PARALLEL OPTIMIZATION OF INTRA MODE SELECTION IN HEVC USING OPEN MP

by

PRATIK DEVENDRAKUMAR MEHTA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2014

Acknowledgements

Firstly I would like to thank Dr. Rao from the bottom of my heart for being my supervisor and his constant support and inspiration throughout the thesis. I would like to thank Dr. Alan Davis and Dr. Howard Russell for serving on my committee.

I would also like to thank Jamil-Ur-Rehman, Ye Zhang and also thesis by alumni of MPL lab Sudeep Gangavati and Tushar Saxena whose research has been an inspiration to my work.

I would also like to thank Rohit Rawat, Soumitro Auddy for their expert opinion. I am also thankful to my lab mates: Kushal Shah, Abhijith Jagannath, Harshdeep Jain, Sapna Vasudevan, Tuan Ho and Abhishek Thungaraj for their valuable inputs throughout the research.

Last but not least, I thank my father, Devendrakumar Mehta and my mother Asmita Mehta for encouraging me to take up thesis and for their continuous support and motivation.

October 27, 2014

Abstract

PARALLEL OPTIMIZATION OF

INTRA MODE SELECTION

IN HEVC USING

OPEN MP

Pratik Mehta, M.S

The University of Texas at Arlington, 2014

Supervising Professor: K.R.Rao

HEVC aims at providing an efficient video coding algorithm which not only provides high performance, but also improves coding efficiency. It also provides high compression efficiency as compared to its counter parts, but at the cost of an increase in complexity. The increased number of intra prediction modes in HEVC allows better compression and more flexible block representation. The purpose of the thesis is to optimize the mode decision process using Open MP to reduce the encoding time without further increase in complexity. This includes implementing a parallel computing method that will calculate 35 distinct intra prediction modes in parallel and compare them to decide the three best possible modes depending on RD cost values as compared to a single mode in H.264. There are over 11900 different intra modes according to different PB size as shown in Table 5-1 which provides higher level compression but at the cost of increase in the encoding time . Experimental results were compared with the HEVC reference software HM 9.1 for different standard test sequences with respect to various quantization parameters. The proposed algorithm was evaluated using different metrics. These include encoding time, BD- PSNR (Bjontegaard Delta Peak Signal to Noise Ratio) and BD-bitrate (Bjontegaard Delta bitrate), rate distortion (RD curve) graph and percentage reduction in encoding time. There is a 22-

57% reduction in encoding time compared to reference software calculations with negligible increase in bitrate and negligible decrease in PSNR.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

1.1 Video Compression basics

A video is basically a group of images which in turn is nothing but group of pixels. Multimedia files are large and consume lots of disk space. The file size makes it time consuming to move them from place to place over networks or distribute them over the internet. The basic use of video compression is to shrink video files and make them smaller and more practical to store and share. H.264 [24] and HEVC [1][11][16][17] are popular examples. The video compression works by removing repetitious or redundant information, effectively summarizing the contents of a file in a way that preserves as much of the original data as possible [20]. Consider a sequence of image as shown in Fig 1-1 of a person walking on a street with an umbrella [19]. The uncompressed video contains information for every pixel, in every frame while compressed video contains less information because similar pixels are grouped together. Therefore by recognizing that all pixels in the background remain the same and only the person with umbrella is in motion, the compressed video significantly reduces the file size.

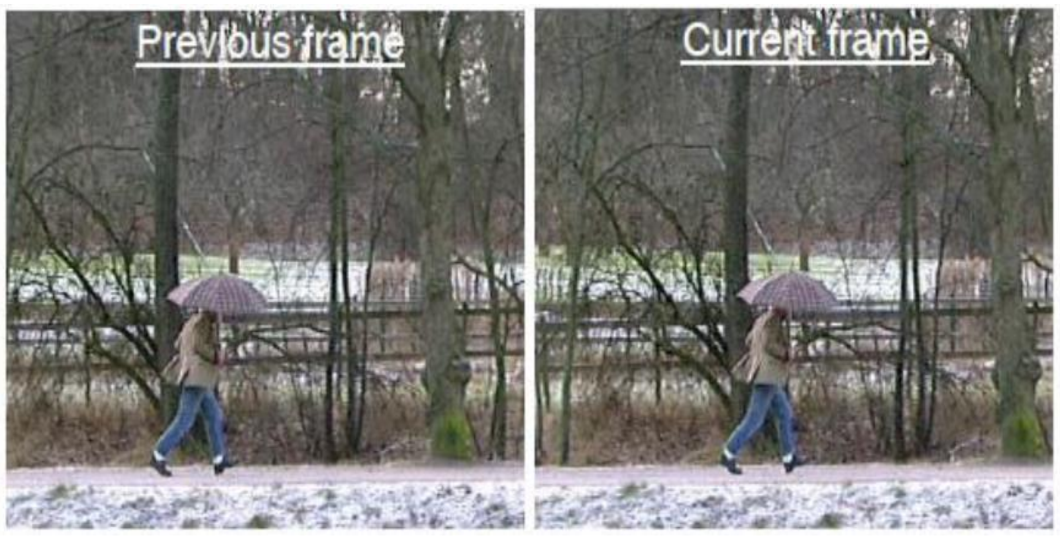


Figure 1-1 Similarity of successive pictures [42]

The modern video technology provides an extremely high image quality with moderate compression compared to the technologies that were used in the past. Modern data compression techniques offer the possibility to store or transmit the vast amount of data necessary to represent videos and images in an efficient and robust way. Fig. 1-2 shows the reduced time interval in which the compressed video or image file can be transmitted as compared to the uncompressed file while taking into consideration the bandwidth limitation.

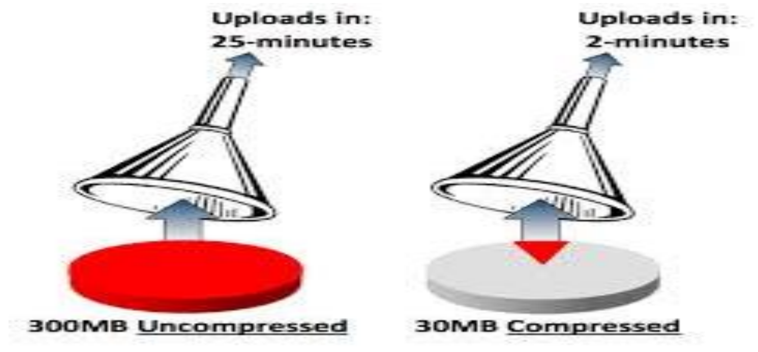


Figure 1-2 Basics of video compression [19]

Video is basically a collection of frames that are displayed quickly in succession such that the user gets the feeling of movement of images in real time. A typical video file contains image, audio and metadata. Each of these properties can be compressed, since all of them are made up of 0's and 1's. Video is transmitted as electrical signals which move around via air i.e. radio waves, microwaves, etc. or via cable i.e. HDMI, co-axial cables, etc. However as shown in Fig. 1-2 the amount of signal that can be transmitted is limited by the bandwidth of the medium through which it is transmitted [21]. Therefore different compression techniques have been developed to compress the signal effectively without much reduction in quality.

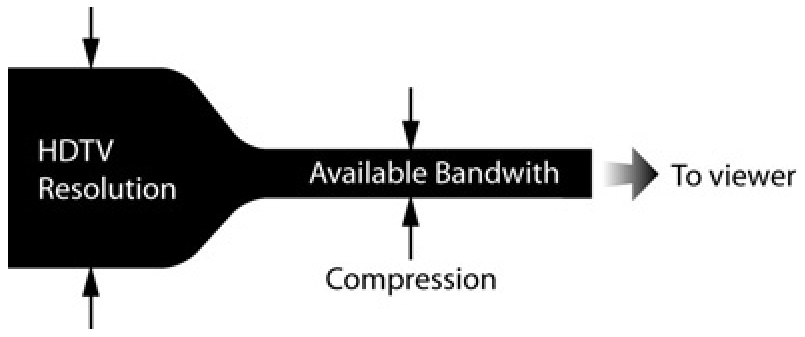


Figure 1-3 Bandwidth requirements [19]

As shown in Fig. 1-3 video coding has evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 [29] and H.263 [30]; ISO/IEC produced

MPEG-1 [31] and MPEG-4 visual [32]. The two organizations jointly produced the H.262/MPEG-2 [33] video and H.264/MPEG-4 AVC [34] standards. The most recent project of the ITU-T VCEG and ISO/IEC has been released called HEVC. These evolutions have enabled maximizing compression capability and are improving other characteristics such as data loss robustness.

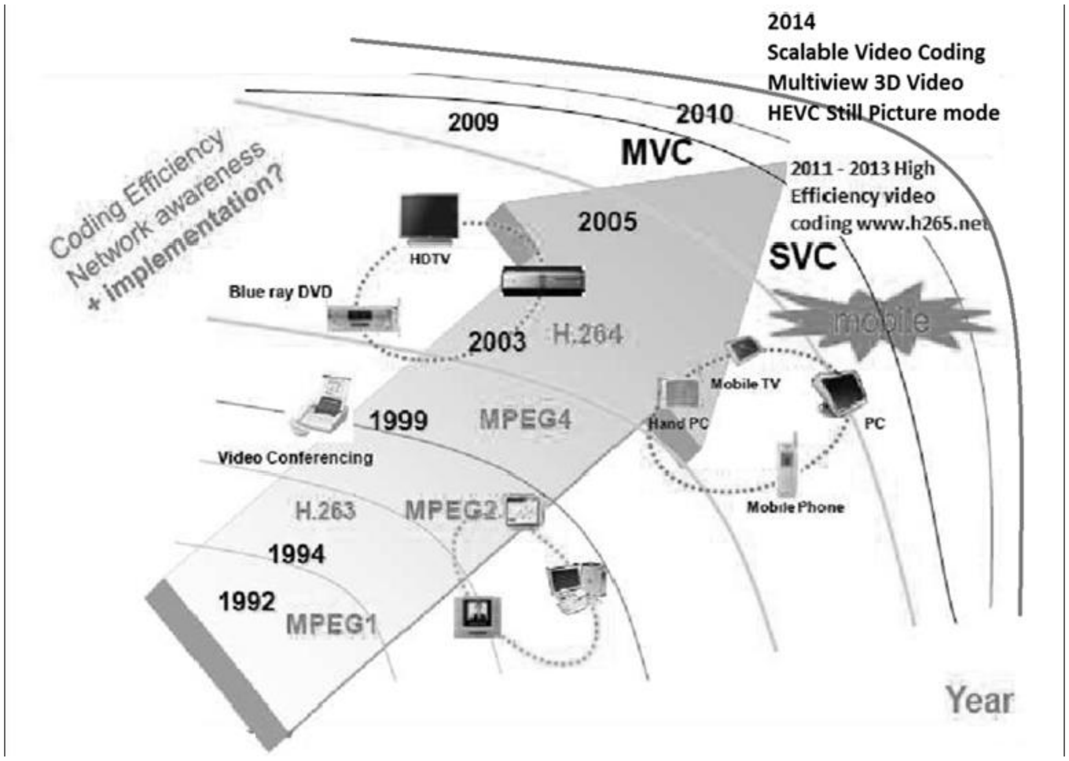


Figure 1-4 Evolution of video coding standards [41]

Video is an electronic medium for the recording, copying and broadcasting of moving visual images. However, an increasing demand of HD videos is creating even a stronger needs for higher coding efficiency standards. Moreover the traffic caused by video applications targeting mobile devices and tablet-PCs is imposing a severe challenge on today's networks.

1.2 Summary

The overall chapter gives basic introduction about video coding technique and its application. It also explains the evolution in video coding standards in brief. The next chapter will introduce HEVC and its basic components in detail.

1.3 Thesis outline

The following chapters will explain further about HEVC and proposed algorithm in detail. Chapter 2 will explain about different blocks in HEVC and their significance. Chapter 3 explains intra prediction technique in detail. Chapter 4 will concentrate of need of parallel programming technique and will give brief overview of Open MP technique. Chapter 5 will explain about actual implementation of parallel programming technique for intra prediction in HEVC. It will also provide comparative analysis of proposed algorithm with reference software HM 9.1 [11]. Chapter 6 outlines the conclusion and further possible research.

Chapter 2

High Efficiency video coding

In April 2010, a Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T ISO/IEC started its work on a new standard for High Efficiency Video Coding (HEVC). HEVC has been designed to address essentially all existing applications of H.264 [27] and to particularly focus on issues such as increased use of parallel processing techniques and increased video resolution. Three profiles, namely main, main intra and main 10 bit profiles have been finalized as the final draft international standard (FDIS) by JCT-VC in Jan. 2013. Apart from that various extensions such as 3D video, scalable video coding (SVC) are under development. While the highest performance gain also comes with associated high complexity requirements, just marginally lowering performance also brings high coding gains [9][4][6]. Coding gains in HEVC are due to both advanced inter and intra predictions.

HEVC [1][11][16][17] implements the same hybrid approach as H.264 [27] which includes both temporal and spatial predictions. It aims at 50% compression gain over H.264 while maintaining similar video quality [5]. It requires half the bandwidth compared to H.264 for high quality video transmission. H.264 divides the image into 16 by 16 pixels, while HEVC divides the image into varying block sizes up to 64 by 64 pixels. This larger block size typically enables better compression. Various features are introduced in the HEVC standard to enhance parallel processing capability or to modify the structuring of slice data for packetization purposes [2]. Either the image is divided into various parts like tiles in which the processor works on one of them, or the wave front method where each processor handles one line of blocks in the image, or make use of a dependent slice method in which dependent slice can only be decoded if part of a previous slice has been decoded.

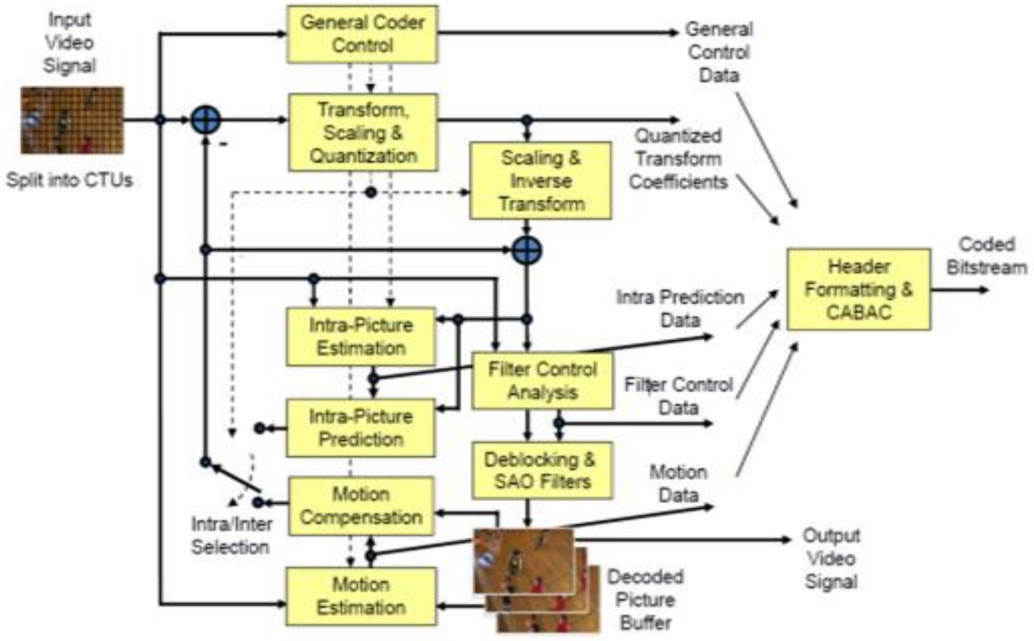


Figure 2-1 Encoder block diagram of HEVC [1]

As shown in Fig 2-1, theFigure 2-1 Encoder block diagram of HEVC [1] standard shows block diagram of HEVC encoder in which each picture is partitioned into blocks of different sizes and the same is conveyed to the decoder. In the given sequence intra prediction is applied to the very first picture which uses spatial redundancy of the picture while for rest of the frames temporal redundancy is exploited using inter prediction [1].

The encoding process of inter-picture prediction consists of choosing motion data which comprises of the selected reference picture and motion vector (MV) to be applied for predicting the samples of each block [1]. The encoder and decoder generate identical inter prediction signals by applying motion compensation (MC) using the motion vector (MV) and mode decision data, which are transmitted as side information.

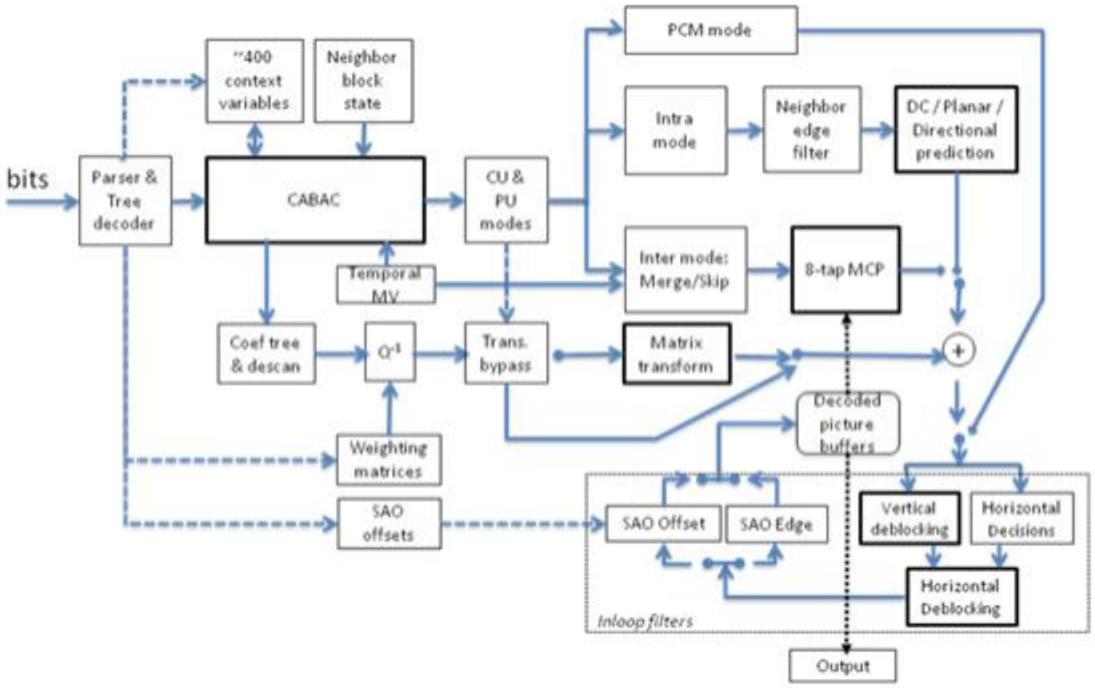


Figure 2-2 Decoder block diagram of HEVC [1]

The residual signal of intra/inter prediction which is the difference between the original and predicted blocks is further transformed by a linear spatial transform which is scaled, quantized, entropy coded and transmitted along with prediction information. This residual signal is also inverse transformed, inverse quantized and filtered to duplicate the decoder processing loop and added with the predicted signal to produce a decoded picture which is stored in the buffer for further predictions. As shown in Fig. 2-2 in the block diagram of the HEVC decoder, the residual signal is added to the prediction, and the result is fed to the deblocking filter [40] to reduce the visual artifacts and finally stored in decoded picture buffer which can be used for further decoding of remaining pictures [1].

A non-linear amplitude mapping is introduced in the inter-prediction loop after the deblocking filter called the sample adaptive offset (SAO). The goal is to improve the reconstruction of the original signal amplitudes. It uses a look-up table that is described by a few parameters which can be described by histogram analysis at the encoder side [1].

HEVC basically uses the YCrCb color space with 4:2:0 color format and 8 bps (bits per color sample). As shown in Fig. 2-3 the sampling of 4:2:0 in which pixels are subsampled in both horizontal and vertical dimensions by a factor of 2. Theoretically, the chroma pixel is positioned between the rows and columns.

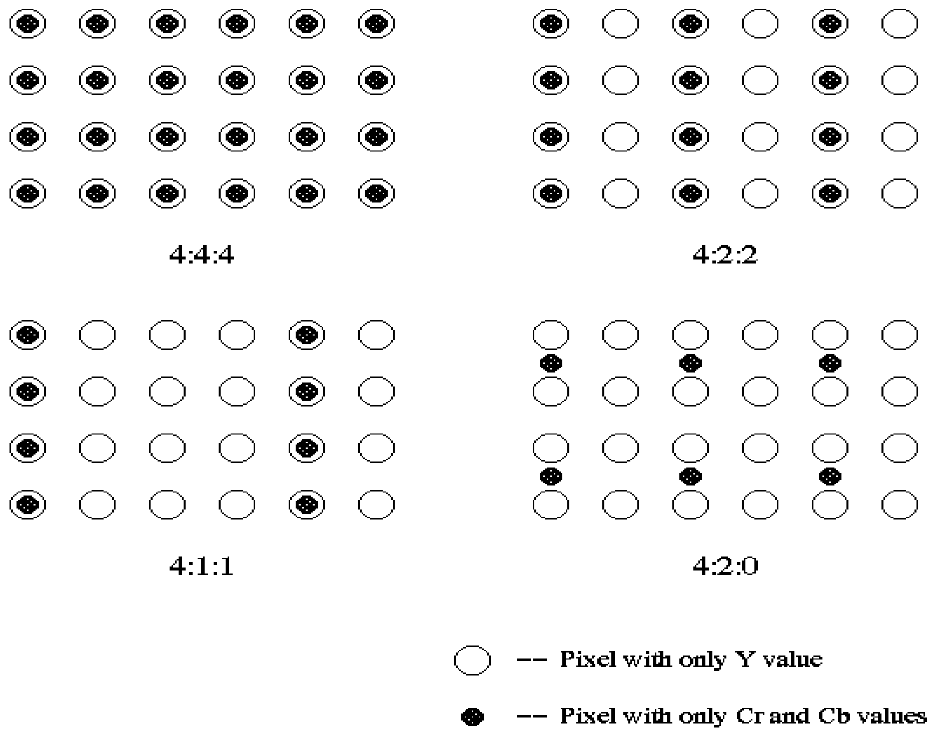

4:4:4
4:2:2
4:1:1
4:2:0
-- Pixel with only Y value
-- Pixel with only Cr and Cb values

Figure 2-3 Chroma subsampling [38]

The 4:2:0 chroma format supported in the version 1 profiles has chroma information that is half resolution both in the horizontal and vertical dimensions. This has been typical for consumer entertainment use, but the demands of higher-quality applications and screen content coding require use of 4:4:4 format with full-resolution chroma representations, or of the 4:2:2 format in which half-resolution horizontal but full-resolution vertical chroma sampling is used [43].

2.1 Quad based tree structure

Unlike H.264 which contains 16 by 16 size macroblocks, HEVC employs a quad tree structure which contains a coding tree unit (CTU), size of which is selected by the encoder and can be larger than the traditional macroblock.

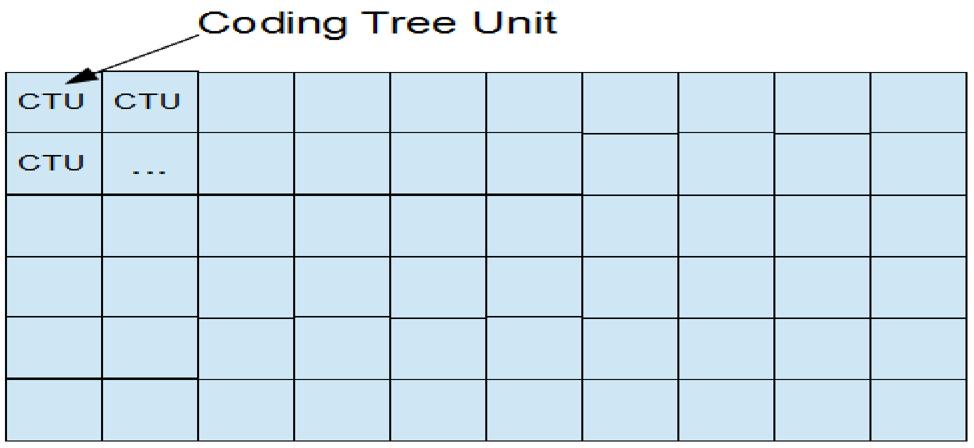


Figure 2-4 Division of an image into CTU [13]

Fig.2-4 shows the basic division of image into multiple CTUs. The width and height of CTU are signaled in a sequence parameter set hence all the CTUs in a video sequence have the same size i.e. 64 by 64, 32 by 32, or 16 by 16 [13] as shown in *Fig. 2-5*.

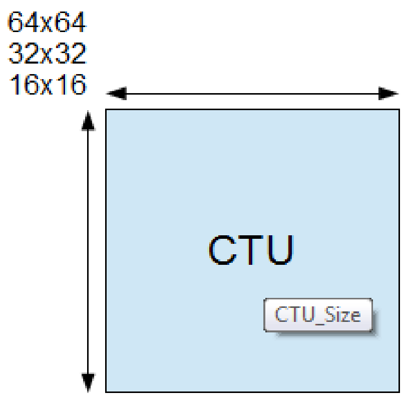


Figure 2-5 Different sizes of CTU [13]

Each coding unit basically consists of luma and chroma prediction blocks and each block is called a coding tree block (CTB). This has the same size as CTU. However, CTBs are too big to decide the type of prediction method to be used. So CTBs are further divided into coding blocks (CB), at which decision is taken whether to perform inter-picture or intra-picture prediction [13]. CBs are good enough for prediction type decision but too large to store motion vectors. Thus each CB can be split into prediction blocks (PB) differently depending on the temporal and/ or spatial predictability.

In either case, the prediction residual i.e. the difference between the original input signal and the predication signal, is transform coded using a variable block size integer DCT (Discrete cosine transform). Note that according to the residual quad tree (RQT), the coding blocks can be further subdivided into smaller transform blocks, such that the block sizes for prediction and for DCT coding do not have to be the same. This is shown by Fig.2-6 in which for block labeled 7 with transform block sizes in the range of 4x4 to 32x32 for both the luma and chroma components are supported. The transform kernel for each supported transform block size is given by a separate integer approximation of the 2D DCT-II (type-II Discrete Cosine Transform) of the corresponding block size [36].

As shown in Fig. 2-6 division of CTU into CU (square shaped blocks) which are further subdivided into prediction unit (square/non square shaped blocks) and transform units (square shaped blocks). The divisions of the 4kx2k block in Fig 2-7 sequence *Traffic* [4] shows the coding block (white) and nested transform block (red) structure resulting from recursive quadtree partitioning.

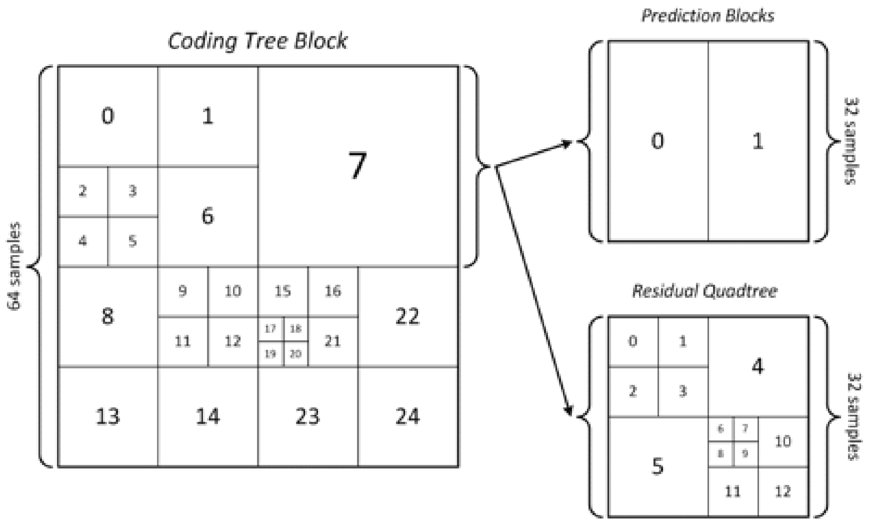


Figure 2-6 Prediction blocks and residual quadtree structure [36]

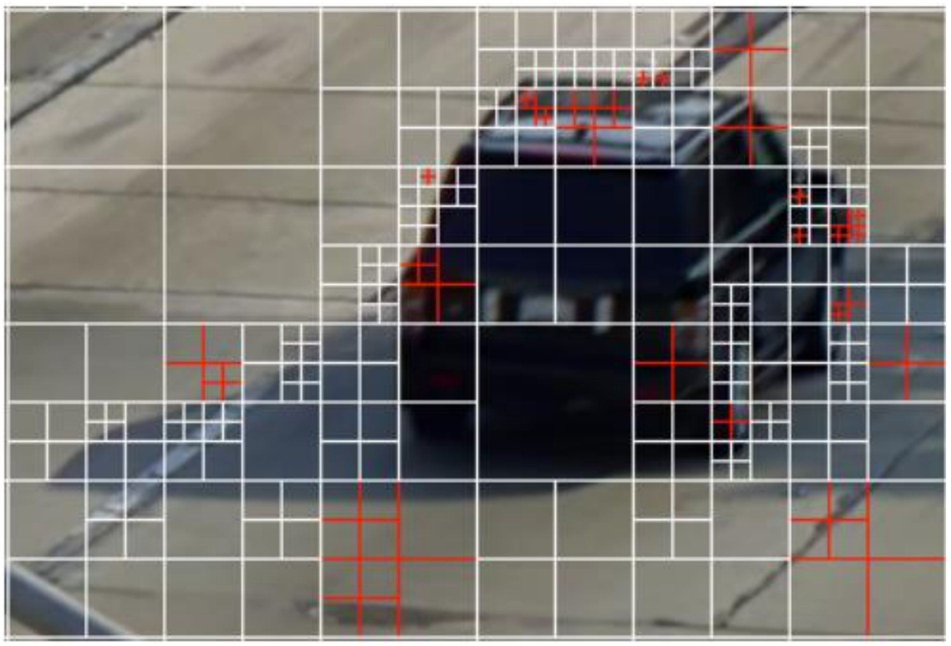


Figure 2-7 Example of division of image into quadtree coding structure [4]

2.2 Transform scaling and Quantization

Fig. 2-1 shows the residual signal is obtained by taking the difference between the input pixels and their predicted values. A two-dimensional transform of the residue is computed, then the transform values (coefficients) are quantized and then entropy encoded. The decoder performs the reverse operation.

The two-dimensional transform is usually a DCT operation as it concentrates most of the signal energy in the low index AC coefficients. After quantization, most of the small magnitude AC coefficients become zero, resulting in a sparse matrix of quantized coefficients. The 2-D matrix is reordered into a 1-D vector with a zigzag mapping as shown in Fig. 2-8, so that the vector has all the non-zero coefficients at the beginning followed by a long tail of zero coefficients. The entropy encoder can then efficiently compress the information [39]

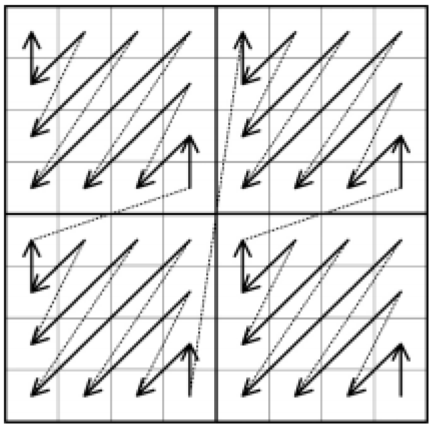


Figure 2-8 Diagonal scan pattern in 8x8 TB [56]

2.3 Entropy coding

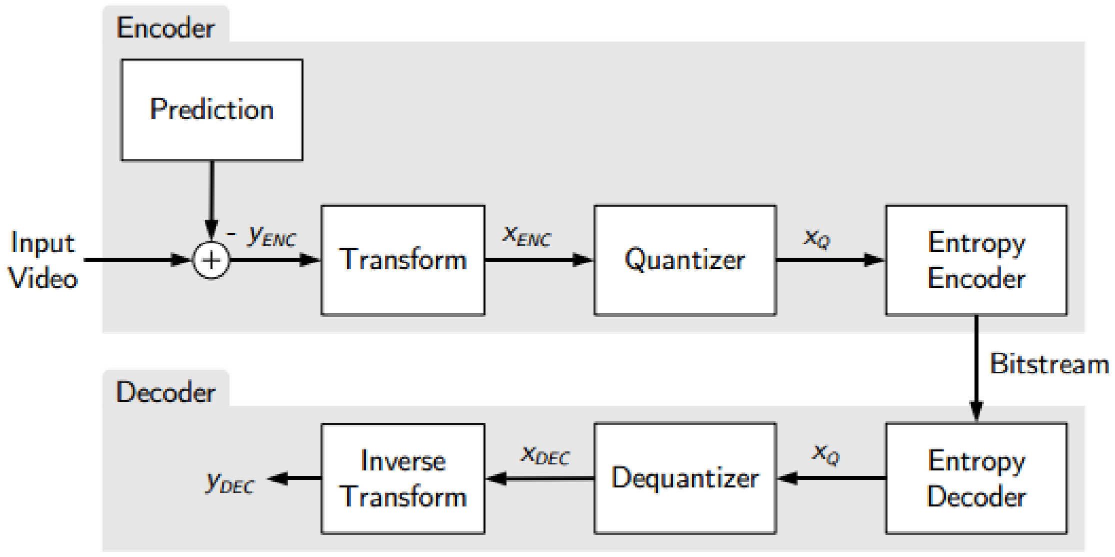


Figure 2-9 Example of division of image into quadtree coding structure [39]

Context adaptive binary arithmetic coding (CABAC) is used for entropy coding [1]. The algorithm of context-based adaptive binary arithmetic coding (CABAC) has been developed within the joint standardization activities of ITU-T and ISO/ IEC for the design and specification of the video coding standard H.264/AVC. During the later stage when the scalable extension of H.264 / AVC or HEVC was designed, another feature of CABAC has proven to be useful.

CABAC has been adopted as normative part of H.264 /AVC as well as of the HEVC draft standard. The other method specified in H.264/AVC [27] is a low-complexity entropy-coding technique based on the usage of context-adaptively switched sets of variable-length codes, so-called Context-Adaptive Variable-Length Coding (CAVLC). Compared to CABAC, CAVLC offers reduced implementation costs at the price of lower compression efficiency. For TV signals in standard- or high-definition resolution, CABAC typically provides bit-rate savings of 10-20% relative to CAVLC at the same objective video quality. Note that for the HEVC draft, CABAC is the only entropy coding method [37]. The CABAC scheme has undergone several improvements to improve its throughput speed especially for parallel processing architectures and its compression performance, and to reduce its context memory requirements [1]

The design of CABAC involves the key elements of binarization, context modeling, and binary arithmetic coding.

2.2 Deblocking filter [40]

In a coding scheme that uses block-based prediction and transform coding, discontinuity can occur in the reconstructed signal at the block boundaries. Visible discontinuities at the block boundaries are known as blocking artifacts. A major source of blocking artifacts is the block-transform coding of the prediction error followed by coarse quantization. Moreover, in a motion compensated prediction process, predictions for adjacent blocks in the current picture might not come from adjacent blocks in the previously coded pictures, which create discontinuities at the block boundaries of prediction signals. Similarly, when applying intra prediction, the prediction process of adjacent blocks might be different causing discontinuities at the block boundaries of the prediction signal.

The HEVC draft standards define two in-loop filters that can be applied sequentially to the reconstructed picture. The first one is the deblocking filter and the second one is the sample adaptive offset filter (SAO) that is currently included into the main profile.

The deblocking filter in HEVC has been designed to improve the subjective quality while reducing the complexity. It constitutes a major part of decoder complexity in H.264/AVC standard. Therefore special consideration has been taken to maintain the subjective and objective quality while reducing decoder complexity.

2.3 Slices, Tiles and Wavefront parallel processing

H.264/ AVC support slices, which were introduced mainly to prevent loss of quality in case of transmission errors and also to parallelize decoder. A slice can either be an entire picture or a region of picture as shown in part (a) of Fig. 2-10. It can be decoded independently from other slice of the same picture. However, if more number of slices is used to increase parallelism, it can incur significant coding losses. Also the number of slices is determined by the encoder and if the decoder relies on slices to obtain real-time performance, it may not achieve this if it receives a video sequence with only one or few slices per frame.

In the HEVC standard the introduction of a tile was yet another innovation compared to the H.264/AVC standard. Tiles divide the frame into a grid of rectangular regions that can independently be encode or decoded as shown in part (b) of Fig. 2-10. Unlike slices, the tiles increase the compression ratio and also the losses in compression efficiency at the boundaries is negligibly small. The use of tiles thus opens up new possibilities to increase encoding and decoding of video data using parallel processing on modern multi-core desktop and mobile platforms. Tiles are independently decodable regions of picture that are encoded with some shared header information. Hence they provide parallelism at a more coarse level and no sophisticated synchronization of threads is necessary for their use [1]. Tiles are basically zero overhead slices since it needs to send tile information once for sequence based on resolution. In the case of slices, the header needs to be sent at every slice, and it can constitute as an overhead to the bitrate in low to medium bitrate use cases.

Wavefront parallel processing (WPP) divides the slides into rows of CTUs. The first row is processed in the ordinary way; the second row can begin to process only after a few decisions have been made for the first row and so on as shown in part (c) of Fig. 2-10. It may provide better compression compared to the tiles and also avoid some visual artifacts that may be incurred by tiles [1].

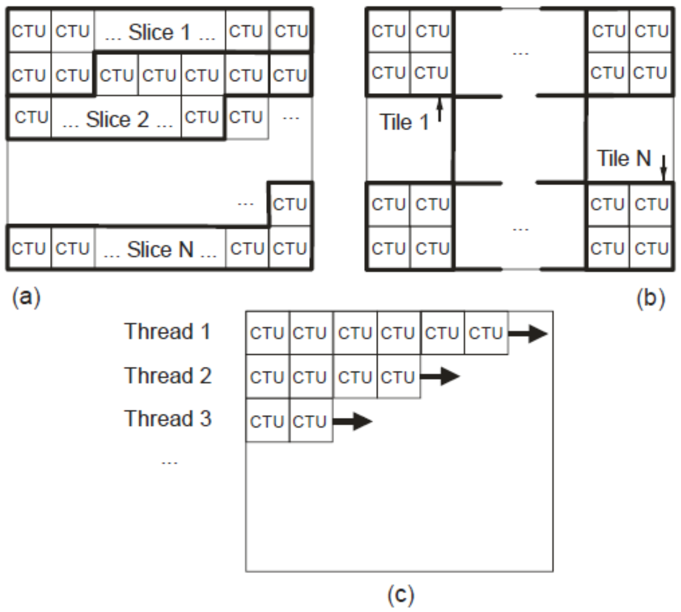


Figure 2-10 Subdivision of picture into (a) slices (b) tiles and (c) illustration of wavefront parallel processing [1]

2.4 Intra prediction

In H.264/AVC, intra coding is based on spatial extrapolation of samples from previously decoded image blocks, followed by discrete cosine transform (DCT) based transform coding. HEVC utilizes the same principle, but further extends it to be able to efficiently represent wider range of textural and structural information in images.

Intra Prediction Mode	Associated Names
0	Planar
1	DC
2..34	Angular (N), $N = 2 \ldots 34$

Figure 2-11 Intra prediction modes in HEVC [3]

There is also emphasis on avoiding introduction of artificial edges with potential blocking effects. This is achieved by adaptive smoothening of the reference samples and smoothening the generated prediction boundary samples and smoothening the generated prediction boundary samples for DC and directly horizontal and vertical modes. As shown in Fig. 2-12, the intra prediction in HEVC has 33 different angular modes, while mode 0 and mode 1 are termed as the planar and the DC mode respectively [3].

Since the main focus of this thesis is on intra prediction it is explained in further detail in next chapter.

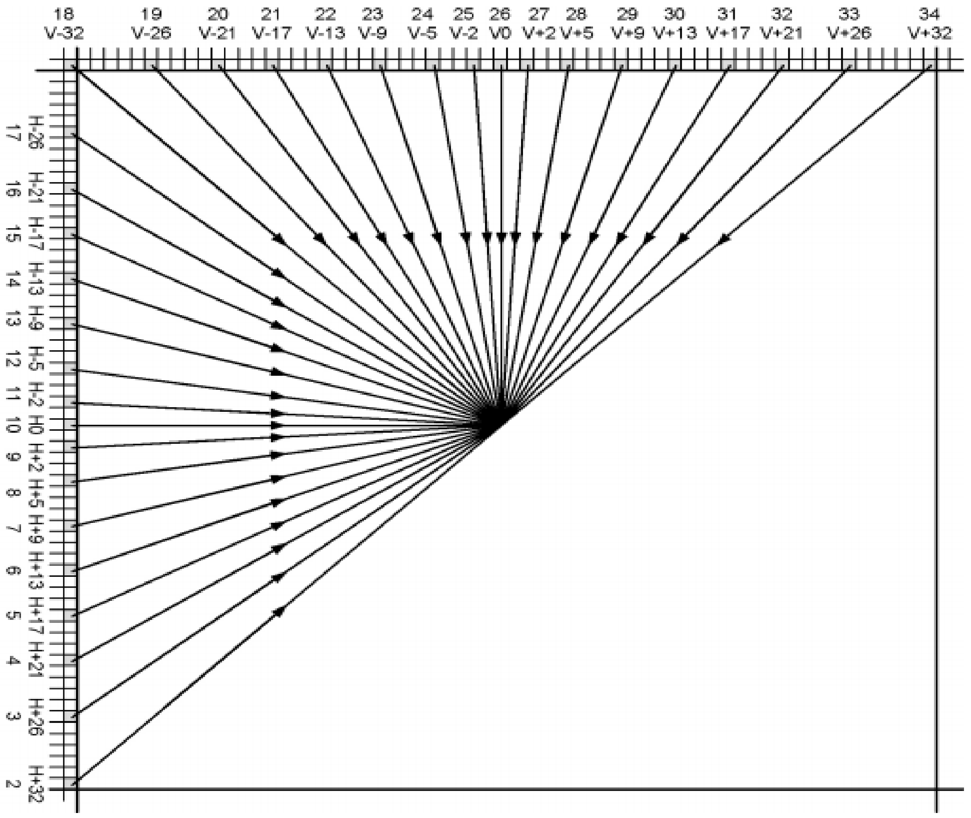


Figure 2-12 HEVC intra prediction modes

2.5 Summary

This chapter gives an overview of the basic HEVC components including a brief introduction about intra prediction. The next chapter will explain the HEVC intra prediction process and its evolution from the H.264 intra prediction method.

Chapter 3

Intra Prediction

3.1 Overview

In H.264, intra prediction [22][24][25][26] is based on spatial extrapolation of samples from previously decoded image blocks, followed by integer discrete cosine transform (DCT) [23] based coding [E3]. HEVC uses the same principle, but further extends it to efficiently representing a wider range of textural and structural information in images. HEVC contains several elements for improving the efficiency of intra prediction over earlier approaches. The introduced methods can model accurately different structures as well as smooth regions with gradually changing sample values.

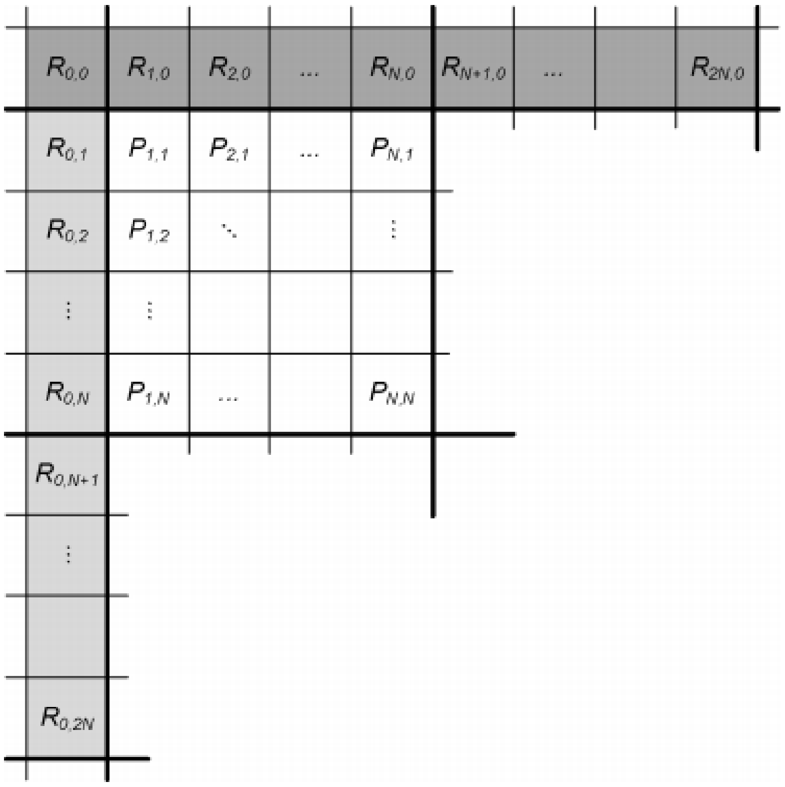


Figure 3-1 Reference samples $R_{x,y}$ used in prediction to obtain predicted samples $P_{x,y}$ for a block of size N by N samples [3].

As shown in Fig. 3-1 the reference samples located on top left and above of the image block to be predicted and denoted by $R_{x,y}$ while the predicted block is denoted by $P_{x,y}$ where (x,y) denotes the position of the predicted sample value. In some cases neighboring reference samples may be unavailable for intra prediction. Hence in such cases missing reference samples on the top boundary are obtained by copying the closest available reference sample [3]. In Fig. 3-1 the missing reference samples on the top boundary are obtained by copying the closest available reference sample from the left while the missing reference samples on the left boundary are generated by copying the reference samples below.

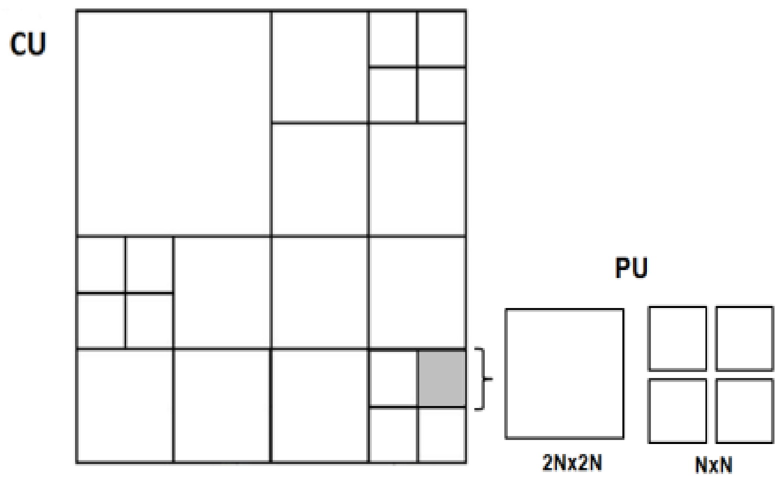


Figure 3-2 Types of partitioning of intra coded CU into PUs [54]

The HEVC emerging standard defines that a frame is divided into large coding units (LCU) which are then partitioned into coding units (CU) using a quad tree structure. Each leaf of the CU can also be further partitioned into prediction units (PUs) as shown in Fig. 3-2, and each PU can deploy a different prediction direction. An intra-coded CU can consist of one 2Nx2N PU or four NxN Pus [54].

HEVC introduces 33 angular prediction modes along with planar and DC prediction modes as shown in Fig. 2-11. The number and angularity of prediction directions are selected to provide a good tradeoff between encoding complexity and coding efficiency [7]. In HEVC there are four effective intra prediction block sizes ranging from 4 by 4 to 32 by 32 samples, each of which supports 33 distinct prediction directions. In order to further simplify the process, all sample locations within one prediction block are projected to a single reference row or column depending on the directionality of the selected prediction mode for example, using the left reference column for angular modes 2 to 17 and the above reference row for angular modes 18 to 34 [3].

Each predicted sample $P_{x,y}$ is obtained by projecting its location to the reference row of pixels applying the selected prediction direction and interpolating a value for the sample at 1/32 pixel accuracy using linear interpolation between two closest reference samples as shown in equation (3.1) [53] below.

$$P_{x,y} = ((32-w_y).\ R_i + w_y.R_{i+1}+16) >> 5 \qquad (3.1)$$

where R_i is the i^{th} reference sample on the reference row, R_{i+1} is the consecutive reference sample, and w_y is the weighting between the two reference samples corresponding to the projected sub-pixel location between R_i and R_{i+1}. The reference sample index I, and the weighting parameter w_y are calculated based on the projection displacement d associated with the selected prediction direction (describing the tangent of the prediction direction in units of 1/32 sample and having a value from -32 to +32) as shown in Fig. 2-11 [53] is as follows.

$$\begin{aligned} c_y &= (y.d) >> 5 \\ w_y &= (y.d)\ \&\ 31 \qquad (3.2) \\ i &= x+c_y \end{aligned}$$

In the above equation >> denotes bit shift operation to right and & denotes the bitwise AND operation. It should be noted that both c_y and w_y parameters depend only on coordinate y and the selected prediction displacement d. Thus, both parameters remain constant when calculating predictions for one line of samples within the prediction block as shown in Fig. 2-12. This makes the sample prediction process to have very low computational requirements as in order to derive the predicted value for specific sample only equation (3.1) needs to be evaluated.

Fig. 3-3 shows 9 different modes supported by H.264 while Table 3-1 shows comparison of number of prediction modes supported by HEVC and H.264 corresponding to different block sizes [26]. In order to be able to represent structures with various directional properties, H.264/AVC defines up to nine different prediction modes for a given block. The maximal set of modes includes eight directional properties and a mode predicting the block with the average (DC) value of the reference pixels. As seen in Fig. 3-3 different directionalities are supported so that video encoders can choose the mode that provides the best RD performance. For example, if the image block that is coded exhibits a strong vertical structures, such as vertical stripes, the prediction mode 0 (vertical mode) would most likely give better compression capability than the other modes [53].

Table 3-1 Comparing HEVC Intra luma prediction modes for 64x64 LCU with H.264/AVC Intra modes for a 64x64 image region [26]

Prediction size	Total Intra Angular modes	
	HEVC/H.265(64x64)	H.264/AVC(16x16)
64x64	4	NA
32x32	35	NA
16x16	35	4
8x8	35	9
4x4	18	9
Total No. of Modes	7808	16x(16x9+4x9+4)=2944

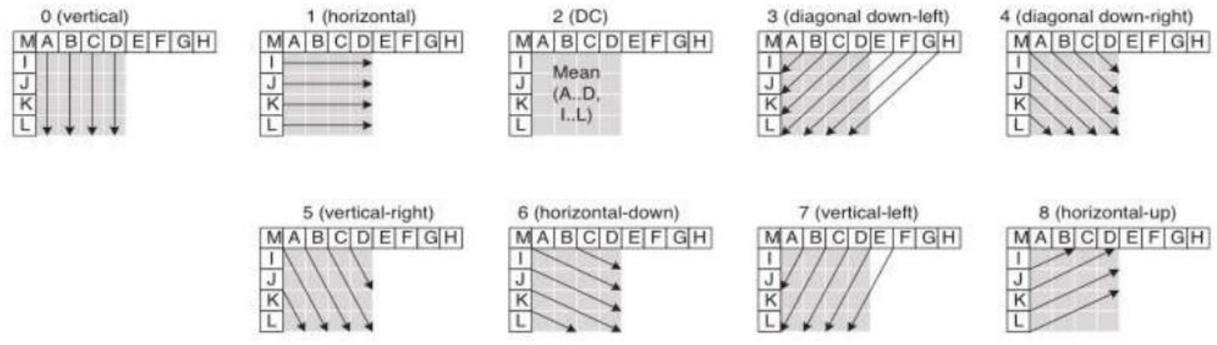


Figure 3-3 H.264 intra prediction modes [E1]

It is observed that the 9 intra prediction modes supported in H.264/ AVC with different directionalities is not flexible enough to represent complex structures or image segments. To mitigate this, HEVC extends the set of directional prediction modes of H.264/AVC providing increased flexibility and more accurate predictions for the sampled values. The increased prediction accuracy provides significant reductions in residual energy of intra coded block and improving coding efficiency [53]. Table 3-2 shows number of intra prediction modes supported by HEVC corresponding to different PU sizes.

Table 3-2 Luma intra prediction modes supported by different PU sizes [27]

PU Size	Intra prediction Modes
4x4	0-16, 34
8x8	0-34
16x16	0-34
32x32	0-34
64x64	0-2, 34

For the chroma component of an intra PU, the encoder selects the best chroma prediction mode among five modes including Planar, DC, Horizontal, Vertical and a direct copy of the intra prediction mode for the luma component. The mapping between intra prediction direction and intra prediction mode number for chroma is shown in Table 3-3 [50].

Table 3-3 Mapping between intra prediction direction and intra prediction mode for chroma [50]

Intra_chroma_pred_mode	Intra prediction direction				
	0	26	10	1	X (0<=X<=34)
0	34	0	0	0	0
1	26	34	26	26	26
2	10	10	34	10	10
3	1	1	1	34	1
4	0	26	10	1	X

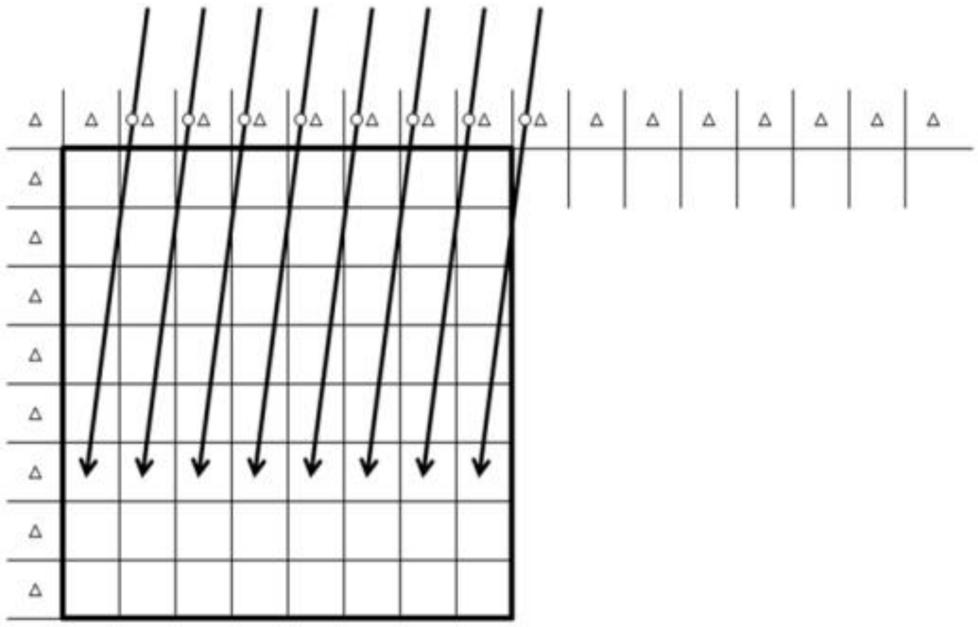


Figure 3-4 An example of angular prediction when operating on sixth row of an 8x8 block [53]

3.2 Intra mode decision process

As previously mentioned, HEVC supports a total 33 angular prediction modes as well as planar and DC prediction for luma intra prediction for all the PU sizes. In H.264/AVC mode coding approach, a single most probable mode was derived based on relative RD cost value. In H.264/AVC mode coding approach, a single most probable mode was derived based on relative RD cost value. HEVC defines the three most probable modes for each PU based on the modes of the neighboring PUs. The selected number of the most probable modes makes it possible to indicate one of the 32 remaining modes by a fixed length code, as the distribution of the mode probabilities outside of the set of most probable modes is found to be relatively uniform [3].

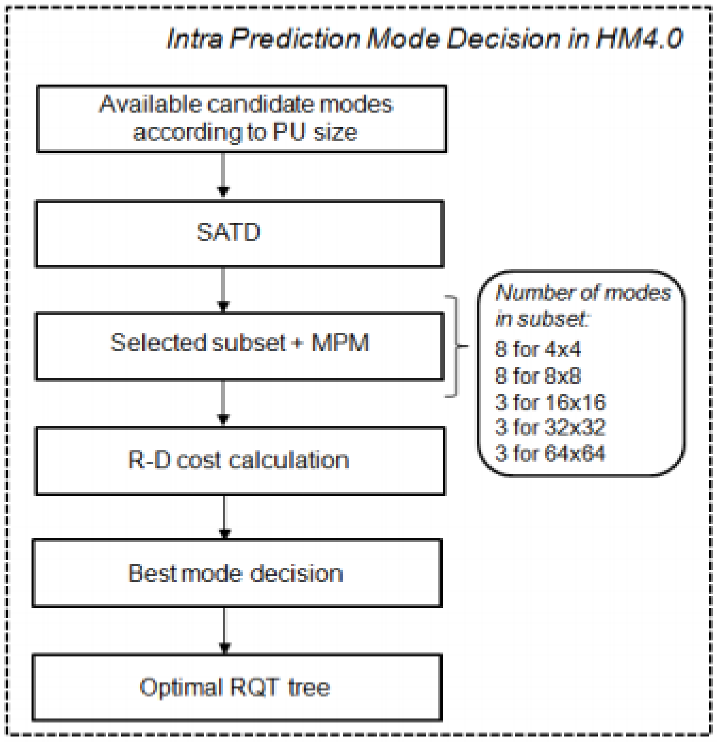


Figure 3-5 Intra prediction mode decision in HM4.0 [54]

The HM4.0 [55] standard defines a simplified version of the intra prediction process which helps to reduce the number of intra prediction modes to be evaluated. The algorithm is based on deciding the best possible subset of modes that yield the smallest sum of the absolute transformed difference (SATD) between the original pixels and the predicted pixels. Depending on the intra modes of the left and top neighboring blocks or PUs, a most probable mode (MPM) is also added to this subset [54].

Finally, the R-D cost of each prediction mode belonging to this subset is computed and the mode with the best R-D cost is selected to encode the PU.

3.3 Summary

This chapter explains the HEVC intra prediction method and its advantage over the H.264 intra prediction method to represent complex structures or image segments. The next chapter will further explain the basics of parallel processing and also the Open MP API.

Chapter 4

Parallel programming

4.1 Parallel computing

The main aim of parallel computing in this thesis is to design a scheme that will calculate 35 intra prediction modes as shown in Fig. 2-12 in parallel. This will reduce the overall encoding time without much increase in complexity.

In the early days of computing, programs were serial, that is, a program consisted of a sequence of instructions, where each instruction executed one after the other. It ran from start to finish on a single processor. Fig. 4-1 shows a typical scenario for a serial computation. In serial computation, a problem to be solved is divided into discrete set of instructions which are executed one at a time.

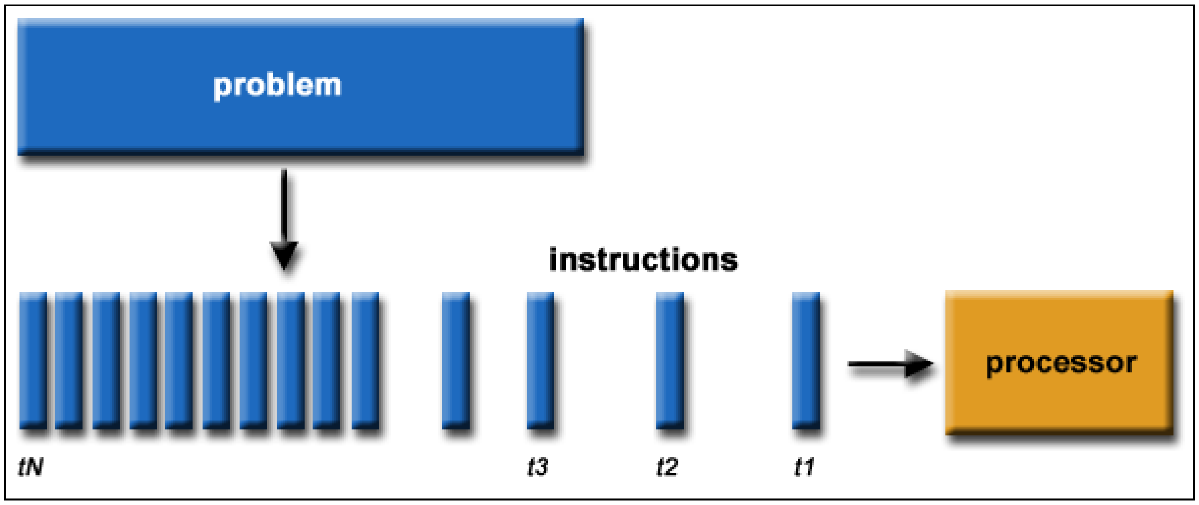


Figure 4-1 Serial computation model [57]

Parallel programming has been developed as a means for improving performance and efficiency. In the simplest sense, parallel computing is the simultaneous use of multiple computer resources to solve a computational problem. In a parallel program, the processing is broken up into parts, each of which can be executed concurrently. The instructions from each part run simultaneously on different CPUs as shown in Fig. 4-2. These CPUs can exist on a single machine, or they can be CPUs in a set of computers connected via a network.

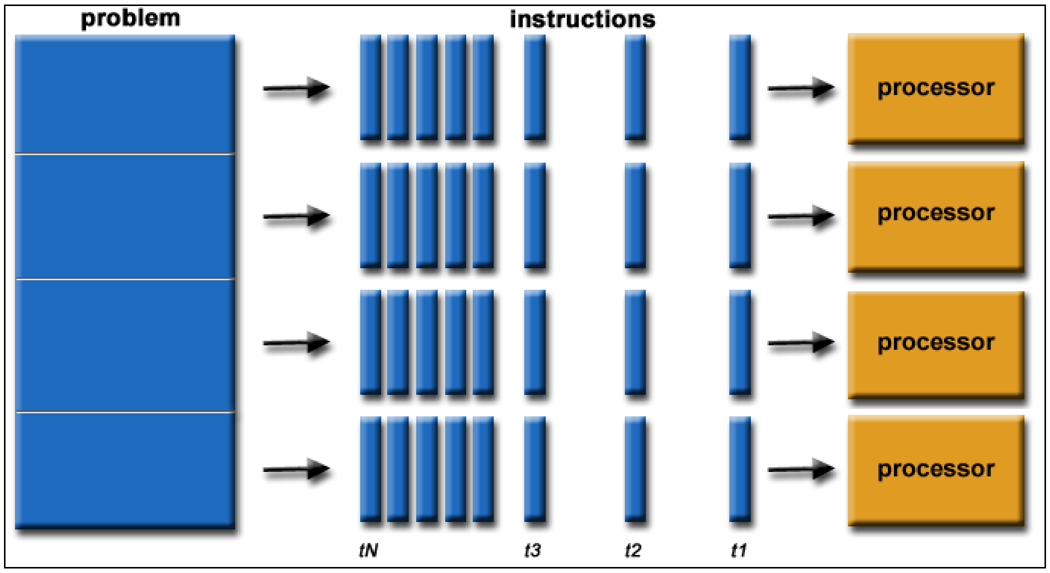


Figure 4-2 Parallel computation model [57]

4.2 Parallel programming models

4.2.1 Shared memory [59]

Shared memory is the fastest inter process communication mechanism. The operating system maps a memory segment in the address space of several processes. In the simplest sense, it is an extra piece of memory that is attached to some address spaces for their owners to use. As a result, all of these processes share the same memory segment and have access to it. Consequently, race conditions may occur if memory accesses are not handled properly. Figure 4-3 shows two processes and their address spaces. It also has a shared memory attached to both address spaces. Both process 1 and process 2 can have access to this shared memory as if it is part of their own address space. In some sense, the original address space is "extended" by attaching this shared memory.

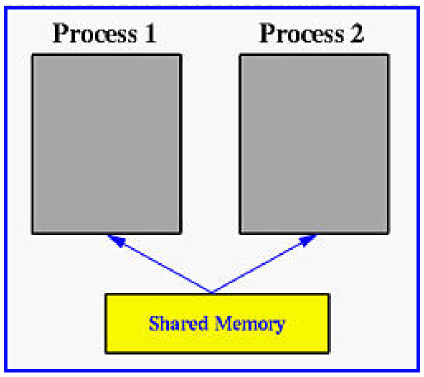


Figure 4-3 Shared memory [59]

4.2.2 Threads

A Thread is the smallest unit of processing that can be performed in an operating system. In most modern operating systems, a thread exists within a process i.e. a single process may contain multiple threads [60]

Multitasking allows processes to run concurrently, while multithreading allows sub-processes to run concurrently. Basically, an operating system with multitasking capabilities will allow two programs to run seemingly at the same time. On the other hand, a single program with multithreading capabilities will allow individual sub-processes (or threads) to run seemingly at the same time. For example, an operating system manages each application program in the PC system (Microsoft word, Microsoft excel or Internet browser etc.) as a separate task. The operating system gives a task a turn at running, and then requires it to wait while another program gets a turn. But it switches between tasks so fast; they appear to run many programs simultaneously. If the program initiates a request like writing a file to the printer or reading a file then it creates a thread.

4.2.3 Distributed memory / Message passing

The message passing interface is a communication system that was designed by a group of researchers to supply programmers with a standard for distributed-memory parallel programming that is portable and usable on a variety of platforms

Message passing in distributed memory systems implies that multiple processes are initiated and run usually on different CPUs to completion. These processes do not have anything in common, and each has its own memory space. Thus information exchanges require communication of data, for which MPI (message passing interface) was, interfaced [61].

4.2.4 Data parallel model

In the task-parallel model represented, the user specifies the distribution of iterations among processors and then the data travels to the computation. Each code is executed on one processor by default. While in data- parallel programming model, the user specifies the distribution of arrays among processors, and then only those processors owning the data will perform the computation. The code is executed on every processor in parallel by default [62].

4.3 Points to remember before developing a parallel program

The goal of parallel processing is to reduce the processing time of several tasks as compared to its serial execution without much increase in complexity. But sometimes, the desired result is not achieved and there is an increase in the processing time or its complexity instead of improving its performance. Hence there are few factors that should be kept in mind before developing a parallel program.

4.3.1 Amount of work to be parallelized

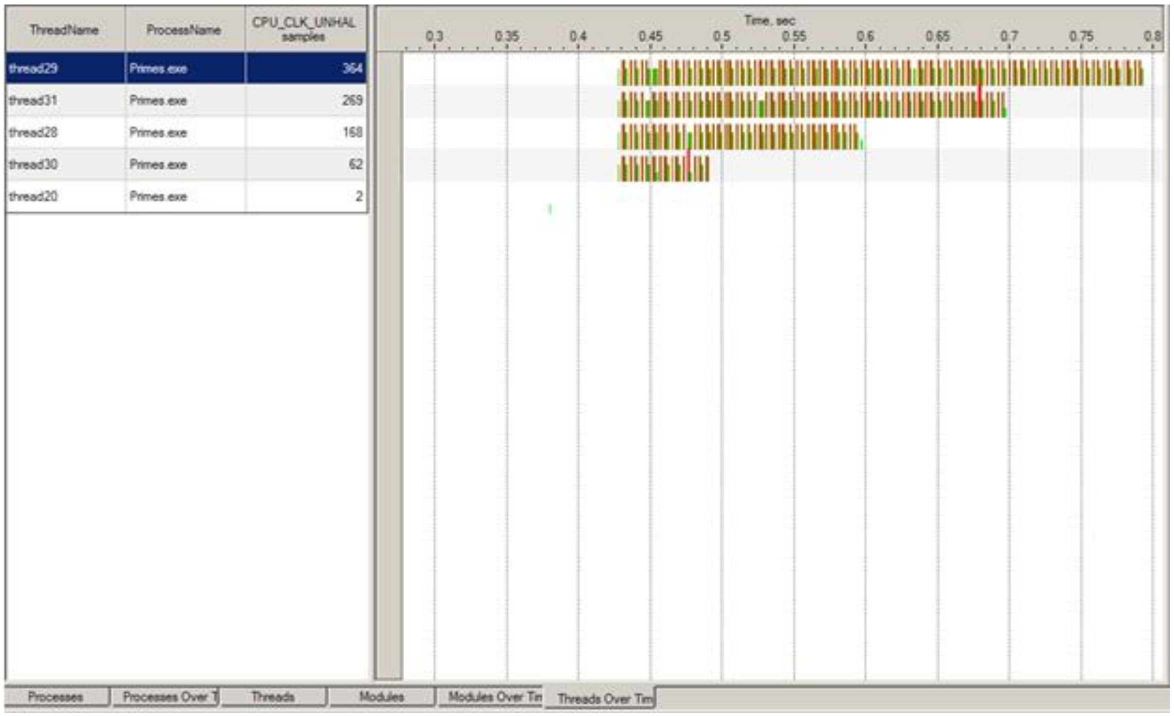
An important requirement for parallelism is to make sure that the program must have enough work that can be performed in parallel. To benefit from parallelism, the total amount of processor – intensive work in a program must be large enough to minimize the overheads of parallelism.

4.3.2 Task granularity

If a program does a lot of parallelizable work, then proper care must be taken that a task is broken into appropriately sized chunks that can be executed in parallel. If more number of chunks are created than required, the overhead of managing and scheduling the chunks will be large. While if less number of chunks are created than required, some cores on the machine will be idle.

4.3.3 Load balancing [63]

Unequal thread workloads occur whenever one thread requires more time to do a given amount of work, thereby diminishing performance. End users experience load imbalances as sluggish program response. Such imbalances often have characteristic visual profiles in system monitoring and software development tools. For example, on modern operating systems that have graphical performance monitoring tools, thread imbalances can show up rather dramatically



ThreadName	ProcessName	CPU_CLK_UNHAL samples
thread29	Primes.exe	364
thread31	Primes.exe	269
thread28	Primes.exe	168
thread30	Primes.exe	62
thread20	Primes.exe	2

Figure 4-4 Poorly load balanced threads as displayed in the Intel VTune Performance analyzer [63]

4.3.4 Memory allocation and garbage collection

Some programs spend a lot of time in memory allocations and garbage collections. Unfortunately, allocating memory in an operation that may require synchronization, since it must be assured that memory regions allocated to different threads may not overlap. Also, allocating the memory typically means that it must be also assured that garbage collection work is implemented in order to reclaim the memory that has been freed. If the garbage collection dominates the running time of the program, them it may scale up the overall processing time.

4.3.5 Race around condition [64]

A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.

Thus, to prevent the race conditions from occurring, the shared variables must be locked so that only one thread can have access to them at a time.

4.3.6 Deadlocks [64]

A deadlock occurs when two threads each lock a different variable at the same time and then try to lock the variable that the other thread already locked. As a result, each thread stops executing and waits for the other thread to release the variable. Because each thread is holding the variable that the other thread wants, nothing occurs, and the threads remain deadlocked. So it must be ensured that the shared variable is properly unlocked by a thread before it attempts to lock another shared variable.

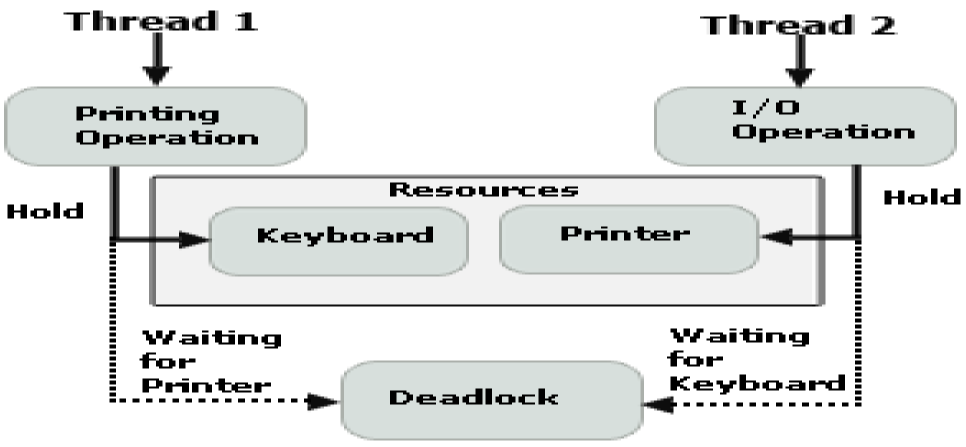


Figure 4-5 Deadlock condition [65]

As shown in Fig. 4-5, two threads have the printing and I/0 operations at the same time. But thread 1 needs printer operation that is hold up by thread 2. In the same way, thread 2 needs keyboard operation that is hold up by thread 1. In this situation CPU become idle and the deadlock condition occurs because no thread is executed until the holdup resources are free [65].

4.4 OpenMP [72]

4.4.1 Introduction

OpenMP (Open Multi-processing) [72] is an API that supports multi-platform shared multiprocessing programming in C, C++ and FORTRAN (Formula translating system) on most processor architectures and operating systems. It consists of a set of compiler directives, library routines and environment variables that influence run time behavior.

The OpenMP API uses the fork-join model for parallel execution in which multiple threads of execution perform tasks defined implicitly or explicitly by OpenMP directives [14] as shown in Figure 8. The OpenMP API is intended to support programs that will execute correctly both as parallel programs and as sequential programs. It supports multi-platform shared memory multiprocessing programming.

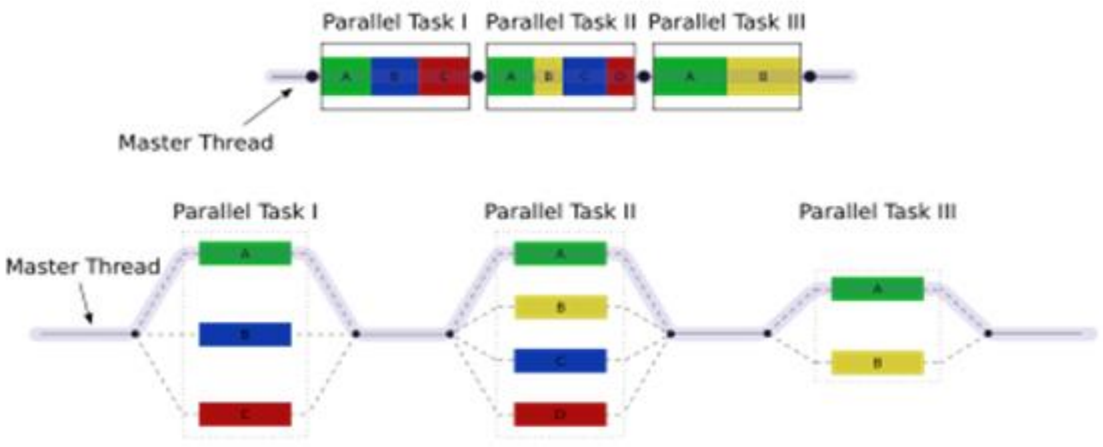


Figure 4-6 Illustration of multithreading in OpenMP [12]

The OpenMP API provides a relaxed- consistency, shared-memory model. All OpenMP threads have access to a place to store and to retrieve a variable, called the memory. In addition, each thread is allowed to have its own temporary view of memory which allows the thread and the memory for every reference to a variable. Each thread also has access to another type of memory that must not be accessed by other threads, called thread private memory [33].

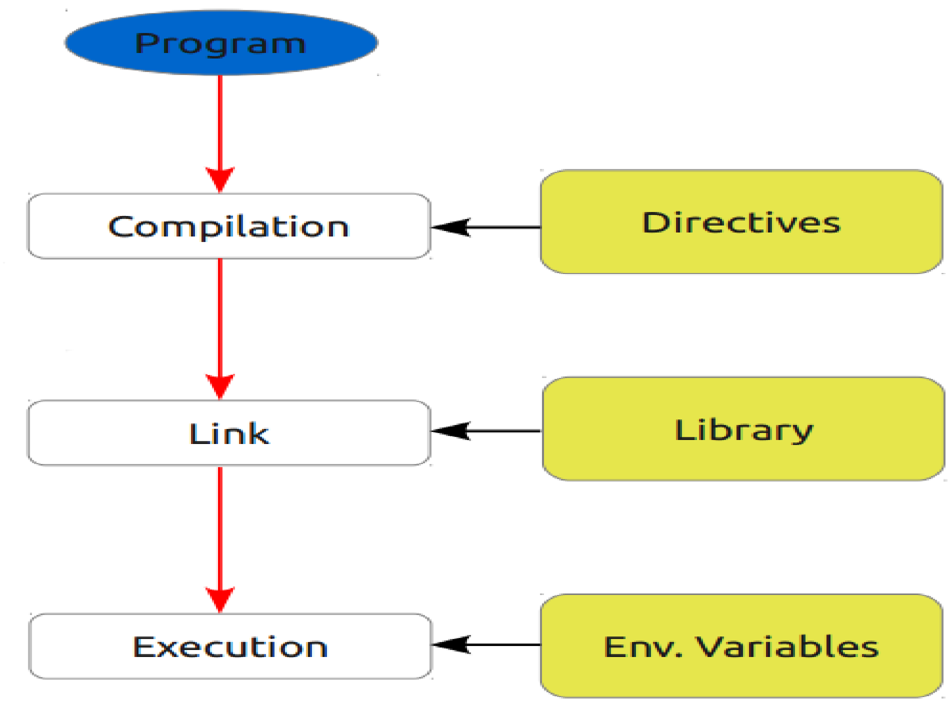


Figure 4-7 OpenMP components [66]

As shown in Fig. 4-6, the OpenMP API is comprised of three distinct components

4.4.1.1 Compiler directives [69]

Compiler directives appear as a comment in source code and are ignored by compilers unless specified by an appropriate compiler flag. They are basically responsible for spawning a parallel region, dividing a block of codes among threads, distributing loop iterations between threads and synchronization of work among threads.

4.4.1.2 Runtime library routines [67]

OpenMP provides several run-time library routines to help manage the program in parallel mode. Many of these run-time library routines have corresponding environment variables that can be set as defaults. The run-time library routines are used for changing these factors to assist in controlling the program.

4.4.1.3 Environment variables [68]

The Visual C++ implementation of the OpenMP standard includes environment variables. These environment variables are read at program startup and modifications to their values are ignored at runtime. Examples of environment variables are OMP_DYNAMIC, OMP_NESTED, OMP_SCHEDULE, etc.

In the following sections a few of important OpenMP directives will be explained that serve as important information for parallelizing code using OpenMP and they have been implemented in the thesis.

4.4.2 OpenMP directives

4.4.2.1 The OpenMP parallel pragma

This directive will set up a team of threads including master thread, all of which will execute the block following the directive in parallel. Unlike other directive, this directive leaves it up to the programmer as to how to partition the work. For example:

```
#include <omp.h>
…
void main ()
{
#pragma omp parallel
// do parallel work
}
```

As shown in this example, the pragma directive is initiated to execute the task in parallel below it, which will be further executed by all threads. The function main () is run by a master thread which will then branch off into many threads to execute the task in parallel. It should be noted that include file “omp.h” defines all OpenMP functions and should be included to initiate them.

4.4.2.2 The OpenMP single Pragma

In some cases only one thread is required to execute some part code, even though that code is part of the parallel or other work sharing block.

For example:

```
#pragma omp single
{
}
```

4.4.2.3 The OpenMP barrier pragma

The barrier directive helps to synchronize all the threads in a team. When a thread reaches barrier directive, it will wait at the point until all other threads have reached the barrier, and then continues executing the code after the barrier in parallel.

For example:

#pragma omp barrier *newline*

4.4.2.4 The OpenMP critical pragma

It basically allows only one thread to enter a particular part of code at a time, while others wait.

For example:

```
#pragma omp critical [(name)]
{
//code
}
```

The thread waits at the start of a critical region identified by a given name until no other thread in the program is executing a critical region with that same name. The critical directive supports no open MP clauses.

4.4.2.5 The OpenMP for pragma

It basically breaks up a C/C++ *for* loop, assigning various iterations to various threads.

For example:

```
#pragma omp parallel for [clauses]
//For_statement
```

Each thread executes one or more of the iterations and executes in parallel. The programmer must make sure that the iterations are independent. It supports many clauses like firstprivate, lastprivate, nowait, ordered, private, reduction, and schedule. Some of this clause will be explained later on.

4.4.2.6 The OpenMP ordered pragma

It basically specifies the code under a parallelized *for* loop that must be executed in sequential order.

For example:

```
void main()
{
#pragma omp parallel for
for (int i=0; i<2;i++)
  {
///some parallelized code
#pragma omp ordered
printf("-----")
  }
}
```

As shown in this example, it applies to statement block immediately following it which is a print statement in the given code. Thus, threads for loop iteration will execute print statement sequentially rather than executing it in parallel.

4.4.3 OpenMP clauses [70]

Some directives mentioned above have clauses associated with them. The brief description about some of the important OpenMP clauses is described below.

4.4.3.1 private clause

It declares variables to be private to each thread in a team. Private copies of the variable are initialized from the original object when entering the region.

4.4.3.2 firstprivate clause

It provides a superset of the functionality provided by the private clause. Each private data object is initialized with the value of the original object.

4.4.3.3 lastprivate clause

It provides a superset of the functionality provided by the private clause. The original object is updated with the value of the private copy from the last sequential iteration of the associated loop, or the lexically last section construct, when exiting the region.

4.4.3.4 default clause

It enables programmer to affect the data-scope attributes of variables.

4.4.3.5 reduction clause

It performs a reduction on scalar variables.

4.4.3.6 ordered clause

The structured block following an ordered directive is executed in the order in which iterations would be executed in a sequential loop.

4.4.3.7 schedule clause

It specifies how iterations of *for* loop are divided among the threads of the team.

4.4.3.8 nowait clause

It indicates that an implementation may omit the barrier at the end of the work sharing region.

4.4.3.9 collapse (n) clause

It specifies the number of loops that are associated with the OpenMP loop construct for collapsing.

4.4.3.10 shared clause

It shares variables among all the threads in a team.

4.4.3.11 untied clause

It indicates that a resumed task does not have to be executed by same thread executing it, before it was suspended.

4.4.3.12 final (*expr*)

The expr is evaluated, and if the value is true, then this task and all its descendant tasks are non-deferred (not executed in parallel).

4.4.4 OpenMP Environment variables [71]

4.4.4.1 omp_dynamic

It specifies whether the OpenMP run time can adjust the number of threads in parallel region.

4.4.4.2 omp_nested

It specifies whether nested parallelism is enabled, unless nested parallelism is enabled or disabled with omp_set nested

4.4.4.3 omp_num_threads

It sets the maximum number of threads in the parallel region, unless overridden by omp_set_num_threads or num_threads.

4.4.4.4 omp_schedule

It modifies the behavior of the schedule clause when schedule (runtime) is specified in *for* or parallel *for* directive.

4.5 Summary

The chapter gives a brief introduction of parallel computing and its components. It also gives an overview of the openMP API that has been used in this thesis. The next chapter will further explain about how parallel computing using openMP has been implemented in this thesis to parallelize 35 different intra prediction modes along with results.

Chapter 5

Algorithm, Implementation and Results

5.1 Overview

The HEVC standard has proven to be able to deliver high compression ratio, suitable for a wide range of applications including HDTV broadcasting. The increased number of intra prediction modes in the HEVC standard allows better compression and more flexible block representation, on top of which advanced prediction and transform concepts can be built. It has been shown that significant visual and about 40% better PSNR performance improvement can be achieved over H.264/AVC [5].

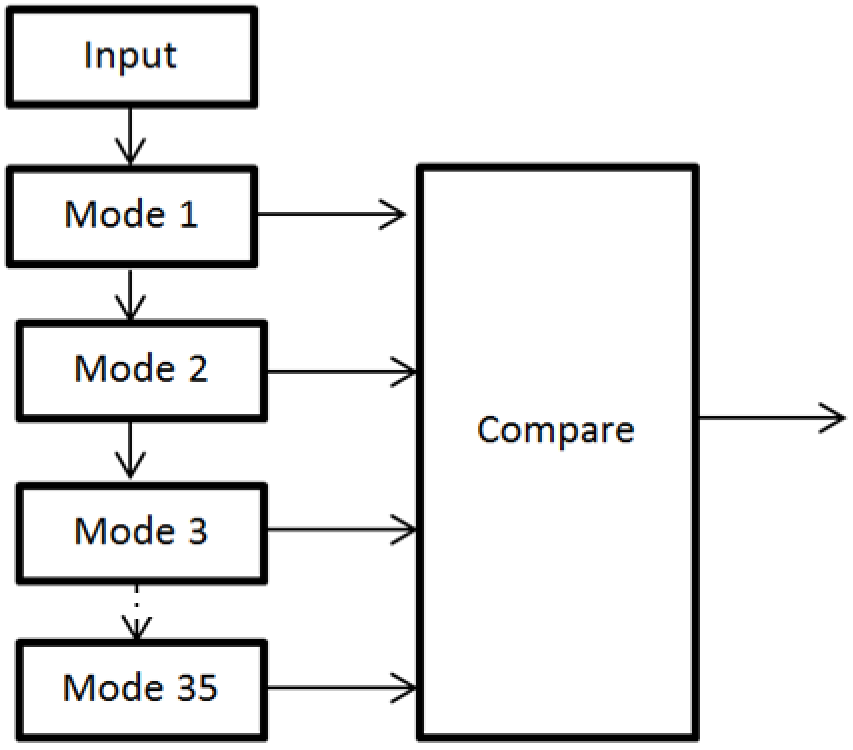


Figure 5-1 Serial processing of intra prediction modes in HEVC [75]

Fig.5-1 shows a serial implementation of 35 different intra modes supported by HEVC. As mentioned in Section 3.2 about intra mode decision process, HEVC defines the three most probable modes depending on RD cost values as compared to single mode in H.264. There are over 11900 possibilities of splitting a single CU as shown in Table 5-1. It is not a small feat for the encoder to perform this computation and come up with the best mode that costs the least and also has minimum distortion and bitrate [74]. Thus, in this thesis, effort has been taken to reduce the encoding time. This time can be saved if a mode decision process is executed in parallel instead of serials. It must be ensured that with reduction in encoding time there is not much variation in peak signal to noise ratio (PSNR) and bitrate

Table 5-1 Current problem- complexity and encoded time [74]

Size of a PB	Number of PBs in 64x64 CU	Number of modes to be tested in one PB	Total number of modes to be tested at this level
32x32	4	35	140
16x16	16	35	560
8x8	64	35	2240
4x4	256	35	8960
Total			11,900

5.2 Proposed solution by parallel processing of intra mode decision process

There are number of papers and approaches available on reducing the encoding time of HEVC by modifying the intra mode decision process. Fast intra prediction mode decision can be done using parallel processing approach [75]. This has been proposed by J.Rehman and Zhang which aims at parallelizing eight different H.264 intra modes to reduce the encoding time (see Fig. 3-3). The main objective here is to further implement the parallel approach using Open MP API to parallelize 35 different intra prediction modes in HEVC leading to reduction in the encoding time.

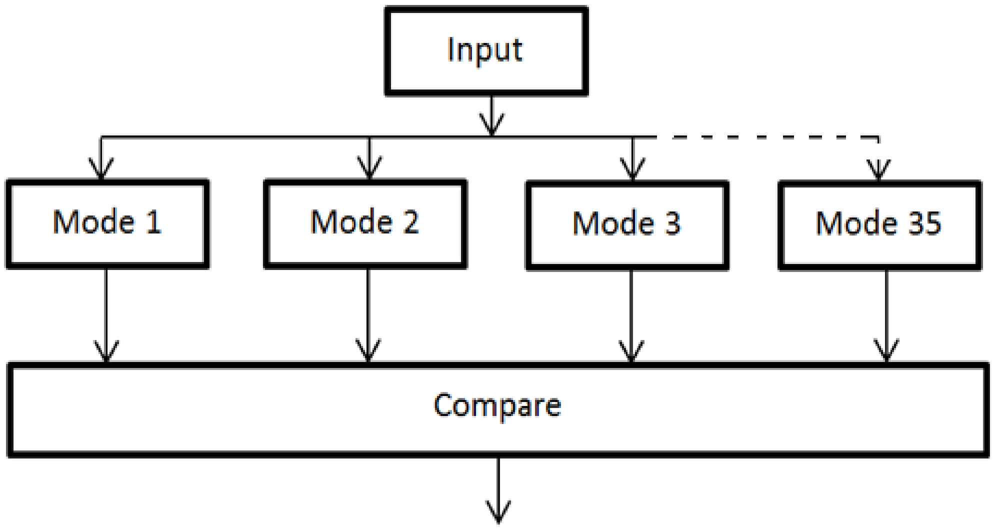
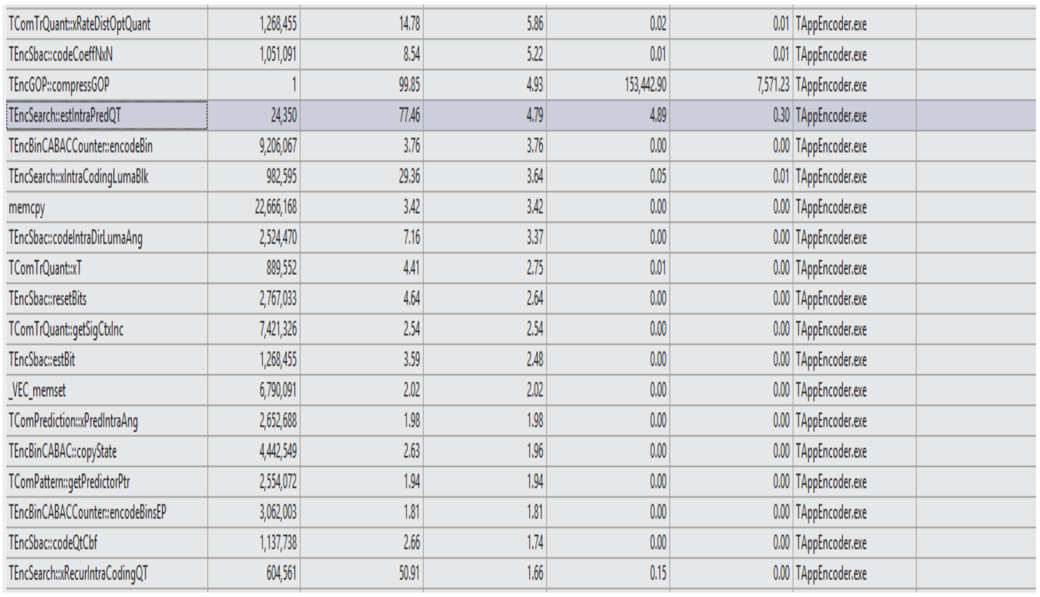


Figure 5-2 Parallel processing of intra prediction modes in HEVC [75]

As shown in Fig. 5-2, a parallel approach has been implemented to reduce the encoding time required for decision making process. It aims at dividing the modes among multiple threads, calculating RD cost values and other decision parameters corresponding to each thread simultaneously, then comparing it. The approach is in contrast to the method shown in Fig. 5-1 which calculates decision making parameters corresponding to each mode in sequential order. The parallel approach described in Fig.5-2 helps in reducing the encoding time without much increase in complexity and affecting PSNR and bitrate parameters of the sequence.

A maximum of two threads are run in parallel on two different cores to eliminate data dependency. Also the data dependency in other functions must be reduced which relate to work sharing among threads during the decision making process.

The data based parallelism is achieved by finding the hotspot in the HM 9.1 reference software. These hotspots are further made to run in parallel by dividing the total work of a hotspot into two different threads equally by using OpenMP software. This division of work is done by enhancing the code of the hotspot under consideration. Also it is preferable to maintain load balancing between the two threads for optimum results. Thus it may increase software complexity and produce an additional overhead of thread creation for each hotspot in the reference software. This is further discussed in future implementation section of this thesis.

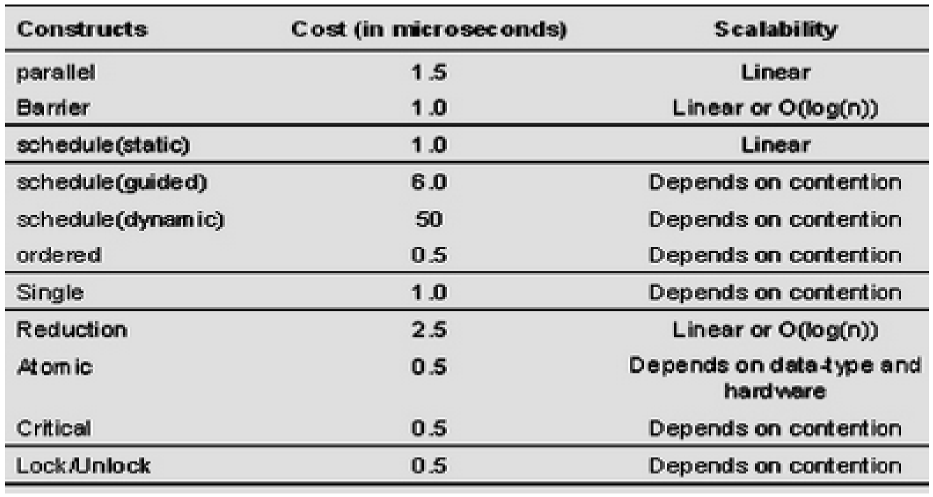


TComTrQuant::xRateDistOptQuant	1,268,455	14.78	5.86	0.02	0.01	TAppEncoder.exe	
TEncSbac::codeCoeffNxN	1,051,091	8.54	5.22	0.01	0.01	TAppEncoder.exe	
TEncGOP::compressGOP	1	99.85	4.93	153,442.90	7,571.23	TAppEncoder.exe	
TEncSearch::estIntraPredQT	24,350	77.46	4.79	4.89	0.30	TAppEncoder.exe	
TEncBinCABACCounter::encodeBin	9,206,067	3.76	3.76	0.00	0.00	TAppEncoder.exe	
TEncSearch::xIntraCodingLumaBlk	982,595	29.36	3.64	0.05	0.01	TAppEncoder.exe	
memcpy	22,666,168	3.42	3.42	0.00	0.00	TAppEncoder.exe	
TEncSbac::codeIntraDirLumaAng	2,524,470	7.16	3.37	0.00	0.00	TAppEncoder.exe	
TComTrQuant::xT	889,552	4.41	2.75	0.01	0.00	TAppEncoder.exe	
TEncSbac::resetBits	2,767,033	4.64	2.64	0.00	0.00	TAppEncoder.exe	
TComTrQuant::getSigCtxInc	7,421,326	2.54	2.54	0.00	0.00	TAppEncoder.exe	
TEncSbac::estBit	1,268,455	3.59	2.48	0.00	0.00	TAppEncoder.exe	
_VEC_memset	6,790,091	2.02	2.02	0.00	0.00	TAppEncoder.exe	
TComPrediction::xPredIntraAng	2,652,688	1.98	1.98	0.00	0.00	TAppEncoder.exe	
TEncBinCABAC::copyState	4,442,549	2.63	1.96	0.00	0.00	TAppEncoder.exe	
TComPattern::getPredictorPtr	2,554,072	1.94	1.94	0.00	0.00	TAppEncoder.exe	
TEncBinCABACCounter::encodeBinsEP	3,062,003	1.81	1.81	0.00	0.00	TAppEncoder.exe	
TEncSbac::codeQtCbf	1,137,738	2.66	1.74	0.00	0.00	TAppEncoder.exe	
TEncSearch::xRecurIntraCodingQT	604,561	50.91	1.66	0.15	0.00	TAppEncoder.exe	

Figure 5-3 Profiler instrumentation data

To implement data based parallelism one needs to find hotspot in the reference software. A profiler tool is available in Visual studio. This is used to determine exactly the amount of time which is consumed by each function. As shown in Fig. 5-3, the function named ‘esIntraPredQT’ (highlighted in Fig 5-3) which was responsible for calling the functions to calculate the RD cost values and storing them in a buffer. This function consumes most of the time for mode decision process. Hence efforts are taken to parallelize that particular section of code to improve the overall performance by reducing the encoding time.

However the main drawback of OpenMP is overhead caused while entering a parallel section of the code. As shown in Fig.5-4 there is a definite overhead included while using different openMP parameters. Thus by calculating the overall overhead time during parallelization process and subtracting it from overall encoding time, actual estimation of amount of time saved after implementing the parallel processing approach can be determined.



Constructs	Cost (in microseconds)	Scalability
parallel	1.5	Linear
Barrier	1.0	Linear or O(log(n))
schedule(static)	1.0	Linear
schedule(guided)	6.0	Depends on contention
schedule(dynamic)	50	Depends on contention
ordered	0.5	Depends on contention
Single	1.0	Depends on contention
Reduction	2.5	Linear or O(log(n))
Atomic	0.5	Depends on data-type and hardware
Critical	0.5	Depends on contention
Lock/Unlock	0.5	Depends on contention

Figure 5-4 Overhead cost for OpenMP parameters

5.3 Results

5.3.1 Test conditions

The performance of the proposed parallel approach in intra prediction was evaluated using HEVC reference software (HM 9.1) [11]. The intra main profile was selected by setting the value of intra period in configuration file as 1 to make sure that all frames are intra coded and frame rate is 50 fps. The approach was evaluated by using different QP values of 22, 27, 32 and 37 on standard test sequences [15] recommended by JCT-VC as shown in Table 5-2.

Table 5-2 gives a brief overview of the sequences used and the corresponding parameters.

Table 5-2 Test Sequences used [15]

No.	Sequence Name	Resolution	Type	Approx. size (MB)	No. of frames
1	RaceHorses_416x240_30.yuv	416x240	WQVGA	44	30
2	BasketballDrillText_832x480_50.yuv	832x480	WVGA	294	30
3	SlideEditing_1280x720_30.yuv	1280x720	SD	405	30
4	Kimono1_1920x1080_24.yuv	1920x1080	HD	729	30
5	PeopleOnStreet_2560x1600_30.yuv	2560x1600	WQHD	900	30

5.3.2 Encoding Time Gain

The proposed parallel approach used to reduce encoding time provides a reduction in encoding time of about 22-57 % as compared to the unmodified encoder of HM 9.1 [11]. To analyze the results standard test sequences [15] are considered using different quantization parameters (QP) as suggestedby JCTVC. The results are described in Fig 5-4 through Fig 5-9

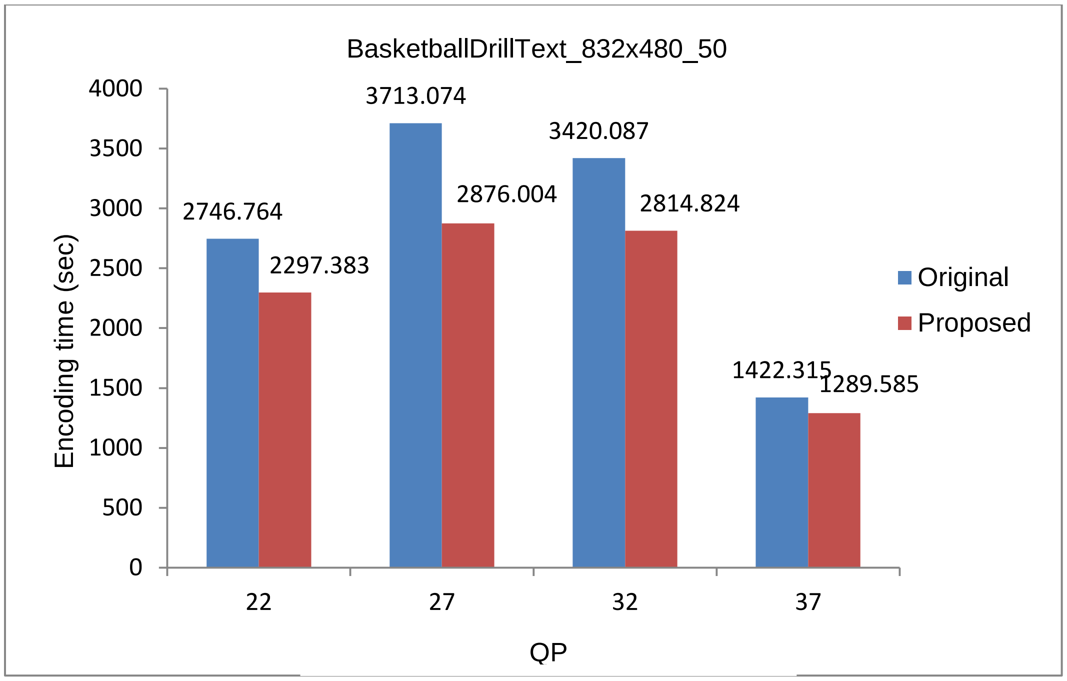


Figure 5-5 Encoding time vs quantization parameter for BasketballDrillText

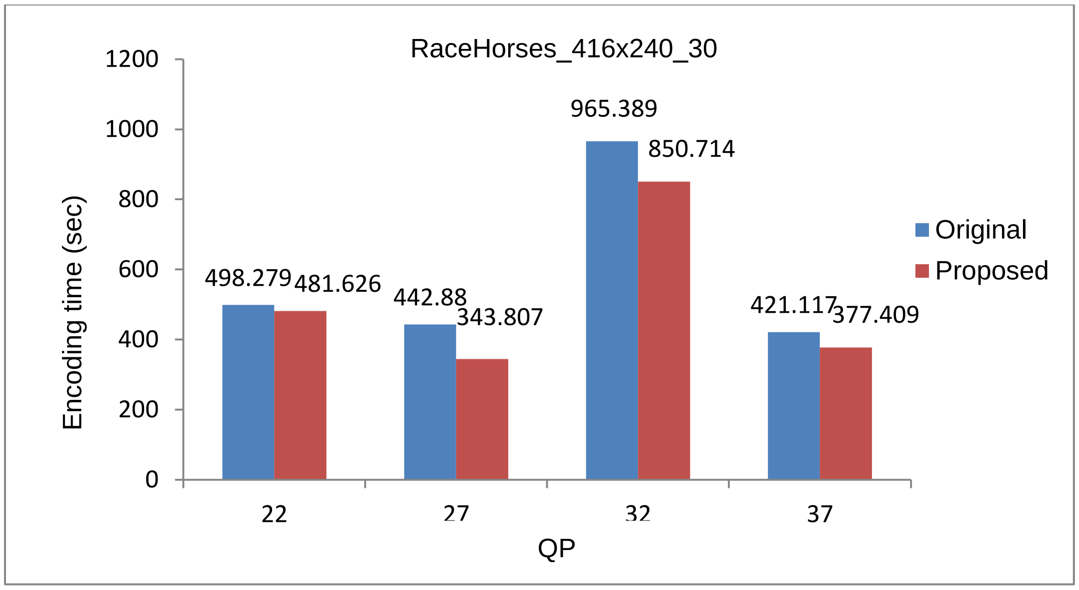


Figure 5-6 Encoding time vs quantization parameter for RaceHorses

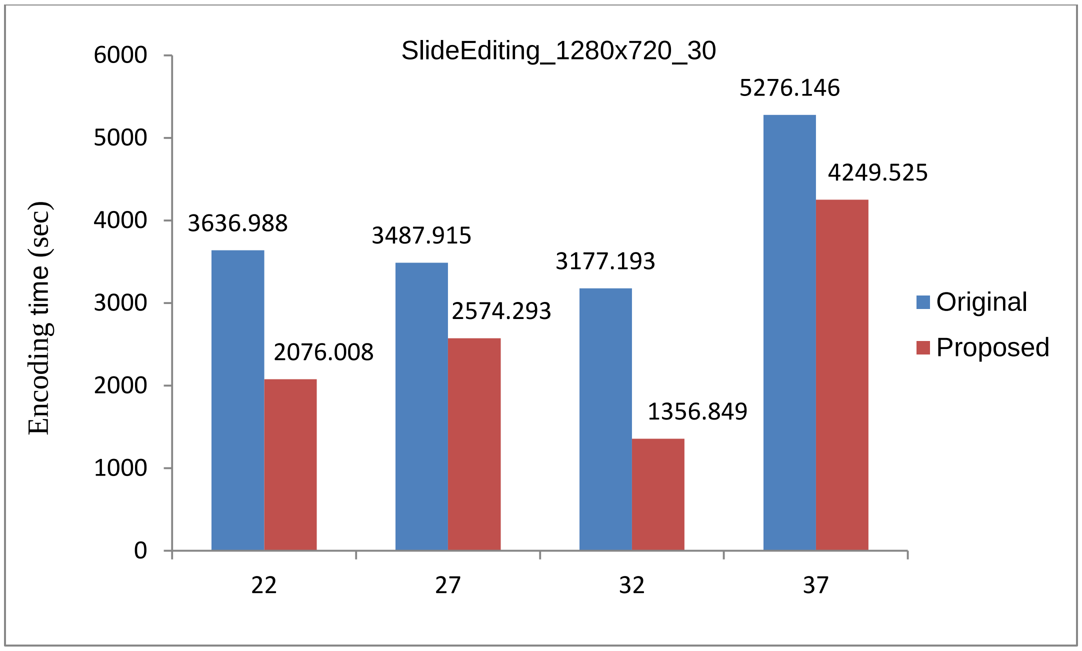


Figure 5-7 Encoding time vs quantization parameter for SlideEditing

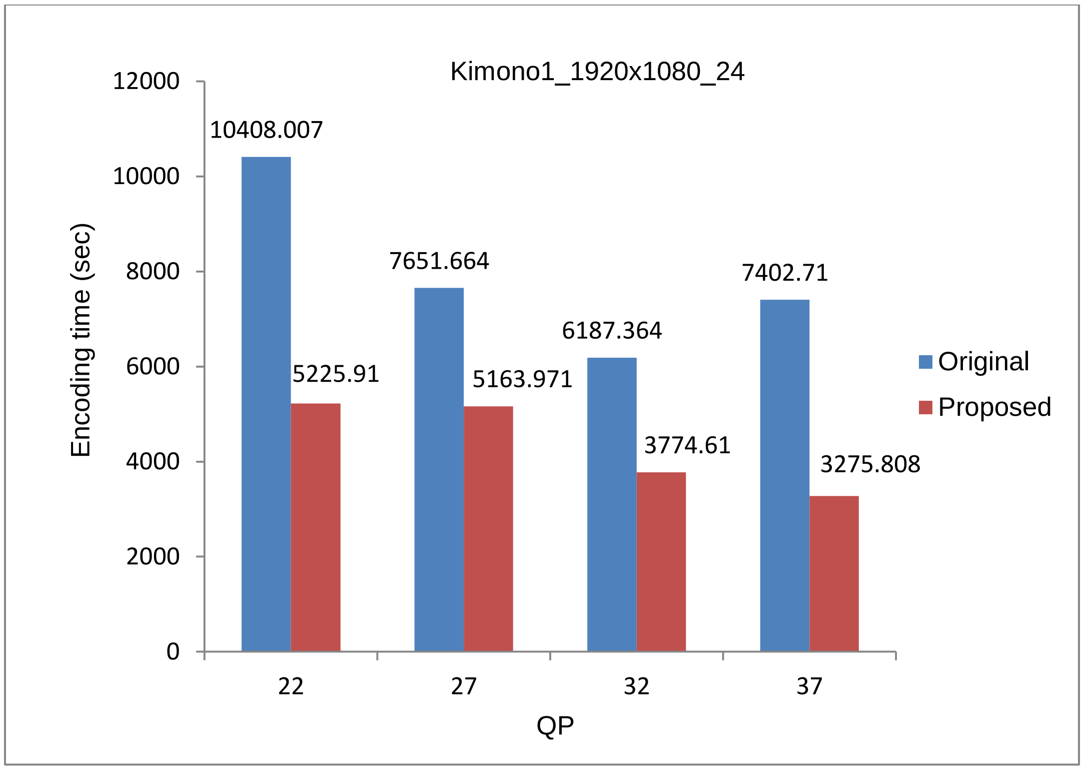


Figure 5-8 Encoding time vs quantization parameter for Kimono

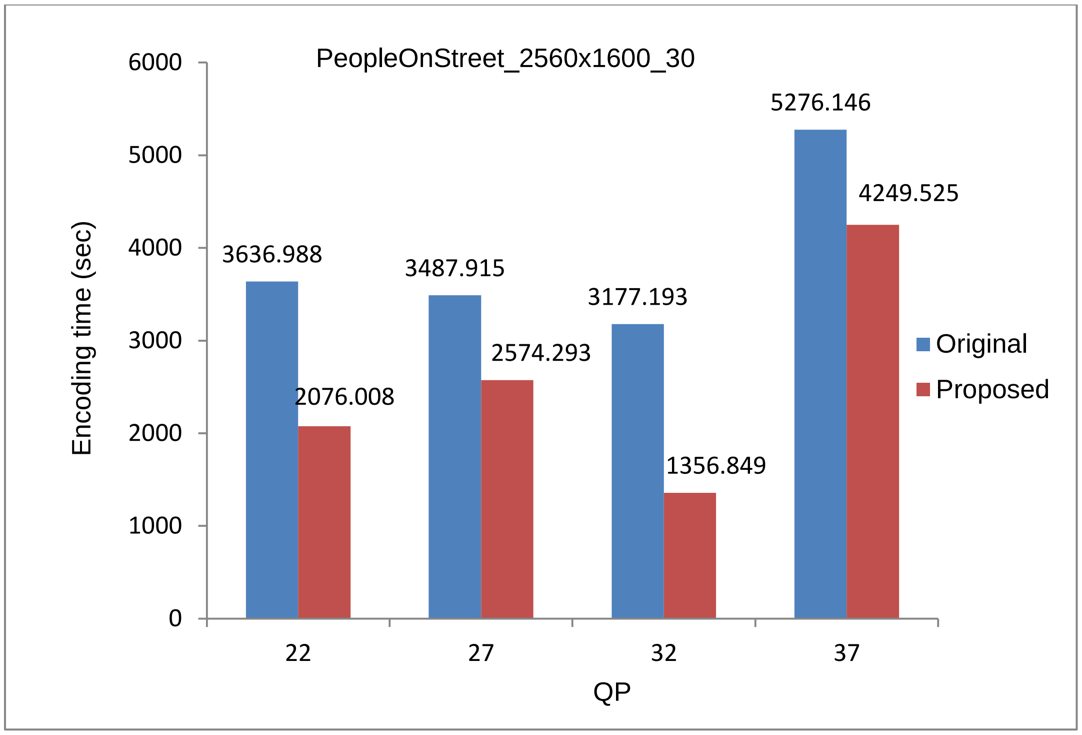


Figure 5-9 Encoding time vs quantization parameter for PeopleOnStreet

5.3.3 BD-PSNR

BD-PSNR (see Appendix C) is a curve fitting program based on the rate and distortion of the video sequence. Based on the rate distortion (R-D) curve fitting, it helps to provide a good evaluation of the R-D performance. However BD-PSNR has a critical drawback that it does not take the coding complexity into account. Clearly, for practical video applications, especially for those on handheld devices, coding complexity has to be considered while evaluating overall coding performance [77]. As expected there is a slight decrease in PSNR for the proposed algorithm, making BD-PSNR negative. The results for BD-PSNR are describes as follows from Fig. 5-10 through 5-14

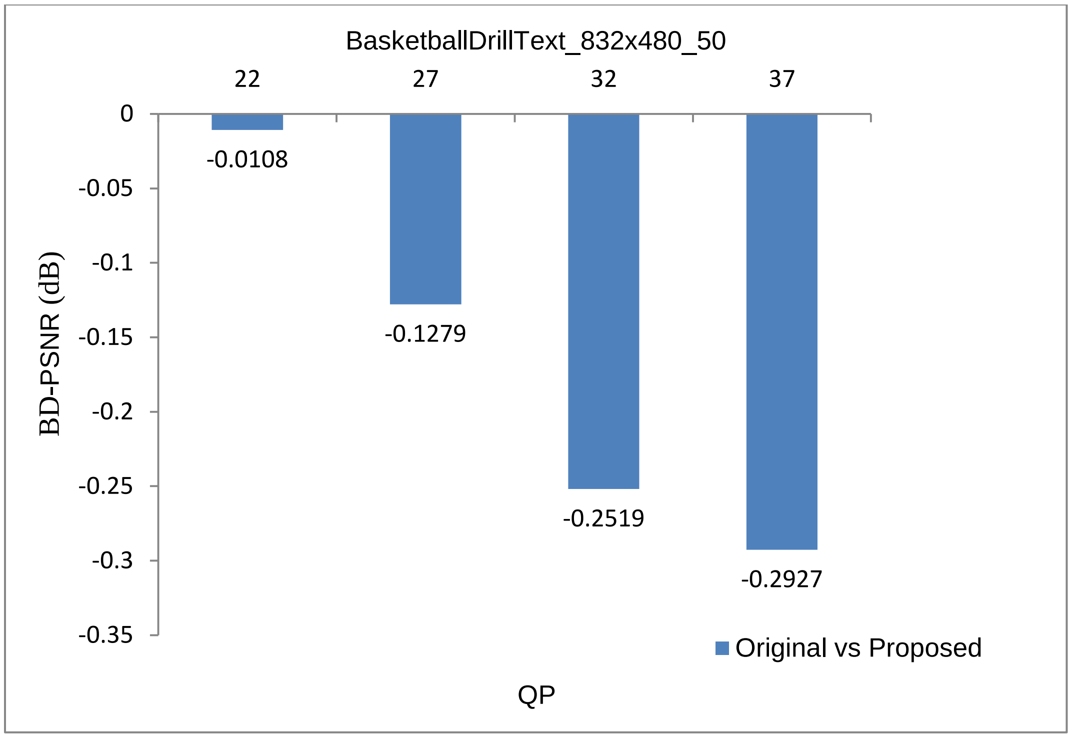


Figure 5-10 BD-PSNR vs quantization for BasketballDrillText

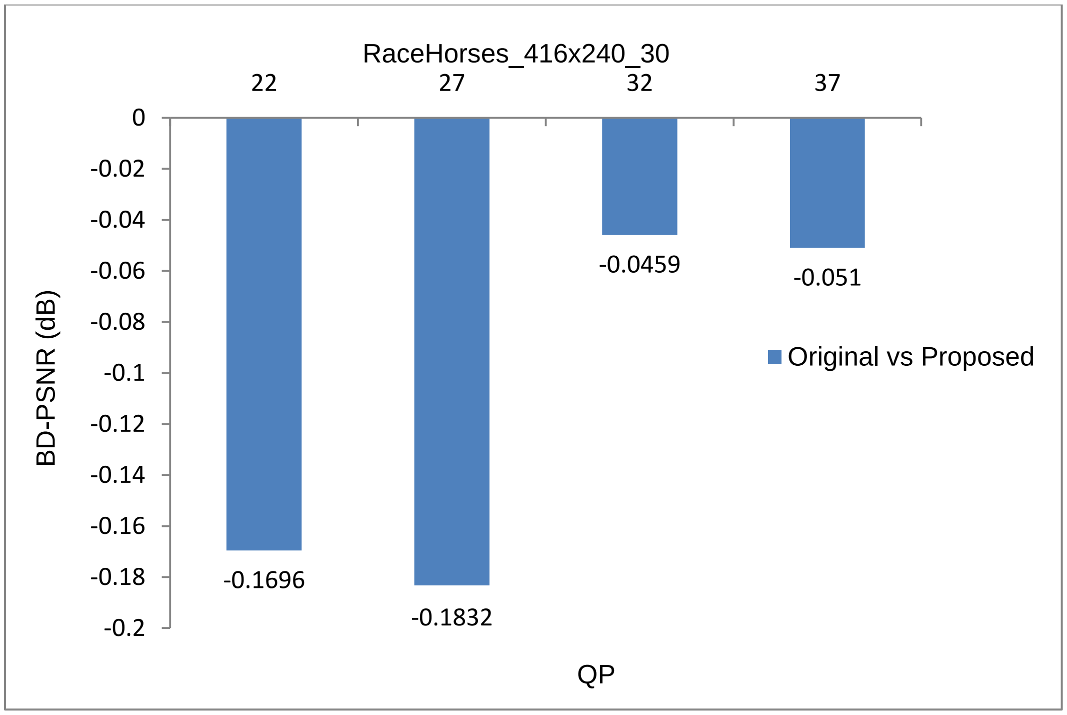


Figure 5-11 BD-PSNR vs quantization for RaceHorses

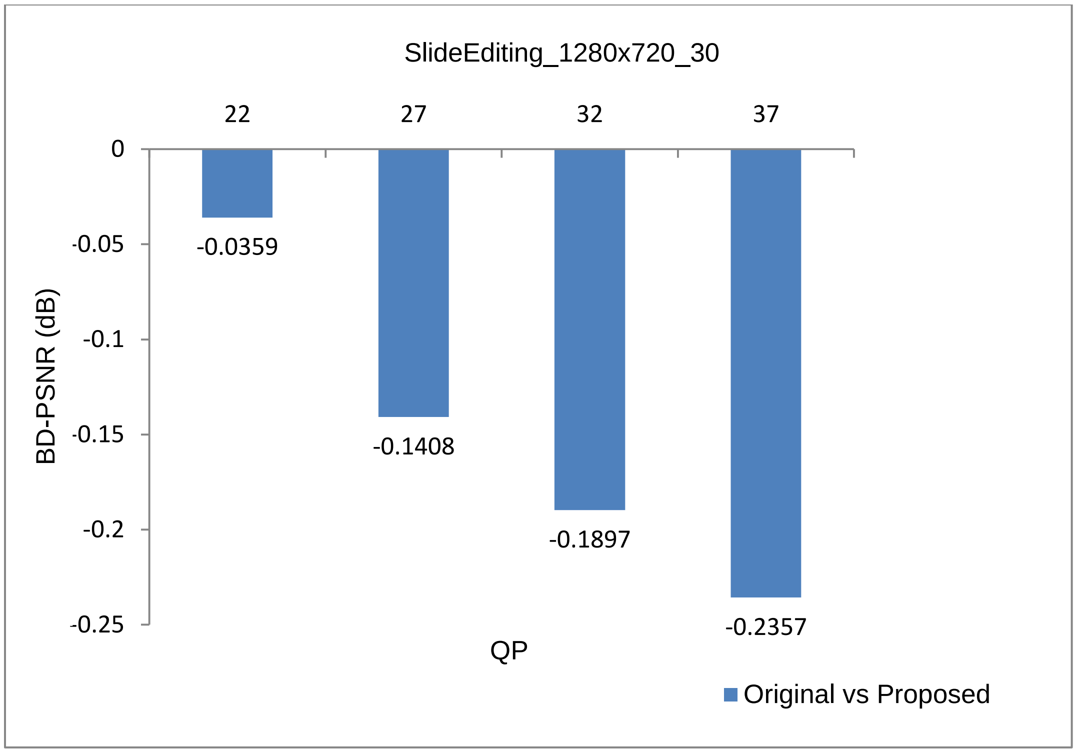


Figure 5-12 BD-PSNR vs quantization for SlideEditing

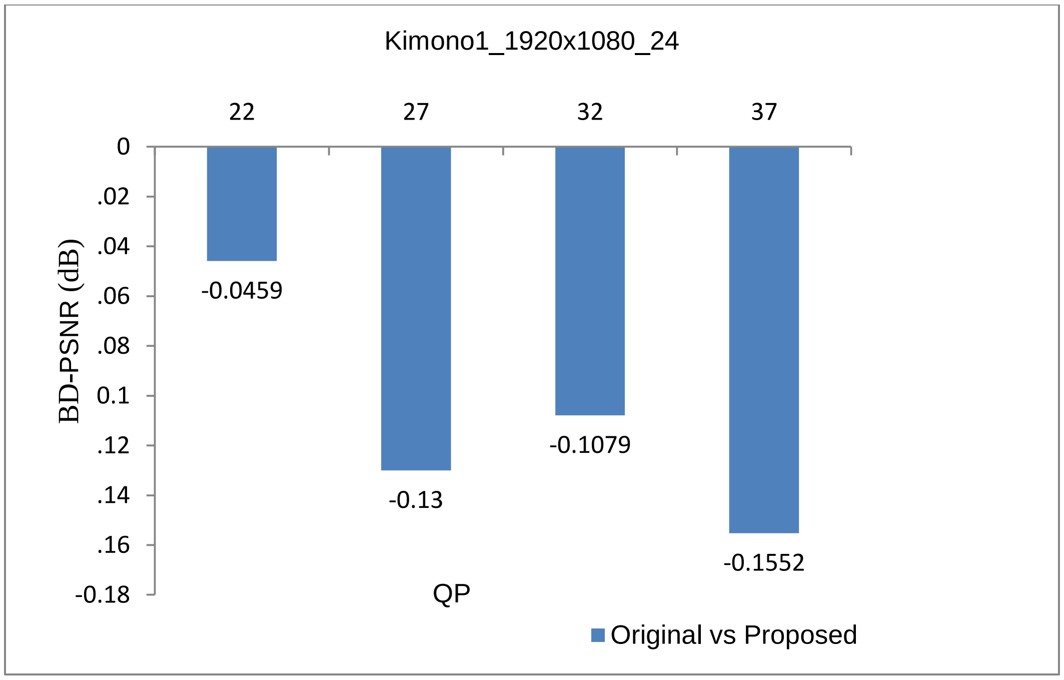


Figure 5-13 BD-PSNR vs quantization for Kimono

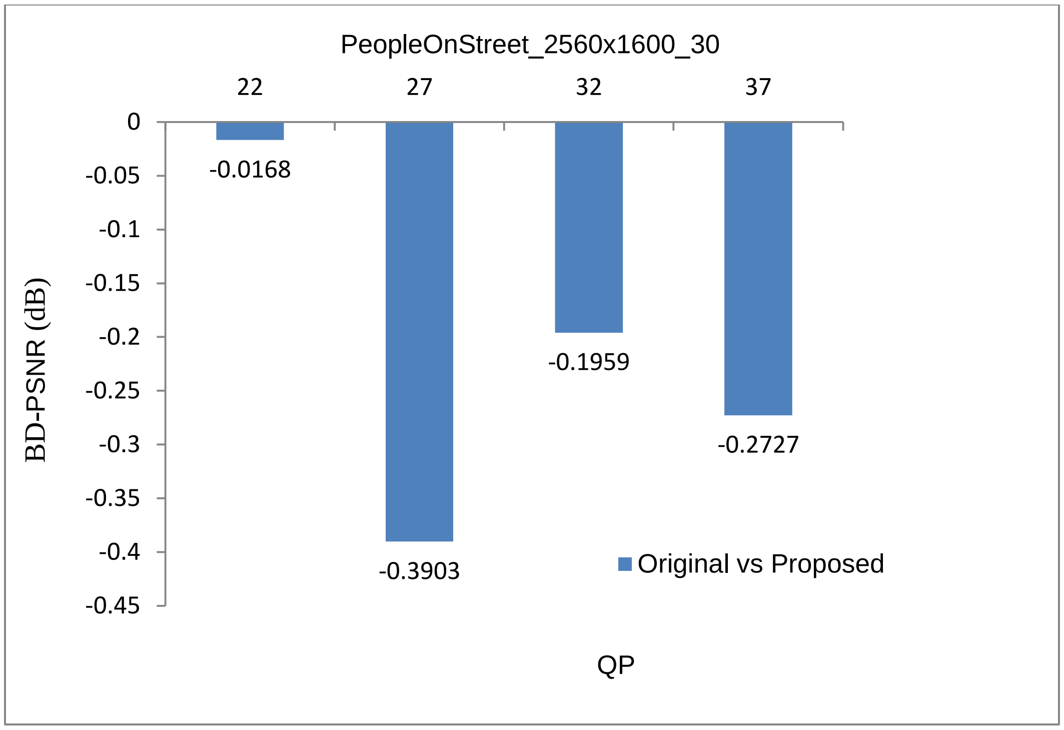


Figure 5-14 BD-PSNR vs quantization for PeopleOnStreet

5.3.4 BD- bitrate

Similar to BD-PSNR , BD-bitrate is also a metric to determine the quality of encoded video sequence. The relation between BD-PSNR and BD-bitrate is clearly shown in Fig. 5-15.As expected there is slight increase in bitrate for the proposed algorithm, making BD-bitrate positive as shown in results below from Fig. 5-16 through Fig. 5-20.

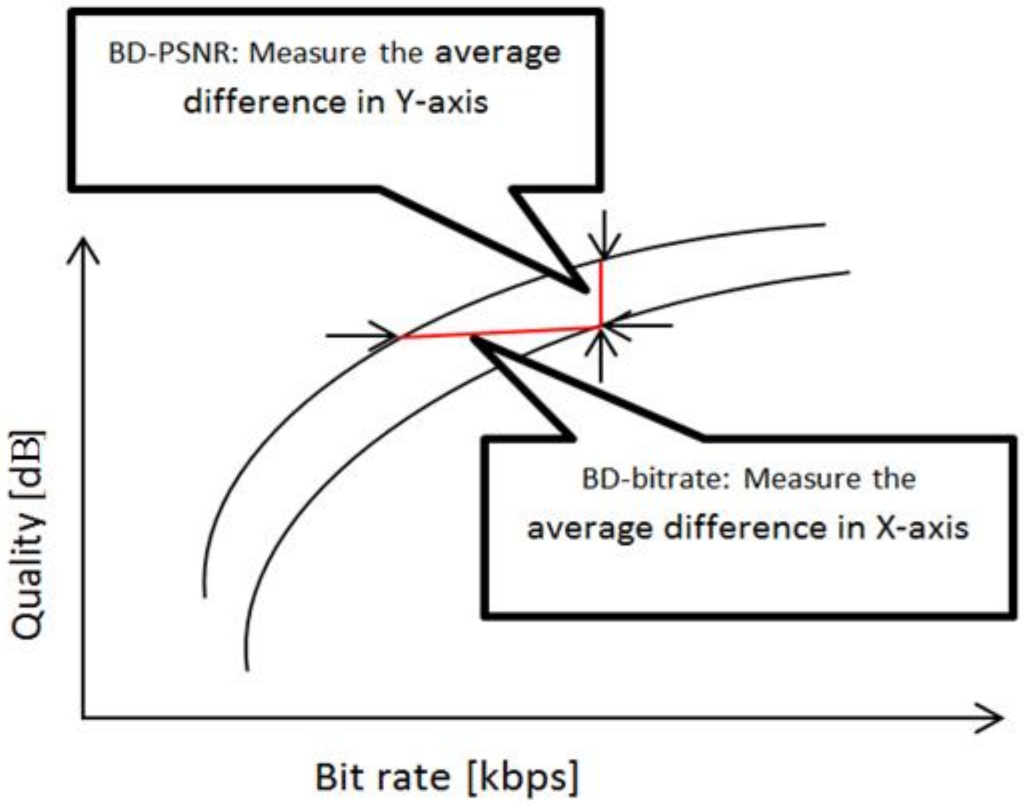

BD-PSNR: Measure the average difference in Y-axis
BD-bitrate: Measure the average difference in X-axis
Quality [dB]
Bit rate [kbps]

Figure 5-15 Measurement of BD-bitrate and BD-PSNR [78]

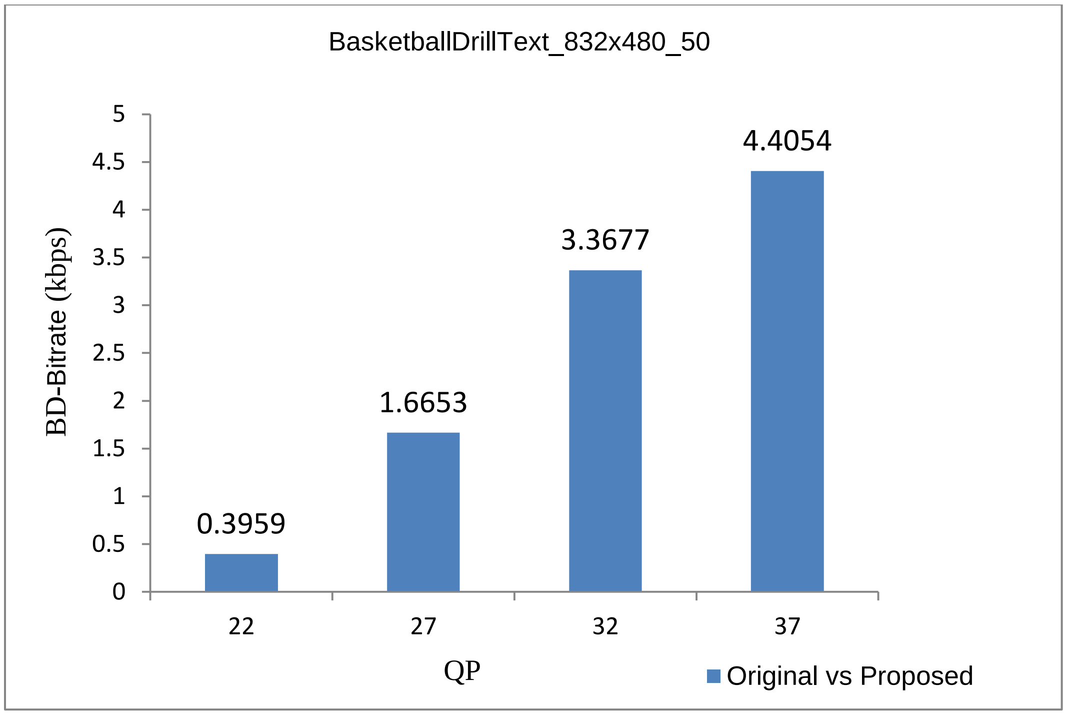


Figure 5-16 BD-Bitrate vs quantization for BasketballDrillText

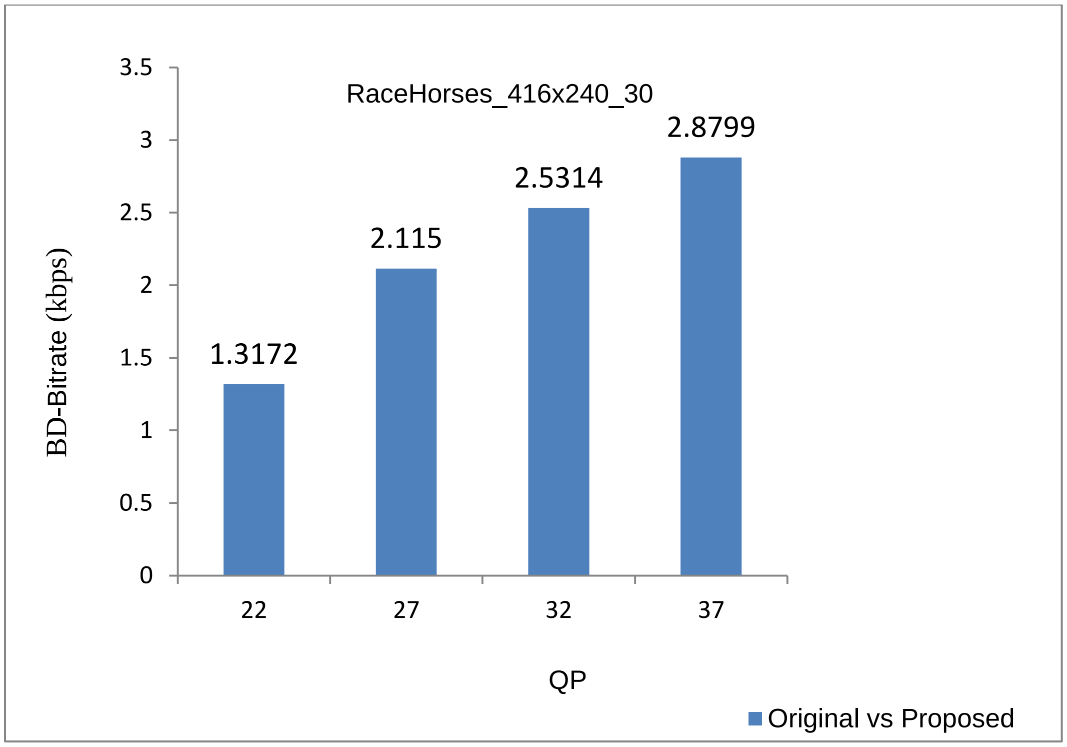


Figure 5-17 BD-Bitrate vs quantization for RaceHorses

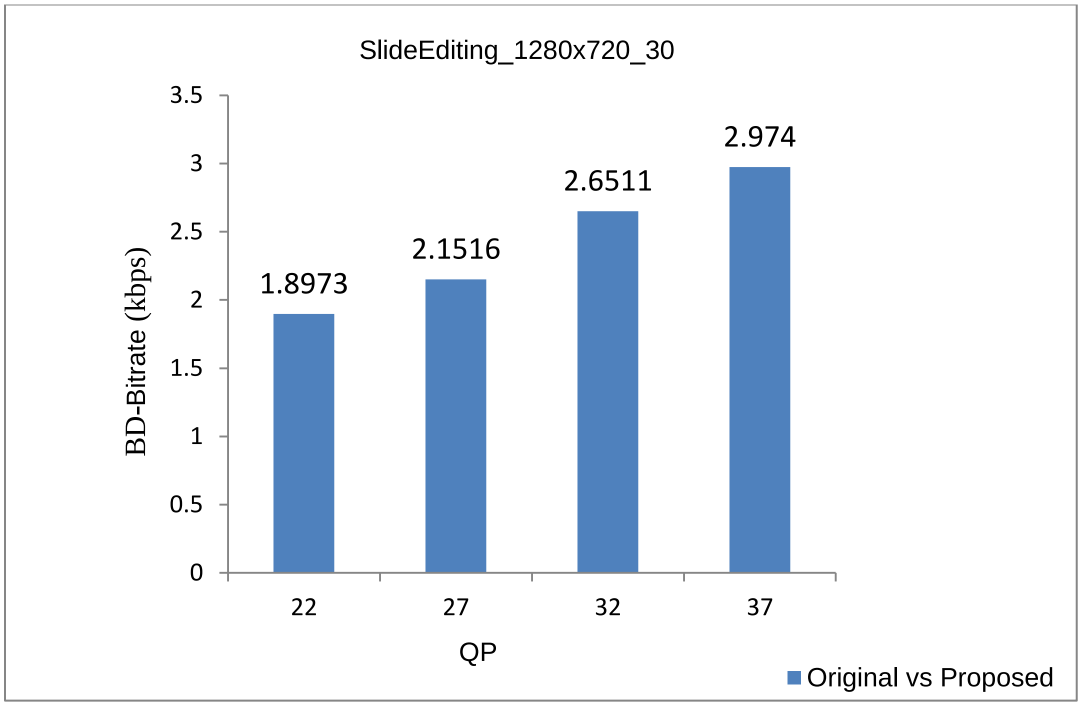


Figure 5-18 BD-Bitrate vs quantization for SlideEditing

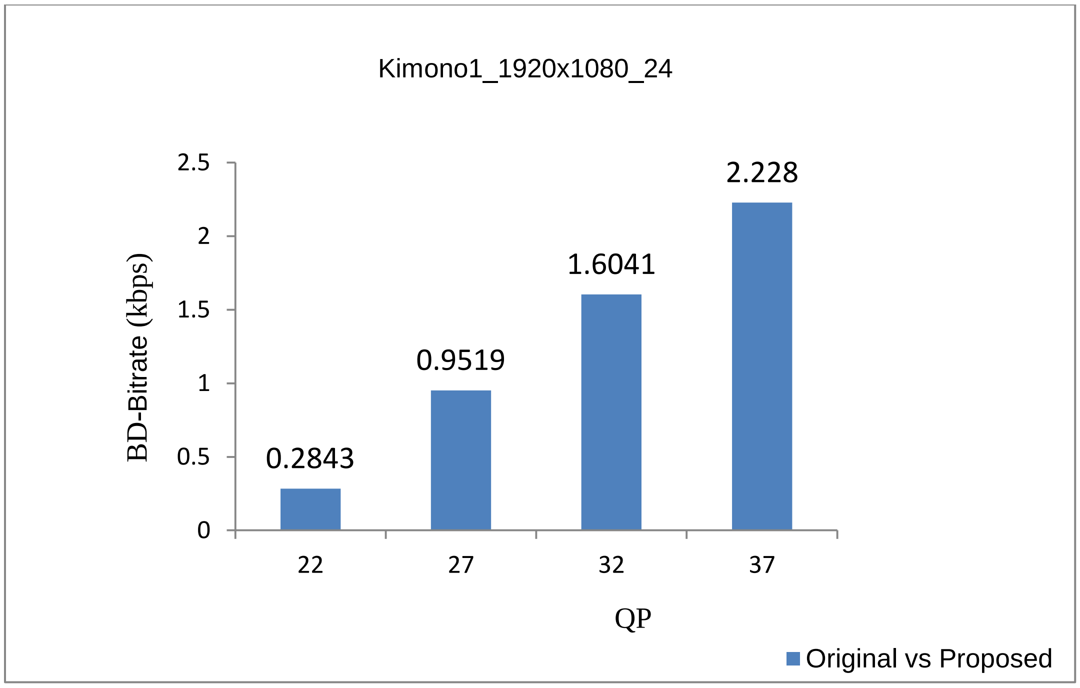


Figure 5-19 BD-Bitrate vs quantization for Kimono

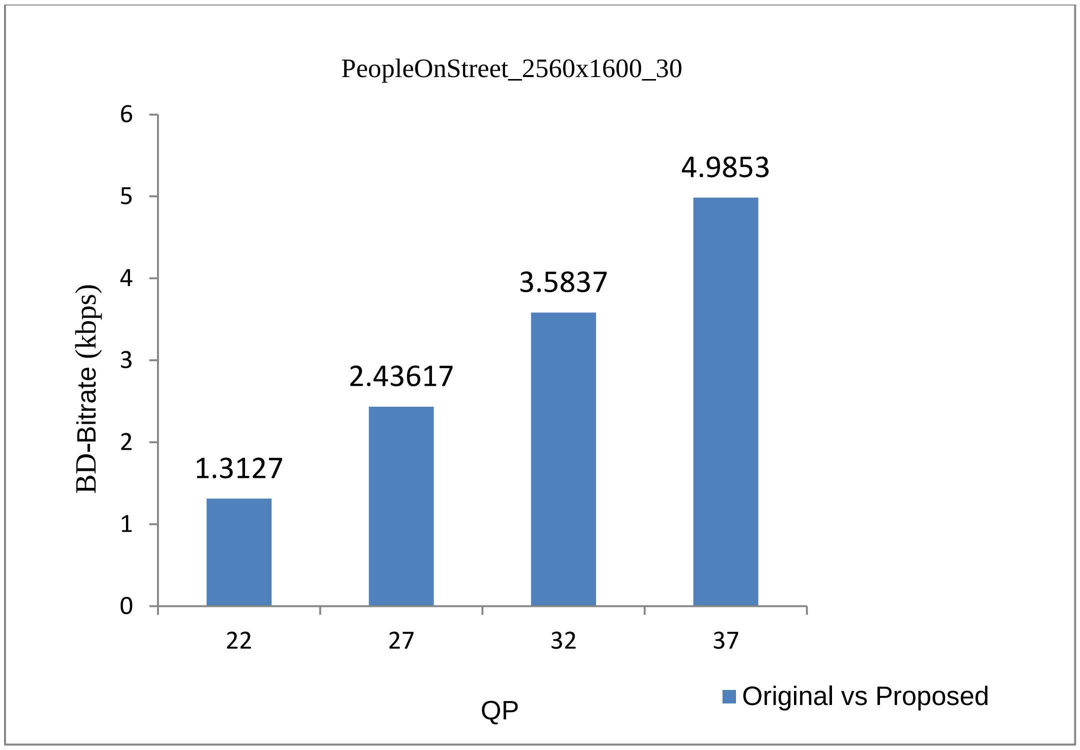


Figure 5-20 BD-Bitrate vs quantization for PeopleOnStreet

5.3.5 Rate Distortion Plot (RD Plot)

Rate distortion plot is basically used to evaluate the performance of an algorithm in terms of variation in PSNR value with respect to bitrate. As can be seen through Fig. 5-21 to Fig. 5-25, there is negligible loss in PSNR and negligible increase in bit rate for the proposed algorithm.

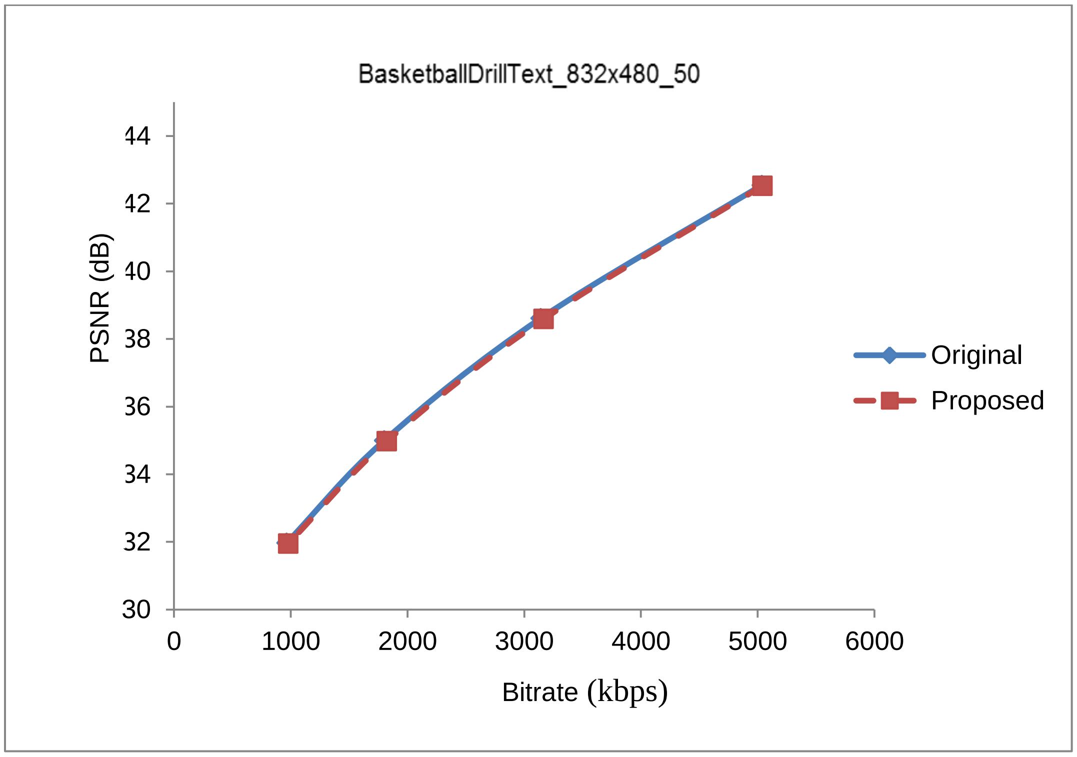


Figure 5-21 Rate distortion plot for BasketballDrillText

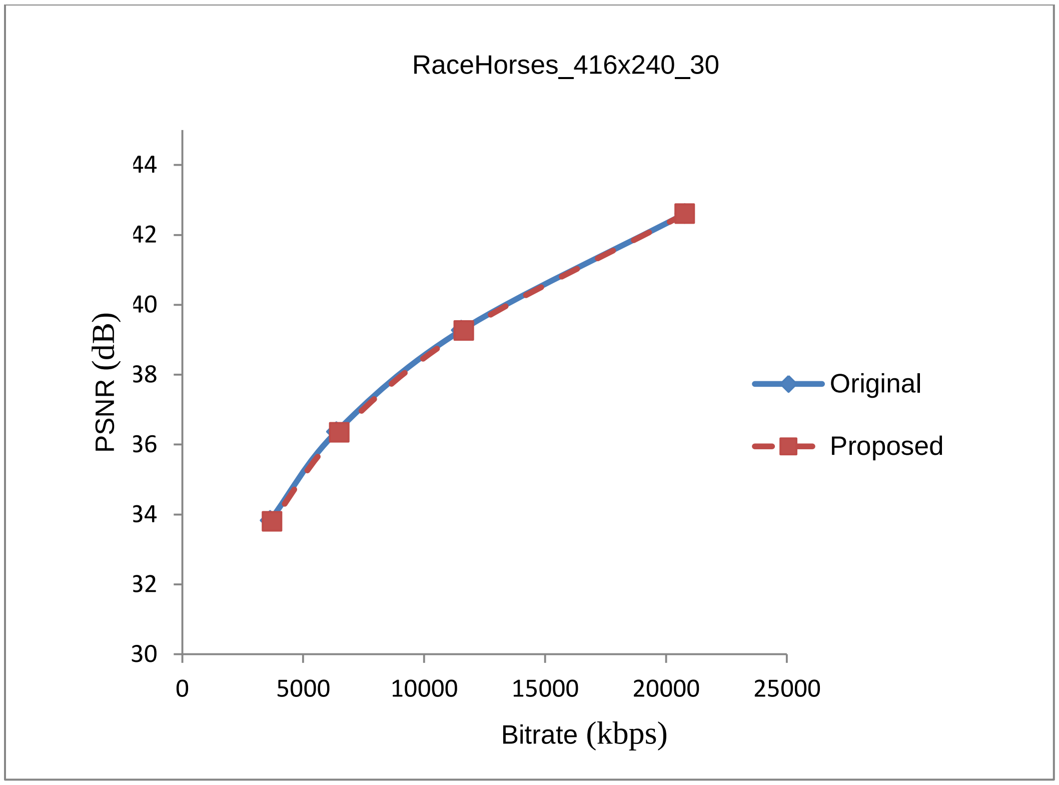


Figure 5-22 Rate distortion plot for RaceHorses

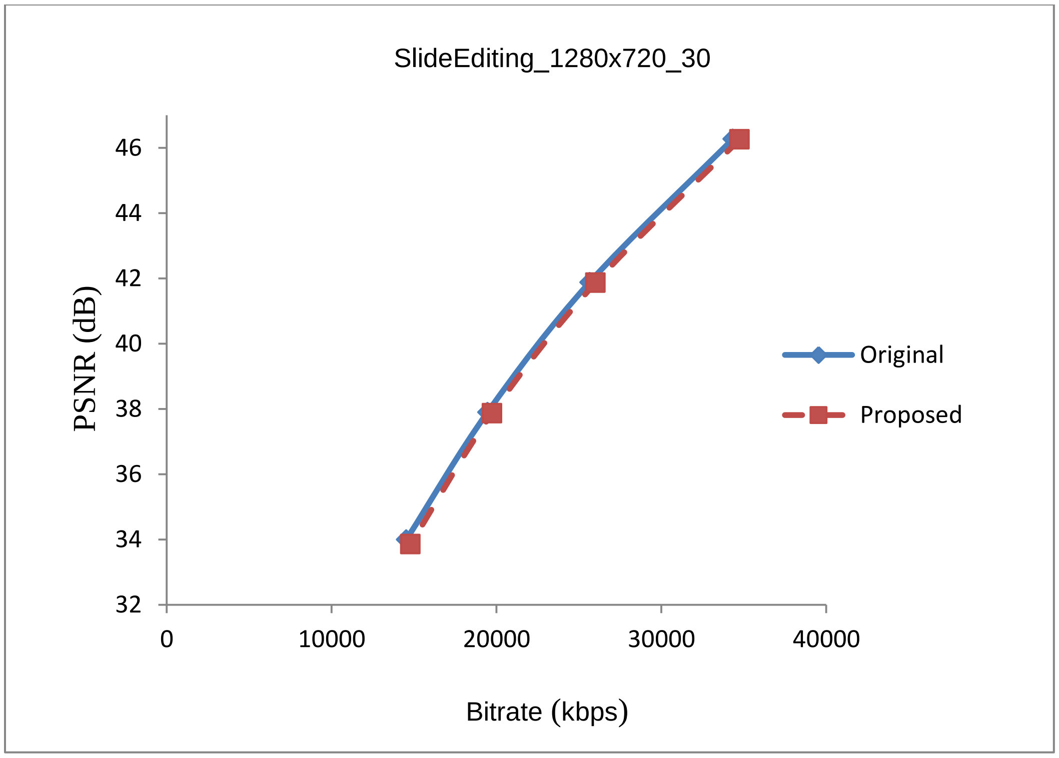


Figure 5-23 Rate distortion plot for SlideEditing

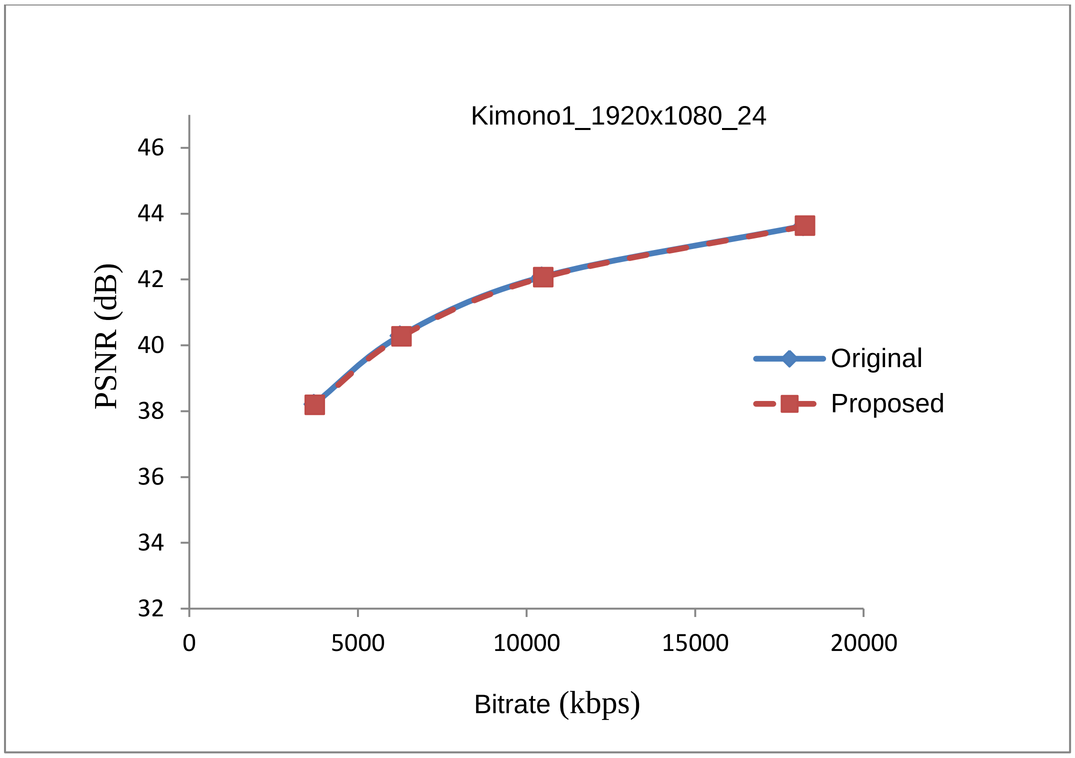


Figure 5-24 Rate distortion plot for Kimono

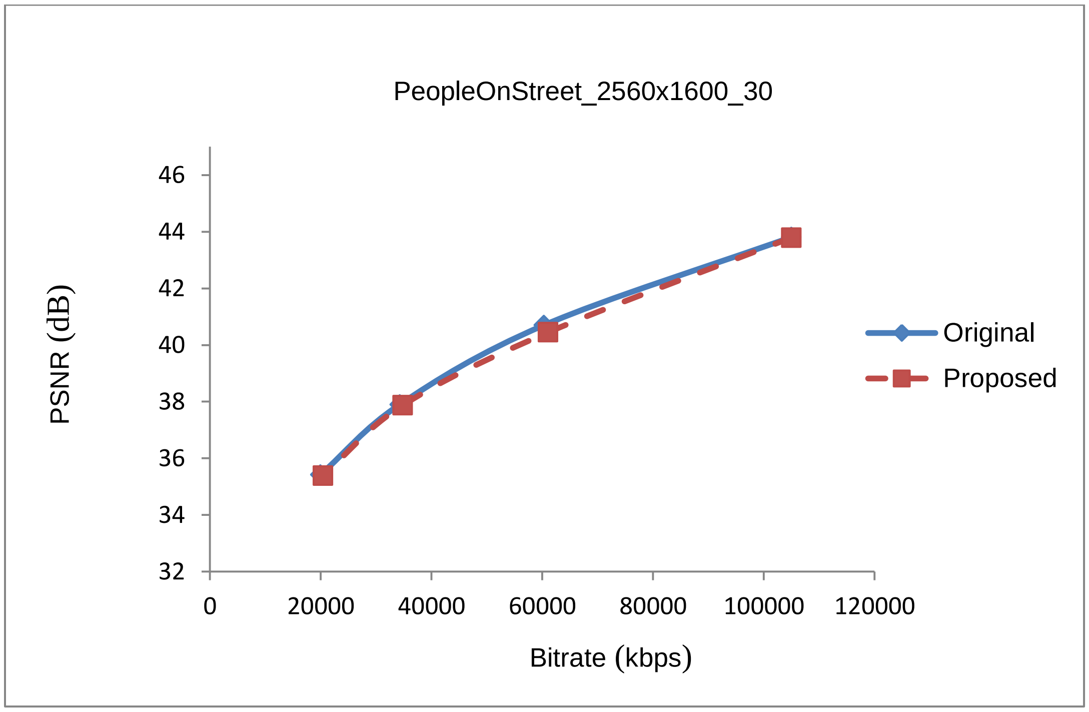


Figure 5-25 Rate distortion plot for PeopleOnStreet

5.3.6 Bit stream size gain

Figures 5-26 through 5-31 show the bit stream size of the HM 9.1 software vs proposed algorithm results for different video sequences. It can be concluded that there is a negligible increase in bit stream size as compared to the HM 9.1 software.

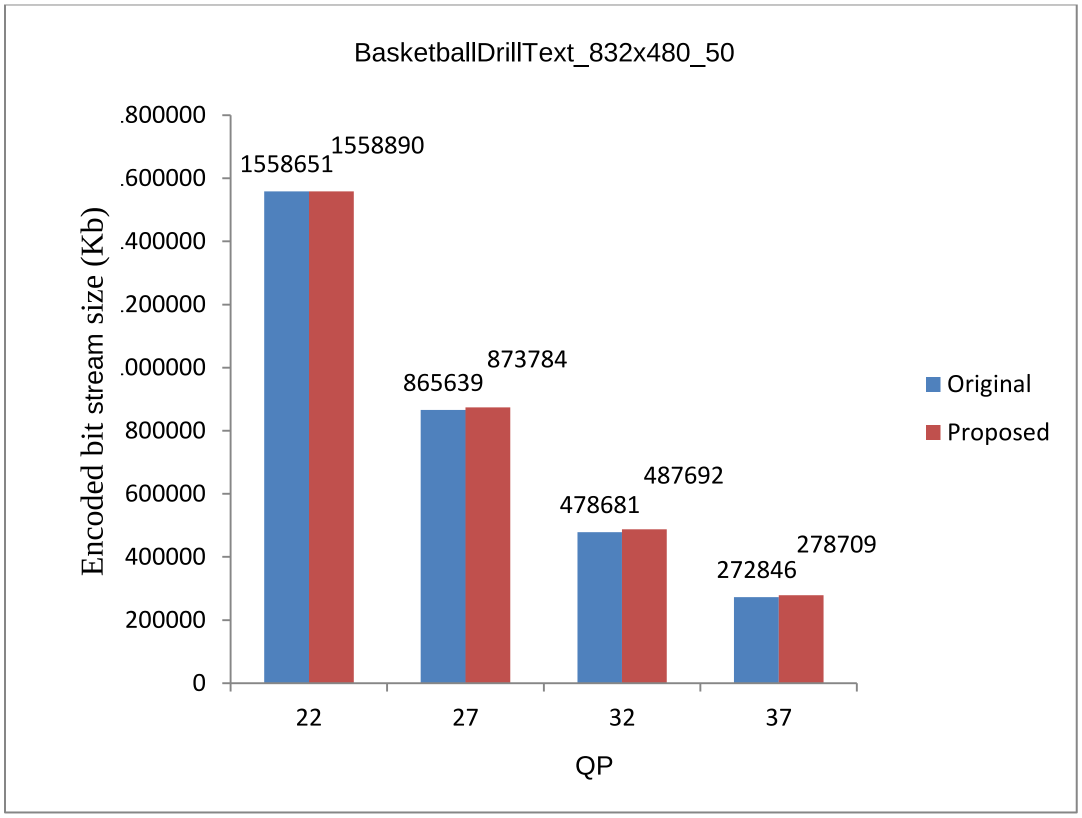


Figure 5-26 Encoded bit stream size vs quantization parameter for BasketballDrillText

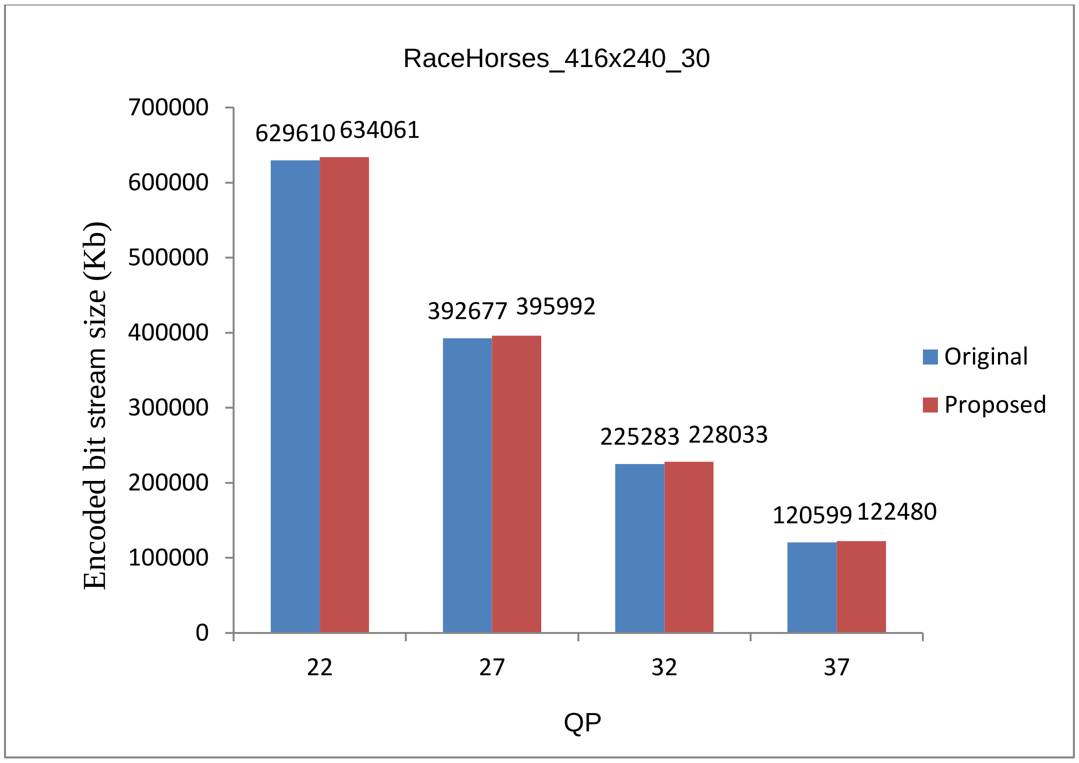


Figure 5-27 Encoded bit stream size vs quantization parameter for RaceHorses

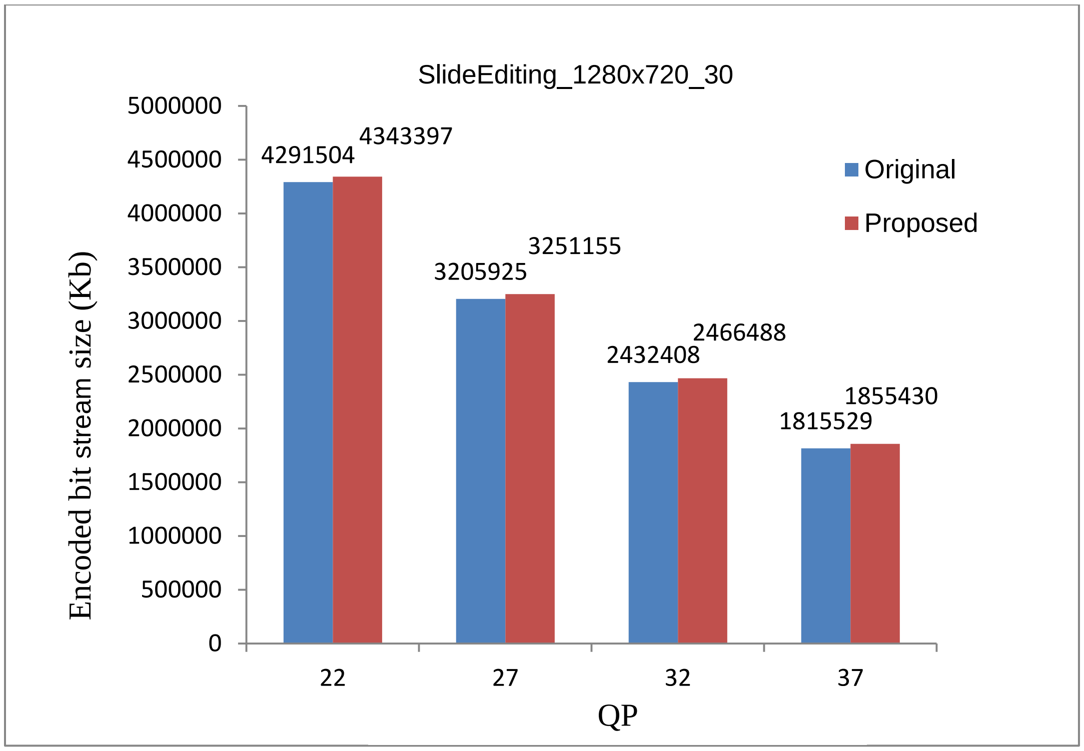


Figure 5-28 Encoded bit stream size vs quantization parameter for SlideEditing

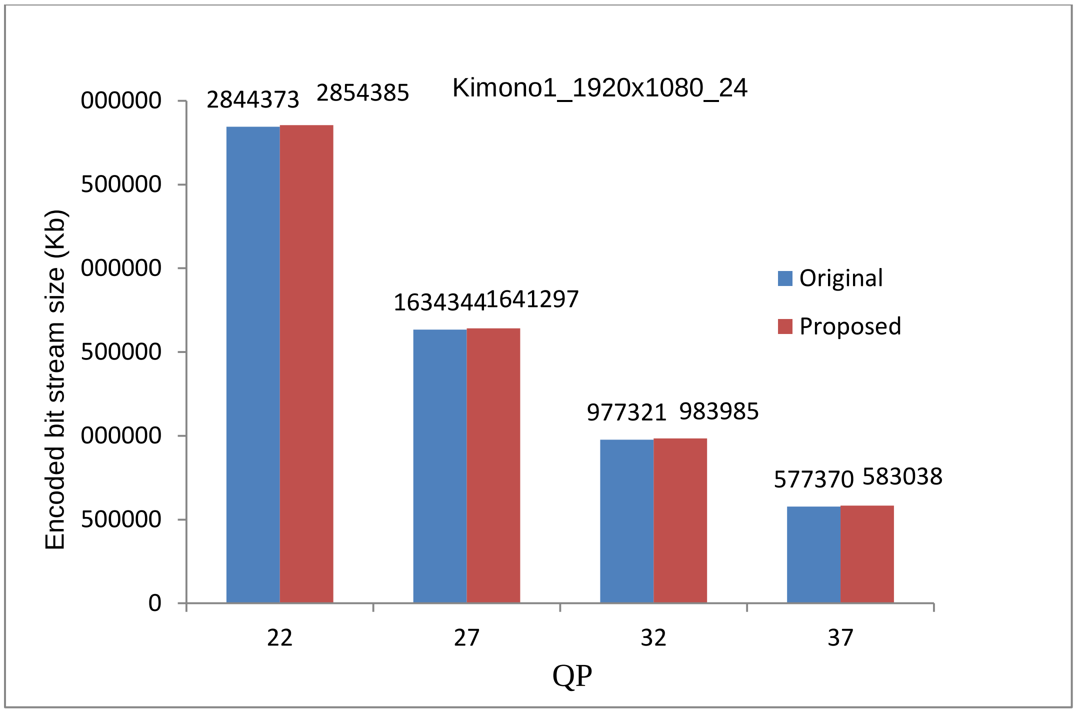


Figure 5-29 Encoded bit stream size vs quantization parameter for RaceHorses

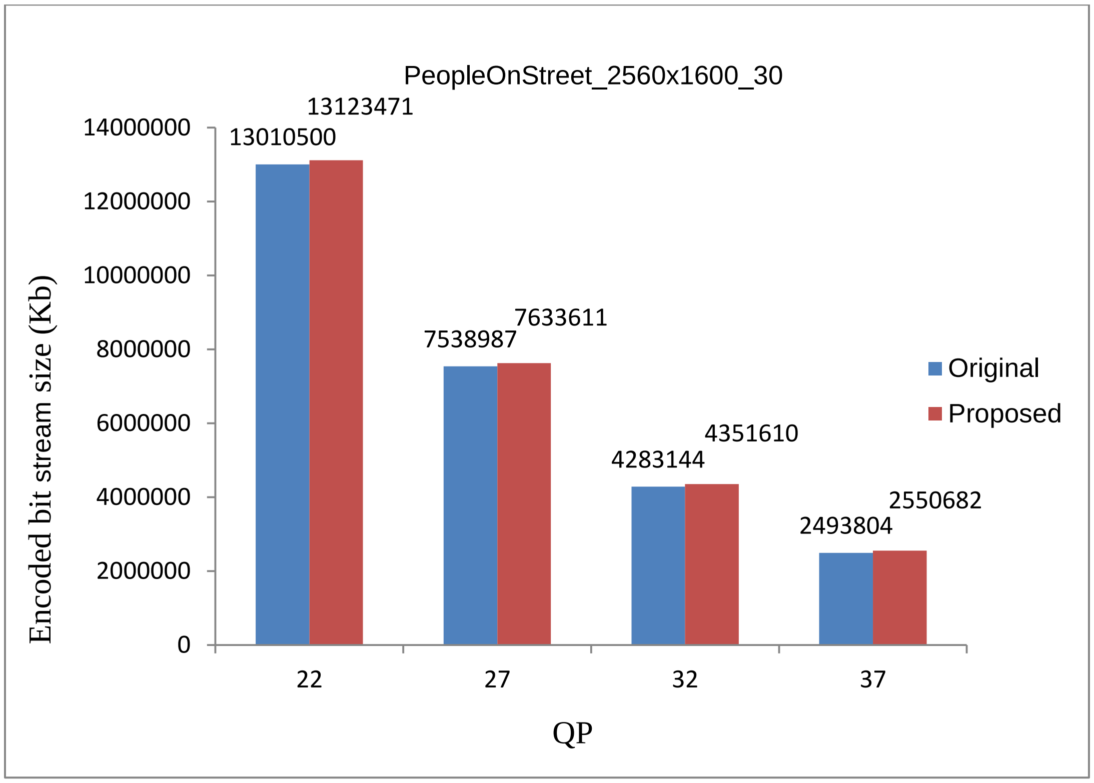


Figure 5-30 Encoded bit stream size vs quantization parameter for PeopleOnStreet

5.3.7 Percentage reduction in encoding time

Figures 5-31 through 5-35 show the reduction in encoding time from 9-59% reduction for different QP values corresponding to different test video sequences as compared to the HM 9.1 algorithm.

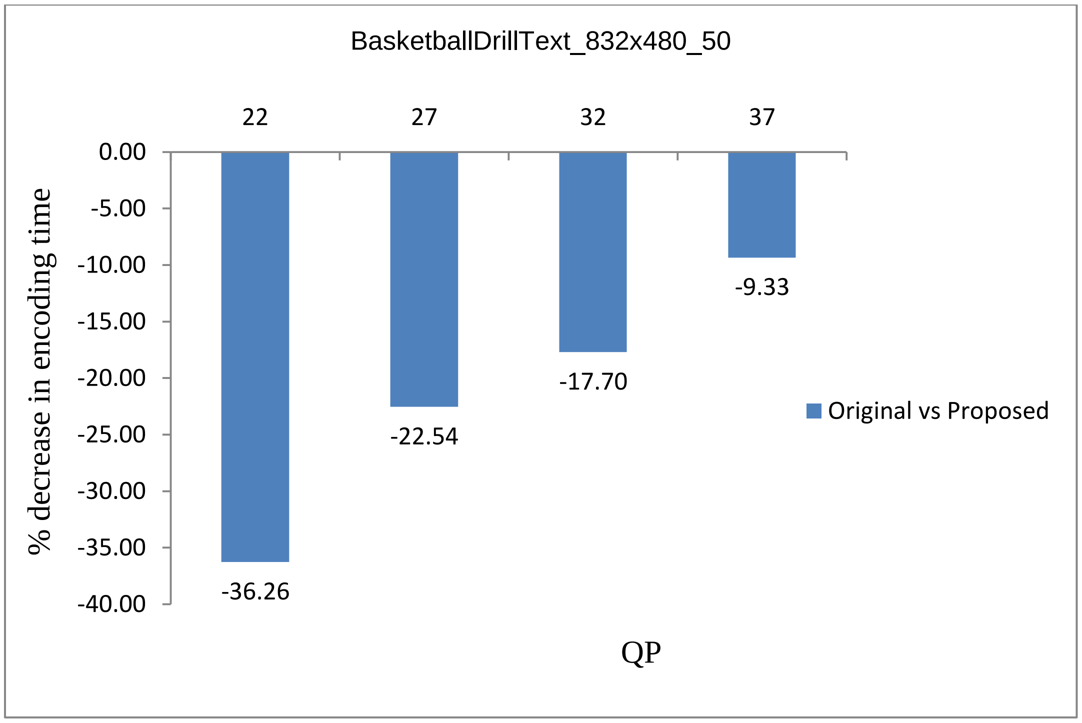


Figure 5-31 Percentage reduction in encoding time for BasketballDrillText

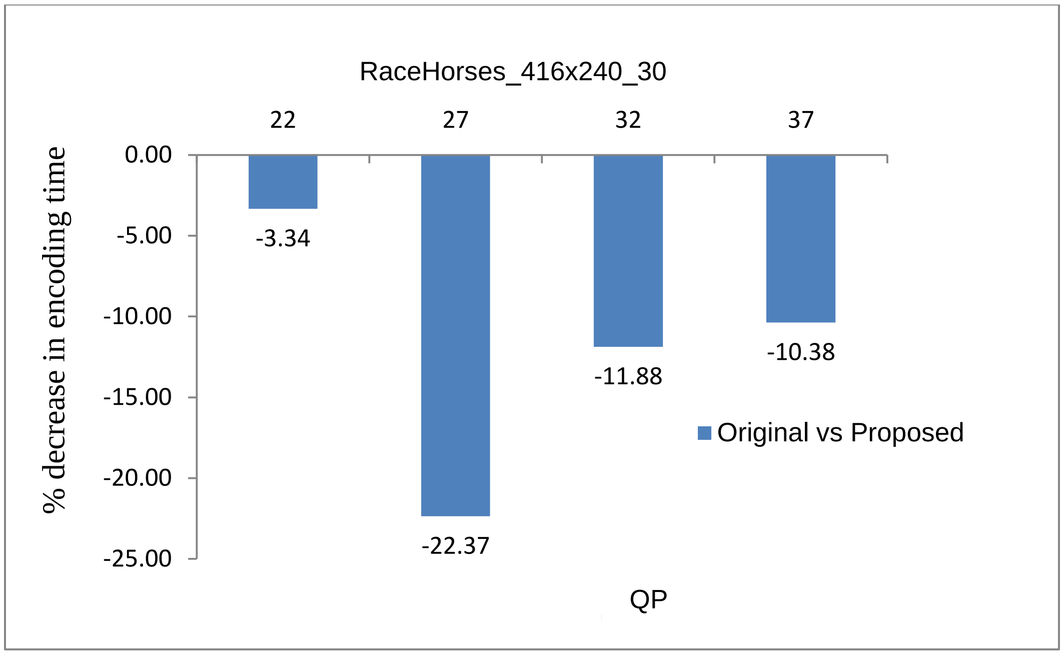


Figure 5-32 Percentage reduction in encoding time for RaceHorses

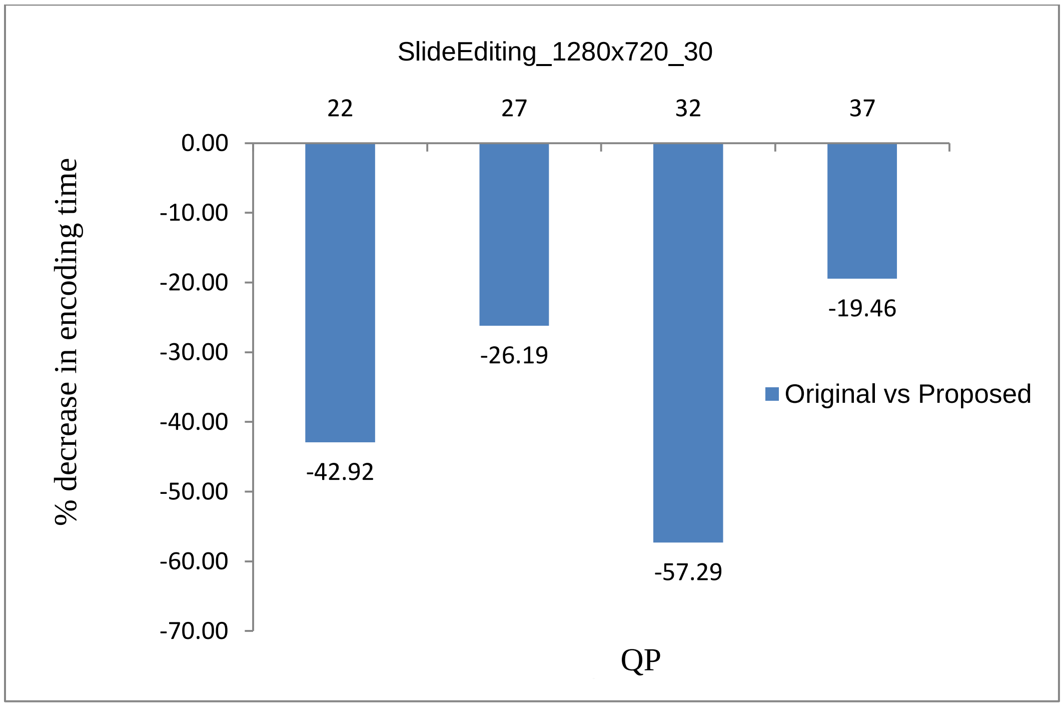


Figure 5-33 Percentage reduction in encoding time for SlideEditing

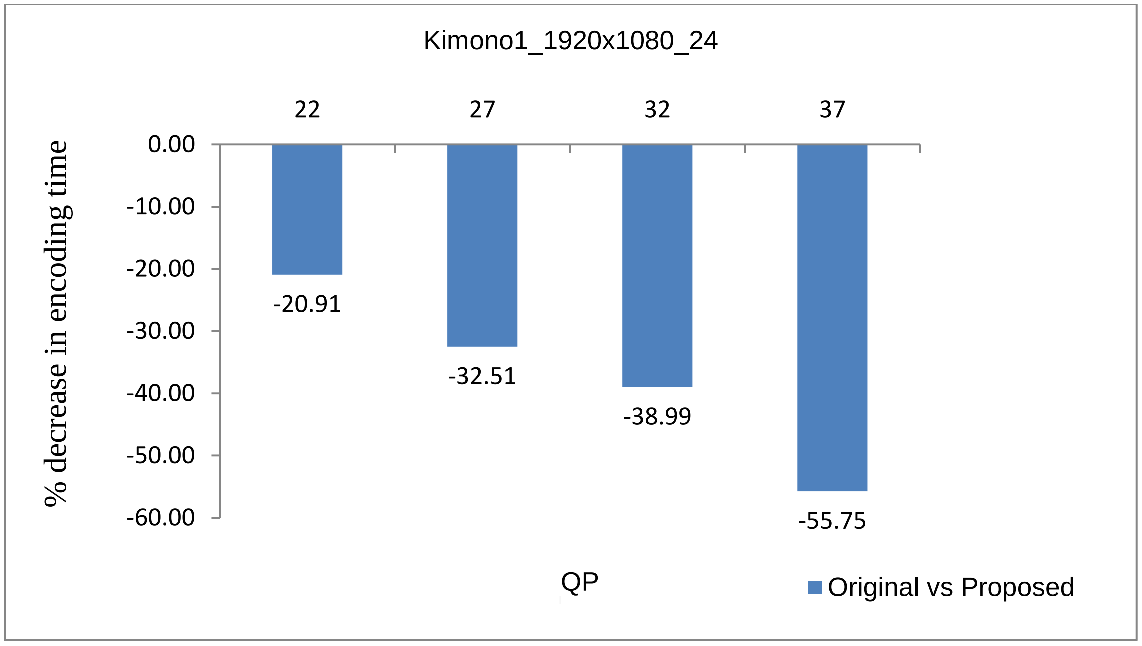


Figure 5-34 Percentage reduction in encoding time for Kimono

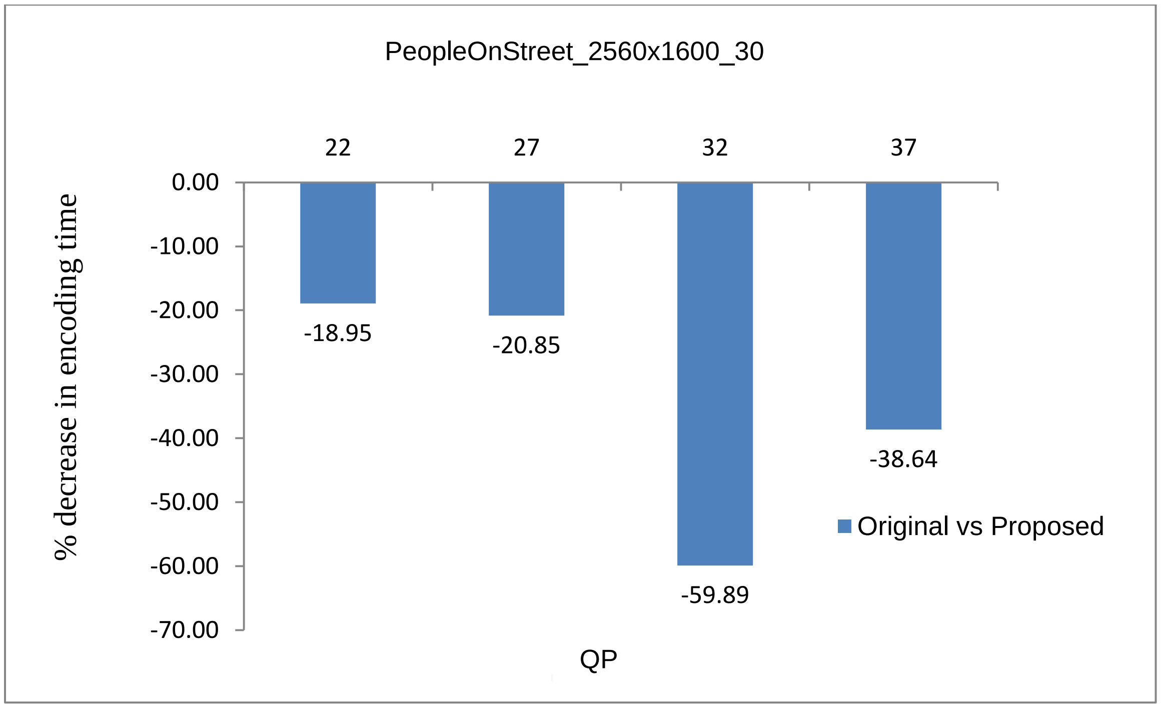


Figure 5-35 Percentage reduction in encoding time for PeopleOnStreet

5.4 Summary

This chapter describes the actual implementation of the parallel optimization of intramode selection in the HEVC standard using OpenMP. It also shows the results in terms of encoding time, BD-bitrate, BD-PSNR, rate distortion, bitstream size and percentage reduction in encoding time corresponding to different QP values for various standard test sequences. The next chapter will describe conclusions and future analysis to improve the proposed algorithm.

Chapter 6

Conclusions and Future work

6.1 Conclusions

The main aim of this thesis is to introduce parallel optimization of mode decision process for intra prediction in the HEVC standard. The increase in the number of angular modes in intra prediction has no doubt improved performance but at the cost of an increase in processing time. The proposed parallel approach gives an overview of the possible performance improvement in the HEVC standard. The results show that encoding time can by reduced by approximately 35-40% on average as compared to the HM 9.1 encoder. There is a negligible drop in image quality with a slight increase in bitrate for different values of the quantization parameter values on various test sequences. Apart from encoding time the parameters that are taken into consideration to evaluate the proposed technique are BD-PSNR, BD-bitrate, bitstream size and rate distortion plot.

6.2 Future Work

As discussed in Chapter 4, the proposed method is introduced to get an estimate of the effect of parallel approach in reduction of encoding time. There are many other effective techniques to implement parallelism to different sections of the software that consumes more processing time.

Parallel programming by Open MP is much simpler to understand and easier to debug as compared to other parallel processing techniques It also also helps to improve the portability between multiple platforms [72]. But the Fork-join model of the Open MP program may impose an additional thread in nested loops during thread creation [79]. For example, if there is a *parallel do* directive on an inner loop, then it will incur the parallel overhead n-1 times, where n is the number of iterations [80]. If the inner loop is parallelized, the iteration is much lower but the overhead of work sharing is much higher [81]. There are many approaches to solve this overhead problem but care must be taken to eliminate thread dependency as it might result in increase in encoding time as compared to original software. The encoding time mentioned in Chapter 5 has been determined by deducting the calculated overhead time from the actual encoding time.

In parallel computing model like Computer Unified Device Architecture (CUDA) [82] invented by NVIDIA, the thread creation overhead can be reduced as CUDA threads are extremely light weight, with very low creation overheads and switching time.

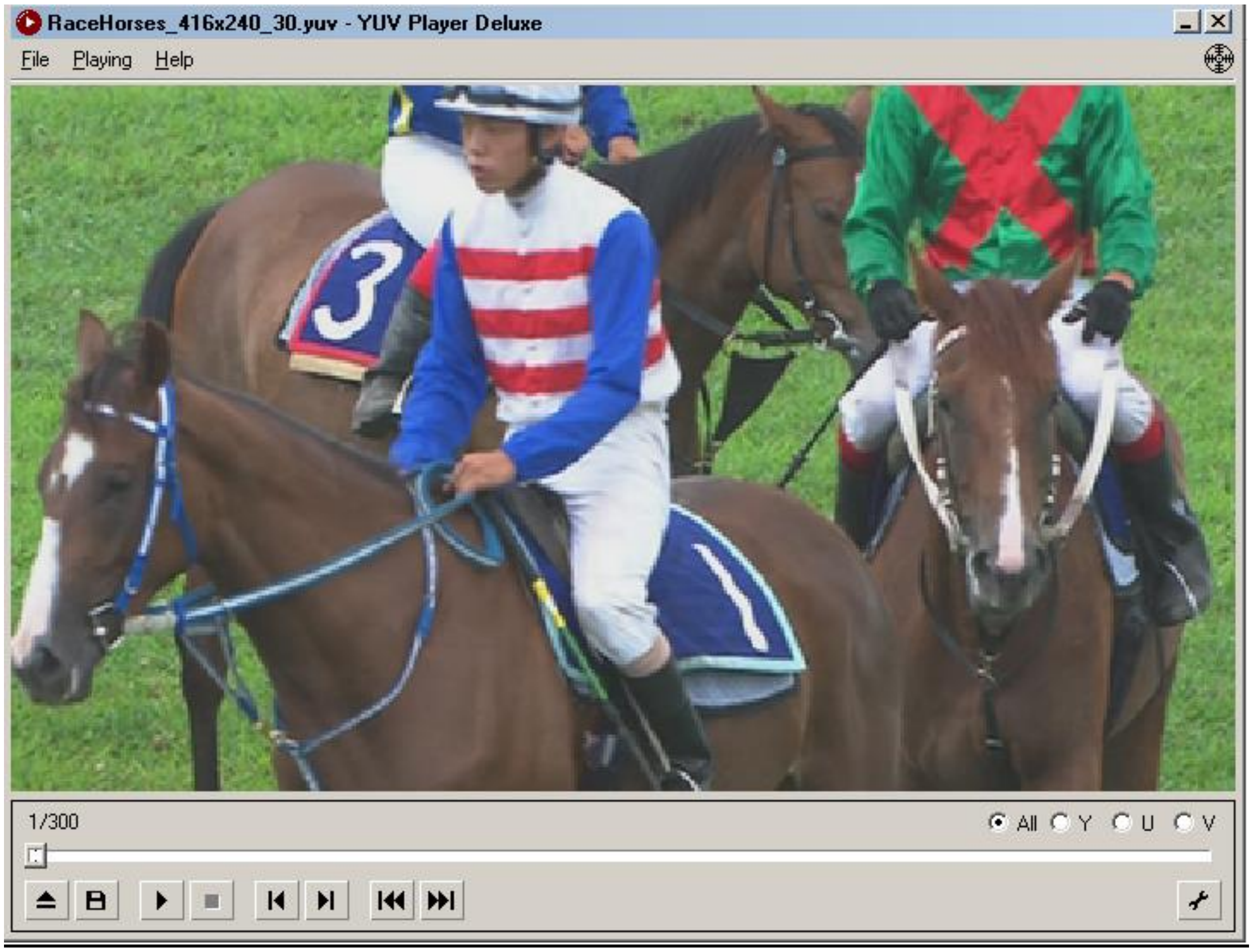
The use of POSIX threads (pthreads) [84] provides thread pool which reduces overhead in thread creation as they already pre-allocated before the master thread begins dispatching threads to work. The tasks are processed in order, usually faster and can be done by creating a thread per task [83].

Appendix A

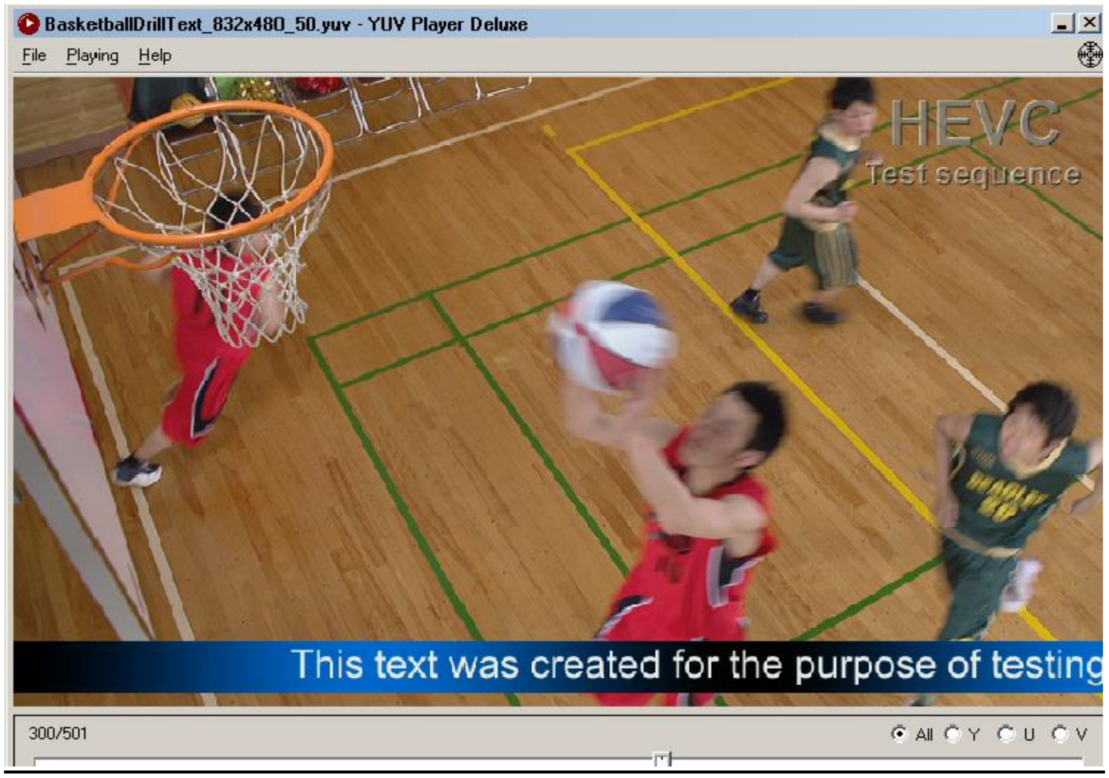
Test Sequences [15]

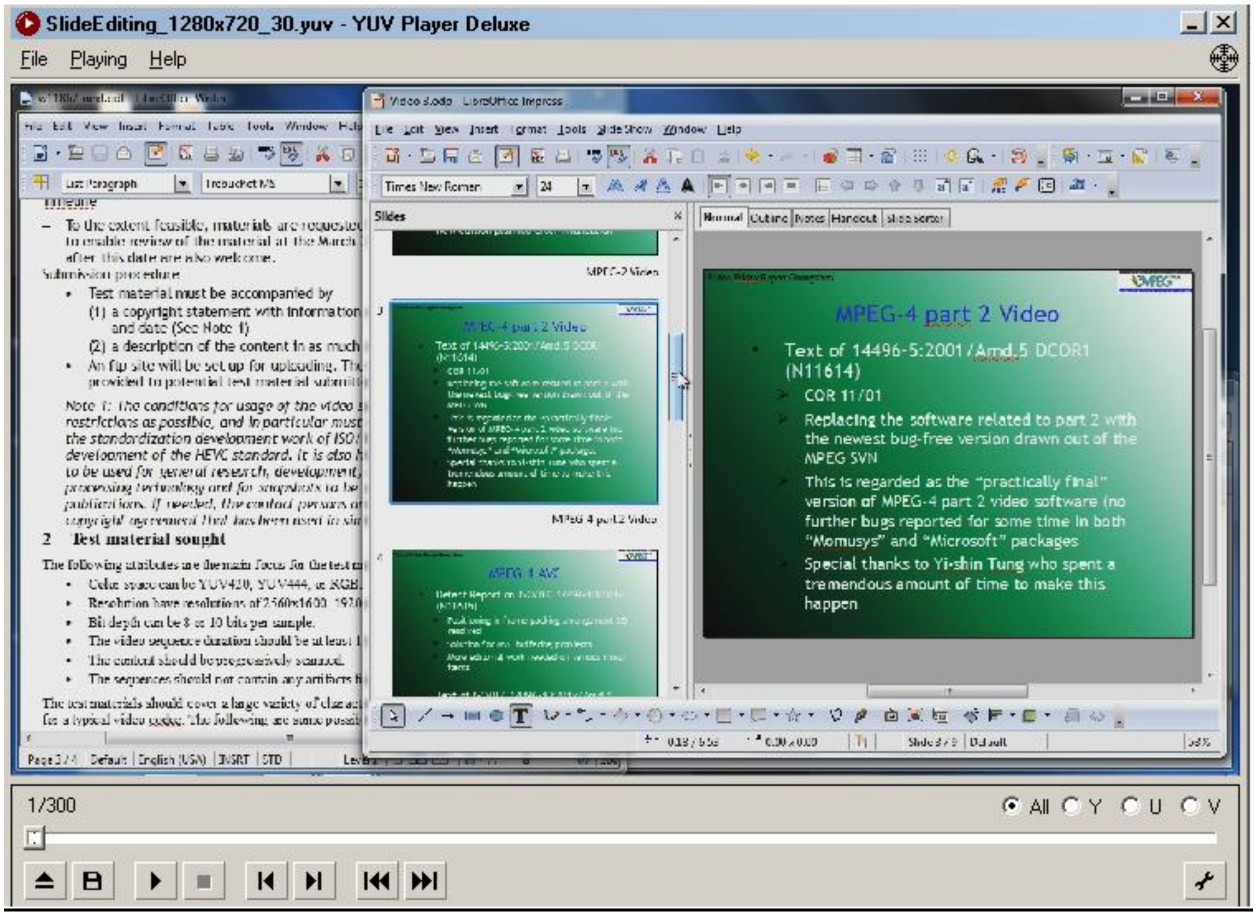
The following standard test sequences have been used in chapter 5 to obtain the results for the proposed algorithm. These test sequences are arranged in increasing order of video resolution as mentioned in Table 5-2.

A.1 Racehorses



A.2 BasketballDrillText


BasketballDrillText_832x480_50.yuv - YUV Player Deluxe
File
Playing
Help
HEVC
Test sequence
This text was created for the purpose of testing
300/501
All
Y
U
V



MPEG-4 part 2 Video

- Text of 14496-5:2001/Amd.5 DCOR1 (N11614)
  - COR 11/01
  - Replacing the software related to part 2 with the newest bug-free version drawn out of the MPEG SVN
  - This is regarded as the "practically final" version of MPEG-4 part 2 video software (no further bugs reported for some time in both "Momusys" and "Microsoft" packages
  - Special thanks to Yi-shin Tung who spent a tremendous amount of time to make this happen

timeline

– To the extent feasible, materials are requested
to enable review of the material at the March
after this date are also welcome.

Submission procedure

- Test material must be accompanied by
  (1) a copyright statement with information
  and date (See Note 1)
  (2) a description of the content in as much
- An ftp site will be set up for uploading. The
  provided to potential test material submitt

Note 1: The conditions for usage of the video s restrictions as possible, and in particular must the standardization development work of ISO/ development of the HEVC standard. It is also h to be used for general research, development, processing technology and for snapshots to be publications. If needed, the contact persons a copyright agreement that has been used in sim

2 Test material sought

The following attributes are the main focus for the test m

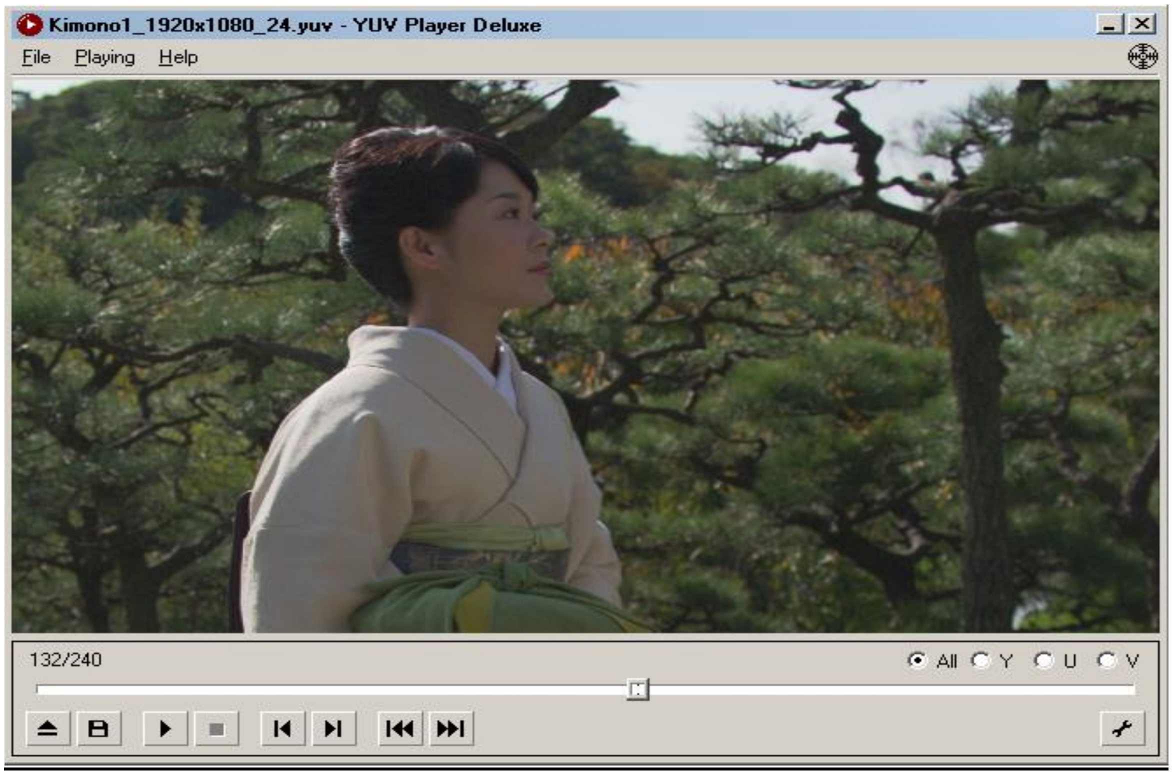
- Color space can be YUV420, YUV444, or RGB.
- Resolution have resolutions of 2560x1600, 1920
- Bit depth can be 8 or 10 bits per sample.
- The video sequence duration should be at least 1
- The content should be progressively scanned.
- The sequences should not contain any artifacts fr

The test materials should cover a large variety of charac for a typical video codec. The following are some possib

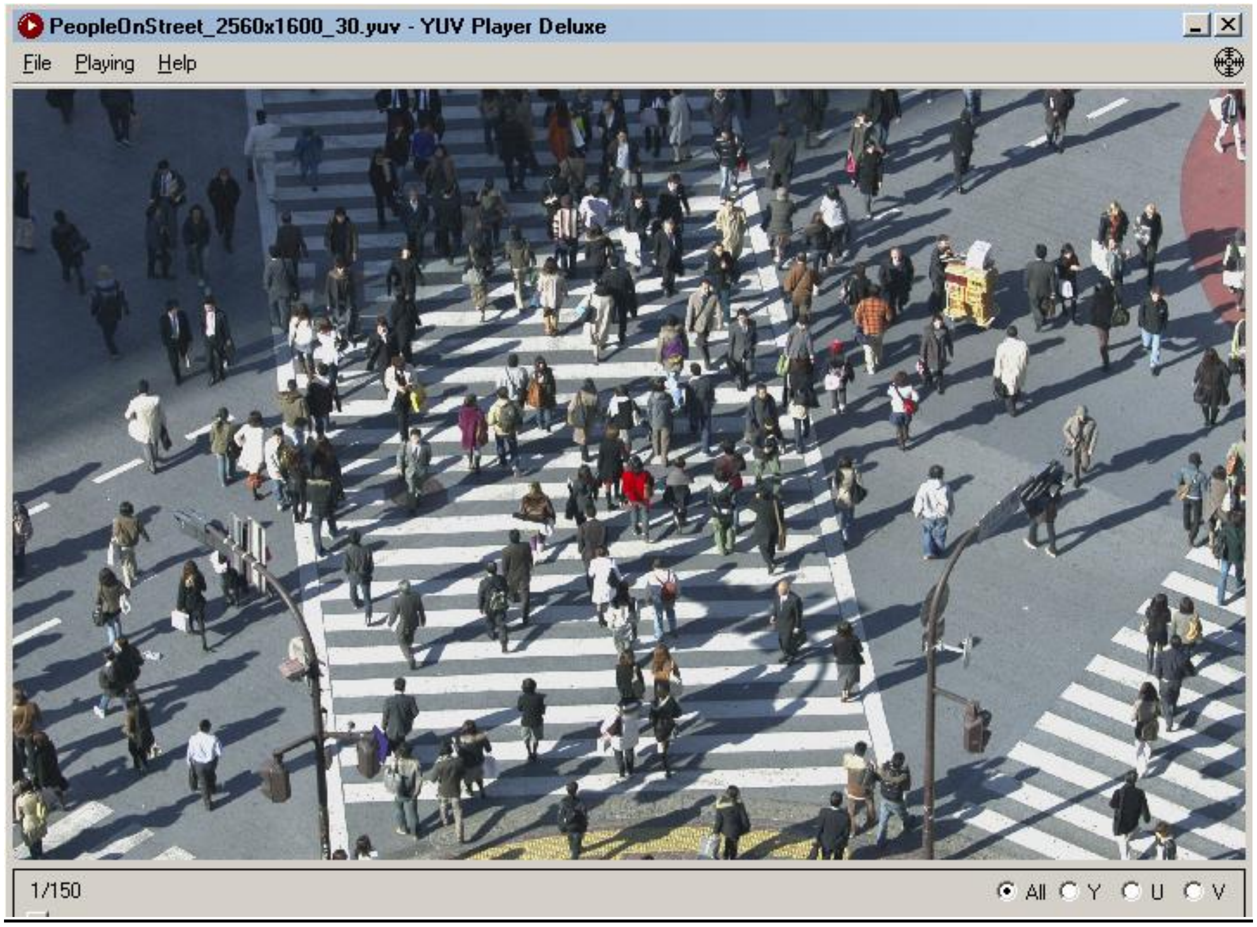
Page 3 / 4 Default English (USA) INSRT STD

Slide 3 / 9 Default

A.4 Kimono


Kimono1_1920x1080_24.yuv - YUV Player Deluxe
File
Playing
Help
132/240
All
Y
U
V

A.5 PeopleOnStreet



Appendix B

Test Conditions

The proposed thesis was implemented on Intel i5 processor at 2.67 GHz frequency. The RAM size was 4. GB and operating system is 64 bit Window8.1 OS. The reference software that was used was HM 9.1 [11].

Appendix C

BD-PSNR and BD-bitrate

ITU - Telecommunications Standardization Sector
STUDY GROUP 16 Question 6
Video Coding Experts Group (VCEG)

Document VCEG-M33
Filename: VCEG-M33.doc
Generated: 26 March '01

Thirteenth Meeting: Austin, Texas, USA, 2-4 April, 2001

Question:	Q.6/SG16 (VCEG)

Source: Gisle Bjontegaard
Telenor Satellite Services
P.O.Box 6914 St.Olavs plass
N-0130 Oslo, Norway
Tel: +47 23 13 83 81
Fax: +47 22 77 79 80
Email: gisle.bjontegaard@telenor.com

Title: Calculation of average PSNR differences between RD-curves

Purpose: Proposal

Introduction

VCEG-L38 defines "Recommended Simulation Conditions for H.26L". One of the outcomes is supposed to be RD-plots where PSNR and bitrate differences between two simulation conditions may be read. The present document describes a method for calculating the average difference between two such curves. The basic elements are:

- Fit a curve through 4 data points (PSNR/bitrate are assumed to be obtained for QP = 16,20,24,28)
- Based on this, find an expression for the integral of the curve
- The average difference is the difference between the integrals divided by the integration interval

IPR

"The contributor(s) are not aware of any issued, pending, or planned patents associated with the technical content of this proposal."

Fitting a curve

A good interpolation curve through 4 data points of a "normal" RD-curve (see figure 1) can be obtained by:

SNR = (a + b*bit + c*bit2)/(bit + d)

where a,b,c,d are determined such that the curve passes through all 4 data points.

This type of curve is well suited to make interpolation in "normal" luma curves. However, the division may cause problems. For certain data (Jani pointed out some typical chroma data) the obtained function may have a singular point in the range of integration - and it fails.

Use of logarithmic scale of bitrate

When we look at figure 1, the difference between the curves is dominated by the high bitrates.

- The range (1500-2000) gets 4 times the weight of the range (375-500) even if they both represent a bitrate variation of 33%

Hence it was considered to be more appropriate to do the integration based on logarithmic scale of bitrate. Figure 2 shows a plot where "Logarithmic x-axes" is used in the graph function of Excel. However, this function has no flexibility and only allows factors of 10 as units.

In figure 3 I first took the logarithm of bitrates and the plot has units of "dB" along both axes. The factor between two vertical gridlines in the plot is: $10^{0.05} = 1.122$ (or 12.2%). Could this be an alternative way of presenting RD-plots?

Interpolation with logarithmic bitrate scale

With logarithmic bitrate scale the interpolation can also be made more straight forward with a third order polynomial of the form:

SNR = a + b*bit + c*bit2 + d*bit3

To further improve the approximation accuracy a piece-wise cubic interpolation is proposed as an alternative (See page 282 in Chapter 9 of the book cited at the end).

This result in good fit and there is no problems with singular points. This is therefore the function I have used for the calculations in VCEG-M34. However, for integration of luma curves the results are practically the same as with the first integration method which was used for the software distributed by Michael regarding the complexity experiment.

In the same way we can do the interpolation to find Bit as a function of SNR:

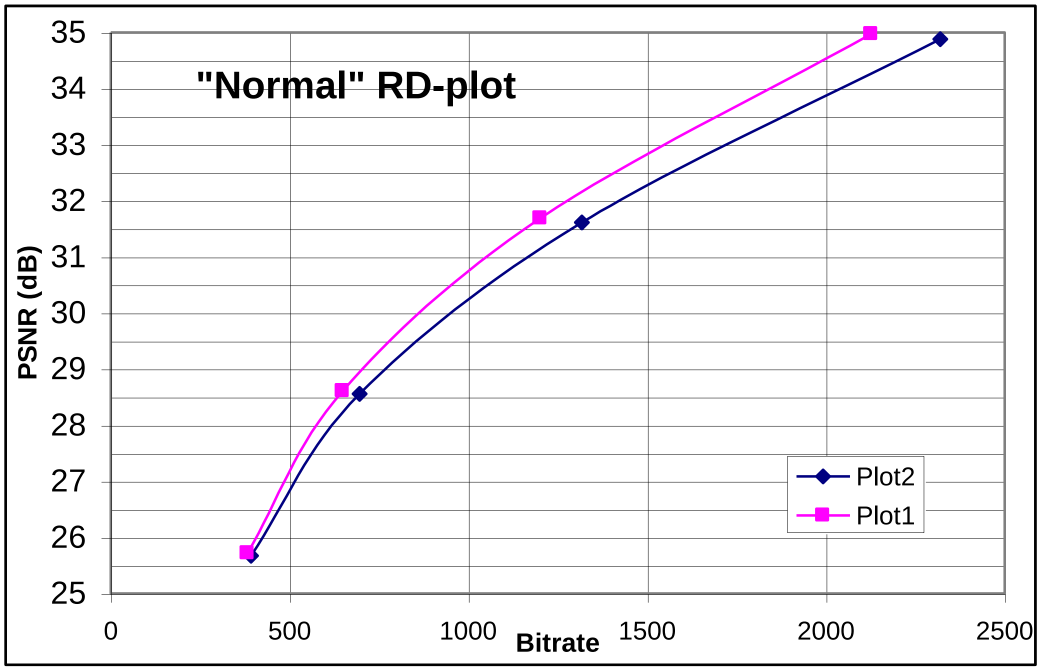
$\mathrm{SNR} = a + b*\mathrm{SNR} + c*\mathrm{SNR}^2 + d*\mathrm{SNR}^3$

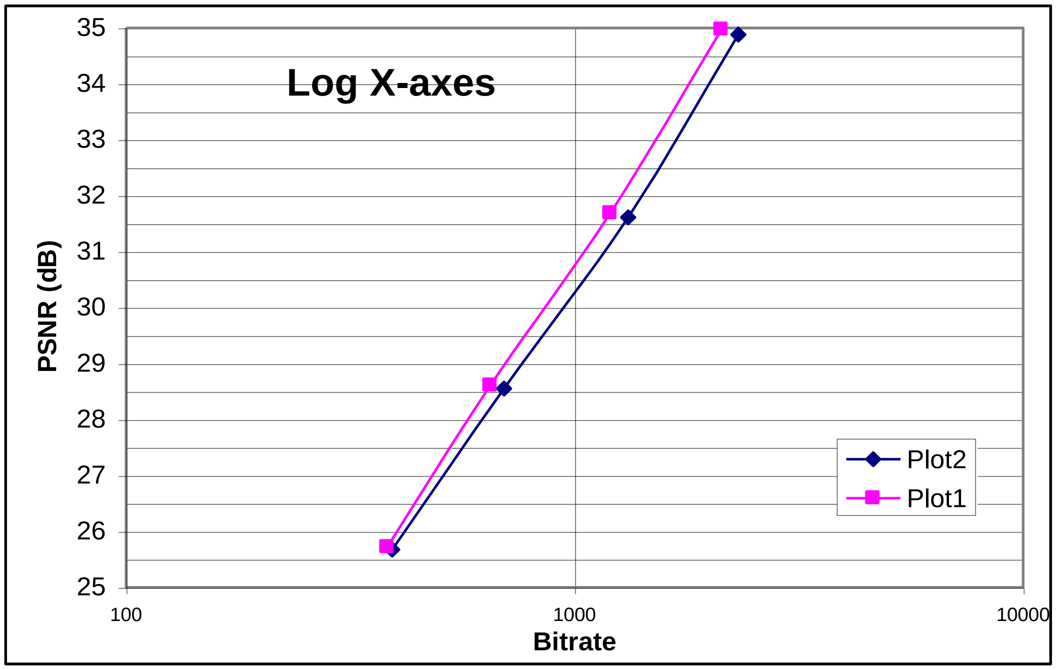
In this way we can find both:

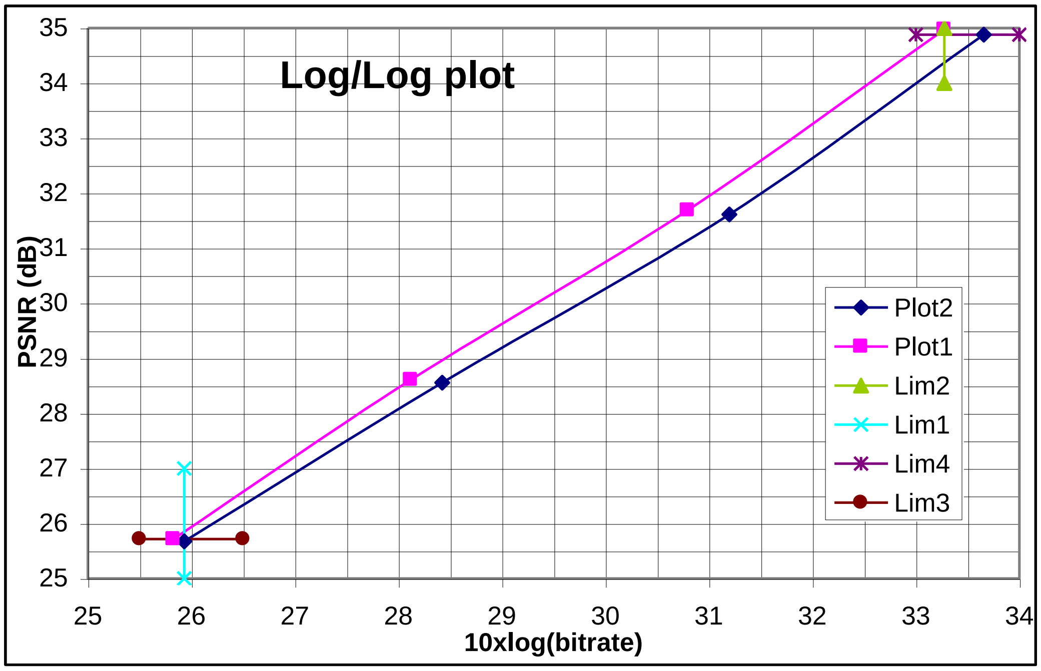
- Average PSNR difference in dB over the whole range of bitrates
- Average bitrate difference in % over the whole range of PSNR
- On request from Michael average differences are found over the whole simulation range (see integration limits in figure 3) as well as in the middle section - called mid range.
- As a result VCEG-M34 shows 4 separate data tables.

Conclusions

- It is proposed to include this method of finding numerical averages between RD-curves as part of the presentation of results. This is a more compact and in some sense more accurate way to present the data and comes in addition to the RD-plots.
- The distinction between "total range" and "mid range" does not seem to add much and it is therefore proposed to use "total range" only.
- From the data it is seen that relation between ΔSNR and Δbitrate is well represented by **0.5 dB = 10%** or **0.05 dB = 1%** It is therefore proposed to calculate either change in bitrate or change in PSNR.
- Should it be considered to present RD-plots as indicated in figure 3?


"Normal" RD-plot
PSNR (dB)
35
34
33
32
31
30
29
28
27
26
25
0
500
1000
1500
2000
2500
Bitrate
Plot2
Plot1


Log X-axes
PSNR (dB)
35
34
33
32
31
30
29
28
27
26
25
100
1000
10000
Bitrate
Plot2
Plot1



Here is a document about BD-PSNR which has been referenced by many Video Engineers. You can download it at http://wftp3.itu.int/av-arch/video-site/

The matlab code for computing BD-Bitrate and BD-PSNR is found in this link: http://www.mathworks.com/matlabcentral/fileexchange/27798-bjontegaardmetric/content/bjontegaard.m

```
function avg_diff = bjontegaard(R1,PSNR1,R2,PSNR2,mode)

%BJONTEGAARD    Bjontegaard metric calculation
%   Bjontegaard's metric allows to compute the average gain in PSNR or the
%   average per cent saving in bitrate between two rate-distortion
%   curves [1].
%   Differently from the avsnr software package or VCEG Excel [2] plugin this
%   tool enables Bjontegaard's metric computation also with more than 4 RD
%   points.
%
%   R1,PSNR1 - RD points for curve 1
%   R2,PSNR2 - RD points for curve 2
%   mode -
%       'dsnr' - average PSNR difference
%       'rate' - percentage of bitrate saving between data set 1 and
%               data set 2
%
%   avg_diff - the calculated Bjontegaard metric ('dsnr' or 'rate')
%
%   (c) 2010 Giuseppe Valenzise
%
%   References:
%
```

```
%   [1] G. Bjontegaard, Calculation of average PSNR differences between
%       RD-curves (VCEG-M33)
%   [2] S. Pateux, J. Jung, An excel add-in for computing Bjontegaard metric and
%       its evolution

% convert rates in logarithmic units
lR1 = log(R1);
lR2 = log(R2);

switch lower(mode)
   case 'dsnr'
       % PSNR method
       p1 = polyfit(lR1,PSNR1,3);
       p2 = polyfit(lR2,PSNR2,3);

       % integration interval
       min_int = min([lR1; lR2]);
       max_int = max([lR1; lR2]);

       % find integral
       p_int1 = polyint(p1);
       p_int2 = polyint(p2);

       int1 = polyval(p_int1, max_int) - polyval(p_int1, min_int);
       int2 = polyval(p_int2, max_int) - polyval(p_int2, min_int);

       % find avg diff
       avg_diff = (int2-int1)/(max_int-min_int);

   case 'rate'
       % rate method
       p1 = polyfit(PSNR1,lR1,3);
       p2 = polyfit(PSNR2,lR2,3);

       % integration interval
       min_int = min([PSNR1; PSNR2]);
       max_int = max([PSNR1; PSNR2]);

       % find integral
       p_int1 = polyint(p1);
       p_int2 = polyint(p2);

       int1 = polyval(p_int1, max_int) - polyval(p_int1, min_int);
       int2 = polyval(p_int2, max_int) - polyval(p_int2, min_int);

       % find avg diff
       avg_exp_diff = (int2-int1)/(max_int-min_int);
       avg_diff = (exp(avg_exp_diff)-1)*100;
end
```

BD-PSNR and BD-BITRATE are described graphically in Chapter 6 A. Tabatabai et al, “Compression performance analysis in HEVC”, in the book, V. Sze, M. Budagavi, and G.J. Sullivan, “High efficiency vodeo coding (HEVC): algorithms and architectures”, Springer, 2014.

Appendix D

The code for the proposed algorithm

The following section of HEVC code had been modified to implement the proposed algorithm.

```
Void
TEncSearch::estIntraPredQT( TComDataCU* pcCU,
                            TComYuv*    pcOrgYuv,
                            TComYuv*    pcPredYuv,
                            TComYuv*    pcResiYuv,
                            TComYuv*    pcRecoYuv,
                            UInt&       ruiDistC,
                            Bool        bLumaOnly )
{
  UInt    uiDepth        = pcCU->getDepth(0);
  UInt    uiNumPU        = pcCU->getNumPartInter();
  UInt    uiInitTrDepth  = pcCU->getPartitionSize(0) == SIZE_2Nx2N ? 0 : 1;
  UInt    uiWidth        = pcCU->getWidth (0) >> uiInitTrDepth;
  UInt    uiHeight       = pcCU->getHeight(0) >> uiInitTrDepth;
  UInt    uiQNumParts    = pcCU->getTotalNumPart() >> 2;
  UInt    uiWidthBit     = pcCU->getIntraSizeIdx(0);
  UInt    uiOverallDistY = 0;
  UInt    uiOverallDistC = 0;
  UInt    CandNum;
  Double  CandCostList[ FAST_UDI_MAX_RDMODE_NUM ];
  Double  BuffCandCostList[ FAST_UDI_MAX_RDMODE_NUM ];

 // Int totalThreads=0;
//omp_set_num_threads (2);
 // Fun_count++;
// printf("function count is %d\n\n",Fun_count);

//int iCPU = omp_get_num_procs();
//printf("number of PU is %d\n\n",iCPU);
  //===== set QP and clear Cbf =====
  if ( pcCU->getSlice()->getPPS()->getUseDQP() == true)
  {
    pcCU->setQPSubParts( pcCU->getQP(0), 0, uiDepth );
  }
  else
  {
    pcCU->setQPSubParts( pcCU->getSlice()->getSliceQp(), 0, uiDepth );
  }

  //===== loop over partitions =====
  UInt uiPartOffset = 0;

  for( UInt uiPU = 0; uiPU < uiNumPU; uiPU++, uiPartOffset += uiQNumParts )
  {
    //===== init pattern for luma prediction =====
    Bool bAboveAvail = false;
    Bool bLeftAvail  = false;
    pcCU->getPattern()->initPattern   ( pcCU, uiInitTrDepth, uiPartOffset );
    pcCU->getPattern()->initAdiPattern( pcCU, uiPartOffset, uiInitTrDepth, m_piYuvExt, m_iYuvExtStride,
m_iYuvExtHeight, bAboveAvail, bLeftAvail );

    //===== determine set of modes to be tested (using prediction signal only) =====
    Int numModesAvailable     = 35; //total number of Intra modes
    Pel* piOrg         = pcOrgYuv ->getLumaAddr( uiPU, uiWidth );
    Pel* piPred        = pcPredYuv->getLumaAddr( uiPU, uiWidth );
    UInt uiStride      = pcPredYuv->getStride();
    UInt uiRdModeList[FAST_UDI_MAX_RDMODE_NUM];
            UInt BuffuiRdModeList[FAST_UDI_MAX_RDMODE_NUM];

    Int numModesForFullRD = g_aucIntraModeNumFast[ uiWidthBit ];

    Bool doFastSearch = (numModesForFullRD != numModesAvailable);
            //printf("value of doFastSearch is %d\n\n",doFastSearch);
    if (doFastSearch)
    {
      assert(numModesForFullRD < numModesAvailable);

      for( Int i=0; i < numModesForFullRD; i++ )
      {
        CandCostList[ i ] = MAX_DOUBLE;
                          BuffCandCostList[i]= MAX_DOUBLE;
      }
      CandNum = 0;

  Int modeIdx=0;

  Fun_count++;
  printf("function count is %d\n\n",Fun_count);

#pragma omp parallel for shared (uiStride,uiWidth, uiHeight, bAboveAvail, bLeftAvail)
      for( modeIdx = 0; modeIdx < numModesAvailable; modeIdx++ )
      {
      UInt  uiMode = modeIdx;
              UInt temp;
              printf("mode is %d \n\n",uiMode);
            totalThreads = omp_get_num_threads();
            printf("Total number of threads are %d\n",totalThreads);
            //  printf("check point 1 is here\n\n");

        predIntraLumaAng( pcCU->getPattern(), uiMode, piPred, uiStride, uiWidth, uiHeight, bAboveAvail,
bLeftAvail );
             // }
     //   printf("check point 2 is here\n\n");
        // use hadamard transform here
        UInt uiSad = m_pcRdCost->calcHAD(g_bitDepthY, piOrg, uiStride, piPred, uiStride, uiWidth,
uiHeight );
   //       printf("check point 3 is here\n\n");
        UInt   iModeBits = xModeBitsIntra( pcCU, uiMode, uiPU, uiPartOffset, uiDepth, uiInitTrDepth );
            //            printf("check point 4 is here\n\n");
        Double cost      = (Double)uiSad + (Double)iModeBits * m_pcRdCost->getSqrtLambda();
  //       printf("check point 5 is here\n\n");
//#pragma omp critical
            //  {
                          BuffuiRdModeList[modeIdx]= uiMode;
                          BuffCandCostList[modeIdx]=cost;

              }
```

```
          for( modeIdx = 0; modeIdx < numModesAvailable; modeIdx++ )
          {
    CandNum = CandNum + xUpdateCandList( BuffuiRdModeList[modeIdx],
BuffCandCostList[modeIdx], numModesForFullRD, uiRdModeList, CandCostList );
   //}
          }

#if FAST_UDI_USE_MPM
   Int uiPreds[3] = {-1, -1, -1};
   Int iMode = -1;
   Int numCand = pcCU->getIntraDirLumaPredictor( uiPartOffset, uiPreds, &iMode );
   if( iMode >= 0 )
   {
    numCand = iMode;
   }


//# pragma omp parallel for reduction (+:numModesForFullRD)
   for( Int j=0; j < numCand; j++)

   {
    Bool mostProbableModeIncluded = false;
    Int mostProbableMode = uiPreds[j];

    for( Int i=0; i < numModesForFullRD; i++)
    {
     mostProbableModeIncluded |= (mostProbableMode == uiRdModeList[i]);
    }
    if (!mostProbableModeIncluded)
    {
     uiRdModeList[numModesForFullRD++] = mostProbableMode;
    }
   }

#endif // FAST_UDI_USE_MPM
  }
  else
  {
   for( Int i=0; i < numModesForFullRD; i++)
   {
    uiRdModeList[i] = i;
   }
  }

  //===== check modes (using r-d costs) =====
#if HHI_RQT_INTRA_SPEEDUP_MOD
  UInt   uiSecondBestMode  = MAX_UINT;
  Double dSecondBestPUCost = MAX_DOUBLE;
#endif

  UInt    uiBestPUMode  = 0;
  UInt    uiBestPUDistY = 0;
  UInt    uiBestPUDistC = 0;
  Double  dBestPUCost   = MAX_DOUBLE;

//# pragma omp parallel for
```

```
    for( UInt uiMode = 0; uiMode < numModesForFullRD; uiMode++ )
    {
      // set luma prediction mode
      UInt uiOrgMode = uiRdModeList[uiMode];

      pcCU->setLumaIntraDirSubParts ( uiOrgMode, uiPartOffset, uiDepth + uiInitTrDepth );

      // set context models
      if( m_bUseSBACRD )
      {
        m_pcRDGoOnSbacCoder->load( m_pppcRDSbacCoder[uiDepth][CI_CURR_BEST] );
      }

      // determine residual for partition
      UInt   uiPUDistY = 0;
      UInt   uiPUDistC = 0;
      Double dPUCost   = 0.0;
#if HHI_RQT_INTRA_SPEEDUP
      xRecurIntraCodingQT( pcCU, uiInitTrDepth, uiPartOffset, bLumaOnly, pcOrgYuv, pcPredYuv,
pcResiYuv, uiPUDistY, uiPUDistC, true, dPUCost );
#else
      xRecurIntraCodingQT( pcCU, uiInitTrDepth, uiPartOffset, bLumaOnly, pcOrgYuv, pcPredYuv,
pcResiYuv, uiPUDistY, uiPUDistC, dPUCost );
#endif

      // check r-d cost
      if( dPUCost < dBestPUCost )
      {
#if HHI_RQT_INTRA_SPEEDUP_MOD
        uiSecondBestMode  = uiBestPUMode;
        dSecondBestPUCost = dBestPUCost;
#endif
        uiBestPUMode  = uiOrgMode;
        uiBestPUDistY = uiPUDistY;
        uiBestPUDistC = uiPUDistC;
        dBestPUCost   = dPUCost;

        xSetIntraResultQT( pcCU, uiInitTrDepth, uiPartOffset, bLumaOnly, pcRecoYuv );

        UInt uiQPartNum = pcCU->getPic()->getNumPartInCU() >> ( ( pcCU->getDepth(0) + uiInitTrDepth )
<< 1 );
        ::memcpy( m_puhQTTempTrIdx,  pcCU->getTransformIdx()       + uiPartOffset, uiQPartNum * sizeof(
UChar ) );
        ::memcpy( m_puhQTTempCbf[0], pcCU->getCbf( TEXT_LUMA     ) + uiPartOffset, uiQPartNum *
sizeof( UChar ) );
        ::memcpy( m_puhQTTempCbf[1], pcCU->getCbf( TEXT_CHROMA_U ) + uiPartOffset, uiQPartNum
* sizeof( UChar ) );
        ::memcpy( m_puhQTTempCbf[2], pcCU->getCbf( TEXT_CHROMA_V ) + uiPartOffset, uiQPartNum
* sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[0], pcCU->getTransformSkip(TEXT_LUMA)     +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[1], pcCU->getTransformSkip(TEXT_CHROMA_U) +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[2], pcCU->getTransformSkip(TEXT_CHROMA_V) +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
      }
```

```
#if HHI_RQT_INTRA_SPEEDUP_MOD
      else if( dPUCost < dSecondBestPUCost )
      {
        uiSecondBestMode  = uiOrgMode;
        dSecondBestPUCost = dPUCost;
      }
#endif
    } // Mode loop

#if HHI_RQT_INTRA_SPEEDUP
#if HHI_RQT_INTRA_SPEEDUP_MOD
    for( UInt ui =0; ui < 2; ++ui )
#endif
    {
#if HHI_RQT_INTRA_SPEEDUP_MOD
      UInt uiOrgMode   = ui ? uiSecondBestMode  : uiBestPUMode;
      if( uiOrgMode == MAX_UINT )
      {
        break;
      }
#else
      UInt uiOrgMode = uiBestPUMode;
#endif

      pcCU->setLumaIntraDirSubParts ( uiOrgMode, uiPartOffset, uiDepth + uiInitTrDepth );

      // set context models
      if( m_bUseSBACRD )
      {
        m_pcRDGoOnSbacCoder->load( m_pppcRDSbacCoder[uiDepth][CI_CURR_BEST] );
      }

      // determine residual for partition
      UInt   uiPUDistY = 0;
      UInt   uiPUDistC = 0;
      Double dPUCost   = 0.0;
      xRecurIntraCodingQT( pcCU, uiInitTrDepth, uiPartOffset, bLumaOnly, pcOrgYuv, pcPredYuv,
pcResiYuv, uiPUDistY, uiPUDistC, false, dPUCost );

      // check r-d cost
      if( dPUCost < dBestPUCost )
      {
        uiBestPUMode  = uiOrgMode;
        uiBestPUDistY = uiPUDistY;
        uiBestPUDistC = uiPUDistC;
        dBestPUCost   = dPUCost;

        xSetIntraResultQT( pcCU, uiInitTrDepth, uiPartOffset, bLumaOnly, pcRecoYuv );

        UInt uiQPartNum = pcCU->getPic()->getNumPartInCU() >> ( ( pcCU->getDepth(0) + uiInitTrDepth )
<< 1 );
        ::memcpy( m_puhQTTempTrIdx,  pcCU->getTransformIdx()       + uiPartOffset, uiQPartNum * sizeof(
UChar ) );
        ::memcpy( m_puhQTTempCbf[0], pcCU->getCbf( TEXT_LUMA     ) + uiPartOffset, uiQPartNum *
sizeof( UChar ) );
```

```
        ::memcpy( m_puhQTTempCbf[1], pcCU->getCbf( TEXT_CHROMA_U ) + uiPartOffset, uiQPartNum
* sizeof( UChar ) );
        ::memcpy( m_puhQTTempCbf[2], pcCU->getCbf( TEXT_CHROMA_V ) + uiPartOffset, uiQPartNum
* sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[0], pcCU->getTransformSkip(TEXT_LUMA)     +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[1], pcCU->getTransformSkip(TEXT_CHROMA_U) +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
        ::memcpy( m_puhQTTempTransformSkipFlag[2], pcCU->getTransformSkip(TEXT_CHROMA_V) +
uiPartOffset, uiQPartNum * sizeof( UChar ) );
      }
    } // Mode loop
#endif

    //--- update overall distortion ---
    uiOverallDistY += uiBestPUDistY;
    uiOverallDistC += uiBestPUDistC;

    //--- update transform index and cbf ---
    UInt uiQPartNum = pcCU->getPic()->getNumPartInCU() >> ( ( pcCU->getDepth(0) + uiInitTrDepth ) <<
1 );
    ::memcpy( pcCU->getTransformIdx()       + uiPartOffset, m_puhQTTempTrIdx,  uiQPartNum * sizeof(
UChar ) );
    ::memcpy( pcCU->getCbf( TEXT_LUMA     ) + uiPartOffset, m_puhQTTempCbf[0], uiQPartNum *
sizeof( UChar ) );
    ::memcpy( pcCU->getCbf( TEXT_CHROMA_U ) + uiPartOffset, m_puhQTTempCbf[1], uiQPartNum *
sizeof( UChar ) );
    ::memcpy( pcCU->getCbf( TEXT_CHROMA_V ) + uiPartOffset, m_puhQTTempCbf[2], uiQPartNum *
sizeof( UChar ) );
    ::memcpy( pcCU->getTransformSkip(TEXT_LUMA)     + uiPartOffset,
m_puhQTTempTransformSkipFlag[0], uiQPartNum * sizeof( UChar ) );
    ::memcpy( pcCU->getTransformSkip(TEXT_CHROMA_U) + uiPartOffset,
m_puhQTTempTransformSkipFlag[1], uiQPartNum * sizeof( UChar ) );
    ::memcpy( pcCU->getTransformSkip(TEXT_CHROMA_V) + uiPartOffset,
m_puhQTTempTransformSkipFlag[2], uiQPartNum * sizeof( UChar ) );
    //--- set reconstruction for next intra prediction blocks ---
    if( uiPU != uiNumPU - 1 )
    {
      Bool bSkipChroma  = false;
      Bool bChromaSame  = false;
      UInt uiLog2TrSize = g_aucConvertToBit[ pcCU->getSlice()->getSPS()->getMaxCUWidth() >> ( pcCU-
>getDepth(0) + uiInitTrDepth ) ] + 2;
      if( !bLumaOnly && uiLog2TrSize == 2 )
      {
        assert( uiInitTrDepth  > 0 );
        bSkipChroma  = ( uiPU != 0 );
        bChromaSame  = true;
      }

      UInt    uiCompWidth   = pcCU->getWidth ( 0 ) >> uiInitTrDepth;
      UInt    uiCompHeight  = pcCU->getHeight( 0 ) >> uiInitTrDepth;
      UInt    uiZOrder      = pcCU->getZorderIdxInCU() + uiPartOffset;
      Pel*    piDes         = pcCU->getPic()->getPicYuvRec()->getLumaAddr( pcCU->getAddr(), uiZOrder );
      UInt    uiDesStride   = pcCU->getPic()->getPicYuvRec()->getStride();
      Pel*    piSrc         = pcRecoYuv->getLumaAddr( uiPartOffset );
      UInt    uiSrcStride   = pcRecoYuv->getStride();
      for( UInt uiY = 0; uiY < uiCompHeight; uiY++, piSrc += uiSrcStride, piDes += uiDesStride )
      {
        for( UInt uiX = 0; uiX < uiCompWidth; uiX++ )
        {
          piDes[ uiX ] = piSrc[ uiX ];
        }
      }
      if( !bLumaOnly && !bSkipChroma )
      {
        if( !bChromaSame )
        {
          uiCompWidth   >>= 1;
          uiCompHeight  >>= 1;
        }
        piDes         = pcCU->getPic()->getPicYuvRec()->getCbAddr( pcCU->getAddr(), uiZOrder );
        uiDesStride   = pcCU->getPic()->getPicYuvRec()->getCStride();
        piSrc         = pcRecoYuv->getCbAddr( uiPartOffset );
        uiSrcStride   = pcRecoYuv->getCStride();
        for( UInt uiY = 0; uiY < uiCompHeight; uiY++, piSrc += uiSrcStride, piDes += uiDesStride )
        {
          for( UInt uiX = 0; uiX < uiCompWidth; uiX++ )
          {
            piDes[ uiX ] = piSrc[ uiX ];
          }
        }
        piDes         = pcCU->getPic()->getPicYuvRec()->getCrAddr( pcCU->getAddr(), uiZOrder );
        piSrc         = pcRecoYuv->getCrAddr( uiPartOffset );
        for( UInt uiY = 0; uiY < uiCompHeight; uiY++, piSrc += uiSrcStride, piDes += uiDesStride )
        {
          for( UInt uiX = 0; uiX < uiCompWidth; uiX++ )
          {
            piDes[ uiX ] = piSrc[ uiX ];
          }
        }
      }
    }

    //=== update PU data ====
    pcCU->setLumaIntraDirSubParts     ( uiBestPUMode, uiPartOffset, uiDepth + uiInitTrDepth );
    pcCU->copyToPic                   ( uiDepth, uiPU, uiInitTrDepth );
//          Fun_count++;
  } // PU loop

//  printf("total outer functions are %d\n\n",Fun_count);
  if( uiNumPU > 1 )
  { // set Cbf for all blocks
    UInt uiCombCbfY = 0;
    UInt uiCombCbfU = 0;
    UInt uiCombCbfV = 0;
    UInt uiPartIdx  = 0;
    for( UInt uiPart = 0; uiPart < 4; uiPart++, uiPartIdx += uiQNumParts )
    {
      uiCombCbfY |= pcCU->getCbf( uiPartIdx, TEXT_LUMA,     1 );
      uiCombCbfU |= pcCU->getCbf( uiPartIdx, TEXT_CHROMA_U, 1 );
      uiCombCbfV |= pcCU->getCbf( uiPartIdx, TEXT_CHROMA_V, 1 );
    }
    for( UInt uiOffs = 0; uiOffs < 4 * uiQNumParts; uiOffs++ )
    {
      pcCU->getCbf( TEXT_LUMA     )[ uiOffs ] |= uiCombCbfY;
      pcCU->getCbf( TEXT_CHROMA_U )[ uiOffs ] |= uiCombCbfU;
      pcCU->getCbf( TEXT_CHROMA_V )[ uiOffs ] |= uiCombCbfV;
    }
  }

  //===== reset context models =====
  if(m_bUseSBACRD)
  {
    m_pcRDGoOnSbacCoder->load(m_pppcRDSbacCoder[uiDepth][CI_CURR_BEST]);
  }

  //===== set distortion (rate and r-d costs are determined later) =====
  ruiDistC                   = uiOverallDistC;
  pcCU->getTotalDistortion() = uiOverallDistY + uiOverallDistC;
}
```

Appendix E

Acronyms

Acronyms

API: Application Programming Interface

AVC: Advanced Video Coding

AVS: Audio Video Coding Standard

BD: Bjontegaard Delta

CABAC: Context Adaptive Binary Arithmetic Coding

CB: Coding Block

CE: Consumer Electronics

CPU: Central Processing Unit

CSVT: Circuits and Systems for Video Technology

CTB: Coding Tree Block

CTU: Coding Tree Unit

CU: Coding Unit

CUDA: Compute Unified Device Architecture

DCC: Data Compression Conference

DCT: Discrete Cosine Transform

DST: Discrete Sine Transform

FDIS: Final Draft International Standard

FORTRAN: Formula Translating System

HD: High Definition

HDMI: High Definition Multimedia Interface

HEVC: High Efficiency Video Coding

ICASSP: International Conference on Acoustics, Speech and Signal Processing

ICIP: International Conference on Image Processing

IEC: International Electrotechnical Commission

ISO: International Organization for Standardization

ITU-T: International Telecommunication Union – Telecommunication Standardization Sector

JCT-VC: Joint Collaborative Team on Video Coding

MC	Motion Compensation
MCP	Motion Compensated Predication
MPEG	Moving Picture Experts Group
MPI	Message Passing Interface
MPM	Most Probable Mode
MVC	Multiview video coding
NGVC	Next Generation Video Coding
OPENMP	Open Multiprocessing
PB	Prediction Block
PCM	Pulse Code Modulation
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
RD	Rate Distortion
SAO	Sample Adaptive Offset
SATD	Sum Of Absolute Transformed Difference
SIMD	Single Instruction Multiple Data
SVC	Scalable Video coding
TB	Transform Block
VCEG	Video Coding Experts Group
VCIP	Visual Communication and Image Processing

References

[1] G.J. Sullivan et al, "Overview of the high efficiency video coding (HEVC) standard", IEEE Trans. CSVT, vol. 22, pp.1649-1668, Dec.2012.

[2] C.C.Chi et al, "Parallel scalability and efficiency of HEVC parallelization approaches", IEEE Trans. CSVT, vol. 22, pp.1827-1838, Dec.2012.

[3] J. Lainema et al, "Intra coding of the HEVC standard", IEEE Trans. CSVT,vol.22, pp.1792-1801, Dec.2012.

[4] F. Bossen et al, "HEVC complexity and implementation analysis", IEEE Trans. CSVT, vol. 22, pp.1685-1696, Dec.2012.

[5] P. Hanhart et al, "Subjective quality evaluation of the upcoming HEVC video compression standard" SPIE Applications of digital image processing XXXV, vol.8499, pp.8499-30, Aug.2012.

[6] J.-R Ohm, et al, "Comparison of the coding efficiency of video coding standards- including high efficiency video coding (HEVC)" , IEEE Trans. CSVT , vol.22, pp.1669-1684, Dec.2012.

[7] X. Zhang, S. Liu and S. Lei, "Intra mode coding in HEVC standard", Visual Communications and Image Processing, VCIP 2012, pp. 1-6, San Diego, CA, Nov.2012.

[8] Y.Duan, "An optimized real time multi-thread HEVC decoder", Visual Communications and Image Processing, VCIP 2012, pp.1, San Diego, CA, Nov.2012.

[9] G. Correa et al, "Performance and computational complexity assessment of high efficiency video encoders", IEEE Trans. CSVT, vol.22, pp.1899-1909, Dec.2012.

[10] A.Saxena, F. Fernandes and Y. Reznik, "Fast transforms for intra-prediction-based image and video coding," in Proc. IEEE Data Compression Conference (DCC'13), pp.13-22, Snowbird, UT, Mar.2013.

[11]HEVC open source software (encoder/decoder) https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-9.1-dev/

[12] Introduction to parallel computing https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[13] Information about quad tree structure of HEVC http://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/

[14] Guide into OpenMP: Easy multithreading programming for C++ http://bisqwit.iki.fi/story/howto/openmp/

[15] Website for downloading test sequence for research purposes http://media.xiph.org/video/derf/

[16] Information on developments in HEVC NGVC- Next generation video coding http://bisqwit.iki.fi/story/howto/openmp/

[17] F. Bossen, D. Flynn and K. Suhring (July 2012), "HEVC reference software manual" online available: http://phenix.int-evry.fr/jct/doc_end_user/documents/6_Torino/wg11/JCTVC-F634-v2.zip

[18] JCT-VC documents are publicly available at http://ftp3.itu.ch/av-arch/jctvc-site and http://phenix.it-sudparis.eu/jct/

[19] Information about basics of video compression
http://desktopvideo.about.com/od/videoonyourwebsite/qt/compress.htm

[20] Video research project
http://www1.curriculum.edu.au/videoresearch/compression.htm

[21] Video compression basics
http://wolfcrow.com/blog/what-is-video-compression/

[22] T.L Silva et al, ”HEVC intra coding acceleration based on tree inter-level mode correlation”, SPA 2013 Sep.2013, Poznan, Poland

[23] A. Saxena and F. Fernanades, “Mode dependent DCT/DST for intra prediction in block based image/video coding”, IEEE ICIP, pp. 1685-1688, Sept. 2011.

[24] H. Zhang and Z. Ma, ”Fast intra prediction for high efficiency video coding ”, Pacific Rim Conf. on Multimedia, PCM2012, Singapore, Dec.2012.

[25] M. Zhang, C. Zhao and J. Xu, ”An adaptive fast intra mode decision in HEVC ”, IEEE ICIP 2012, pp.221-224, Orlando, FL, Sept.- Oct.2012.

[25] K. Chen et al, ”Efficient SIMD optimization of HEVC encoder over X86 processors”, APSIPA, pp. 1732-1745, Los Angeles, CA, Dec. 2012.

[26] Y. Kim et al, “A fast intra-prediction method in HEVC using rate-distortion estimation based on Hadamard transform”, ETRI Journal, vol.35, #2, pp.270-280, Apr.2013.

[27] T. Wiegand et al., ”Overview of the H.264/AVC Video Coding Standard”, IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560-576, Jul.2003.

[28] M. Khan et al, “An adaptive complexity reduction scheme with fast prediction unit decision for HEVC Intra encoding”, IEEE ICIP, pp. 1578-1582, Sept. 2013.

[29] ITU-T, “Video Codec for Audiovisual Services at px64 Kbit/s”, ITU-T Recommendation H.261, Version 1: Nov. 1990; Version 2: Mar. 1993.

[30] ITU-T, “Video coding for low bit rate communication”, ITU-T Recommendation H.263, Nov. 1995 (and subsequent editions)

[31] ISO/IEC JTC 1, “Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s – Part 2: Video”, ISO/IEC 11172 (MPEG-1),1993.

[32] ISO/IEC JTC 1, “Coding of audio-visual objects – Part 2: Visual”, ISO/IEC 14496-2 (MPEG-4 Visual version 1), April 1999 (and subsequent editions).

[33] ITU-T and ISO/IEC JTC 1, “Generic coding of moving pictures and associated audio information- Part 2: Video”, ITU-T Recommendation H.262 and ISO/IEC 13818-2 (MPEG @ Video), Nov. 1994.

[34] ITU-T and ISO/IEC JTC 1, “Advanced Video Coding for generic audio-visual services”, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (AVC), May 2003 (and subsequent editions).

[35] K.R. Rao, D.N. Kim and J.J. Hwang, “Video coding standards: AVS China, H.264/MPEG-4 Part 10, HEVC, VP6, DIRAC and VC-1”, Springer, 2014.

[36] Generic quadtree based approach for block partitioning

http://www.hhi.fraunhofer.de/fields-of-competence/image-processing/research-groups/image-video-coding/hevc-high-efficiency-video-coding/generic-quadtree-based-approach-for-block-partitioning.html

[37] Information about context based binary arithmetic coding
http://www.hhi.fraunhofer.de/de/kompetenzfelder/image-processing/research-groups/image-video-coding/statistical-modeling-coding/context-based-adaptive-binary-arithmetic-coding-cabac.html

[38] Basics of video
http://lea.hamradio.si/~s51kq/V-BAS.HTM

[39] Circuit implementation of High-Efficiency video coding tools
http://dspace.mit.edu/bitstream/handle/1721.1/75691/820028934.pdf?sequence=1

[40] A. Norkin et al., "HEVC Deblocking Filter", IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1746-1754, Dec.2012.
A. Norkin et al., "Corrections to HEVC Deblocking Filter", IEEE Trans. Circuits Syst. Video Technol., vol. 23, no. 12, pp. 2141, Dec.2013.

[41] C. Fogg, "Suggested figures for the HEVC specification", ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC- J0292r1, July 2012.

[42] Video compression basics
http://www.nt.tuwien.ac.at/fileadmin/users/psvoboda/video-basics.pdf

[43] G.J. Sullivan et al, "Standardized Extension of High Efficiency Video Coding (HEVC)", IEEE Journal of Selected topics in Signal processing, vol. 7, no. 6, pp.1001-1016, Dec.2013.

[44] Special issue on emerging research and standards in next generation video coding, IEEE Trans. CSVT, vol.22, pp.1646-1909, Dec. 2012.

[45] G.J. Sullivan, "HEVC; The next generation in video compression". Keynote speech, Visual Communications and Image Processing, VCIP 2012, San Diego, CA, 27-30 Nov.2012.

[46] M.T Pourazad et al,"HEVC: The new gold standard for video compression", IEEE CE magazine vol.1, issue 3, pp. 36-46, July 2012.

[47] J-R. Ohm, T. Wiegand and G.J. Sullivan, "Video coding progress; The high efficiency video coding (HEVC) standard and its future extensions", IEEE ICASSP, Tutorial, Vancouver, Canada, 2013.

[48] Several papers on HEVC in the poster session IVMSP-P3: Video coding II, IEEE ICASSP 2013, Vancouver, Canada, June 2013.

[49] HEVC tutorial by I.E.G. Richardson: http://www.vcodex.com/h265.html

[50] JCT-VC documents can be found in JCT-VC document management system http://phenix.int-evry.fr/ict

[51] VCEG & JCT documents available from
http://wftp3.itu.int/av-arch in the video-site and jvt-site folders

[52] HEVC encoded bit streams
ftp://ftp.kw.bbc.co.uk/hevc/hm-11.0-anchors/bitstreams/

[53] J. Lainema and K. Ugur, "Angular Intra Prediction in High Efficiency Video Coding", IEEE 13th International Workshop on Multimedia Signal Processing (MMSP) , pp. 1-5, Oct. 2011.

[54] T. Silva, L.Agostini and L.Cruz, "Fast HEVC Intra Prediction Mode Decision Based on Edge Direction Information", 20th European Signal Processing Conference (EUSIPCO 2012), pp. 1214-1218, Aug. 2012.

[55] High Efficiency Video Coding Test Model Available in:
https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-4.0/,2011

[56] J. Sole et al, "Transform Coefficient Coding in HEVC", IEEE Trans. CSVT, vol. 22, no.12 pp.1765-1777, Dec.2012.

[57] Introductions to parallel programming
https://computing.llnl.gov/tutorials/parallel_comp/

[58] Introduction to parallel programming and MapReduce
https://courses.cs.washington.edu/courses/cse490h/07wi/readings/IntroductionToParallelProgrammingAndMapReduce.pdf

[59] Information about shared memory
http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/what-is-shm.html

[60] Definition of thread
http://www.techopedia.com/definition/27857/thread

[61] Information about message passing interface
http://www.hpcvl.org/faqs/programming/mpi-message-passing-interface

[62] Information about Data parallel programming model
http://insidehpc.com/2006/03/what-is-data-parallel-programming/

[63] Information about Load imbalance
https://software.intel.com/en-us/articles/load-balancing-between-threads

[64] Description about race around condition and deadlocks
http://support.microsoft.com/kb/317723

[65] Information and example for deadlock
http://www.roseindia.net/java/thread/deadlocks.shtml

[66] Open MP components
http://www.capsl.udel.edu/courses/cpeg421/2012/slides/openmp_tutorial_04_06_2012.pdf

[67] OpenMP Run time library routines
https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-D3FC1F0B-DD99-4176-B9B5-56EEE72E81A7.htm

[68] OpenMP environment variable
http://msdn.microsoft.com/en-us/library/6sfk977f.aspx

[69] OpenMP tutorial
https://computing.llnl.gov/tutorials/openMP/#Abstract

[70] Information about OpenMP clauses
https://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_par_dirs.htm

[71] Information regarding OpenMP environment variables
http://msdn.microsoft.com/en-us/library/tt15eb9t.aspx

[72] Official website for information on OpenMP
http://openmp.org/wp/

[72] Official website for information on OpenMP
http://openmp.org/wp/

[73] D.P. Kumar, “Intra Frame Luma Prediction using Neural Networks in HEVC”, website: http://www-ee.uta.edu/Dip/Courses/EE5359/Dilip_Thesis_Document.pdf, Thesis, University of Texas at Arlington, UMI Dissertation Publishing, May 2013

[74] M.Mrak, A. Gabriellini , D.Flynn, “Parallel processing for combined intra prediction in high efficiency video coding”, IEEE ICIP, pp.3489 -3492, Sept. 2011.

[75] J.Rehman, Y. Zhang,”Fast Intra Prediction mode decision using parallel processing”, Proceedings of the fourth international conference on machine learning and cybernetics, pp.5094-5098, Aug. 2005

[76]Overhead in openMP parameters
http://www.embedded.com/design/mcus-processors-and-socs/4007155/Using-OpenMP-for-programming-parallel-threads-in-multicore-applications-Part-2

[77] X. Li et al, “Rate-Complexity-Distortion evaluation for hybrid video coding”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 21, pp. 957- 970, July 2011.

[78] V.Sze, M.Budagavi, G.J. Sullivan, “High Efficiency Video Coding (HEVC)- Algorithm and Architectures”, Springer, 2014.

[79] Spiros N Agathos, P. Hadjidoukas and V. Dimakopoulos,“Task based execution of Nested OpenMP loops”, Springer, 2012.

[80] R.Chandra et al, “Parallel programming in OpenMP”, Academic Press, 2001.

[81] R.Eigenmann, B. Supinski, “OpenMP in a New Era of Parallelism”, 4th International Workshop, IWOMP 2008 West Lafayette, IN, USA proceedings, Springer 2008.

[82] Getting Started with CUDA
http://www.nvidia.com/content/cudazone/download/Getting_Started_w_CUDA_Training_NVISION08.pdf

[83] Multithreaded programming guide
http://docs.oracle.com/cd/E19253-01/816-5137/ggedn/index.html

[84] Information about Pthread
http://pubs.opengroup.org/onlinepubs/007908775/xsh/pthread.h.html

[85] H. Jain,” Fast intra mode decision in high efficiency video coding”, M.S. Thesis, EE Dept., UTA, Aug 2014. Access from www.uta.edu/faculty/krrrao/dip

[86] S. Gangavati, ”Complexity reduction of H.264 using parallel programming”, M.S. Thesis,EE Dept.,UTA, Dec 2012. Access from www.uta.edu/faculty/krrrao/dip

[87] T. Saxena, “Reducing the encoding time for H.264 baseline profile using parallel programming techniques”, M.S. Thesis, EE Dept., UTA, June 2012. Access from www.uta.edu/faculty/krrrao/dip

Biographical Information

Pratik Devendrakumar Mehta was born in Nes, Gujarat, India in 1987. After completing his schooling at Saraswati Vidyalaya, Bhayander in 2003, he went on to obtain his Diploma of Engineering in Industrial Electronics, Bhausaheb Vartak Polytechnic, India from 2003-2007. After which he completed his bachelors in Electronics Engineering from 2007-2010 and then went on to work at Infosys Technologies Limited as System Engineer till 2012.

He joined University of Texas at Arlington, USA to pursue his Masters in Electrical Engineering in Fall 2012. He also worked as research assistant in Multimedia Processing Lab. He work for 2 semesters as Coop at Blackberry as Field test specialist from Jan 2014 to Aug 2014.