub>				A <sub>0,0</sub>	a <sub>0,0</sub>	b <sub>0,0</sub>	c <sub>0,0</sub>	A <sub>1,0</sub>				A <sub>2,0</sub>
d <sub>-1,0</sub>				d <sub>0,0</sub>	e <sub>0,0</sub>	f <sub>0,0</sub>	g <sub>0,0</sub>	d <sub>1,0</sub>				d <sub>2,0</sub>
h <sub>-1,0</sub>				h <sub>0,0</sub>	i <sub>0,0</sub>	j <sub>0,0</sub>	k <sub>0,0</sub>	h <sub>1,0</sub>				h <sub>2,0</sub>
n <sub>-1,0</sub>				n <sub>0,0</sub>	p <sub>0,0</sub>	q <sub>0,0</sub>	r <sub>0,0</sub>	n <sub>1,0</sub>				n <sub>2,0</sub>
A <sub>-1,1</sub>				A <sub>0,1</sub>				A <sub>1,1</sub>				A <sub>2,1</sub>
A <sub>-1,2</sub>				A <sub>0,2</sub>	a <sub>0,2</sub>	b <sub>0,2</sub>	c <sub>0,2</sub>	A <sub>1,0</sub>				A <sub>2,2</sub>

Figure 1.8 Integer and fractional sample positions for luma interpolation. The green locations are integer sample positions. Using the interpolation filters, the yellow sub-samples are calculated. Finally, the blue sub-samples are calculated vertically

(SATD) is used for fractional motion vectors. Moreover, the rate is chosen as the motion vector difference cost. Thus, the prediction parameters that results in the minimum cost can be estimated as follows:

$$P_{ME} = (mv^*, mvp^*) = \underbrace{\arg \min}_{\forall mv \in MV_{search}, \forall mvp \in \{mvp_A, mvp_B\}} \{D(mv) + \lambda \cdot R(mvp - mv)\} \quad (1.3)$$

where the two derived motion vector predictor candidates are denoted by  $mvp_A$  and  $mvp_B$ . As mentioned before, these predictors are selected from neighboring PUs using Eq. (1.2). In addition,  $MV_{search}$  is a set of paired integers that determines the displacement motion vector  $mv$ .

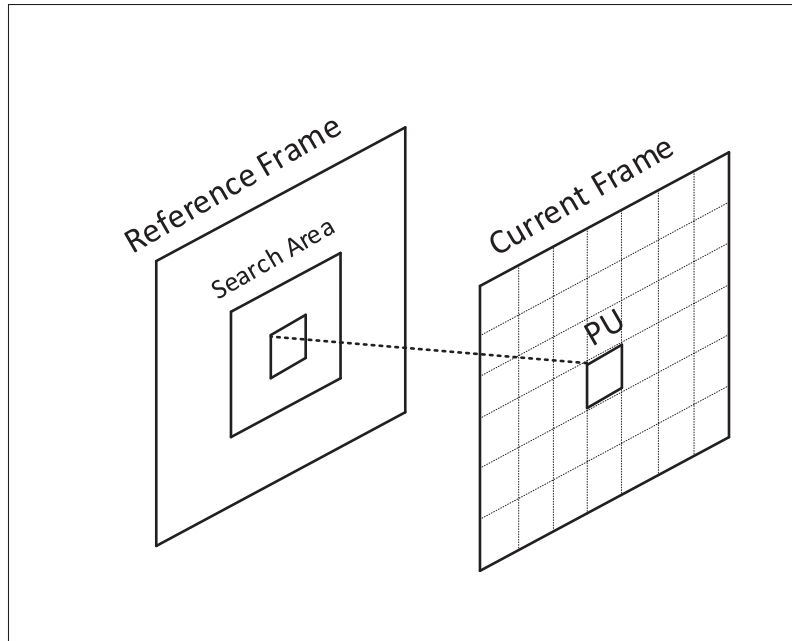


Figure 1.9 Motion estimation in a search range around the co-located block in the reference picture

Furthermore,  $\lambda$  is the Lagrangian weighting factor, used to minimize the distortion measure  $D$  and the required rate  $R$  jointly. The calculation of Eq. (1.3) can be performed by different methods called search algorithms. The most straightforward search method is an exhaustive fullsearch algorithm. However, more complex methods like Test Zone Search (TZS) can be used to find a suboptimal result with less computations. Fullsearch and TZS are explained in the following sections.

Furthermore, performing fractional motion estimation for the whole search range imposes a great amount of calculations. In order to overcome this problem, first, the rate-constrained motion estimation is performed for integer motion vectors, and then the fractional motion vector is determined around the best integer motion vector. Therefore, Eq. (1.3) can be calculated by integer motion estimation followed by fractional motion estimation. In the integer motion estimation, the motion estimation is searching for the best motion vector in the original pixels. However, the fractional motion estimation is pointing to the pixel that result from the interpolation filter and should be calculated by the encoder.

### 1.2.3.1 Fullsearch

Fullsearch is the most straightforward algorithm to perform RCME. This algorithm exhaustively evaluates motion vectors of a rectangular area. In the Eq. (1.3), the  $MV_{\text{search}}$  set covers a square area, determined by the search range (SR) variable as:

$$MV_{\text{search}} = \left\{ (x, y) \mid |x| \leq \text{SR}, |y| \leq \text{SR} \right\} \quad (1.4)$$

The above equation is representing a rectangular search range. In addition,  $SR$  value can be fixed (defined by users) or adaptively changed by the motion estimation algorithm. For each MV in the  $MV_{\text{search}}$ , the distortion is calculated by SAD.

Since performing fractional motion estimation for the whole search range imposes a great amount of calculations, Eq. (1.3) is calculated by integer motion estimation followed by fractional motion estimation using Eq. (1.5) and Eq. (1.6).

$$P_{\text{IME}} = (\mathbf{imv}^*, \mathbf{mvp}^*) = \underbrace{\arg \min}_{\substack{\forall \mathbf{imv} \in MV_{\text{search}}, \\ \forall \mathbf{mvp} \in \{\mathbf{mv}_A, \mathbf{mv}_B\}}} \{ \text{SAD}(\mathbf{imv}) + \lambda_i \cdot R(\mathbf{mvp} - \mathbf{imv}) \} \quad (1.5)$$

$$P_{\text{FME}} = (\mathbf{fmv}^*, \mathbf{mv}^*) = \underbrace{\arg \min}_{\substack{\forall \mathbf{fmv} \in \{(\mathbf{imvx} + x, \mathbf{imvy} + y)\}, \\ x, y \in \{0, \pm \frac{1}{4}, \pm \frac{1}{2}, \pm \frac{3}{4}\}}} \{ \text{SATD}(\mathbf{fmv}) + \lambda_f \cdot R(\mathbf{mvp}^* - \mathbf{fmv}) \} \quad (1.6)$$

where  $\mathbf{imv} = (\mathbf{imvx}, \mathbf{imvy})$ . Moreover,  $\lambda_i$  and  $\lambda_f$  are the Lagrangian factors for integer and fractional RCME respectively.

### 1.2.3.2 TZS a suboptimal search

Suboptimal approaches examine a reduced set of motion vectors (MVs) to decrease the computational complexity but, as their name indicate, they may find suboptimal MVs. Among those, the Test Zone Search (TZS) has been adopted in the HEVC test model implementation. TZS has four different steps: motion vector prediction, initial search, raster search and refinement search.

The motion vector prediction is the first step of TZS. It is used to predict the region near to the best possible result. This will speed up the block search in the next steps by guiding the search to a region where probably the best matching block is located. The motion vector prediction is based on five predictors from the MVs of previously encoded blocks. The left, the upper, the upper-right, the median and the co-located predictors are tested. The best predictor is chosen as the one that results in the lowest RD cost (Jeong *et al.*, 2015). This MV is the best motion vector predictor and used as the center of initial search.

The second step is the initial search. In this step, a search on the positions based on a diamond or square pattern is performed. These patterns are illustrated in fig. 1.10. The central point is at the MV that was obtained at the prediction step. The expansion of search pattern continues until reaching the maximum size of the search grid. The expansion is doubled in each iteration and each iteration is illustrated with the same color in the fig. 1.10. The point with the minimum cost is selected as the best matched point. Furthermore, the search iterations can be stop if no block with a smaller cost is found after three expansion levels.

The raster search is the next step. This step is only executed if the distance between the best block matching found in the initial step and the best block matching on the pattern is greater than a sub-sampling step. The distance is calculated as the number of pixels between the best block matching position and the center of the pattern. In the TZS algorithm, the sub-sample step is called iRaster and it is equal to 5. The raster search, performs a sparse fullsearch over the search area. The sub-sampling of iRaster is performed both horizontally and vertically. The raster search pattern is illustrated in fig. 1.11.



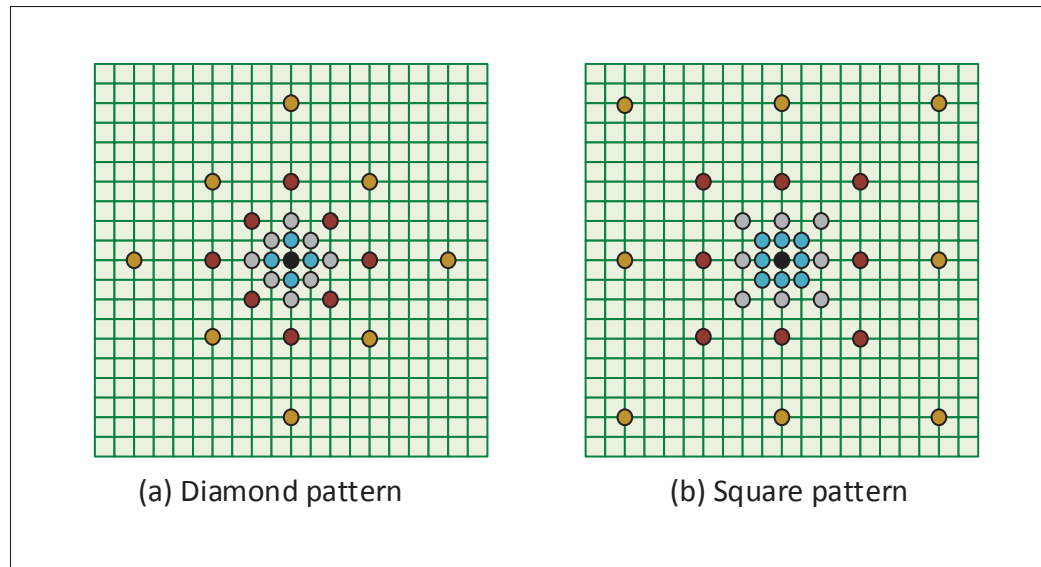


Figure 1.10 TZS initial search patterns

The final step is the refinement around the best position obtained until now. the refinement tries to find a better block match by checking a smaller region. The refinement uses the same search pattern as in the initial search but updates the center of the search area to the best result obtained in its last iteration. Thus, the maximum number of search points is not fixed. If no block with smaller cost is found after two expansion levels, the refinement is finished. A simplified flowchart for TZS algorithm is illustrated in fig. 1.12.

As we can anticipate, the number of iterations depends on different parameters such as the selected initial search center and pixel values. Thus, different number of iterations are performed for different PUs. For instance, a spatial area of a frame with complex motion will require more iterations than a relatively still area. However, the overall complexity of the algorithm is significantly less than fullsearch.

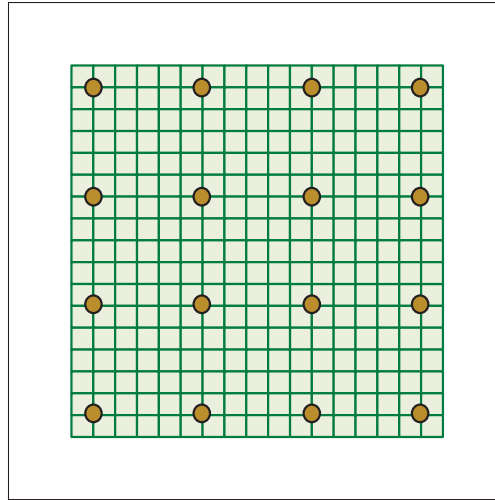


Figure 1.11 Raster Scan with iRaster equals to 5

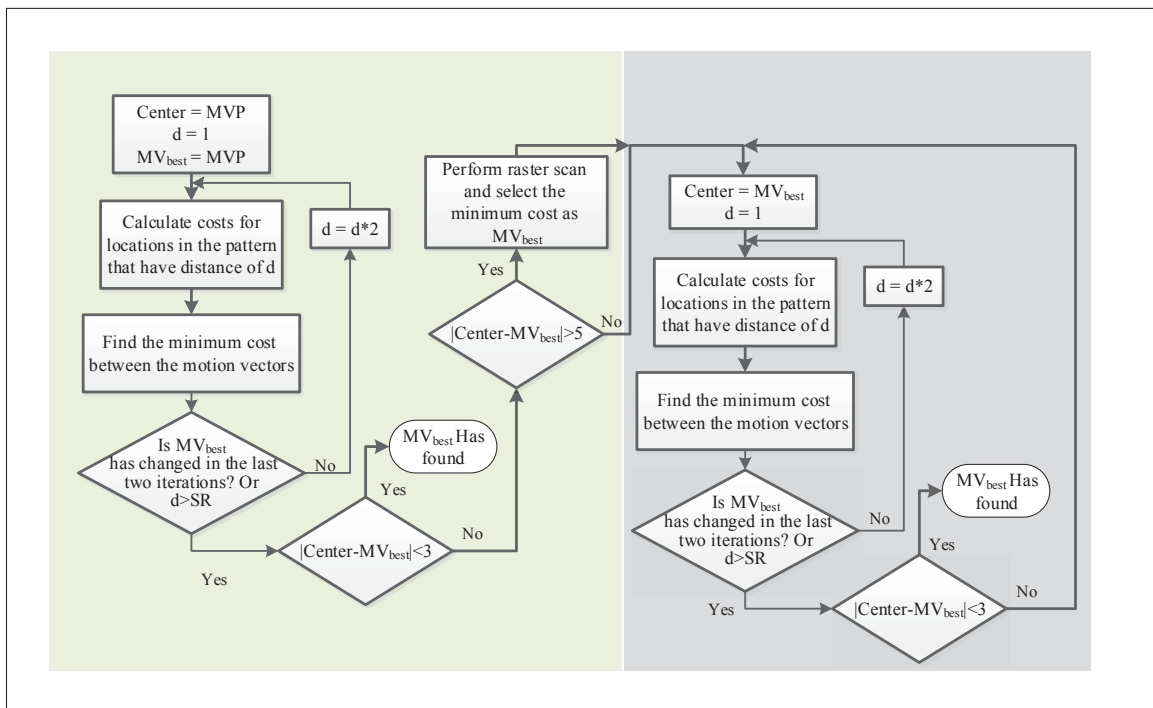


Figure 1.12 TZS algorithm flowchart

### 1.3 Rate-distortion optimization (RDO)

In this section, we present the formulation for HEVC RDO. The RDO has to find the best compromise between low reconstructed distortion and low signaling rate among all possible choices. The frame consists of CTUs; therefore, the encoder selects the best possible parameters for each CTU to minimize the total cost of the frame calculated as:

$$J_{\psi} = \min_{p_i \in P_{CTU}, \forall i \in 1..N_{CTU}} \left\{ \sum_{i=1}^{N_{CTU}} J_{CTU_i}(p_i) \right\} \quad (1.7)$$

Where  $J_{\psi}$  is the minimum cost of the frame,  $N_{CTU}$  is the number of CTUs in the frame and  $P_{CTU}$  is the set of all possible parameters of the CTU. Parameters are such as frame partitioning, quadrees and prediction modes for CTUs. Moreover,  $J_{CTU_i}$  is the cost of the  $i^{th}$  CTU of the frame when CTUs are numbered in raster scan order. The parameter set  $P_{CTU}$  is very large, thus, solving this optimization problem is very intensive and almost impossible due to numerous possible coding parameter combinations.

In order to solve this problem, the RDO is performed for each CTU instead of the whole frame. Since the parameters of one CTU affect the cost of succeeding CTUs, the parameters of previously coded CTUs are required for optimizing the cost of current CTU. The minimum cost of a frame can be then calculated by the collective costs of frame CTUs as:

$$J_{\psi} \approx \sum_{i=1}^{N_{CTU}} \min_{p_i \in P_{CTU}} \{J_{CTU_i}(p_i) \mid p_1, p_2, \dots, p_{i-1}\} \quad (1.8)$$

According to Eq. (1.8), we can calculate the minimum cost of the frame by calculating minimum cost of each CTU given the parameters of previous CTUs.

Furthermore, the cost of the frame's  $i^{th}$  CTU is obtained by summing the cost of its CUs. As we illustrated previously in fig. 1.4, each CU can be split into four smaller sub-coding units (Sub-CUs). Therefore, each CU can be considered as the root for the smaller Sub-CUs and for each Sub-CU this assumption can be repeated. Thus, the calculation is performed using a

recursive function starting with the smallest leaf of the quadtree. At each depth, we have to decide if splitting into smaller Sub-CUs results lower cost or not.

For a CU leaf in the CTU quadtree, no further CU splitting is possible. The cost of a non-splitting CU is denoted by  $J'_{CU}$  and is calculated by Eq. (1.9).

$$J'_{CU}(d, p_d) = D(d, p_d) + \lambda \cdot R(d, p_d), \quad d = 0, \dots, \text{Depth}_{\max} - 1, p_d \in P_{CU} \quad (1.9)$$

where,  $d$  is the depth and  $p_d$  is parameters at depth  $d$ .  $P_{CU}$  is the set of possible parameters for a CU and will be discussed shortly. The Lagrangian weighting factor  $\lambda$  in Eq. (1.9) is determined by the type of distortion measure (D) used and the quantization parameter (QP). Moreover, R is the required rate to reconstruct the complete CU. The value of  $R(d, p_d)$  is the bitrate of the CU if it is coded using  $p_d$ .

Additionally,  $J_{CU}(d, p_d)$  is a function for the minimum CU cost in the depth  $d$  of quadtree structure. For each depth, a cost from the sum of splitting Sub-CUs and a cost from non-splitting CU are calculated. The minimum of these two costs is considered as the minimum cost for the CU at that depth. This determines if a CU should be split to Sub-CU or not. Thus, for each CU depth, the cost is calculated as follows:

$$J_{CU}(d, p_d) = \min_{p_d, p_{d+1,i} \in P_{CU}, \forall i \in 1..4, d=0, \dots, \text{Depth}_{\max}-1} \left\{ J'_{CU}(d, p_d), \sum_{i=1}^4 J_{CU(i)}(d+1, p_{d+1,i}) \right\} \quad (1.10)$$

where the  $J_{CU(i)}$  denotes the  $i^{th}$  Sub-CU and  $p_{d+1,i}$  is its parameters. Consequently, the problem has been simplified to finding the minimum cost of CUs in all depths. Furthermore, to calculate the cost of the quadtree ( $J_{CTU}$ ) in Eq. (1.8), we calculate the cost of a CU with the depth 0 as the root of the CTU.

As we mentioned earlier,  $P_{CU}$  set has all the possible parameters that a CU can have. This set of possible prediction modes is defined by the HEVC standard. In addition, this parameter set may change by user request (e.g. disabling asymmetric prediction modes in the inter-prediction). The complete set of  $P_{CU}$  has the following elements:

$$\begin{aligned}
 P_{CU} &= \{P_{Intra}, P_{Inter}\}, \\
 P_{Inter} &= \{P_{2N \times 2N}, P_{2N \times N}, P_{N \times 2N}, P_{N \times N}, P_{2N \times uN}, P_{2N \times nD}, P_{nL \times 2N}, P_{nR \times 2N}\}, \\
 P_{Intra} &= \{P_{2N \times 2N}, P_{4 \times 4}\}
 \end{aligned} \tag{1.11}$$

Intra-prediction modes are outside the scope of this project and are not discussed further. Considering the inter-prediction modes depicted in fig. 1.6; it is clear that the prediction unit consists of prediction partition blocks. For each partition, the best prediction is found by the RCME. When all of the elements in Eq. (1.11) are calculated, the best prediction mode is determined by selecting the parameter with the lowest cost.

It is recognized that to satisfy the dependencies in each CTU, the RDO process has to start from smaller CU sizes in z-scan order. The order of RDO validation in a CTU is illustrated in fig. 1.13. In other words, the validation of a CU cost is not possible until the validation is performed on its Sub-CUs. There is an exception for CUs in maximum depth since they are not allowed to split further.

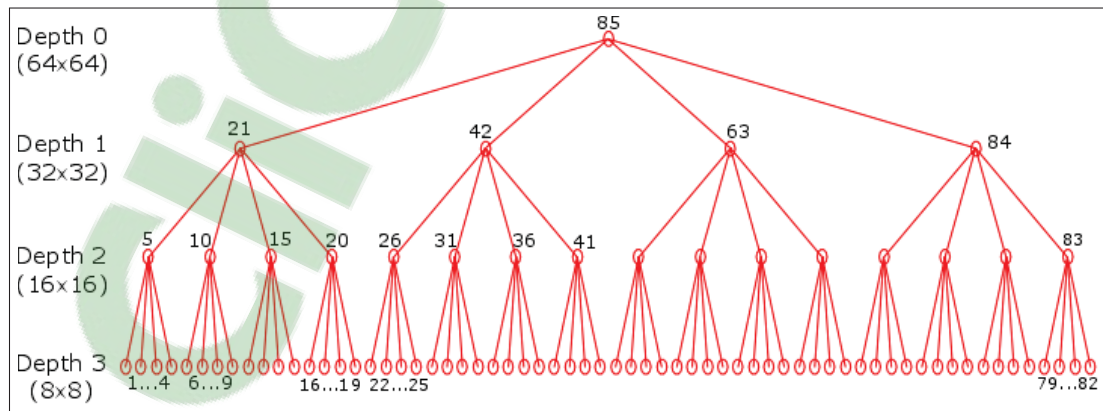


Figure 1.13 RDO processing order in CTU

For more clarity, the RDO process of one CU is depicted in fig. 1.14. The calculations of Sub-CUs are performed if the depth is less than the maximum depth. The inter-prediction is performed for elements of  $P_{\text{Inter}}$  set and the prediction parameter with the minimum cost is selected as the best inter-prediction mode. As well, the best intra-prediction mode is selected between the  $P_{\text{Intra}}$  set elements. Moreover, the cost of the Sub-CUs is already calculated since the same RDO process is performed for Sub-CUs recursively. Lastly, the best prediction parameters of the CU at this depth is selected between the possible modes and a split/non-split decision is made.

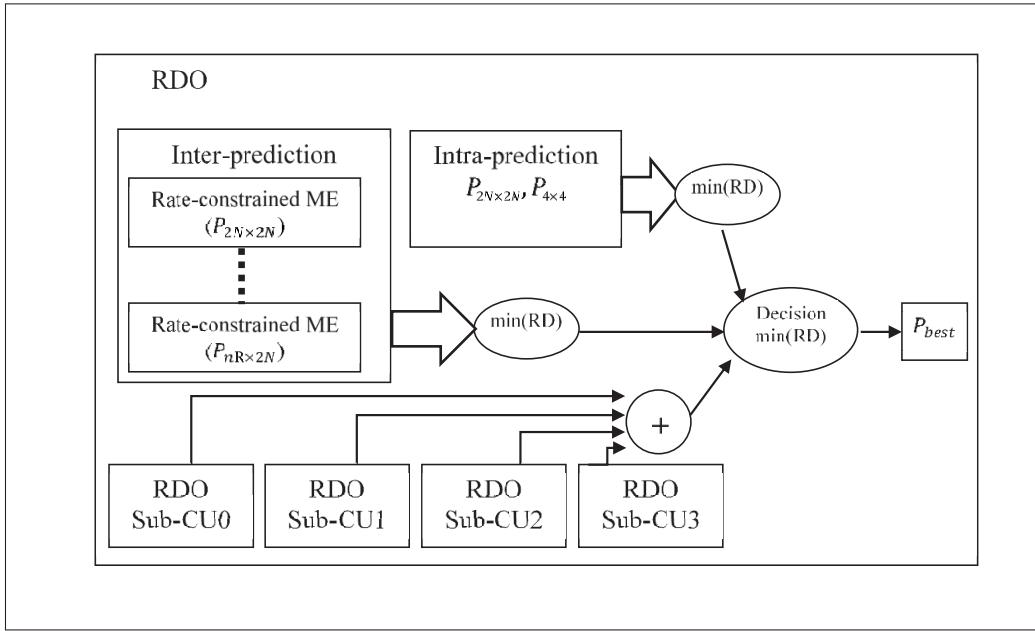


Figure 1.14 RDO block diagram for a CU and possible Sub-CUs

Since the RDO process is not explicitly specified by the HEVC standard, the implementation can use different approaches to make a different trade-off between the computational complexity and the RD performance. For instance, the encoder can skip the calculation of some modes to reduce the computational complexity but at a cost of reduced RD performance.

## 1.4 Parallel tools in HEVC

As mentioned before, the HEVC standard has several high-level parallel tools that facilitate parallel processing. Parallel methods from coarse to fine grain degree are as follows.

- **Group of pictures (GOP) parallelization:**

In this level of parallelization, several GOPs are processed in parallel. In video coding, a GOP structure, specifies the order in which intra- and inter-frames are organized. The GOP is a collection of successive frames. By definition, the decoder should be enabled to decode each GOP separately, thus, there is no dependency between GOPs. This allows the encoding of GOPs concurrently. This provides a degree of parallelism equal to number of GOPs which are processed in parallel. For instance, to encode 40 frames in GOPs with 4 frame in each GOP the degree of parallelism is 10. However, for this kind of parallelization the GOPs' frames should exist before encoding. As a result, it is not appropriate for low-delay encoding. Furthermore, the encoding complexity of GOPs are not the same and it may cause an unbalanced core workload. An example of GOP parallelization is illustrated in fig. 1.15.

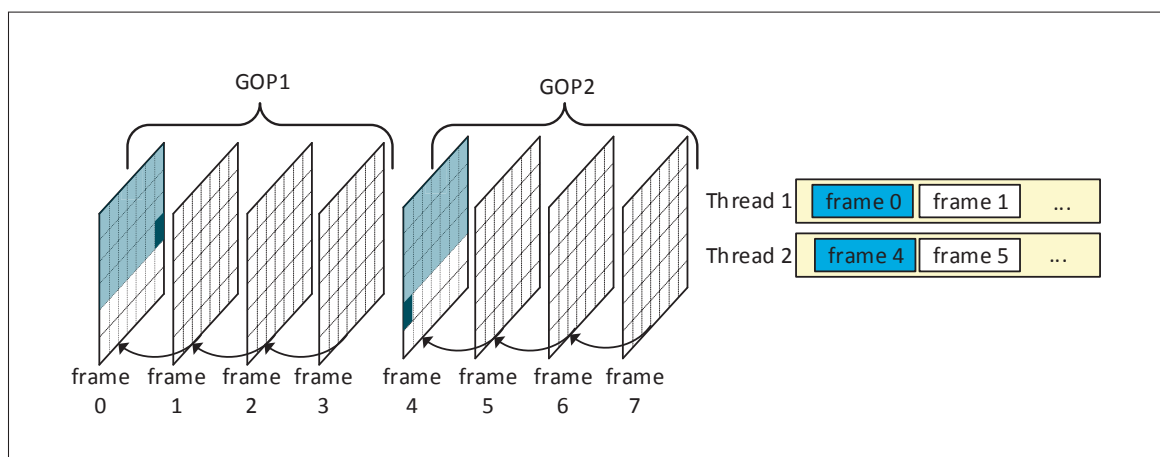


Figure 1.15 GOP parallelization

- **Frame parallelization:**

The frame parallelism is the processing of a single GOP frames. The I-frames can be encoded

independently because the encoder is allowed to only use intra-prediction in them. However, the P- and B-frames use both inter- and intra-prediction. Thus, these frames are dependent on their reference frames. Thus, the frames that are used as reference should be encoded before. This will limit the parallelization scalability and it is limited by the number of concurrent pictures that can be processed in parallel. Thus, the GOP structure and availability of the reference frames has an impact on this type of parallelization. Moreover, the picture level parallelism increases the encoding latency. In fig. 1.16 an example for frame parallelization of a GOP with the IPBBP frame structure is illustrated. The dependencies between frames are depicted by the arrows. The degree of parallelism (DOP) in frame parallelization is:

$$\text{DOP} = N_{\text{Frames}} \quad (1.12)$$

where,  $N_{\text{Frames}}$  is the number of frames that are available and processed in parallel.

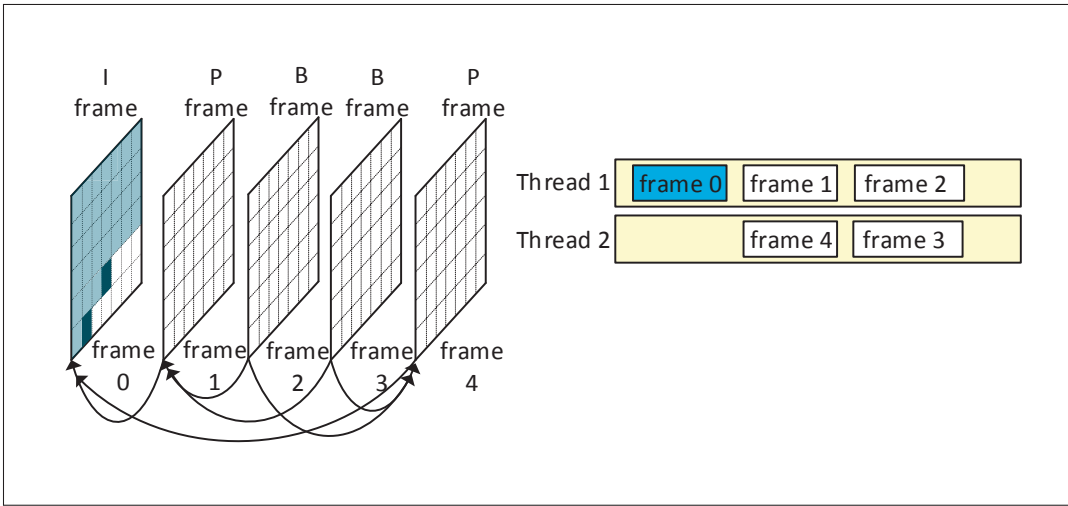


Figure 1.16 Frame parallelization

- **Tiles parallelization:**

As it is illustrated in fig. 1.3, tiles are used to divide a picture horizontally and vertically into multiple sub-pictures. Prediction dependencies are broken at tile boundaries and tiles can be processed in parallel. In addition, entropy coding and reconstruction dependencies are not



allowed across a tile boundary, the only exception is in-loop filtering that is permitted through the boundaries. Tiles usually provide higher coding efficiency relative to slices since spatial distances in tiles is reduced which leads to a better spatial correlation between samples inside a tile. However, the coding efficiency loss typically increases with the number of tiles, due to the breaking of dependencies and resetting CABAC context at the beginning of each tile. The degree of parallelism for tiles is:

$$\text{DOP} = N_{\text{Tiles}} \quad (1.13)$$

where,  $N_{\text{Tiles}}$  is the number of tiles in the frame that are independently processed in parallel.

- **Slice parallelization:**

As definition, a slice in a picture is an independently decodable part and the only potential dependency is in-loop filtering across the edge of slices. However, slices can be used for parallel processing. The use of multiple slices, however, reduces the coding efficiency significantly due to the restrictions of prediction and entropy coding outside slice borders. In addition, the overhead of a slice is not negligible and using several slices will reduce the RD performance of the encoder. Thus, slice-level parallelism is only beneficial with a small number of slices per picture.

- **Block level parallelization:**

Block-Level parallelization is fine grained. This technique is implemented in a pipeline manner where one core is dedicated for prediction, one for entropy coding and so on. In this way, blocks will be coded concurrently on different cores. The main difficulty of this technique is imbalanced tasks, which requires an elaborate scheduling. The HEVC specification includes a special tool called wavefront parallel processing (WPP) in which each CTU row of a picture is assumed as a separate partition. In WPP mode, each CTU row is processed relative to its above CTU row with a delay of two CTUs. In this way, no dependency between following CTU rows are broken at the partition boundaries except the CABAC context variables at the end of the row. To alleviate the RD performance loss because of CABAC initialization at the start of each CTU row, the content of the adapted CABAC context variables are propagated.

The propagation is from the second encoded CTU of the above CTU row to the first CTU of the current row. Therefore, performance loss of WPP is negligible (Chi *et al.*, 2012b). Using WPP, parallel threads up to the number of CTU rows in a picture can work simultaneously. Therefore, the degree of parallelism of WPP is limited to the height of frame and CTU.

$$\text{DOP} = \frac{\text{Height}_{\text{Frame}}}{\text{Height}_{\text{CTU}}} \quad (1.14)$$

The main drawbacks of WPP are the limitation on the number of parallel threads and the unbalanced processing time for different rows. WPP is illustrated in fig. 1.17.

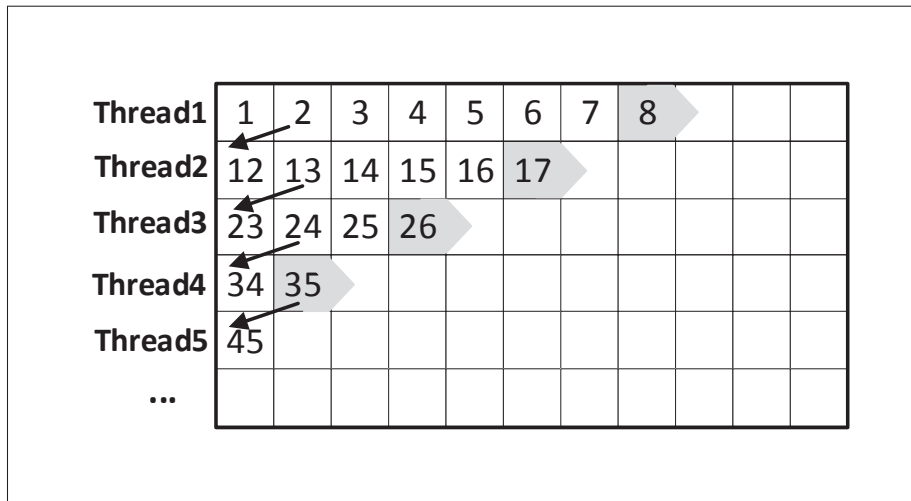


Figure 1.17 Wavefront parallel processing. Grayed blocks are processed in parallel

Slice parallelization in HEVC is not considered an efficient parallelization tool compared to tiles and WPP. The drawbacks and advantages of WPP and Tiles tools are summarized in table 1.2.

Table 1.2 HEVC high-level parallel tools comparison

<b>WPP</b>	<b>Advantages</b>	Satisfactory RD performance. No boundary artifacts.
	<b>Drawbacks</b>	Low workload for start and end of the frame (ramp-up and ramp-down). Unbalanced workload and stall of rows below. High synchronization overhead between rows. Medium scalability. Decoder support is required.
<b>Tiles</b>	<b>Advantages</b>	Number of partitions can be freely chosen (However, limited by the RD penalty). No synchronization between tiles required. Region of interest can be defined. No boundary artifacts.
	<b>Drawbacks</b>	Lower RD performance (entropy and prediction dependencies inhibited outside the tile). Boundary artifacts. Medium scalability. Decoder support is required.

## 1.5 Summarize

In this chapter, we explained the fundamentals of HEVC. The coding process of HEVC is based on removing redundant information in several stages. We explained how the frame is partitioned into smaller blocks. The PU is the smallest building block that is used to predict a region of the frame based on an already encoded part of the image. In inter-prediction, the best prediction is achieved by RCME. Furthermore, the RDO process determines the best possible parameters. Moreover, the existing parallel tools in HEVC were discussed. Table 1.2 summarized existing tools with their drawbacks and advantages. In this chapter, we provided also an overview of the HEVC standard to explain the parts related to our project. More detailed description of the HEVC standard is available in (Sullivan *et al.*, 2012), (Sze *et al.*, 2014) and (Wien, 2015). In Chapter 2, we focus on parallelization of HEVC and we present the state-of-the-art approaches on this topic.



## CHAPTER 2

### LITERATURE REVIEW ON HEVC PARALLEL PROCESSING METHODS

In this chapter, we review the most important previous works on parallelization of video coding standards. Although we consider different parts of HEVC, our focus is on parallel motion estimation in HEVC publications. Since the principles of the RCME in H.264 and HEVC are similar, we also consider the ideas for the previous video coding standard that are applicable to HEVC. We can classify the parallel HEVC publications into different groups according to their objective or the means of achieving that goal. In fig. 2.1, we categorize papers related to parallel processing HEVC encoder based on the papers' objectives. In this chapter, the coarse-grained and fine-grained parallelization literature is investigated briefly. Parallel rate-constrained motion estimation falls into the fine-grained category and is mentioned separately since it is the basis of our project. Furthermore, in the discussion section, the shortcomings and drawbacks of existing state-of-the-art methods are discussed. In the next chapter, we introduce methods to improve them.

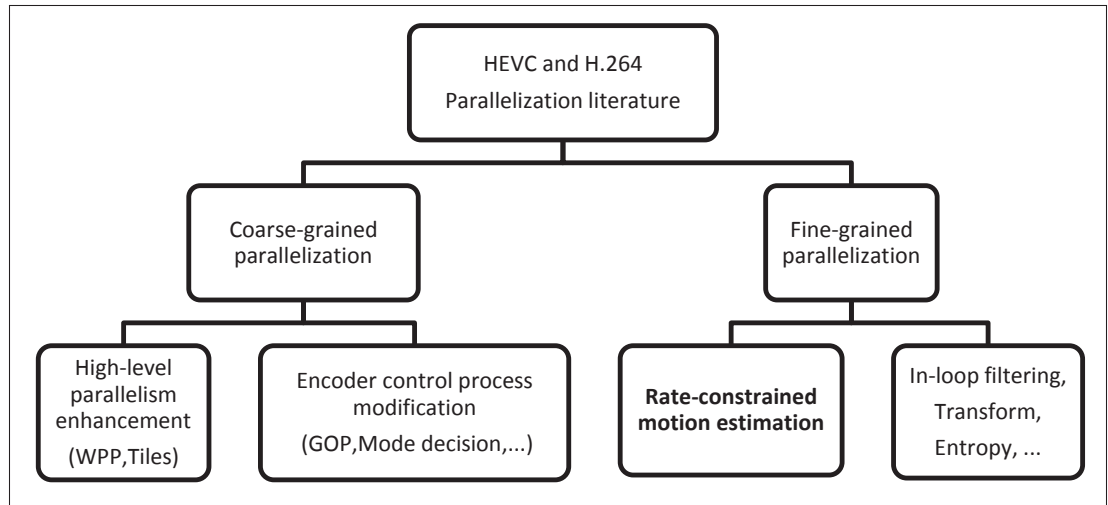


Figure 2.1 Parallel video coding literature categorization

## 2.1 Coarse-grained parallelization

In the approaches of this group, the main goal is to achieve speedup in the encoder by improving HEVC high-level parallel tools like wavefront and tiles. In (Chi *et al.*, 2012a), a more efficient wavefront algorithm called overlapped wavefront (OWF) that improves the performance of WPP is presented. The authors focused on the ramp-up and ramp-down inefficiencies in the whole process. In the proposed OWF method, when a thread has finished processing a CTU row in the current picture and no more rows are available, they start processing the next picture instead of waiting for the current picture to finish. The OWF puts a boundary on the maximum vertical CTU search range in the RCME stage to ensure that its whole reference area is available. In fig. 2.2, OWF is illustrated. In this figure, the gray rectangle shows the available reference region for the next frame. This region is the only available search area for the next frame's CTUs. To evaluate the results, they implement a parallel HEVC decoder that supports multiple parallelization strategies. Using OWF on a parallel platform with 12 cores (2 sockets 6 core each) performs 28% faster than WPP. The same researchers, in (Chi *et al.*, 2012b), implemented OWF on a decoder using TILE-Gx36 platform and achieved 116 fps 4K real-time decoding.

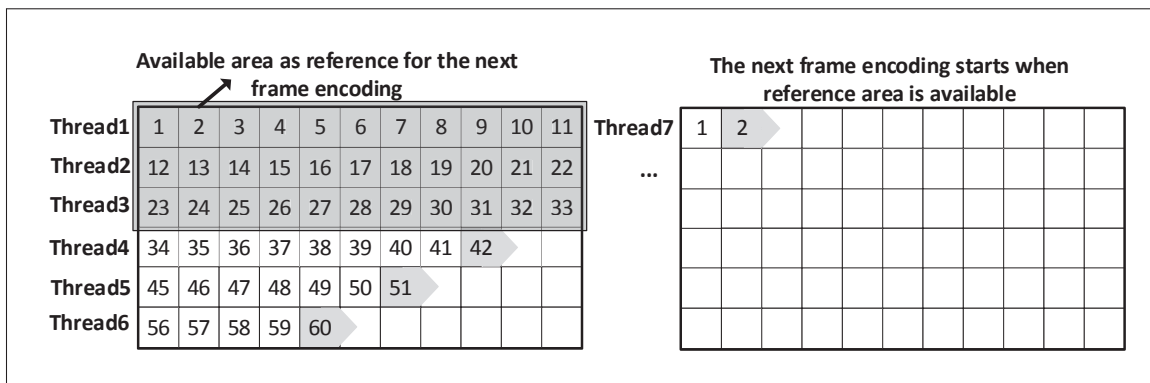


Figure 2.2 Overlapped wavefront. Adapted from (Chi *et al.*, 2012a)

In (Chen *et al.*, 2014), the authors proposed to use of OWF in a group of pictures (GOP) in order to achieve higher speedup. They named this Inter-Frame Wavefront (IWF). In this paper, they analyzed the WPP and came to the conclusion that there is no possible way to improve parallelization of intra-prediction using a WPP-based algorithm, but inter-prediction is more flexible for that goal. They used an IBBP coding structure as illustrated in fig. 2.3. In addition, they limit the search of vertical motion vectors by four CTUs which caused a small 0.1% Bjøntegaard-Delta Rate (BD-Rate) loss. The results are compared to a serial implementation by the authors, and there is no comparison with other works.

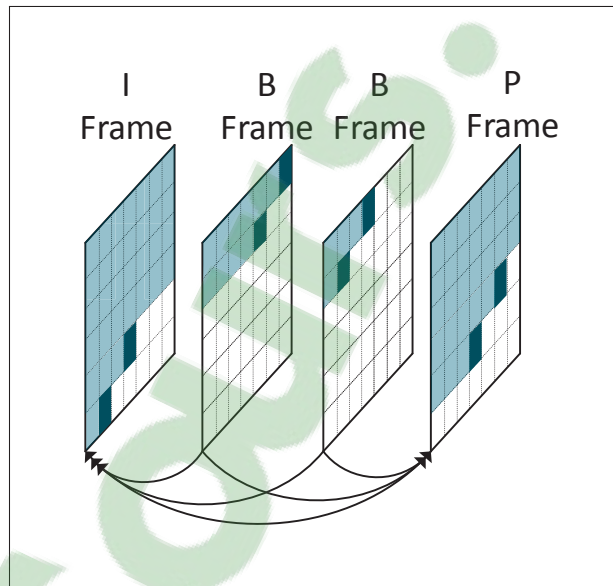


Figure 2.3 Inter-frame wavefront.  
Adapted from (Chen *et al.*, 2014)

Dong *et al.* proposed a Dynamic Macroblock-level Scheduling (DMS) instead of a conventional row based scheduling in WPP (Dong *et al.*, 2013). The main motivation for their work is to reduce the idle threads in a row based processing. Also, penalties for excessive number of threads and synchronization between them is high. By using the DMS scheme they reduce both the unbalance and synchronization delay among multiple threads. In DMS, the basic scheduling unit is a MB and each thread can select any available MB to encode it. After a

thread has finished processing a MB, if the MB on the right of it is available it has the highest priority than any other available MB otherwise it will encode another MB on the row below. The results show a 9% speedup compared to the row-based processing.

In (Zhang *et al.*, 2014), the authors implemented a parallel HEVC encoder on a Tilera platform. They improved memory localities based on the Tilera hardware. In addition, they reduced the CTU size to  $16 \times 16$  and  $32 \times 32$  to alleviate the WPP scalability issues but this caused up to 15% BD-Rate penalty for  $16 \times 16$  CTU size. Furthermore, like IWF they used frame-level parallelization to increase speed. Although they mentioned this implementation has more degree of parallelization than simple WPP, they did not indicate the overall quality loss. However, it can be inferred that it leads to more than 15% BD-Rate loss, which is completely unacceptable for most applications.

In order to improve tiles' performance, Blumenberg *et al.* proposed an algorithm which is performed in two stages to define the vertical and horizontal tile boundaries in order to group the highly correlated samples into the same tile partition (Blumenberg *et al.*, 2013). The first stage acquires the coordinates of the vertical boundaries, while the second one acquires the horizontal boundaries' coordinates. The variance of each CTU must be calculated and results in a picture variance map. Based on this map, the tile boundaries are determined. Their results show that the proposed algorithm is able to reduce the inherent coding efficiency losses of using tiles when compared to the conventional uniformly spaced tile partitions. The improvement is about 0.2% BD-Rate for this algorithm compared to uniform tiles.

In (Shafique *et al.*, 2014), output quality and efficient core allocation is considered to determine tiles size and position. In this method the position and size of each tile can be different. The tile formation and mapping (TFM) technique first combines the neighboring samples at the corners of the frame to generate a so-called master tile. Afterwards, it derives so-called secondary tiles from the master tile to cover the complete video frame and assign them to cores for processing. This will determine the number of encoding cores required for sustaining the HEVC's encoding workload such that the number of cores and power consumption is minimized.



Some of the previous works introduce a parallel framework for the whole encoder process to improve parallel processing. In (Heng *et al.*, 2014) an HEVC UHD real-time encoder using three workstations and a high-speed network is introduced. The encoding process is optimized with data parallelism and uses GPU for motion estimation. The method utilizes temporal parallelism for GOP and spatial parallelism of pictures through slicing. It consists of four elements: image analyzer, encoders, bitstream controller and a network switch. There is one PC for the image analyzing and bitstream control, two PCs (32 cores each) for encoding. The purpose of the image analyzer is to handle a 4K video input and perform motion estimation on them by GPU. Each encoder performs the encoding of a GOP. With multiple encoders processing different GOPs (32 frames), temporal parallelization is realized. Using slices, all dependencies are broken and slices can be processed in parallel. Results show about 15794x speedup compared to HEVC test model HM 11.0 and 13x compared to x265 with a PSNR loss of 0.49 dB and 0.03 dB respectively.

In (Ahn *et al.*, 2013) a slice/tile load balancing algorithm based on a picture complexity model is introduced. A model based on analyzing the standard sequences that estimates the complexity of inter/intra/merge depending on size of block is built. Using this model, the encoder can regulate size of the slice or tile according to complexity of CTUs inside that part. The average speedup of this method is 12% for the slice and 3.8% for the tiles with around 2% BD-Rate degradation.

In (Koziri *et al.*, 2016), the authors focused on the problem of proper slice sizing to reduce load imbalances among threads. Using existing ideas for H.264/AVC they have developed a fast dynamic approach to decide on load distribution in the HEVC. They developed a heuristic called TSLB (time-based slice load balancer) which assigns load based on the time complexity of the previous frame. They achieved an improvement specially for the case of Low-Delay by exploiting GOP structure. Based on their experimental results, the load imbalance is reduced 10% in many cases.

In (Łuczak *et al.*, 2012), a modified CTU processing order is proposed. By processing the CTU in a diamond group of MBs, they achieved a higher degree of parallelization. The idea is to modify the process starting point. Instead of starting the process from the upper left corner, it starts from the center of the image and grows in a diamond shape. Since all MBs have already-coded neighbors, each group of MBs can be processed at the same time. This algorithm grows faster than WPP and allows better utilization of the multi-core hardware. Here we mention that although this method yields about 4x parallel processing speedup, it requires extensive changes to the standard and it is not HEVC-compliant. In addition, other features like entropy coding, slices and tiles are not defined in this work.

## **2.2 Fine-grained parallelization**

According to the previous section, coarse-grain parallelization methods are covering the CTU row level and frame level parallelization of HEVC for all encoding steps. However, fine-grained parallelization can be achieved when the parallel processing is performed at the CTU level or on smaller blocks. Moreover, we divided the literature based on the process that is performed on the block. The rate-constrained motion estimation process is discussed separately from the other processes.

### **2.2.1 Intra-prediction, in-loop filter, transformation, entropy coder**

In this section, we present some important parallel methods in the literature for intra-prediction, in-loop filtering, transformation and entropy coder.

In (Zhao *et al.*, 2013), fine granularity parallelism for HEVC intra coding is realized and the degree of parallelism is increased to the coding unit level. The authors used a directed acyclic graph (DAG) to visualize the dependency relationship and parallel execution. Using, DAG a model is built and dependencies are preserved. Similarly, in (Yan *et al.*, 2014a) an intra-prediction parallelization framework is proposed. In the proposed framework, the encoding

starts by running parallel threads to perform intra-prediction, with each thread processing one CTU row. Another thread starts entropy coding and processes CTUs in raster order.

Meher *et al.* introduced an enhanced architecture for the implementation of integer discrete cosine transform (DCT) of different lengths (Meher *et al.*, 2014). The authors show that an efficient constant matrix multiplication scheme can be used to derive parallel architectures for 1-D integer DCT. Also, the proposed structure can be reusable for DCTs of lengths 4, 8, 16, and 32 with a throughput of 32 DCT coefficients per cycle irrespective of the transform size. The results show that the proposed architecture involves nearly 14% less area-delay product (ADP) compared to the direct implementation of the reference algorithm.

In (de Souza *et al.*, 2014), a method is proposed for the utilization of the maximum level of parallelism in an HEVC deblocking filter. A highly optimized CPU parallel implementation and a GPU implementation of the HEVC deblocking filter is the basis of their work. Also, load-balancing between CPU and GPU is considered. Their main work is to optimize existing techniques using SIMD and prevent diverging branches in the GPU implementation. They achieve 9 to 17 times speedup relative to sequential implementation of deblocking filter.

Since the CABAC process is highly sequential, almost all of the papers on this part are based on logic level hardware implementations like FPGA. For instance, in (Zhou *et al.*, 2013) the throughput of CABAC is increased by 31%~34% with the proposed pre-normalization, hybrid path coverage (HPC), bypass bin splitting (BPBS) and state dual-transition (SDT) schemes. For evaluation, they implement this CABAC entropy coder on silicon in a 65 nm video encoder chip.

### **2.2.2 Parallel rate-constrained motion estimation**

Since motion estimation (ME) is used in several video processing fields, there are many works on the ME search algorithms. Furthermore, the main goal of these papers is to find the appropriate matching block in a reference frame in order to find real motion vectors. However, in video compression, the main goal is to find the motion vector with the least rate-distortion

cost using RCME. Here, we review the parallel rate-constrained ME publications in the video compression field.

Fast block motion estimation algorithms such as the Enhanced Predictive Zonal Search (EPZS) (Tourapis, 2002) and the UMHexagonS (Chen *et al.*, 2006) are not suitable for parallel ME (Cheung *et al.*, 2009). These algorithms employ spatiotemporal correlation by using neighboring blocks in order to predict MVs; they thus increase data dependencies in ME. As a result, fullsearch block matching is the most straightforward algorithm that is used for parallel ME in the literature.

As an early attempt to perform parallel ME for heterogeneous architectures, Lee *et al.* proposed a multi-pass method to unroll and rearrange the multiple nested loops (Lee *et al.*, 2007). They perform the integer ME with a two-pass process on the GPU. Furthermore, Chen & Hang proposed a five-step algorithm to perform parallel ME on heterogeneous architectures for the H.264 standard (Chen & Hang, 2008). This algorithm calculates motion vectors of the blocks of an entire frame in parallel. The main advantage of this method is that the calculation of the distortion values are computed in a bottom-up manner. Therefore, the distortion of bigger blocks are calculated from smaller blocks. The process is as illustrated in fig. 2.4 and is as follows:

- 1- Divide the MB into 4x4 blocks and calculate the SAD value of each 4x4 block in parallel for all motion vectors inside the search range.
- 2- Merge these 4x4 block SADs to form the 4x8, 8x4, 8x8, 8x16, 16x8, and 16x16 block SADs, respectively.
- 3- For each block size, determine the minimum SAD of all MVs. Select it as the integer-pixel motion vector (IMV).
- 4- Calculate SAD of fractional pixel motion vectors, near the best IMV.
- 5- Determine the final MV value according to fractional-pixel motion vector (FMV).

The five-step algorithm is used as the base algorithm by other researches. However, Chen & Hang ignored the effect of MVP on the cost of Eq. (1.3) and their proposed algorithm

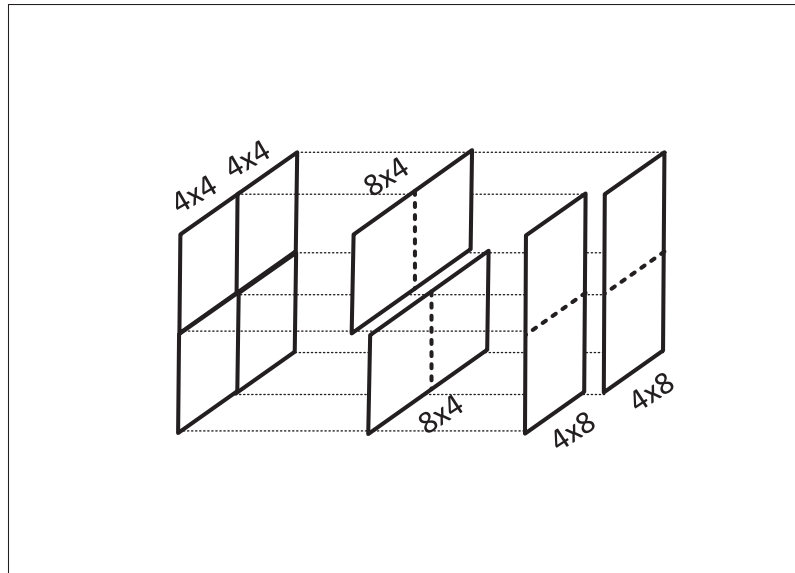


Figure 2.4 Variable block bottom-up SAD calculation

selects the best motion vector only based on distortion. As mentioned afterwards by Rodríguez-Sánchez *et al.* and Momcilovic *et al.*, this algorithm is not suitable for real applications unless serious improvement is applied (Rodríguez-Sánchez *et al.*, 2012; Momcilovic *et al.*, 2014b). Therefore, spatial and temporal information is used to predict the MVP in most of the recent works.

In Xue *et al.* (2017), a method named multilevel resolution motion estimation (MLRME) is introduced. In this method, by combining the advantages of local fullsearch and downsampling they have reduced the complexity of the fullsearch algorithm. In the search method, they used zero as motion vector predictor. While their implementation is faster than fullsearch, they have reported 10% increase in execution time compared to fast search. They mentioned two reasons for this slowdown. First, complexity of the MLRME method is higher than the fast search. Second, the heuristic search and early termination condition in the fast search help reducing the search time. Furthermore, the experimental results showed 1.5% increase in BD-Rate.

In (Yu *et al.*, 2012) and (Yan *et al.*, 2014b), the authors introduced methods to infer the MVPs of PUs inside a CTU. Since there is a dependency on spatial information, these works provide

a method to perform parallelization at the PU level just for a CTU. Thus, the processing of CTUs is performed in normal sequential order. Therefore, the parallelization degree of these methods is limited.

However, temporal information can be used without adding any restriction to the algorithm. In (Momcilovic *et al.*, 2014a), using the zero MVP as predictor is investigated for H.264. They show that by selecting the most probable predictor, performance can improve significantly compared to (Chen & Hang, 2008). Furthermore, by investigation the co-located MV and zero MVP for different search ranges they improve their proposed method.

Furthermore, using the previous frame motion vectors to predict MVPs of the current frame has been investigated in (Rodríguez-Sánchez *et al.*, 2012) and (Momcilovic *et al.*, 2014b) for the H.264 standard. Rodríguez-Sánchez *et al.* improved the previous method by using co-located block motion vector as MVP of the current block. Further improvements on using co-located motion information is achieved in (Yi *et al.*, 2010), (Gao & Zhou, 2014) and (Ma *et al.*, 2014). They proposed co-located MB based motion estimation (CMME) and motion vector extrapolation based approach (MVEA) respectively for the H.264 standard. In CMME, motion information of the previous frame's MB at the same coordinates is used to estimate the MVP. This improves the PSNR by 0.8 dB compared to (Chen & Hang, 2008). Yi *et al.*, estimated the MVP using overlapping MVs from previous frame (Yi *et al.*, 2010). They achieve slight RD performance improvement compared to their previous work using MVP zero. In (Ma *et al.*, 2014), a similar method is implemented by using the average of co-located motion vectors of the previous frame as MVP of current PU. However, adapting these algorithms for HEVC is challenging since the size of coding blocks in the HEVC is variable while in H.264 the MB size is fixed. Furthermore, the motion vector prediction derivation is different in HEVC and may have different effects on RD performance.

In (Wang *et al.*, 2013), RCME is divided into two stages. First, they calculate the variable block PU sizes and store 3 best MVs based on SADs values. Next, the CPU performs mode decision, RDO process, reconstructs the image, and copies the reconstructed image into the

GPU for deblocking filter and interpolation for the next frame. In each step, one CTU row of data is processed. This allows MVP derivation from top row.

Similarly, Wang *et al.*, divided the RCME process into two steps. The first step is done before the encoding of frame starts and the calculations in this step are done in GPU. In the second step, the mode decision is performed in a sequential manner in the CPU (Wang *et al.*, 2014). The GPU acts as a co-processor and pre-calculates the values that are required by the CPU in the RDO process. Their method is illustrated in fig. 2.5. Like (Momcilovic *et al.*, 2014a), fractional values are pre-calculated based on the zero MVP. However, SATD values are not calculated by the GPU and CPU will calculate interpolated values if required. This framework results in better RD performance compared to co-located methods. Similarly, in (Radicke *et al.*, 2014), the authors achieved an improved RD performance. In this work, a wavefront architecture is utilized to improve speed. However, the authors mention 20% of the time is wasted on the stall for transferring large amount of SAD value between CPU and GPU. In Khemiri *et al.* (2017), the authors optimized SAD calculations on the GPU. Furthermore, instead of using a sparse pattern like (Radicke *et al.*, 2014), they used rectangular area like fullsearch. However, even by performing several optimizations in their implementation, they achieved only 23% time reduction with 0.4% BD-Rate increase.

A similar method with more GPU optimization is proposed in (Lin & Wu, 2016). The main improvement is due to their design considering the GPU architecture. By using more fine-grained kernels in the GPU, they have modified the proposed method of Chen & Hang to have better GPU utilization. In addition, they use a method to determine the CU splitting in the GPU and transfer the splitting flags to the CPU. This will reduce the complexity of RDO. Their results shows an extra 23% time reduction compared to (Radicke *et al.*, 2014). However, the BD-Rate is increased by 4.6%.

In Kao *et al.* (2016), the authors tried to reduce the amount of data transfer between CPU and GPU by reducing the search window. They propose to use a zero motion vector predictor to find an approximate motion vector. Then, they will transfer the surrounding distortion values

around this motion vector. The CPU has to search on a smaller area and perform the fractional calculation when actual motion vector predictor is available. Their experimental results showed 3.1% increase in BD-Rate and achieved 9x speedup compared to fullsearch.

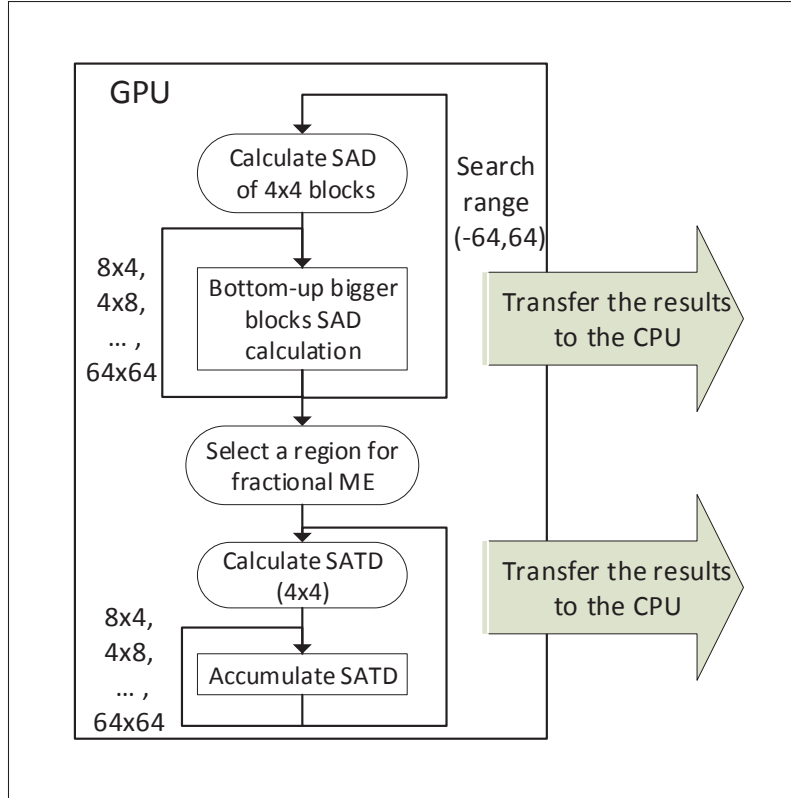


Figure 2.5 Using GPU for distortion pre-calculation.  
Adapted from (Wang *et al.*, 2014)

### 2.3 Discussion

The maximum achievable degree of parallelism is limited to the high-level parallel tools provided by HEVC. Considering the coarse-grained parallelization publications, it is noticeable that these methods improve the parallelization performance of HEVC. However, these methods cannot provide enough parallelization for many-core architectures. These methods are more useful when combined with appropriate fine-grained methods to achieve a high degree of parallelism and preserve RD performance. According to the papers presented in Section 2.2.2,



we can categorize previous works on parallel rate-constrained motion estimation into two categories.

In the first category, the MVP is completely ignored or is predicted. These methods will result in a fine-grained parallelization with a significantly reduced RD performance. In the second category, the GPU is just used to pre-calculate values and no assumption is made for the MVP. These methods can achieve good RD performance, however, the speedup is limited, due to high amount of data transfer between the CPU and the GPU. These problems are discussed in detail in the following sections.

### 2.3.1 Reduced rate-distortion performance

In the first category, the MVP is unknown and the MV cost is completely ignored ((Lee *et al.*, 2007) and (Chen & Hang, 2008)). By removing the MV cost from Eq. (1.3), it becomes:

$$P_{ME} = (mv^*) = \underbrace{\arg \min}_{\forall mv \in MV_{search}} \{D(mv)\} \quad (2.1)$$

Thus, the best MV is selected only based on the distortion. This will result in significant RD performance reduction compared to Eq. (1.3). As it is reported in (Momcilovic *et al.*, 2014b), the BD-Rate of H.264 when ignoring MV cost is increased by 57% which is completely unacceptable.

Moreover, using zero MVP showed significant improvement compared to ignoring the MV cost (Momcilovic *et al.*, 2014a). However, Momcilovic *et al.* mentioned that the performance is highly dependent on the nature of the video sequence. For instance, assuming zero MVP for a motionless sequence achieves high performance while for a sequence with high motions, it will perform very poorly.

Instead of using zero MVP, using temporal information has the advantage of adapting to the sequence motions. More advanced methods tried to predict the MVP based on a function

((Rodríguez-Sánchez *et al.*, 2012), (Momcilovic *et al.*, 2014b), (Yi *et al.*, 2010), (Gao & Zhou, 2014)). The previous frame is used for MVP prediction. Because of consistency in moving objects in a video sequence, using temporal motion information can predict the motion of current frame to some extent. However, in cases where the motion field of the current frame is different or there is no certain consistency between successive frames, these algorithms will fail (Momcilovic *et al.*, 2014b). Furthermore, the efficiency of these algorithms is not investigated for HEVC by other researchers.

Based on the discussion above, we conclude that temporal prediction could be used to obtain high granularity parallelization. Furthermore, a single predictor would not provide good RD performance. Therefore, in Section 3.4 we will propose a novel MTP-RCME method to overcome the aforementioned problems.

### 2.3.2 Bandwidth limitation

In the second category, the GPU is used just for the pre-calculation of the distortion ((Wang *et al.*, 2014), (Wang *et al.*, 2013), (Radicke *et al.*, 2014)). This method can achieve the best RD performance since the parallelization actually would not break any data dependencies. Using the GPU as co-processor will increase the speed. The CPU uses these values and the actual MVP in the RDO step to find the best solution. The main problem of using the GPU in this manner is the high bandwidth required for transferring all the pre-calculated data. For example in (Radicke *et al.*, 2014), the stall of the CPU because of the transferring of the pre-calculated data is 20% of the whole process. Moreover, the memory size that is required for storing the pre-calculated values can be a serious problem when the number of concurrent CTUs is increased. An encoder that uses a method in this category will require a bandwidth computed as follows:

$$\begin{aligned} \text{Bandwidth (bytes/sec)} = & \left\lceil \frac{\text{FrameWidth}}{64} \right\rceil \times \left\lceil \frac{\text{FrameHeight}}{64} \right\rceil \times \\ & N_{\text{PUsInCTU}} \times N_{\text{ReferenceFrames}} \times N_{\text{FramesPerSecond}} \times \\ & (2 \times \text{SearchRange})^2 \times \text{sizeof(Int16)} \end{aligned} \quad (2.2)$$

where,  $\text{FrameWidth}$  is the frame width in pixels,  $\text{FrameHeight}$  is the frame height,  $N_{\text{PUsInCTU}}$  is the total number of PUs in a CTU,  $N_{\text{ReferenceFrames}}$  is the number of reference frames used for inter-prediction,  $N_{\text{FramesPerSecond}}$  is the required number of frames to be encoded in a second and  $\text{SearchRange}$  is the search area used for RCME. Moreover, we are assuming that pre-calculated distortion values are 16-bit integer values. Therefore, assuming encoding a class A video and 30 (fps) with typical encoding parameters:

$$\begin{aligned} \text{FrameWidth} &= 2560, \text{FrameHeight} = 1600, N_{\text{PUsInCTU}} = 425, \\ N_{\text{ReferenceFrames}} &= 4, N_{\text{FramesPerSecond}} = 30, \text{SearchRange} = 32 \end{aligned} \quad (2.3)$$

The required bandwidth to transfer all the calculated distortion values from the GPU to the CPU is 390 GB/s. That is far beyond the maximum theoretical bandwidth of PCI-express(3.0)  $\times 16$  which is 16 GB/s. This means it is not possible to use these methods for real-time encoding of a typical video. Thus, on commodity hardware the bandwidth is the main bottleneck of these methods.

Moreover, to achieve high RD performance in these methods, fractional ME has to be calculated in the second step on the CPU in sequential order. In order to use the GPU for pre-computations if the integer ME and the fractional ME is performed in the GPU it will reduce the RD performance significantly. On the other hand, if to preserve the quality, fractional ME is performed on the CPU, the high computation burden of fractional ME will fall on the CPU. In conclusion, we see that the performance of this algorithm is high as long as the fractional ME is executed on the CPU in sequential order. In Section 3.2 and Section 3.3, our proposed method resolves the aforementioned bandwidth problem, while the RD performance is preserved when an appropriate MVP list is used.



## CHAPTER 3

### PROPOSED PARALLEL ENCODING FRAMEWORK

According to the discussion presented in Section 2.3, several difficulties have to be addressed in order to perform HEVC encoding on a massively parallel architecture. In this chapter, a framework is proposed that resolves these problems.

First, the existing dependencies in HEVC are analyzed in detail to have a better understanding of those dependencies. Using this analysis, a method is introduced which provides fine-grained parallelization of HEVC encoding by performing RCME in two stages. Our method is similar to the proposed by other works. However, we modify the usual two stages approach by considering a multiple predictor rate-constrained motion estimation (MP-RCME) method to reduce the RD loss. In the MP-RCME approach a list of MVP candidates is established for which the optimal MV is computed along with the distortion and rate information. Moreover, we introduce an algorithm with the possibility of performing fractional calculation in the second stage of MP-RCME in the CPU. This method provides a trade-off control between RD performance and speedup that is decided based on the video encoder application. Next, a method is provided to extract the MVP candidates list based on the motion vectors of previous frames. Furthermore, we introduce a nested diamond pattern (NDS) method as a suboptimal motion estimation search method specially designed according to the GPU architecture. Finally, we provide a complete framework using the introduced methods as a fast parallel encoder for massively parallel architectures.

#### 3.1 Dependency analysis

To achieve a better compression, an encoder can reuse the encoded information for each CTU. For instance, to predict a CTU information, several inferences are made based on previously coded CTUs in the left and top of the current CTU. This kind of inferences are not limited to prediction and are performed on several blocks of the encoder.

To analyze the dependencies, we should first determine the granularity of the encoding process. For instance, for frame parallelism, the only dependency for the current encoding frame is the availability of the frame part that is used as a reference frame. For CTU level parallelism, in addition to mentioned dependency, the left, left-top, top and top-right CTUs should be already encoded and their quadtree structure must be determined. The finest granularity is achieved for PU level parallelism. Since we are aiming to provide the highest degree of parallelism, we analyze the dependencies of encoding parallel PUs.

The dependencies that prevent parallel encoding of all PUs of a CTU can be divided into two categories of hard and soft dependencies. We define a dependency as a hard dependency if the process of next step is (almost) impossible until the calculation of previous step is finished. However, a soft dependency can be ignored in order to process the next part in parallel with the current part. Nonetheless, a reduction in the RD performance of the encoded video is anticipated due to ignoring soft dependencies. Based on HEVC structure in Section 1.1, we can determine the type of dependency.

In HEVC, CABAC is the block that is used for generating the final bitstream. Furthermore, in order to generate a valid bitstream the CABAC entropy coding should be performed sequentially. The entropy coder context is updated after coding each CU and is used to encode the next CU. Thus, the state of CABAC cannot be estimated, since an incorrect estimation can make the entire bitstream invalid. Moreover, it is not feasible to calculate all the possible states because of extremely large state space. Also, as explained in Section 1.3, to find the best parameter  $p$  in Eq. (1.9), entropy coding is performed. Consequently, entropy coder is one of the main hard dependencies. In fig. 3.1, the hard dependencies between two consecutive CU depths are depicted using directed acyclic graph (DAG). The entropy coder dependency is presented by solid lines in this figure.

According to Section 1.1.2, the CTU structure is formed by smaller CUs. In the RDO process, in order to decide between possible quadtree structures, to make a decision for split/no-split, the cost of smaller CUs should be calculated first. This RDO dependency can be observed

from Eq. (1.10) and is explained in Section 1.3. This dependency is presented in dotted lines in fig. 3.1 as a hard dependency.

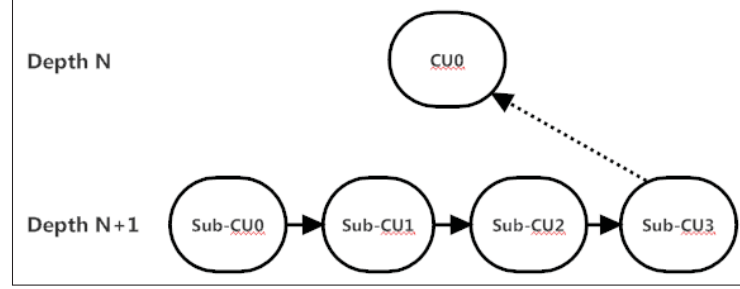


Figure 3.1 RDO hard dependencies between CUs for two consecutive depths

On the contrary, the prediction dependencies are soft dependency and if they are ignored, it only causes RD performance loss. Considering the inter-prediction mode, the MVP of each CU depends on top, left and top-left and top-right CUs as it is illustrated in fig. 3.2. The source of this dependency is Eq. (1.1) and is because of the MVP derivation process as explained in Section 1.2.1.

To investigate the MVP dependency in the RCME, we assume the dependency is removed and the dependent value is substituted by a predicted value. Thus, Eq. (1.3) is modified as:

$$\widehat{mv} = \underbrace{\arg \min}_{\forall mv \in MV_{\text{search}}} \{D(mv) + \lambda \cdot R(\widehat{mvp} - mv)\} \quad (3.1)$$

where  $\widehat{mvp}$  is an assumed MVP that is used instead of the actual MVP. This assumption, will produce  $\widehat{mv}$  which is calculated after RCME. It is clear that:

$$\widehat{mvp} \in \{mvp_A, mvp_B\} \rightarrow \widehat{mv} = mv^* \quad (3.2)$$

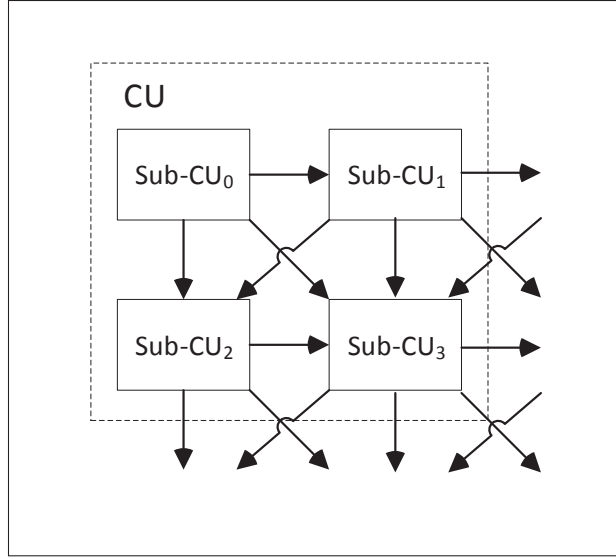


Figure 3.2 Inter-prediction soft dependencies between CUs

It shows that using an incorrect MVP will cause an error in RCME result. Moreover, an exact assumption of  $\widehat{mvp}$  will result in the exact  $mv^*$ . However, it is not possible to calculate the exact error caused by an incorrect assumption. Therefore, using multiple predictors suggests a solution that provides a way to eliminate this dependency. It should be noted that in this project, we only consider inter-prediction dependencies as soft dependency. The other soft dependencies in the HEVC (for instance the intra-prediction block) are assumed as hard dependency and they are kept as they were and are not modified by our proposed methods.

### 3.2 Two stage multiple predictor rate-constrained motion estimation (MP-RCME)

According to the previous section, to keep hard dependencies, encoding is performed in a z-scan order on the CTUs in a frame because the encoding of each CTU depends on all previously coded CTUs. However, it is possible to perform RCME according to Eq. (3.1). To provide a fine-grained parallel RCME, we have to separate the soft dependencies of RCME from the rest of HEVC. In this section, we introduce a method to divide RCME into two steps; first calculate RCME based on a list of assumed MVPs, and secondly choose the result based on



actual MVP as the final value. The first step can be calculated in parallel, however, the second step is performs the RDO process and is to perform in the normal sequential order.

The first stage estimates the prediction cost of a PU based on the most probable MVPs and finds the best parameters based on this assumption. Thus, no actual MVP is required at this time. By separating this part, we can calculate the main part of RCME based on a set of highly probable MVPs. Afterwards, in the second stage, one MV is selected based on the RCME costs when the actual MVP is available.

### 3.2.1 Parallel prediction using MP-RCME

The RCMEs of all the PUs could be calculated in parallel if we could determine the value of MVPs. However, it is not possible unless the calculations are done sequentially. To eliminate this problem we propose using multiple highly probable MVPs to estimate the cost.

The derivation of MVP from neighboring PU is not possible in parallel. Improper MVP assumption will produce an incorrect rate that leads to incorrect MV selection. RCME in Eq. (1.3) can be evaluated for an estimated MVP instead of the actual MVP. To reduce the RD performance loss in case of missed prediction, we propose using an MVP candidates list. This method is called multi-predictor rate-constrained ME. Using this concept, Eq. (1.3) is modified as below:

$$\widehat{mv}_i = \underbrace{\arg \min}_{\forall mv \in MV_{\text{search}}} \{D(mv) + \lambda \cdot R(\widehat{mvp}_i - mv)\} \quad (3.3)$$

where  $\widehat{mvp}_i$  is the  $i^{th}$  candidate from the MVP candidates list consisting of  $N$  probable candidates:

$$\widehat{mvp}_i \in \{\widehat{mvp}_1, \dots, \widehat{mvp}_N\} \quad (3.4)$$

The resulting  $\widehat{mv}_i$  from Eq. (3.3) is associated with the  $\widehat{mvp}_i$ . Furthermore, the distortion of the  $\widehat{mv}_i$  is calculated. As a result, for each MVP candidate from the Eq. (3.4) a parameter is produced as:

$$\widehat{P}_{ME_i} = (\widehat{mv}_i, \widehat{mvp}_i, D(\widehat{mv}_i)) \quad (3.5)$$

Parameters in the Eq. (3.5) are the best rate-constrained motion vector and the corresponding distortion when the  $\widehat{mvp}_i$  is a candidate for the actual MVP. The final result is achieved by producing Eq. (3.5) from calculation of Eq. (3.3) for each member of Eq. (3.4). The complete set of prediction parameters is constructed as:

$$\widehat{P}_{ME} = \{\widehat{P}_{ME_1}, \dots, \widehat{P}_{ME_N}\} \quad (3.6)$$

Eq. (3.5) is analogous to the original RCME. However, it is calculated based on a predicted MVP. Moreover, the  $\widehat{P}_{ME_i}$  is considered as one possible solution and to allow the next step to decide if this is the best solution, we included the distortion value into this parameter. As we explain in the next section, the final decision of MP-RCME has to be made in the RDO process. Differing the decision of choosing the best parameter, allows the parallel calculation of all  $\widehat{P}_{ME}$  parameters of all PUs.

### 3.2.2 Best parameter selection

The second stage of our framework is to use the calculated prediction costs to choose the best mode based on RD. This stage is added to the RDO process and is performed in sequential order before the rest of the RDO. Moreover, we explained in Section 3.1 that it is not possible to preform the RDO process in parallel. Thus, we should perform this stage at low computational cost to minimize its effect on speedup. In this stage, the actual MVP is available. This value is used to find the best MVP candidate which produces minimum cost.

In this stage of the algorithm, the actual values of MVP is determined as:

$$mvp \in \{mvp_A, mvp_B\} \quad (3.7)$$

The  $mvp_A$  and  $mvp_B$  are determined by the MVP derivation process that was explained in Section 1.2.1. The best parameter set, is determined by selecting a set producing the minimum

RD cost. Using calculated values from the previous step, the cost can be calculated as:

$$J_{ME}(i, \mathbf{mvp}) = D(\widehat{\mathbf{mv}}_i) + \lambda \cdot R(\mathbf{mvp} - \widehat{\mathbf{mv}}_i) \quad (3.8)$$

where  $D(\widehat{\mathbf{mv}}_i)$  and  $\widehat{\mathbf{mv}}_i$  are values from previously calculated  $\hat{P}_{ME,i}$ . Also, the  $\mathbf{mvp}$  is recovered from Eq. (3.7). The value of  $J_{ME}$  will determine the cost of  $\widehat{\mathbf{mv}}_i$  if it is calculated with the actual  $\mathbf{mvp}$ . However,  $\widehat{\mathbf{mv}}_i$  was calculated based on an assumed  $\widehat{\mathbf{mvp}}_i$  and the resulting cost might not be the minimum. To find the minimum cost between all predictors, the following minimization is performed:

$$P_{ME} = (\mathbf{mv}^*, \mathbf{mvp}^*) = \underbrace{\arg \min}_{i \in \{1, \dots, N\}, \mathbf{mvp} \in \{\mathbf{mvp}_A, \mathbf{mvp}_B\}} \{J_{ME}(i, \mathbf{mvp})\} \quad (3.9)$$

After selection of the  $P_{ME}$ , the rest of the RDO process is performed as explained in Section 1.3.

### 3.2.3 Offloading MP-RCME to the GPU

As we explained, MP-RCME is providing a method to perform complex parts of RCME in parallel. This method can be used, to offload the RCME part to any device capable of running the parallel algorithm. However, the degree of parallelism for this algorithm is very high and is more efficient for massively parallel architecture. In this section, we propose an algorithm to use MP-RCME method to offload the calculation into GPU.

The process depicted in fig. 3.3 is for encoding a frame with the parallel RCME. In the first step, the data is prepared and transferred to the GPU for distortion calculation. The original image and reference images are transferred before the encoding starts.

Meanwhile, the MVPs candidates list is built in the CPU and is transferred to GPU for performing a MP-RCME. The MVPs candidates list will be used and motion estimation performed in parallel for all PUs.

The final result will be a list of distortion and MV pairs. In the second stage, the calculated pairs will be used in the CPU by the RDO process to make the final decisions.

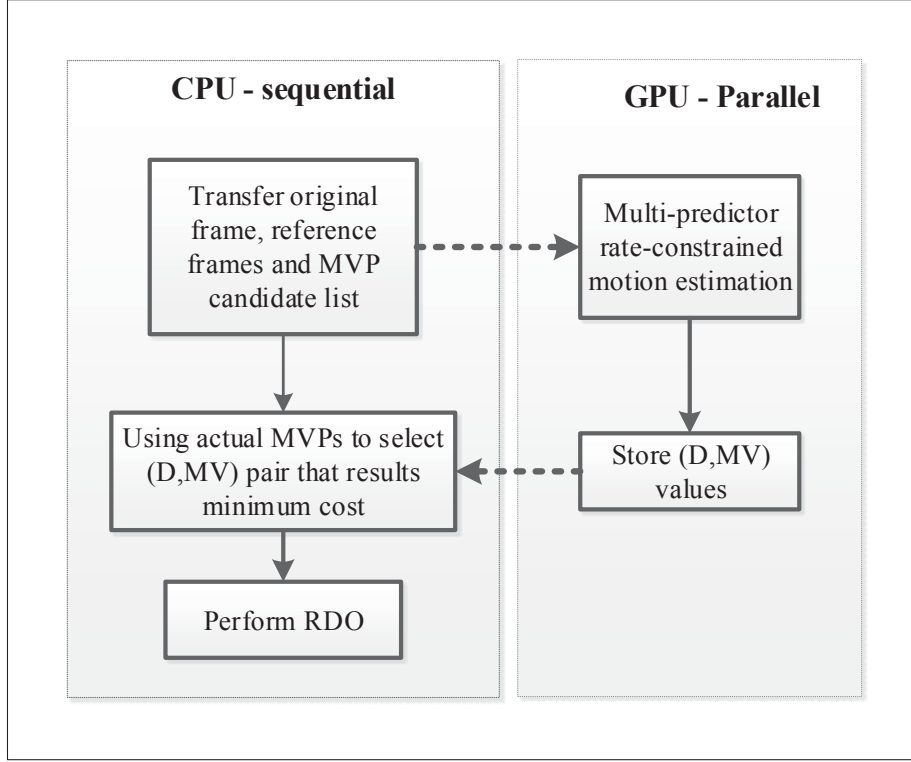


Figure 3.3 Proposed two stage MP-RCME framework

Furthermore, the maximum required bandwidth of this method can be calculated by the following equation:

$$\begin{aligned}
 \text{Bandwidth (bytes/sec)} = & \left\lceil \frac{\text{FrameWidth}}{64} \right\rceil \times \left\lceil \frac{\text{FrameHeight}}{64} \right\rceil \times \\
 & N_{\text{PUsInCTU}} \times N_{\text{ReferenceFrames}} \times N_{\text{FramesPerSecond}} \times \\
 & N_{\text{MVP}} \times 3 \times \text{sizeof}(\text{Int16})
 \end{aligned} \tag{3.10}$$

where  $N_{MVP}$  is the number of MVP candidates described in Eq. (3.4) and the rest of parameters are as explained for Eq. (2.2). To find the bandwidth improvement of our proposed method, it is compared with the bandwidth for the state-of-the-art methods in Eq. (2.2). For instance, let's assume a list of 16 candidates has been used, while, other parameters are the same as for Eq. (2.3). The required bandwidth is calculated as 4.5 GB/s and it is reduced 10x. Furthermore, exhaustive fullsearch or suboptimal fast search methods can be used for RCME in the GPU as it is explained in Section 3.5.

Moreover, the fractional motion estimation can be performed in GPU or it can be postponed to be performed in CPU. The difference between these configurations is discussed in Section 3.3.

### 3.3 MP-RCME with postponed fractional calculation

As it is explained in Section 1.2.3, the RCME is performed at two precisions. First an integer motion vector is calculated, then the motion vector is refined using fractional pel precision. Even with fractional refinement around the integer motion vector, the computational cost of this operation is high. The main reason is that the calculation of interpolated pixels based on the integer pixel is computationally complex. This calculation requires filtering operations as explained in Section 1.2.2. Moreover, the calculation of the cost for each of these fractional positions is complex. In this section, we discuss the effects of performing fractional motion estimation according to the MP-RCME algorithm. As mentioned, the MP-RCME process is divided into two stages. Based on the place of the fractional refinement process, two types of algorithm are presented.

It is possible to perform fractional prediction in the first stage after integer MP-RCME. This configuration is depicted in fig. 3.4. The advantage is to perform all of the RCME process in the GPU and as a result a higher speedup is possible. Since more work is performed by the GPU, the RDO process in the CPU becomes less computationally complex. However, because of assumed MVP, the RD performance will be reduced. Because of a possible incorrect MVP assumption, integer MV is not the optimum result. However, postponing the fractional MV

refinement allows the CPU to compensate that error because the actual MVP is available at that time. Thus, CPU obtains a fractional MV with more accurate MV cost.

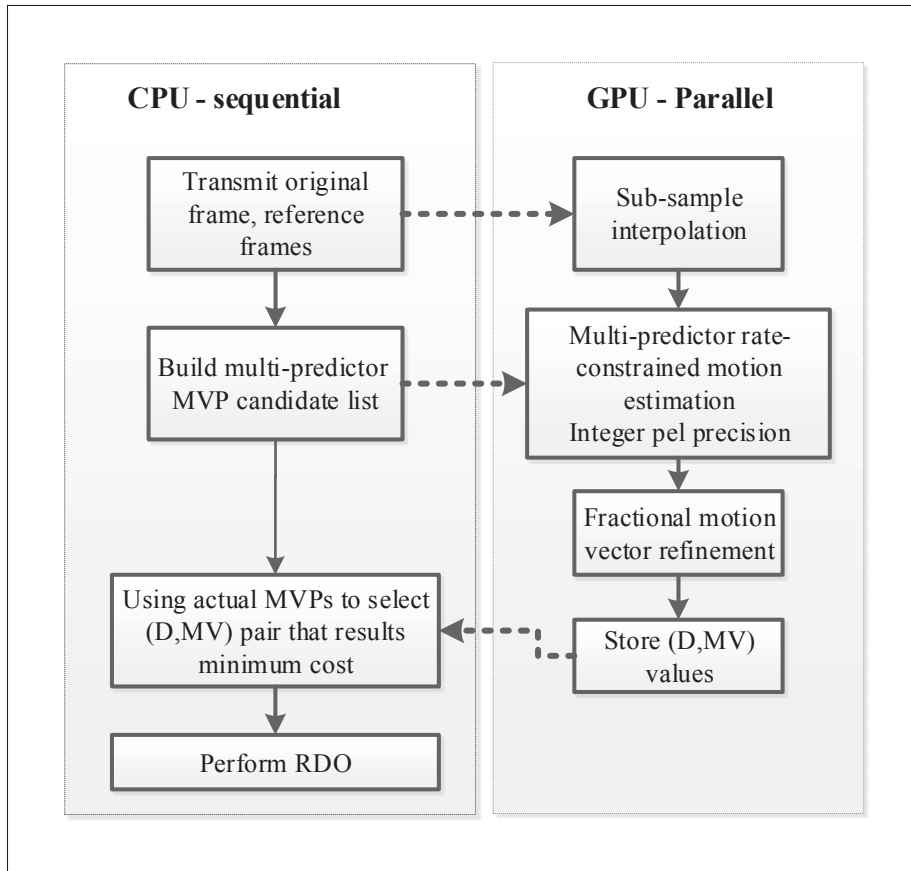


Figure 3.4 MP-RCME with fractional refinement in the GPU

The other configuration is to perform postponed fractional refinement in the second stage. This means, the integer MP-RCME is performed in first stage and the fractional is performed in the RDO. This configuration is depicted in fig. 3.5. The benefit of this configuration is better RD performance because the refinement cost can be calculated based on actual MVP. However, all of the interpolation computations have to be performed in the CPU and sequentially, which is undesirable.

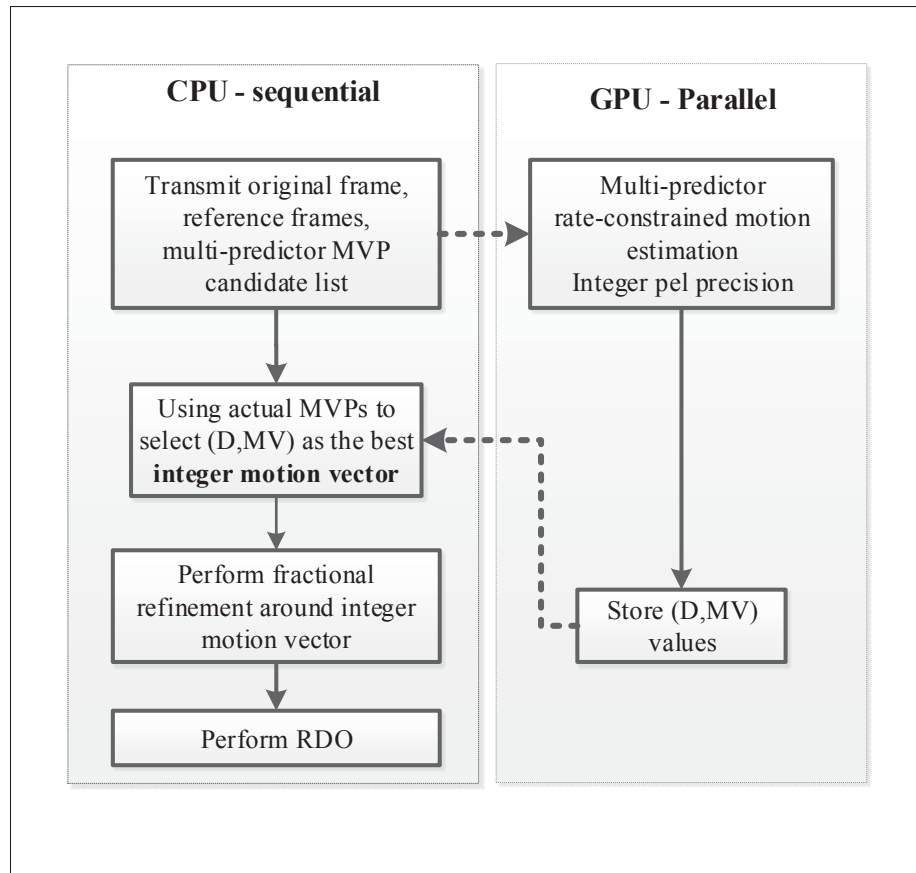


Figure 3.5 MP-RCME with fractional refinement in the CPU

Nevertheless, these two configurations provide different trade-off for RD performance and complexity. The results, for both of these configurations are compared in Chapter 4.

### 3.4 Multiple temporal predictor rate-constrained motion estimation (MTP-RCME)

As it was mentioned in Section 3.2.1, to perform the MP-RCME a list of highly probable MVP candidates is essential. If the MVP candidates list is poorly selected, the RD performance will be reduced significantly. In this section, the MTP-RCME method is introduced to perform the MP-RCME based on temporal information. The MVP candidate list is generated in the first part of MP-RCME, thus, it should have insignificant computational complexity.

In this method the MVs of previous frames are used as MVP for the current frame. As it is mentioned in Section 1.1.2, the coding structure of HEVC is based on a quadtree. Thus, special consideration is required to determine MVPs from MVs of previous frame. The structure of CU in the current frame could be different from that of the previous frame, thus, the number of MVPs that are inferred could be different.

In HEVC, after compression of a frame, MVs are stored in the decoded picture buffer (DPB) to be used later. These MVs are stored in a 4x4 grid and for a CTU of size 64 there are 16 MVs stored in the DPB. For each CTU in the new frame, it is possible to build a list of unique MVs from the previous frame using values in the DPB as follows:

$$\text{MTP} = \{mv_0, \dots, mv_N\} \quad (3.11)$$

where MTP is the set of unique motion vectors extracted from previous frame at the co-located CTU. As a result, the number of  $N$  is not fixed and it is less or equal to 16. Based on the structure of the previous CTU frame, multiple temporal predictor (MTP) sets can represent most probable independent MVP candidates.

Using Eq. (3.11) in the MP-RCME algorithm will provide a better RD performance because of the list of highly probable MTP, while it has the advantage of low bandwidth requirements of MP-RCME.



### 3.5 Suboptimal parallel RCME on the GPU

As it was discussed in Section 2.2.2, to our knowledge, fullsearch is the only search algorithm used in the literature for parallel RCME. The main reason is the higher performance of GPU for data parallel algorithms. Moreover, fullsearch implementation in the GPU is fairly straightforward and building bottom-up distortions is advantageous. Moreover, in the fullsearch algorithm, the starting point of search (also referred as the search center) has less importance because it can be compensated by increasing the search range.

Suboptimal search methods are significantly less complex compared to the fullsearch method. However, the number of iterations depends on the selected search center and on the distortion values, producing unequal number of iterations for different PUs.

The GPU architecture is not able to execute a parallel algorithm efficiently unless it is implemented in a single instruction multiple data (SIMD) paradigm. We propose a suboptimal search method similar to TZS to improve the GPU execution, by removing the different execution paths for different PUs.

#### 3.5.1 GPU architecture considerations

The GPU hardware is designed to execute parallel programs using the single instruction multiple thread (SIMT) computing model. This model is similar to the well-known SIMD computation, however, instead of single operation on different data, a single thread is executing operations on the data. SIMT model executes the same copy of a parallel program (kernel) on different data. The terminology for parallel GPU programming vastly differs for each vendor and based on the framework. In our thesis, we use the Open Computing Language (OpenCL) terminology. Each instance of a program is run by a workitem or thread. Workitems are grouped into workgroups.

However, because of the scheduler architecture in the GPU, workitems are executed in clusters. The number of workitems that are processed in a cluster is 32 for NVIDIA (a thread warp) and

is 64 for AMD (a thread wavefront). To use resources efficiently, the number of workitems in a workgroup should be a multiple of the number of wavefront threads.

Furthermore, a wavefront executes workitems in parallel using the SIMT processing model across the processing unit. Thus, the execution time is affected by divergences in the execution flow. Branching, for example, is achieved by combining all execution paths into a unique sequence of instructions. This implies that the total time to execute a multipath branch is the sum of the execution time for each individual path. As a consequence, even if only one workitem in a wavefront diverges, all the workitems in the wavefront will execute the diverging branch. Thus, all the threads of a wavefront should follow exactly the same execution path to achieve maximum execution efficiency. More detailed information about the graphics processing unit (GPU) hardware architecture is provided in Appendix I.

Considering the GPU architecture, it is clear why TZS is not suitable for GPU. Based on the mentioned concerns, a suboptimal search method is introduced in the following section.

### **3.5.2 Nested diamond search (NDS)**

To achieve a high execution performance for parallel RCME, we have designed a specific search method for the GPU architecture. The proposed NDS method is able to utilize the GPU resources more efficiently.

Considering the constraints mentioned in Section 3.5.1, to map the TZS algorithm into an efficient data-parallel model, we define a fixed search pattern with 64 MVs positions. This search pattern is a modified diamond search pattern as depicted in fig. 3.6.

Using a fixed pattern, defines the base of the algorithm with a fixed number of threads. The number 64 is a multiple of number of threads per wavefront. Thus, each pattern is fitted in one wavefront (for AMD) or two warps (for NVIDIA). Thus, regardless of the hardware in use, the algorithm will be executed efficiently in the GPU.

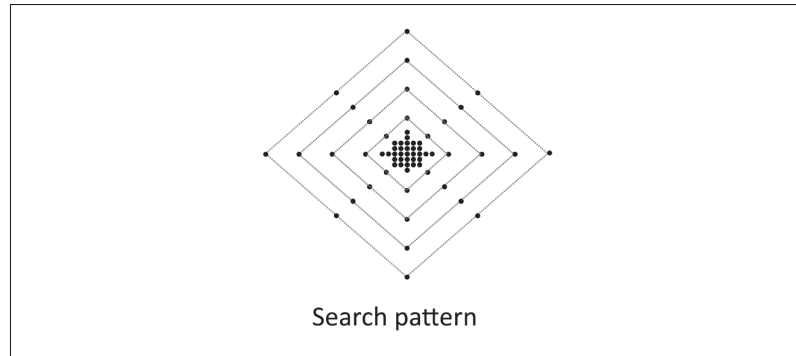


Figure 3.6 Fixed search pattern with 64 MV positions

This pattern is concentrated in the center and surrounded by four embedded diamond patterns with 8 pixels step. The best MV has usually a distance of less than 4 integer pixels with respect to the predictor (Jeong *et al.*, 2015). However, the evaluation of further positions is necessary to prevent falling into a local minimum.

Furthermore, one wavefront performs block matching for all the positions of a PU in the NDS pattern. Hence, each thread is calculating the distortion for one of the MVs in the NDS pattern.

After each iteration, if the termination condition is not met, the center is moved to the best position with the lowest cost and another search iteration is performed by the same workgroup.

Moreover, each CU consists of several PUs but each PU might require a different number of iterations. The algorithm requires special arrangements to prevent performance loss. Thus, we split the CU into all the possible PUs and assign a workgroup to each PU. The structure of workgroups and threads of a CTU is depicted in fig. 3.7.

To match the GPU's data-parallel model, the RCME for each PU is defined by a data structure containing the arguments for this process. The input information for each PU is depicted for some instances in fig. 3.8. Each entry of the job structure contains the position and the size of the PU.

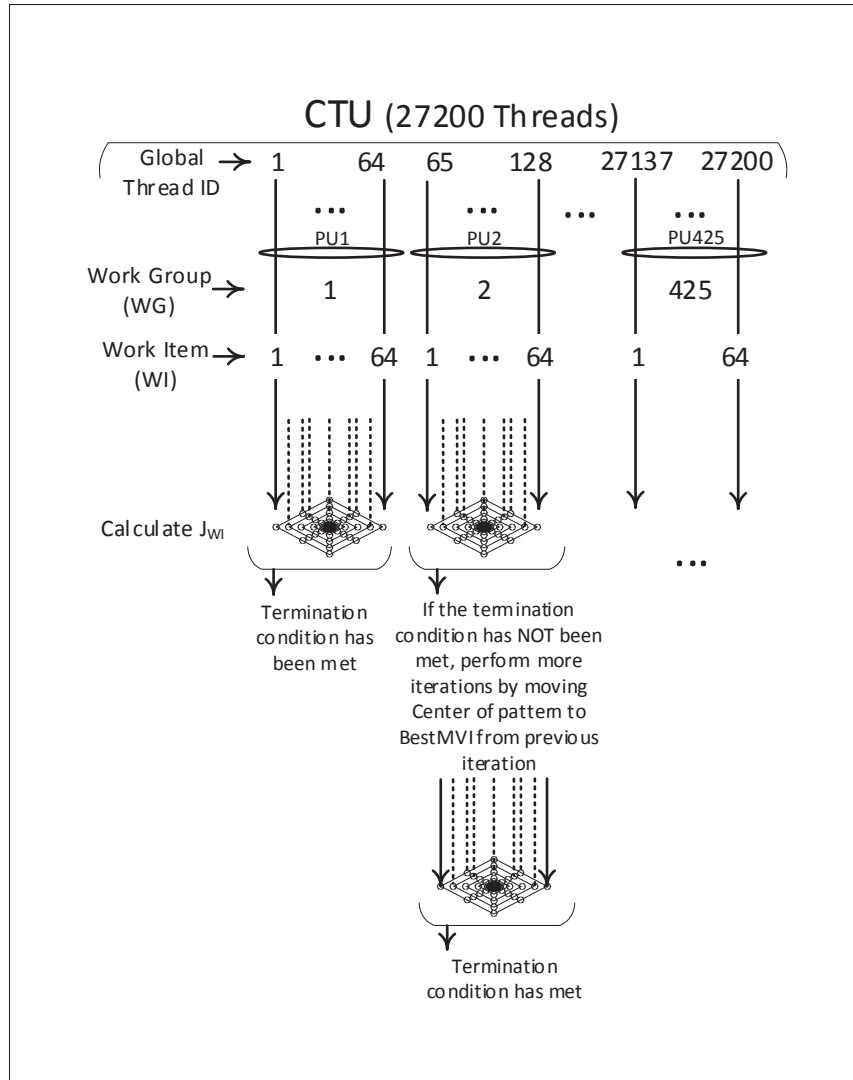


Figure 3.7 Execution path of PUs in NDS

For a CTU, these job structures are precomputed and stored into arrays. When asymmetric mode partitioning (AMP) is not enabled, each CTU consists of 425 possible PUs and accordingly a workgroup is assigned to each PU. Job scheduling and work mapping for a CTU is illustrated in fig. 3.9. The GPU kernel for the NDS method is summarized in fig. 3.10.

To exploit even more the GPU's processing capabilities, we also perform interpolation filtering and fractional pel refinement after the integer motion estimation in the GPU. The reference







PU						...	
WG	1	2	3	4	5	...	425
PU <sub>Location</sub>	0,0	0,0	32,0	0,0	0,32	...	60,56
PU <sub>Size</sub>	64x64	64x32	64x32	32x64	32x64	...	4x8

Figure 3.8 Predefined PU information required in NDS

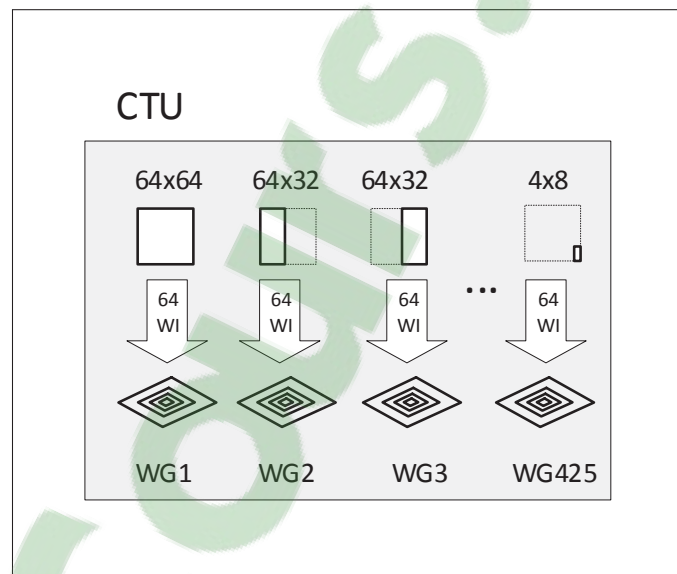


Figure 3.9 Workitem (WI) and workgroup (WG) mapping

frames are updated and interpolated in the GPU right after the reconstruction of each frame. A separate GPU kernel is performing the interpolation filter.

The interpolation filter in the GPU is implemented as a separable filter. For each pixel, sixteen subsamples are generated. The image is partitioned into one-pixel-wide columns with 256

```

1: WG ← get_group_id()      ► PU index (idx)
2: WI ← get_work_id()      ► Position idx in search pattern
3: PUjob ← PUArray[WG]      ► PU size and position
4: SearchPos ← PosArray[WI] ► Search position
5: BestMVI ← MVPi , iter ← 0
6: do
7:   Center ← BestMVI
8:   JME[WI] ← SAD(PUjob, Center + SearchPos)
9:   BestMVI ← argmin(JME[0...63]) ► After barrier
10:  iter++
11: while (iter < 4 and abs(Center - BestMVI) ≥ 2)
12: BestMv = fractionalRefinement(BestMVI)

```

Figure 3.10 NDS GPU kernel algorithm

one-pixel rows, and interpolation of each column is done by one workgroup consisting of 256 workitems. Each workitem is calculating sixteen subsamples for each pixel.

### 3.6 Parallel encoding framework for massively parallel architectures

In this section, we present the structure of our implemented parallel encoder in which our methods are used.

To achieve a complete parallel encoder, we designed the offloading process in two separate threads in the CPU. The main thread is performing the normal RDO process, while the second thread is responsible for communication and offloading tasks to the GPU. Our proposed framework combining all introduced methods is depicted in fig. 3.11.

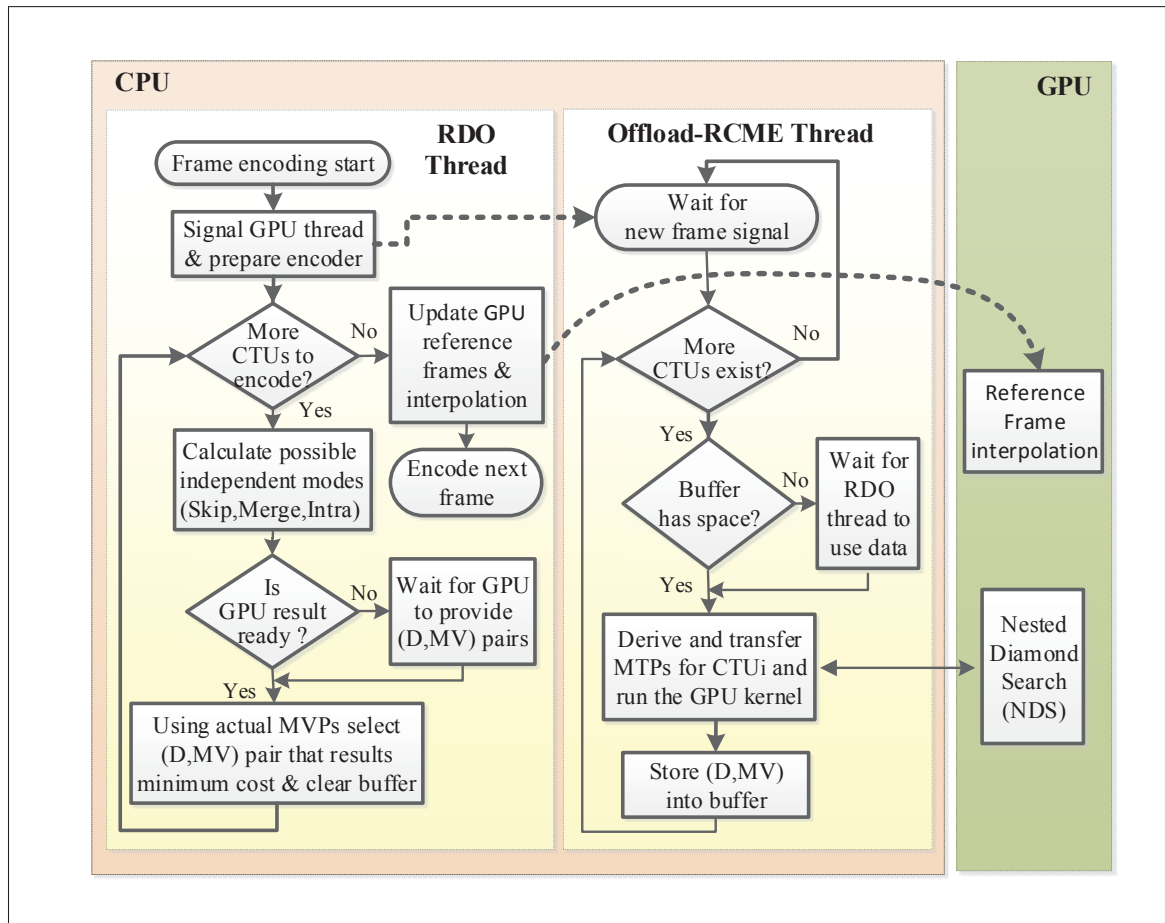


Figure 3.11 HEVC parallel encoding framework

In the RDO thread, the encoder performs all of the encoding process, except the RCME. Instead of RCME, our proposed MTP-RCME is performed in the GPU using NDS method. The offloader thread is responsible for offloading CTU jobs to the GPU and communication with the GPU.

When the MTP-RCME results of each CTU is ready, the offload thread reads them from the GPU and stores them in a buffer. The communication between the two threads are performed using this intermediate buffer.

The RDO thread is waiting until the results are available in the buffer. As soon as the results are available, the RDO thread will start to performing RDO. Meanwhile, the offloading thread is executing concurrently and offloading more CTUs to GPU regardless of the state of the RDO thread.

Moreover, after the encoding of a frame is completed, the RDO thread will transfer the reconstructed image to the GPU and enqueues it in the GPU interpolation kernel.

Using two separate threads, allows to minimize the effect of unbalanced execution time between CPU and GPU. The buffer in the middle eliminates possible stalls. In the next chapter, the results of our proposed methods are compared with state-of-the-art methods.



## CHAPTER 4

### EXPERIMENTAL RESULTS

To evaluate our proposed methods a comprehensive comparison is performed with the state-of-the-art methods. In order to have a fair comparison, both execution speed and RD performance is considered. In the next sections, the utilized metrics for the performance measurement and the setup is explained. Afterwards, the final results achieved by our methods and similar state-of-the-art methods are compared. Finally, the results are discussed and summarized.

#### 4.1 Setup

The HEVC test Model (HM) implementation is provided in C++ to developers (HM1). This implementation does not have parallel implementation. However, it is the comprehensive implementation of the HEVC capabilities. The HEVC test Model (HM) is considered as the anchor implementation for the HEVC and it is used by researchers in academic research. It should be mentioned that HM is not an efficient implementation of the encoder but it can be used as a tool for comparing various proposed algorithms. In addition, Visual Studio in Windows is used as development environment. We have chosen the Visual Studio environment for implementing our primary ideas because it provides a mature development environment. For parallel framework, we decided to use OpenCL since it supports a vast range of GPUs. During this research, we have utilized the servers from Calcul Quebec provided to us, to examine several algorithms. However, the final results for comparing the algorithms are achieved using a hardware with the specifications as shown in table 4.1.

Table 4.1 Hardware specification

<b>CPU</b>	Intel Core i7 4770 @ 3.4 GHz
<b>RAM</b>	24.0 GB
<b>OS</b>	Windows 7 Enterprise (64 bit)
<b>GPU</b>	AMD Radeon R9 270

In order to validate the proposed algorithms, we compare our methods against the state-of-the-art solutions with the same level of parallelism. The most important metric for comparison of video compression is RD performance. The image quality loss is usually measured by the widely used peak signal-to-noise ratio (PSNR). However, comparing two methods and interpreting the results for different rates and quality is difficult. To solve this problem, many researchers in recent years have used two other metrics known as BD-PSNR and BD-Rate. BD-PSNR is the PSNR average and BD-Rate is the rate average over the whole PSNR-rate curve. The details on the calculation of these parameters are mentioned in Appendix II.

For validation of the new methods we use standard video sequences from the standard document of ITU-T and ISO/IEC for HEVC (Bossen, 2013). Detailed information about standard video sequences are presented in Appendix III. Furthermore, parameters of the encoder are selected according to the table 4.2 and are fixed for the anchor method and the proposed methods.

Table 4.2 HEVC encoder configuration

<b>Quantization parameter (QP)</b>	{22, 27, 32, 37}
<b>Number of reference frames</b>	4
<b>Search range</b>	64
<b>Prediction structure</b>	Low Delay P

Performance of an algorithm can be measured according to several aspects. For instance, memory usage and execution time are typical measures. However, for video coding algorithms the RD performance should also be considered as an important metric. The resulting RD of an encoding process is affected by the complexity of the algorithm. For instance, enabling more encoding features allows a better compression and RD performance.

Moreover, in many applications, the amount of memory usage is less important compared to the accessing memory time. Thus, between alternative methods using different memories, the faster algorithm is preferred if the memory is available to run the algorithm. As a result, for

performance measurement of a parallel algorithm, the architecture specifications and memory access complexity should be taken into account.

Considering the points mentioned, we realize that analyzing parallel algorithm is a combination of algorithms and architecture. There are several possible methods to calculate the speedup (Kumar *et al.*, 1994). Nevertheless, speedup metric that is used in our project is achieved by comparing execution time of the whole encoding process of anchor algorithm and the proposed algorithm.

$$\text{Speedup} = \frac{T_{\text{Anchor}}}{T_{\text{Proposed}}} \quad (4.1)$$

It is noted that the RCME is the only part of the encoding process that is performed in parallel and the rest of the encoding process is executed as the anchor version. Moreover, the results in some of the literature are measured by the time reduction (TR). The TR is defined as:

$$\text{TR} = \frac{T_{\text{Anchor}} - T_{\text{Proposed}}}{T_{\text{Anchor}}} \times 100 \quad (4.2)$$

The anchor algorithm is HM test model version 15.0 and the comparison is made with algorithms from the state-of-the-art and our proposed methods. Our comparison consists of methods based on two parameters. The first parameter is methods for resolving MVP prediction. The second parameter is the search method. To have a comprehensive comparison, all of the combinations of these parameters are compared. For the first parameter, the MVP prediction methods of the literature and our proposed method are shown in table 4.3.

Table 4.3 MVP prediction methods

Method	Number of MVPs	MVP selection	Literature
HM	1	original MVP derivation Eq. (1.2)	(Sze <i>et al.</i> , 2014)
Zero	1	$mvp = (0,0)$	(Chen & Hang, 2008) (Momcilovic & Sousa, 2009)
AVG	1	$mvp = \frac{1}{4} \sum_{i=1}^4 mv_i$	(Ma <i>et al.</i> , 2014)
MTP	Multiple	Eq. (3.11)	Proposed

The zero method is corresponding to use a single Null *mv* (with the x component of 0 and the y component of 0). The *mvp* in the AVG method is achieved by averaging co-located MVs from the previous frame. The glsmt is our proposed MP-RCME method in which a list of temporal candidates are used (MTP-RCME) (Hojati *et al.*, 2017a).

For the second parameter, the fullsearch method (FS) and suboptimal methods (TZS, NDS) are compared. The fullsearch implementation is repeatedly used in the literature and we implemented fullsearch on the GPU as mentioned in (Chen & Hang, 2008).

Moreover, there is no previous work in the literature on suboptimal search on the GPU. Thus, we compare our proposed NDS method against the anchor TZS implementation in the HM (Hojati *et al.*, 2017b).

To compare the RD performance of methods, HM-FS is used as anchor since this results in the best RD performance. In this method, the MVP prediction is exactly like HM and the RCME search method is fullsearch. Also, to compare the TR, HM-TZS is used as anchor since this is the fastest possible encoder configuration. In this method, MVP is exactly as HM and the TZS search method is used.

Moreover, MTP-FS is defined as using proposed MTP method for MVP prediction and all of the parallel RCME with fullsearch on the GPU. Similarly, in the MTP-NDS the proposed NDS is used as search method. In MTP-FS-PF method, we use our proposed MTP and perform integer fullsearch on GPU and fractional refinement on the CPU. Results of MTP-FS is compared with MTP-FS-PF (postponed fractional calculation) to provide experimental results for the possible trade-off mentioned in Section 3.3 (combined MTP-RCME with postponed fractional calculation).

## 4.2 Results

To investigate the RD performance of our proposed methods, first the BD-Rate results are reported. To evaluate the effectiveness of MTP-RCME method, a comparison for FS is depicted in table 4.4.

Table 4.4 Comparison of proposed MTP-RCME method with the state-of-the-art methods in terms of RD performance loss

Class	Sequence name	BD-Rate (%) compared to HM-FS		
		Zero-FS	AVG-FS	MTP-FS
D ( $416 \times 240$ )	BQSquare	1.72	1.41	0.39
	BasketballPass	2.01	1.42	0.57
	BlowingBubbles	1.78	1.65	0.28
	RaceHorses	2.6	2.07	1.2
C ( $832 \times 480$ )	BQMall	1.94	1.65	0.56
	BasketballDrill	2.16	1.73	0.49
	Flowervase	2.11	1.52	0.33
	RaceHorses	2.97	2.28	1.48
E ( $1280 \times 720$ )	FourPeople	2.15	1.67	0.62
	Johnny	1.76	1.55	0.75
B ( $1920 \times 1080$ )	Cactus	2.63	1.99	0.82
	Kimono	2.25	1.72	1.26
	ParkScene	2.61	1.93	1.2
A ( $2560 \times 1600$ )	PeopleOnStreet	2.98	2.49	1.21
	<b>Average</b>	<b>2.26</b>	<b>1.79</b>	<b>0.79</b>

Moreover, the MTP-RCME is also effective when used with NDS search method. The results of comparing NDS with HM-FS is presented in table 4.5.

Moreover, the TR of methods compared to the HM-TZS is depicted in table 4.6 and table 4.7.

As it was mentioned in Section 3.3, the calculation of fractional RCME can be performed in either CPU or GPU. The presented results in table 4.4 and table 4.5 are for the configuration with the fractional RCME performed in the GPU. Moreover, in table 4.8 and table 4.9, the RD

Table 4.5 Comparison of proposed NDS method with the state-of-the-art methods in terms of RD performance loss

Class	Sequence name	BD-Rate (%) compared to HM-FS		
		Zero-NDS	AVG-NDS	MTP-NDS
D ( $416 \times 240$ )	BQSquare	2.13	1.34	0.68
	BasketballPass	2.24	1.82	0.98
	BlowingBubbles	1.56	0.77	0.46
	RaceHorses	3.84	3.52	2.15
C ( $832 \times 480$ )	BQMall	3.28	2.76	1.74
	BasketballDrill	3.14	2.59	1.6
	Flowervase	2.08	1.49	0.64
	RaceHorses	3.8	3.39	2.56
E ( $1280 \times 720$ )	FourPeople	3.02	2.39	1.43
	Johnny	2.64	2.24	1.57
B ( $1920 \times 1080$ )	Cactus	2.66	2.01	1.29
	Kimono	3.28	2.49	1.85
	ParkScene	3.18	2.62	1.68
A ( $2560 \times 1600$ )	PeopleOnStreet	3.3	2.62	1.83
	<b>Average</b>	<b>2.87</b>	<b>2.29</b>	<b>1.46</b>

performance and TR is compared when a postponed fractional RCME is performed in the CPU compared to when all of the RCME process is performed in the GPU.

Table 4.6 TR comparison of FS methods with HM-TZS

Class	Sequence name	TR (%) compared to HM-TZS		
		Zero-FS	AVG-FS	MTP-FS
D ( $416 \times 240$ )	BQSquare	36.8	38.2	37.65
	BasketballPass	37.7	37.3	37.6
	BlowingBubbles	37.7	37.5	37.85
	RaceHorses	36.2	36.8	36.6
C ( $832 \times 480$ )	BQMall	41	40.7	41.05
	BasketballDrill	42.1	41.7	41.8
	Flowervase	37.9	39	38.35
	RaceHorses	38.9	37.8	38.15
E ( $1280 \times 720$ )	FourPeople	43.7	43.4	43.3
	Johnny	44	43.7	44.25
B ( $1920 \times 1080$ )	Cactus	43.7	42.7	43.1
	Kimono	41.3	40.9	41.1
	ParkScene	42.6	41.5	42.15
A ( $2560 \times 1600$ )	PeopleOnStreet	41.5	42.6	42.45
	<b>Average</b>	<b>40.4</b>	<b>40.3</b>	<b>40.4</b>

Table 4.7 TR comparison of NDS proposed methods with HM-TZS

Class	Sequence name	TR (%) compared to HM-TZS		
		Zero-NDS	AVG-NDS	MTP-NDS
D ( $416 \times 240$ )	BQSquare	37.1	37.6	37.4
	BasketballPass	37.9	37.7	37.5
	BlowingBubbles	38.2	37.7	37.6
	RaceHorses	36.4	36.4	35.5
C ( $832 \times 480$ )	BQMall	41.4	41.5	40.9
	BasketballDrill	41.9	41.3	41.1
	Flowervase	37.7	38.6	38.4
	RaceHorses	38.5	37.5	37.6
E ( $1280 \times 720$ )	FourPeople	43.2	43.6	43.1
	Johnny	44.8	44.5	45.8
B ( $1920 \times 1080$ )	Cactus	43.5	42.9	42.9
	Kimono	41.3	41.4	41.7
	ParkScene	42.8	42.1	42.5
A ( $2560 \times 1600$ )	PeopleOnStreet	42.3	42.1	42.3
	<b>Average</b>	<b>40.5</b>	<b>40.3</b>	<b>40.1</b>

Table 4.8 The BD-Rate improvement for performing the fractional refinement on the CPU compared to performing on the GPU

Class	Sequence name	BD-Rate (%) improvement		
		Zero-FS	AVG-FS	MTP-FS
D ( $416 \times 240$ )	BQSquare	0.19	0.18	0.27
	BasketballPass	0.24	0.19	0.26
	BlowingBubbles	0.17	0.29	0.27
	RaceHorses	0.23	0.26	0.15
C ( $832 \times 480$ )	BQMall	0.2	0.16	0.22
	BasketballDrill	0.24	0.21	0.22
	Flowervase	0.29	0.26	0.26
	RaceHorses	0.25	0.29	0.27
E ( $1280 \times 720$ )	FourPeople	0.24	0.26	0.24
	Johnny	0.17	0.21	0.24
B ( $1920 \times 1080$ )	Cactus	0.27	0.28	0.17
	Kimono	0.26	0.18	0.3
	ParkScene	0.23	0.23	0.24
A ( $2560 \times 1600$ )	PeopleOnStreet	0.32	0.49	0.16
	<b>Average</b>	<b>0.24</b>	<b>0.25</b>	<b>0.23</b>

Table 4.9 The decrease of TR when performing fractional refinement in the CPU compared to the GPU

Class	Sequence name	TR (%) decrease		
		Zero-FS	AVG-FS	MTP-FS
D ( $416 \times 240$ )	BQSquare	4.21	4.02	4.33
	BasketballPass	4.75	4.47	4.8
	BlowingBubbles	4.95	6.11	6.61
	RaceHorses	3.32	3.61	3.55
C ( $832 \times 480$ )	BQMall	3.91	3.89	4.23
	BasketballDrill	4.81	5	5.17
	Flowervase	5.04	4.96	5.36
	RaceHorses	3.22	3.21	3.46
E ( $1280 \times 720$ )	FourPeople	4.57	4.47	4.86
	Johnny	3.71	3.79	3.91
B ( $1920 \times 1080$ )	Cactus	4.77	4.79	5.19
	Kimono	5.34	5.34	5.64
	ParkScene	5.11	5.14	5.59
A ( $2560 \times 1600$ )	PeopleOnStreet	5.39	5.24	5.73
	<b>Average</b>	<b>4.51</b>	<b>4.57</b>	<b>4.89</b>



### 4.3 Discussion

First, the results are investigated in terms of RD performance. In table 4.4 it is clear that, as it was expected, the worse RD performance is achieved with the approach using a fixed single MVP at zero. Thus, when RCME is performed for Zero-FS the BD-Rate is increased by an average of 2.26%. Also, the RD performance loss is observed regardless of the search method, as it is equal to 2.87% for Zero-NDS in table 4.5.

However, the RD performance loss is less severe by using an averaging method for MVP. The BD-Rate improvement for AVG-FS has an average of 0.47% compared to Zero-FS. A similar improvement is achieved when the average MVP is used for the NDS search. The BD-Rate has improved by 0.58% for AVG-NDS method compared to the basic Zero-NDS method.

Moreover, the maximum improvement is achieved by our proposed MTP method for both search methods. The average BD-Rate improvement of MTP-FS compared to Zero-FS is 1.47%. Moreover, BD-Rate has improved by an average of 1.41% when MTP-NDS method is used compared to Zero-NDS.

As a result, regardless of the search methods, the proposed MTP-RCME method outperforms the Zero MVP method by 1.44% and the AVG method by 0.92%.

Pondering into table 4.4 and table 4.5 shows the effectiveness of MTP-RCME method for all video resolutions. The BD-Rate improvement is consistent for all of video classes.

In addition to RD performance, the TR of methods should be investigated. From, table 4.6 the average TR for Zero-FS is 40.4% compared to HM-TZS. The achieved TR is due to the parallelization of the RCME part of the encoding process. In all of methods, the rest of encoding process is performed in the CPU and sequentially. Subsequently, an average speedup of 1.78 is achieved by our proposed methods when compared to the fastest implementation in HM.

Furthermore, the maximum possible TR for an encoder with parallel RCME is achieved by our proposed methods. The bottleneck of the encoder is the rest of process performed in the CPU. Thus, using high level parallel tools (e.g. WPP) will provide more TR when combined with our parallel RCME methods. Moreover, the GPU load for FS methods is 79% while for our proposed NDS it is reduced to 52%. This, means NDS provides better GPU utilization compared to previous full search (FS) method.

Moreover, in table 4.8 the average of 0.25% in BD-Rate is providing a possibility for trade-off based on the application. Thus from table 4.9, with 4.5% penalty in TR, the BD-Rate can be improved by 0.25%.

The results confirm the effectiveness of our MTP method. It is observed that for all of the configurations, the MTP method gains better RD performance compared to other methods. Moreover, the TR for all of implemented methods are similar. This is because of the CPU becoming the bottleneck of the whole encoding process.

## CONCLUSION AND RECOMMENDATIONS

In this research, we presented our solution to the problem of parallelization of motion estimation in HEVC. Generally, our problem is how we can provide a high degree of parallelization in HEVC to reduce the execution time of HEVC. We saw that existing parallel tools in HEVC are not suitable for this purpose. Furthermore, motion estimation is the most time-consuming part of HEVC and parallelization of this part will permit more speedup compared to other parts. With focus on parallel motion estimation methods in literature, we looked at papers that present solutions for coarse and fine parallelization in the HEVC. Then, the shortcomings of previous works had been discussed.

We proposed a two-stage multi-predictor parallel framework, which is flexible, efficient and can provide a high degree of parallelism suitable for heterogeneous architectures. Furthermore, to compensate for the RD performance loss in parallel framework caused by breaking dependencies, we proposed a multiple temporal predictor rate-constrained motion estimation approach. This approach is using the multiple predictor concept for a list of temporal predictors. As we saw in the experimental results, this method can provide fine parallelism and good RD performance simultaneously.

Moreover, we proposed a well-designed suboptimal search method suitable for the GPU architecture. This method utilizes the GPU efficiently and can be used along the other proposed methods. Finally, we presented experimental results for our implemented methods compared with the state-of-the-art methods and the HM test model. We have published our results in two conference papers showing that our methods achieve a high degree of parallelization while RD performance is insignificantly affected (Hojati *et al.*, 2017a,b).



## APPENDIX I

### GPU PROGRAMMING WITH OPENCL

OpenCL is a software system that lets programmers write a portable program capable of using all resources available in some heterogeneous platform (Kowalik & Puźniakowski, 2012). An heterogeneous platform may include multi-core CPUs, one or more GPUs and other compute devices. In this section, we provide an overview on the concepts that are related to our work. More detailed information is available in (Kowalik & Puźniakowski, 2012) and (Advanced Micro Devices, 2011).

#### 1. OpenCL framework

In principle, OpenCL can be used also for programing homogeneous systems such as Intel multi-core processors, where one core can be a host and the others provide improved performance by parallel processing.

The idea of parallel computation based on heterogeneity is a system with two or more distinct types of processors. The ordinary CPU, called the **host**, is responsible for managing functions of one or more highly parallel processors called the **devices** (for example GPU devices). Devices are responsible for highly parallel processing that accelerates computation. These systems are called heterogeneous because they consist of very different types of processors. The fundamental advantage of OpenCL is its independence from the vendor hardware and operating systems.

Massive parallelism in OpenCL computing is rooted in the concept of the kernel C function. The kernel program is executed on a device (e.g. GPU). The name workitem, in more traditional terminology, means a thread. This kernel code is executed at each point in a data parallel problem domain.

#### Workgroups and workitems

A two-dimensional domain of workitems is shown in Figure-A I-1. Each small square is a workgroup that may have, for example,  $128 \times 128$  workitems. The significance of the workgroup is that it executes together. Also, workitems within a group can communicate for coordination.

#### OpenCL Execution Model

An OpenCL program runs on a host CPU that submits parallel work to compute devices. In a data parallel case, all workitems execute one kernel (a C function). In the more general cases, the OpenCL host program is a collection of kernels and other functions from the run time API

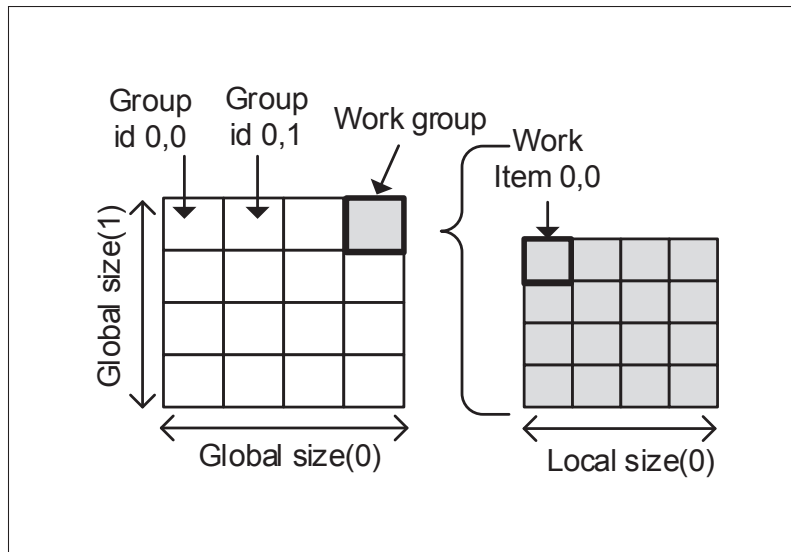


Figure-A I-1 OpenCL 2-D workitems domain.  
Adapted from (Kowalik & Puźniakowski, 2012)

library. The environment in which work items execute is called the context. It includes devices, memories and command queues.

The most frequently used scientific and engineering computing algorithms contain data parallel components. In data parallel computation, one kernel is run on multiple data sets. In this case, OpenCL creates an ID for each work group running on a compute unit (CU) and also an ID for each workitem running on a processing element (PE). The kernel will perform on a portion of data according to the IDs.

### Using OpenCL

The host is usually programed using C/C++ and OpenCL Runtime APIs. The devices run kernels written in the OpenCL. To be able to use the OpenCL, all the elements should be configured correctly. The OpenCL library should be properly installed and available for the corresponding hardware vendor. OpenCL libraries are usually included in hardware drivers. OpenCL application does not need any special compiler. The preferred language for writing OpenCL applications is C or C++, but it is also possible to use this technology with any language that supports linking to binary libraries. There are a number of wrapper libraries for many other languages.

OpenCL programs are usually compiled by a compiler included in drivers during application runtime. There is also the possibility to store a compiled program, but it will work only on the hardware it was compiled for. This functionality is intended for caching programs to speedup application startup time and for embedded solutions.

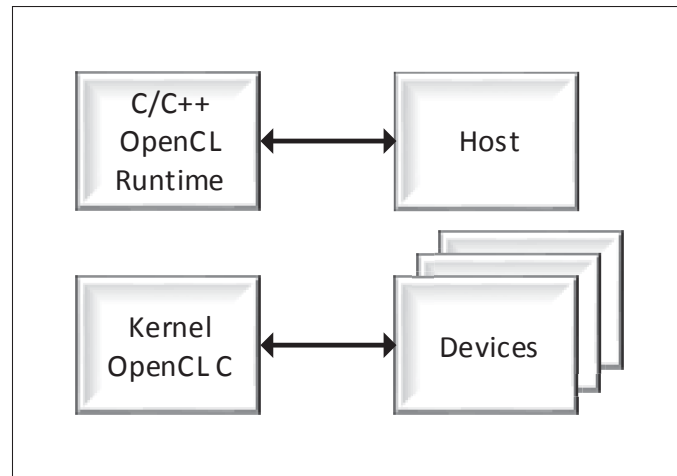


Figure-A I-2 Host and device programming languages. Adapted from (Kowalik & Puźniakowski, 2012, p.31)

The first step in an OpenCL application is to determine which platforms are present. This is performed by querying the platforms at the runtime. It is possible to have many different OpenCL implementations installed on a single system. The application should select the desired OpenCL platform by matching the platform vendor string to the vendor name, for instance, “Advanced Micro Devices, Inc.”.

### Compiling the Host Program

In order to compile the host program on an AMD-based system, the latest AMD Accelerated Parallel Processing SDK (AMD APP SDK) must be installed. The SDK provides all the necessary OpenCL runtime headers and libraries that are required by the host compiler. An environmental variable named AMDAPPSDKROOT is set by the SDK installer. This is the path of the directory in which the AMD APP SDK is installed. Moreover, in the install directory the “include” and “lib” sub-folders, are the runtime headers and libraries respectively. Also, two sub-directories are placed in the “lib”. One library targeted for 32-bit applications, and another for 64-bit applications.

The aforementioned headers and libraries must be included in the project by selecting the applicable setting for the targeted operating system, IDE, and compiler.

#### *Compiling On Windows:*

To compile an OpenCL application on Windows, we have used Visual Studio. According to the settings below, all C++ files must be added to the project.

- Project Properties → C/C++ → Additional Include Directories.

These must include `$(AMDAPPSDKROOT)/include` for OpenCL headers. Optionally, they can include `$(AMDAPPSDKSAMPLESROOT)/include` for SDKUtil headers.

- Project Properties → Linker → Additional Library Directories

Include `$(AMDAPPSDKROOT)/lib/x86` for OpenCL libraries for 32-bit and `$(AMDAPPSDKROOT)/lib/x86_64` for 64-bit.

Moreover, we can include `$(AMDAPPSDKSAMPLESROOT)/lib/x86` for SDKUtil libraries.

- Project Properties → Linker → Input → Additional Dependencies These should include `OpenCL.lib`. Also, it can include `SDKUtil.lib`.

### *Compiling on Linux*

On Linux, gcc or the Intel C compiler must be installed to compile OpenCL applications. Compiling and linking are performed as explained below.

1. Compile all the C++ files (for instance, `Template.cpp`), and get the object files.

For 32-bit object files on a 32-bit or 64-bit object files on 64-bit system:

```
g++ -o Template.o -c Template.cpp -I $AMDAPPSDKROOT/include
```

For building 32-bit object files on a 64-bit system:

```
g++ -o Template.o -c Template.cpp -I $AMDAPPSDKROOT/include
```

2. Link all the generated object files to the OpenCL library and create an executable file.

For linking to a 64-bit library:

```
g++ -o Template Template.o -lOpenCL -L$AMDAPPSDKROOT/lib/x86_64
```

For linking to a 32-bit library:

```
g++ -o Template Template.o -lOpenCL -L$AMDAPPSDKROOT/lib/x86
```

### **AMD GCN hardware overview**

A general OpenCL model for GPU device consists of compute units. Each compute unit can have multiple processing elements (PE). A workitem is executed on a single processing element. The processing elements within a compute unit will execute in lock-step using SIMD execution. However, compute units can execute operations independently (see Figure-A I-3). Different GPU compute devices might have different capabilities (e.g. different number of compute units). However, typically they have a similar design pattern.

As mentioned, AMD GPUs have multiple compute units. The number of compute units and their structure varies with the device family, as well as device designations within a family. Moreover, each of processing elements include ALUs. For the Northern Islands and Southern Islands device families, the ALUs are organized in four processing elements with arrays of 16 ALUs. For each block of 16 workitems, one of these arrays executes a single instruction across each lane. By repeating the instruction over four cycles a wavefront of 64-element vector is executed.



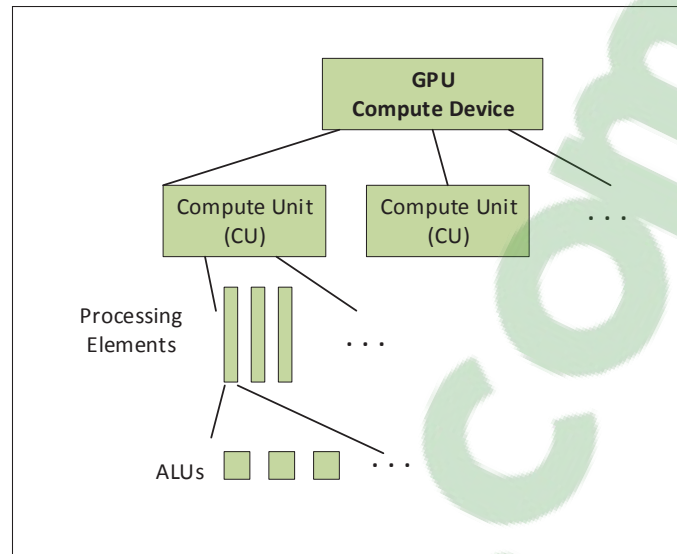


Figure-A I-3 Generalized AMD GPU Compute Device Structure. Adapted from (Advanced Micro Devices, 2011)

On the mentioned family devices, instructions of a wavefront are executed on the PE arrays so that each work-item issues instructions at once in a very-long-instruction-word (VLIW) packet. The branch operation is controlled with instruction and Control Flow module of a processing element.

Based on the aforementioned structure of GPU, all processing elements within a vector unit, execute the same instruction in each cycle. For a typical instruction, 16 processing elements execute the same instruction for 64 workitems over 4 cycles. A block of workitems that are executed together is a wavefront. However, the size of wavefronts can differ on different GPU compute devices according to the manufacturer design.

It is possible for different compute units to execute different instructions because compute units operate independently of each other. Moreover, the different vector units within a compute unit can execute different instructions.

Before discussing flow control, it is necessary to clarify the relationship of a wavefront to a workgroup. If a user defines a workgroup, it consists of one or more wavefronts. A wavefront is a hardware thread with its own program counter. It is capable of following control flow independently of other wavefronts. A wavefront consists of 64 or fewer workitems. The mapping is based on a linear workitem order. On a device with a wavefront size of 64, workitems 0-63 map to wavefront 0, workitems 64-127 map to wavefront 1, etc. For optimum hardware usage, an integer multiple of 64 workitems is recommended.

**Flow Control**

Flow control, such as branching, is achieved by combining all necessary paths as a wavefront. If workitems within a wavefront diverge, all paths are executed serially. For example, if a workitem contains a branch with two paths, the wavefront first executes one path, then the second path. The total time to execute the branch is the sum of each path time. An important point is that even if only one workitem in a wavefront diverges, the rest of the workitems in the wavefront execute the branch. The number of workitems that must be executed during a branch is called the branch granularity. On AMD hardware, the branch granularity is the same as the number of workitems in a wavefront.

## APPENDIX II

### PERFORMANCE MEASURES

Performance of an algorithm can be measured according to several aspects. For instance, memory usage and execution time are typical measures. However, for video coding algorithms, the RD performance should also be considered as an important metric. The resulting RD of an encoding process is affected by the complexity of the algorithm (Bossen *et al.*, 2012). In other words, one algorithm is better than the other if it achieves a similar RD performance in less execution time. In the following section, we explain the main metric for the RD performance measurement.

#### 1. Bjøntegaard Delta Measurements (BD)

It is very useful to have just one number to compare two rate-distortion curves for two different methods of coding (Wien, 2015). To this end, a method is proposed in (Bjøntegaard, 2001, 2008) which has been used in the literature for comparing different video compression methods. In this method, two metrics are proposed: The first one is BD-Rate which shows the rate savings. The second one is BD-PSNR which shows the PSNR improvement of one method over the other one.

To compute the mentioned metrics, in the original proposal, the performance of the two encoding schemes need to be given at four different quantization settings. Then the curves are achieved by interpolating these four points. After that, an integration is performed over the difference of two curves. It should be noted that the calculation is done based on the logarithmic rate to obtain the relative rate difference. The PSNR can be applied directly since it is already in the logarithmic scale.

Figure-A II-1 shows the plots which are used for the BD measurements. The rate axis is linear in these figures. To draw the complete curves an interpolation is used. To do the BD calculations the rate axis is logarithmized. The interpolation is an advanced method based on overlapping points. The vertical and horizontal differences are used to compute the BD-PSNR and BD-Rate respectively (Bjøntegaard, 2008).

In the initial method, a 3-rd order polynomial was used for BD calculations by using the four measurement points (Bjøntegaard, 2001). This method is more effective when used with relatively small quantizer step sizes (e.g. with  $\Delta QP$  of three between the four rate points). This is because the rate-distortion curves are almost liner within a given rate range and can be approximated by a polynomial. Although this is a good approximation for smaller quantization steps, using logarithmic rate axis leads to better results for bigger quantization steps (Zhao *et al.*, 2008). Moreover, JCT-VC suggests to use a piece-wise cubic interpolation to compute

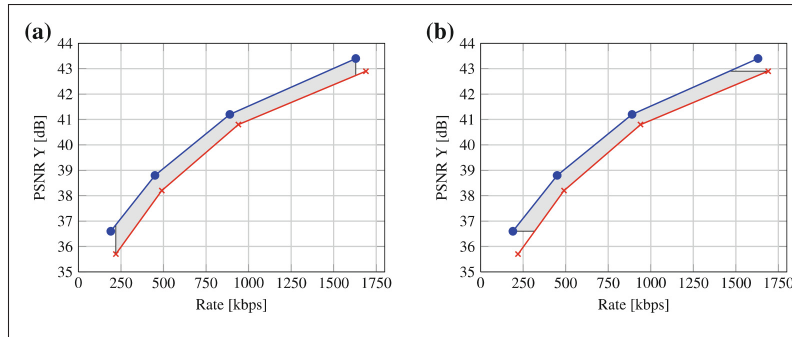


Figure-A II-1 Rate-distortion plot used for computing  
a) BD-PSNR and b) BD-Rate

the BD metrics (Wang *et al.*, 2011). Consequently, the BD-Rate is commonly used in the literature to compare the RD performance of different methods.

## APPENDIX III

### COMMON TEST SEQUENCES

<b>Class (Resolution)</b>	<b>Sequence name</b>	<b>Frame count</b>	<b>Frame rate</b>	<b>Bit depth</b>
A (2560x1600)	PeopleOnStreet	150	30fps	8
B (1920x1080)	Kimono	240	24fps	8
B (1920x1080)	ParkScene	240	24fps	8
B (1920x1080)	Cactus	500	50fps	8
C (832x480)	RaceHorses	300	30fps	8
C (832x480)	BQMall	600	60fps	8
C (832x480)	PartyScene	500	50fps	8
C (832x480)	BasketballDrill	500	50fps	8
D (416x240)	RaceHorses	300	30fps	8
D (416x240)	BQSquare	600	60fps	8
D (416x240)	BlowingBubbles	500	50fps	8
D (416x240)	BasketballPass	500	50fps	8
E (1280x720)	FourPeople	600	60fps	8
E (1280x720)	Johnny	600	60fps	8



## LIST OF REFERENCES

- Joint Collaborative Team on Video Coding Reference Software, ver. HM 15.0. Consulted at <http://hevc.hhi.fraunhofer.de/>.
- Advanced Micro Devices. (2011). AMD Accelerated Parallel Processing OpenCL Programming Guide. Consulted at <http://developer.amd.com/sdks/AMDAPPSDK/documentation>.
- Ahn, Y. J., Hwang, T. J., Sim, D. G. & Han, W. J. (2013). Complexity model based load-balancing algorithm for parallel tools of HEVC. *Visual Communications and Image Processing (VCIP)*, 2013, pp. 1–5.
- Bjøntegaard, G. (2001). *Calculation of average PSNR differences between RD-curves* (Report n°VCEG-M33). 13th Meeting: Austin, Texas, USA.
- Bjøntegaard, G. (2008). Improvements of the BD-PSNR model. *ITU-T 16/Q6, 35 VCEG Meeting, July 2008*.
- Blumenberg, C., Palomino, D., Bampi, S. & Zatt, B. (2013). Adaptive content-based Tile partitioning algorithm for the HEVC standard. *Picture Coding Symposium (PCS)*, 2013, pp. 185–188.
- Bossen, F. (2013). *Common test conditions and software reference configurations* (Report n°JCTVC-L1100). 12th Meeting: Geneva.
- Bossen, F., Bross, B., Suhring, K. & Flynn, D. (2012). HEVC complexity and implementation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1685–1696.
- Bross, B., Han, W. J., Ohm, J. R., Sullivan, G. J., Wang, Y. K. & Wiegand, T. (2013). *High Efficiency Video Coding (HEVC) text specification draft 10*. document JCTVC-L1003, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC).
- Chen, K., Duan, Y., Sun, J. & Guo, Z. (2014). Towards efficient wavefront parallel encoding of HEVC: parallelism analysis and improvement. *Multimedia Signal Processing (MMSP)*, 2014 IEEE 16th International Workshop on, pp. 1–6.
- Chen, W.-N. & Hang, H.-M. (2008). H. 264/AVC motion estimation implementation on compute unified device architecture (CUDA). *Multimedia and Expo, 2008 IEEE International Conference on*, pp. 697–700.
- Chen, Z., Xu, J., He, Y. & Zheng, J. (2006). Fast integer-pel and fractional-pel motion estimation for H. 264/AVC. *Journal of visual communication and image representation*, 17(2), 264–290.

- Cheung, N.-M., Au, O. C., Kung, M.-C., Wong, P. H. & Liu, C. H. (2009). Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(11), 1692–1703.
- Chi, C. C., Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S. & Schierl, T. (2012a). Parallel scalability and efficiency of HEVC parallelization approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1827–1838.
- Chi, C. C., Alvarez-Mesa, M., Juurlink, B., George, V. & Schierl, T. (2012b). Improving the parallelization efficiency of HEVC decoding. *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pp. 213–216.
- de Souza, D. F., Roma, N. & Sousa, L. (2014). OpenCL parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms. *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, pp. 755–759.
- Dong, S., Wang, Z., Wang, R., Wang, W. & Gao, W. (2013). Dynamic MB-level Scheduling for parallel video coding. *Picture Coding Symposium (PCS), 2013*, pp. 145–148.
- Gao, Y. & Zhou, J. (2014). Motion vector extrapolation for parallel motion estimation on GPU. *Multimedia tools and applications*, 68(3), 701–715.
- Heng, T. K., Asano, W., Itoh, T., Tanizawa, A., Yamaguchi, J., Matsuo, T. & Kodama, T. (2014). A highly parallelized H. 265/HEVC real-time UHD software encoder. *Image Processing (ICIP), 2014 IEEE International Conference on*, pp. 1213–1217.
- Hojati, E., Franche, J.-F., Coulombe, S. & Vázquez, C. (2017a). Massively parallel rate-constrained motion estimation using multiple temporal predictors in HEVC. *Multimedia and Expo (ICME), 2017 IEEE International Conference on*, pp. 43–48.
- Hojati, E., Franche, J. F., Coulombe, S. & Vázquez, C. (2017b). Highly Parallel HEVC Motion Estimation Based on Multiple Temporal Predictors and Nested Diamond Search. *International Conference on Image Processing (ICIP2017)*, pp. 2746-2750. doi: 10.1109/ICIP.2017.8296782.
- Jeong, J. H., Parmar, N. & Sunwoo, M. H. (2015). Enhanced test zone search algorithm with rotating pentagon search. *SoC Design Conference (ISOCC), 2015 International*, pp. 275–276.
- Kao, H.-C., Wang, I.-C., Lee, C.-R., Lo, C.-W. & Kang, H.-P. (2016). Accelerating HEVC Motion Estimation Using GPU. *Multimedia big data (bigmm), 2016 IEEE second international conference on*, pp. 255–258.
- Katsigiannis, S., Maroulis, D. & Papaioannou, G. (2013). A GPU based real-time video compression method for video conferencing. *Digital Signal Processing (DSP), 2013 18th International Conference on*, pp. 1–6.



- Khemiri, R., Kibeya, H., Sayadi, F. E., Bahri, N., Atri, M. & Masmoudi, N. (2017). Optimisation of HEVC motion estimation exploiting SAD and SSD GPU-based implementation. *Iet image processing*.
- Kowalik, J. & Puźniakowski, T. (2012). *Using OpenCL: Programming Massively Parallel Computers*. IOS Press.
- Koziri, M., Papadopoulos, P., Tziritas, N., Dadaliaris, A. N., Loukopoulos, T. & Khan, S. U. (2016). Slice-based parallelization in HEVC encoding: realizing the potential through efficient load balancing. *Multimedia signal processing (mmsp), 2016 ieee 18th international workshop on*, pp. 1–6.
- Kumar, V., Grama, A., Gupta, A. & Karypis, G. (1994). Introduction to parallel computing: algorithm design and analysis. Benjamin Cummings/Addison Wesley, Redwood City.
- Lee, C.-Y., Lin, Y.-C., Wu, C.-L., Chang, C.-H., Tsao, Y.-M. & Chien, S.-Y. (2007). Multi-pass and frame parallel algorithms of motion estimation in H. 264/AVC for generic GPU. *Multimedia and Expo, 2007 IEEE International Conference on*, pp. 1603–1606.
- Lin, Y.-C. & Wu, S.-C. (2016). Parallel motion estimation and GPU-based fast coding unit splitting mechanism for HEVC. *High performance extreme computing conference (hpec)*, pp. 1–7.
- Łuczak, A., Karwowski, D., Maćkowiak, S. & Grajek, T. (2012). Diamond scanning order of image blocks for massively parallel HEVC compression. *Computer Vision and Graphics*, 172–179.
- Ma, J., Luo, F., Wang, S. & Ma, S. (2014). Flexible CTU-level parallel motion estimation by CPU and GPU pipeline for HEVC. *Visual Communications and Image Processing Conference, 2014 IEEE*, pp. 282–285.
- Ma, S., Wang, S., Wang, S., Zhao, L., Yu, Q. & Gao, W. (2013). Low complexity rate distortion optimization for HEVC. *Data Compression Conference (DCC), 2013*, pp. 73–82.
- Meher, P. K., Park, S. Y., Mohanty, B. K., Lim, K. S. & Yeo, C. (2014). Efficient integer DCT architectures for HEVC. *IEEE Transactions on Circuits and systems for Video Technology*, 24(1), 168–178.
- Misra, K., Segall, A., Horowitz, M., Xu, S., Fuldseth, A. & Zhou, M. (2013). An overview of Tiles in HEVC. *IEEE Journal of selected topics in signal processing*, 7(6), 969–977.
- Momcilovic, S. & Sousa, L. (2009). Development and evaluation of scalable video motion estimators on GPU. *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pp. 291–296.
- Momcilovic, S., Ilic, A., Roma, N. & Sousa, L. (2014a). Collaborative inter-prediction on CPU+ GPU systems. *Image Processing (ICIP), 2014 IEEE International Conference on*, pp. 1228–1232.

- Momcilovic, S., Ilic, A., Roma, N. & Sousa, L. (2014b). Dynamic load balancing for real-time video encoding on heterogeneous CPU+ GPU systems. *IEEE Transactions on Multimedia*, 16(1), 108–121.
- Radicke, S., Hahn, J., Grecos, C. & Wang, Q. (2014). A highly-parallel approach on motion estimation for high efficiency video coding (HEVC). *Consumer Electronics (ICCE), 2014 IEEE International Conference on*, pp. 187–188.
- Rodríguez-Sánchez, R., Martínez, J. L., Fernández-Escribano, G., Sánchez, J. L. & Claver, J. M. (2012). A fast GPU-based motion estimation algorithm for H. 264/AVC. *International Conference on Multimedia Modeling*, pp. 551–562.
- Shafique, M., Khan, M. U. K. & Henkel, J. (2014). Power efficient and workload balanced tiling for parallelized high efficiency video coding. *Image Processing (ICIP), 2014 IEEE International Conference on*, pp. 1253–1257.
- Sullivan, G. J., Ohm, J.-R., Han, W.-J. & Wiegand, T. (2012). Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1649–1668. doi: {10.1109/TCSVT.2012.2221191}.
- Sze, V., Budagavi, M. & Sullivan, G. J. (2014). *High Efficiency Video Coding (HEVC) Algorithms and Architectures*. Springer.
- Tourapis, A. M. (2002). Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation. *Visual Communications and Image Processing*, 4671(May 2002), 1069–1079. doi: {10.1117/12.453031}.
- Vanne, J., Viitanen, M., Hamalainen, T. D. & Hallapuro, A. (2012). Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1885–1898.
- Wang, F., Zhou, D. & Goto, S. (2014). OpenCL based high-quality HEVC motion estimation on GPU. *Image Processing (ICIP), 2014 IEEE International Conference on*, pp. 1263–1267.
- Wang, J., Yu, X. & He, D. (2011). On BD-rate calculation. *Document JCTVC-F270, Torino*.
- Wang, X.-w., Song, L., Chen, M. & Yang, J.-j. (2013). Paralleling variable block size motion estimation of HEVC on multi-core CPU plus GPU platform. *Image Processing (ICIP), 2013 20th IEEE International Conference on*, pp. 1836–1839.
- Wiegand, T., Sullivan, G. J., Bjøntegaard, G. & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 560–576. doi: {10.1109/TCSVT.2003.815165}.
- Wien, M. (2015). *High efficiency video coding: Coding tools and specification*. Springer.

- Xue, Y.-g., Su, H.-y., Ren, J., Wen, M., Zhang, C.-y. & Xiao, L.-q. (2017). A Highly Parallel and Scalable Motion Estimation Algorithm with GPU for HEVC. *Scientific programming*.
- Yan, C., Zhang, Y., Dai, F., Zhang, J., Li, L. & Dai, Q. (2014a). Efficient parallel HEVC intra-prediction on many-core processor. *Electronics Letters*, 50(11), 805–806.
- Yan, C., Zhang, Y., Xu, J., Dai, F., Zhang, J., Dai, Q. & Wu, F. (2014b). Efficient parallel framework for HEVC motion estimation on many-core processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(12), 2077–2089.
- Yi, G., Shengsheng, Y., Man, P. L. & Jiazhong, C. (2010). Collocated macroblock based motion estimation for H. 264/AVC on GPU. *HKIE Transactions*, 17(3), 15–18.
- Yu, Q., Zhao, L. & Ma, S. (2012). Parallel AMVP candidate list construction for HEVC. *Visual Communications and Image Processing (VCIP)*, 2012 IEEE, pp. 1–6.
- Zhang, S., Zhang, X. & Gao, Z. (2014). Implementation and improvement of wavefront parallel processing for HEVC encoding on many-core platform. *Multimedia and Expo Workshops (ICMEW)*, 2014 IEEE International Conference on, pp. 1–6.
- Zhao, J., Su, Y. & Segall, A. (2008). On the calculation of PSNR and bit-rate differences for the SVT test data. *ITU-T SG16, COM16-C404-E*.
- Zhao, Y., Song, L., Wang, X., Chen, M. & Wang, J. (2013). Efficient realization of parallel HEVC intra encoding. *Multimedia and Expo Workshops (ICMEW)*, 2013 IEEE International Conference on, pp. 1–6.
- Zhou, J., Zhou, D., Fei, W. & Goto, S. (2013). A high-performance CABAC encoder architecture for HEVC and H.264/AVC. *Image Processing (ICIP)*, 2013 20th IEEE International Conference on, pp. 1568–1572.
- Zhou, M., Sze, V. & Budagavi, M. (2012). Parallel tools in HEVC for high-throughput processing. *Proc. of SPIE Vol.*, 8499, 849910–1.