APPLICATION OF OPENMP IN COMPLEXITY REDUCTION OF INTER FRAME CODING IN HEVC

by

KARTHIK SURESH

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 my Professor Dr. K.R.Rao wholeheartedly for being my supervisor and for giving me the inspiration and support needed throughout the timeline of my thesis. I would like extend my heartfelt thanks to Dr. Howard Russel and Dr. Kambiz Alavi for serving on my committee.

I would also like to thank Bhavana Prabhakar, Sudeep Gangavati and all my MPL lab mates for guiding me and providing me with valuable inputs throughout my research.

Last but not least, I would like to thank my father, Suresh.A.N, my mother Krishnaveni.M.V, my grandmother Jayamma.N and all my friends for supporting and motivating me all along.

November 24, 2014

Abstract

APPLICATION OF OPENMP IN COMPLEXITY REDUCTION OF INTER FRAME CODING IN HEVC

Karthik Suresh

University of Texas at Arlington, 2014

Supervising Professor: K.R.Rao

The International Telecommunication Union (ITU-T), Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations working in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) came up with the latest video project known as the High Efficiency Video Coding (HEVC) standard. This standard, also known as the H.265 standard, is based on the same architecture as that of the more widely implemented H.264/AVC (Advance Video Coding). The HEVC has incorporated many improvements like increased bitrate reduction, increased coding efficiency [7] and compression efficiency, but, at the cost of increased complexity. The motion estimation (ME) process in the encoder is the most time-consuming part in both H.264 and the HEVC.

The purpose of this thesis is to make use of parallel programming to achieve faster encoding times with minimal losses. There are certain conditional statements in the HM 13.0 code, which run in a sequential pattern for a long time. Those statements which are having less or no dependencies are chosen as they are ideal for parallel programming. The xTZSearch and TZ8PointDiamondSearch patterns in HM13.0 consist of several conditional statements without dependencies. Running those statements in parallel allows the program to check for different conditions simultaneously and jump directly to the next step rather than checking them one-by-one. OpenMP is the API used to achieve parallel computing by making use of the multicore processor in the computer used to run the test sequences. As expected, there was a reduction of 10% - 25% in encoding time when compared to the original HM13.0. The proposed method was evaluated using different metrics like encoding time, percentage reduction in encoding time, BD-PSNR (Bjontegaard Delta Peak Signal to Noise Ratio), BD-bitrate (Bjontegaard Delta bitrate) and RD (rate distortion). Experimental results based on several video sequences suggest the negligible change in BD-PSNR and BD-bitrate with the implementation of the proposed method. It can be concluded that, with the help of parallel programming, the encoding time of the HEVC encoder can be reduced by a significant amount.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

1.1 Video Compression Basics

A video is a sequence of images put in a proper order. In turn, an image consists of pixels or picture elements which are known as the basic elements defining the picture. The number of pixels along the horizontal and vertical axes gives the respective width and height of the image. Each pixel is also characterized by its color and brightness.

The sequence of images are put together in an order and displayed in quick succession to create the effect of movement of images, which is known as a video. The individual images are also known as frames, and a collection of sequential frames constitute a video. The rate at which each frame is displayed is known as frame rate, which is another important characteristic of a given video.

Video compression is mostly about transmitting information by making use of the redundancies existing between the images. The redundancies that are exploited are temporal and spatial redundancies. In a sequence of images used for a video, there is very little change between each of the images. Temporal redundancy exploits this characteristic for compression purposes. When it comes to a single frame or image, the neighboring pixels for a given pixel will usually have same or similar information. Spatial redundancy exploits this property for compression.

There are three types of frames: intra frame or I – frame, inter frame or P – frame (predicted frame) and the bi-directionally interpolated frame or B – frame. If a frame is compressed using its own pixels for reference, then it is an I – frame. If an I – frame is used as the reference frame for a sequence of successive frames; those frames are called P – frames. If a frame is compressed by considering both I – frame (previous) and P – frames (future) for reference, then it is called a B – frame.

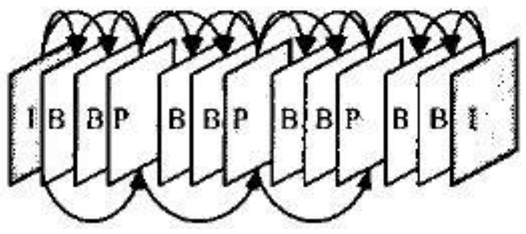


Figure 1-1: I, P and B – frames in a sequence of images [58]

1.2 Necessity for video compression

Usually, videos are large sequences of images that take up a huge amount of memory to store. The bandwidth required to transmit the video is also huge. Hence, there is a need to compress these videos before being transmitted. By developing better compression techniques, the memory and bandwidth requirements needed for video transmission and storage can be reduced. This makes transmission of data in applications like online video streaming, video telephony, etc. easier and economical.

There are several compression schemes that were developed in order to address the aforementioned problem. HEVC [1] [12] [17] [18] is a new video compression scheme that is projected to be widely used around the world soon. Compared to its predecessor H.264 [24] [27], HEVC has about 50% bitrate savings and more paralleling options while maintaining the same visual quality [5].

1.2 Thesis Outline

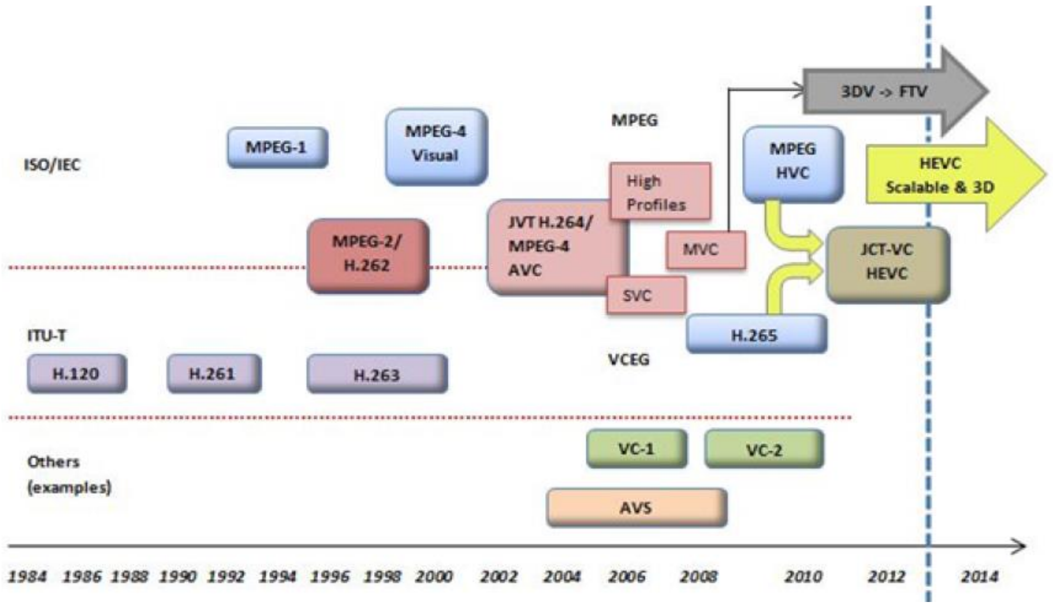


Figure 1-2: Evolution of video compression standards [1]

Chapter 2 details the HEVC compression scheme and gives an overview of the HEVC encoder. Chapter 3 describes the existing inter frame prediction process and a proposal of faster method to do this using paralleling of code structure. Chapter 4 shows the results and conclusions comparing both the existing and the proposed faster scheme. Chapter 5 contains ideas which can be used as future work in improving the performance of the encoder.

Chapter 2

High Efficiency Video Coding

2.1 HEVC design and features

HEVC [1] [12] [17] [18] is the latest video compression standard developed by the Joint Collaborative Team on Video Coding (JCT –VC) during early 2013. It has 3 profiles – main (for 8-bit data), main10 (for 10-bit data) and 4:2:2, 4:4:4 formats (Figure 2-5), bit depths higher than 10 – bit, scalability, and 3D video are being finalized in 2014.

Ultra HD, stereo and multi vision encoding, 3-D video and scalable video coding, which are few of the trending topics in the multimedia processing industry can make use of HEVC and benefit extensively in areas like parallel processing [2][6] using latest architectures, bit-rate savings, coding efficiency and many more. Concepts like wave front parallel processing, tiles and slices are introduced in the new standard. Flexible prediction modes, transform block sizes, improved interpolation and deblocking filters, and better partitioning options make HEVC standout from its predecessor, the H.264 [27]. A combined result of optimizing many processes in the standard led to all these improvements in HEVC. But, these advantages come at the price of higher encoder complexity. Basic design of the HEVC [1] standard remained the same as that of the H.264/AVC with the block based hybrid coding approach being a significant factor in temporal and spatial dependencies. The basic description of the HEVC encoder is shown in Figure 2.1.

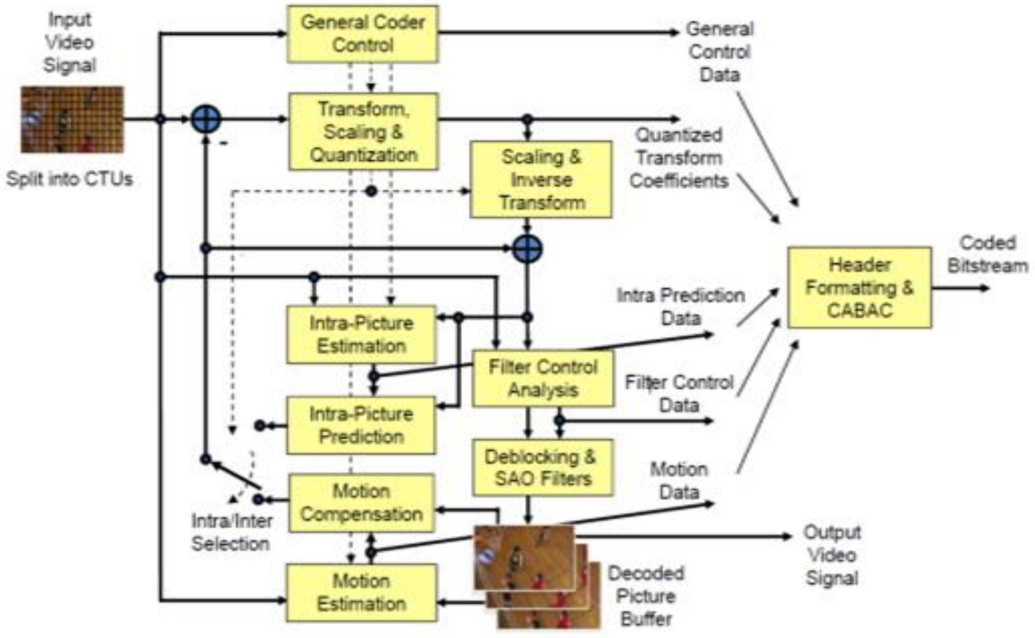


Figure 2-1: HEVC Encoder block diagram [1]

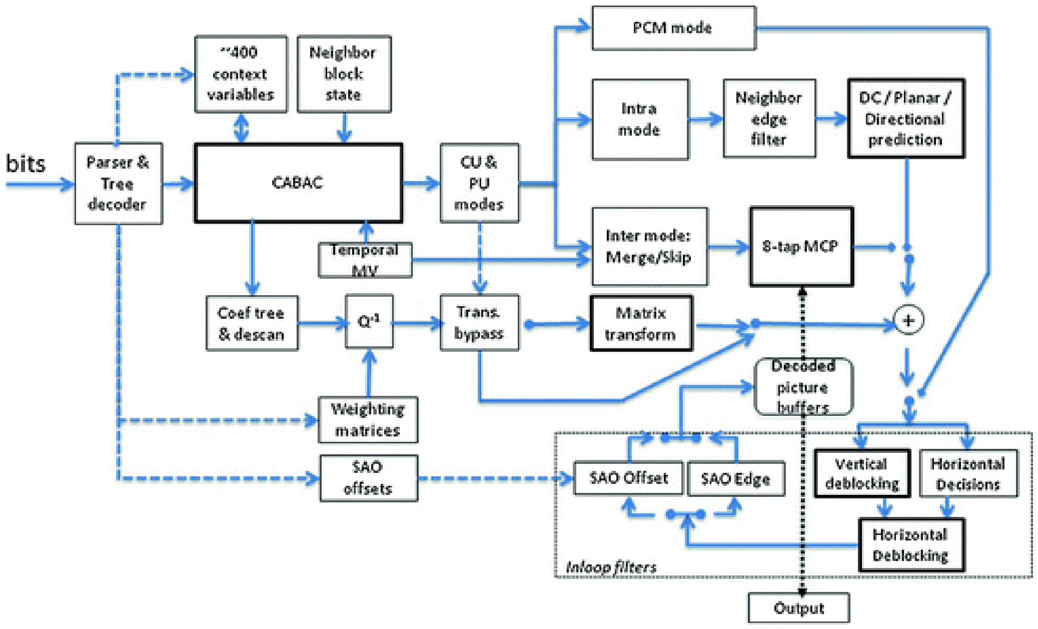


Figure 2-2: HEVC Decoder block diagram [1]

2.1.1 Description of Video Coding in HEVC

A novel hybrid approach (inter/ intra picture prediction and 2-D transform coding) is used in HEVC and has been used in all compression standards since H.261. Each picture is split into block shaped regions and the exact block partitioning will be conveyed to the decoder. The first picture of a video sequence will be coded using only 'intra picture prediction' mode which is a spatial prediction within the frame and the remaining pictures are coded using 'inter picture prediction' mode which is a temporal prediction between the frames.

The encoder duplicates the decoder processing loop such that it generates an identical prediction of a decoder. This is done by inverse scaling and inverse transforming of the encoded data to produce the decoder approximation of the residual signal. This residual signal is then added to the prediction signal and the result of this addition is fed to one or two loop filters which smoothen out the artifacts generally induced by the block-wise processing and quantization step. The final picture representation which is the duplicate of the possible output in the decoder is stored in a 'decoded picture buffer' and is used for prediction of subsequent pictures.

The signal of intra/inter prediction which is the difference between original and predicted block 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 predicted signal to produce decoded picture which is stored in buffer for further predictions. As shown in Figure 2-2 in the block diagram of HEVC [1] decoder, the residual signal is added to the prediction, and the result is fed to the deblocking filter to reduce the artifacts and finally stored in decoded picture buffer which can be used for further decoding of remaining pictures.

Unlike H.264 [27] which contains 16 by 16 size macro blocks, HEVC employs quad tree structure which contains coding tree unit (CTU), size of which is selected by the encoder and can be larger than traditional macro block. HEVC has been designed to address essentially all existing applications of H.264/MPEG-4 AVC and to particularly focus on two key issues: increased video resolution and increased use of parallel processing architectures.

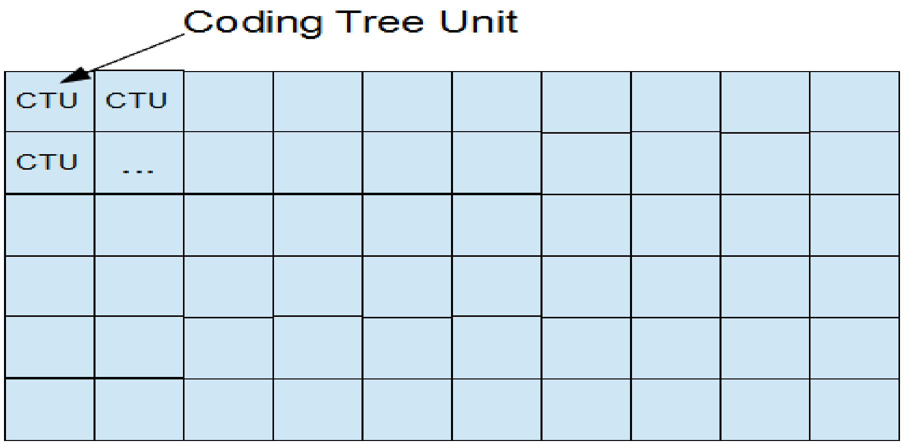


Figure 2-3: Division of an image into CTU [14]

Figure 2-3 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. 64x64, 32x32, or 16 by 16 as shown in Figure 2-4.

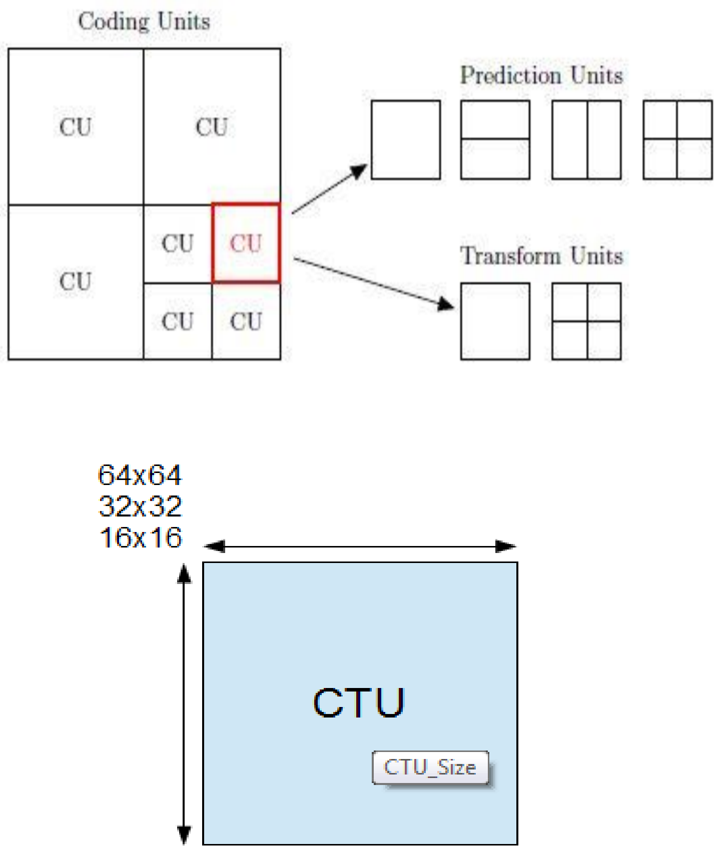


Figure 2-4: Different sizes of CTU [14]

Each coding unit basically consists of luma and chroma prediction blocks and each block is called coding tree block (CTB) having the same size as CTU. But CTBs are too big to decide the type of prediction method to be used. So CTBs are further divided into coding blocks (CB) which are the decision points where decision is taken whether to perform inter-picture or intra-picture prediction. 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.

The HEVC code uses YC_bC_r color space with a 4:2:0 color format with 8 bps (bits per color sample). Y is symbol for luma component, C_b is symbol for the blue chroma component and C_r is symbol for the red chroma component as shown in figure 2-5.

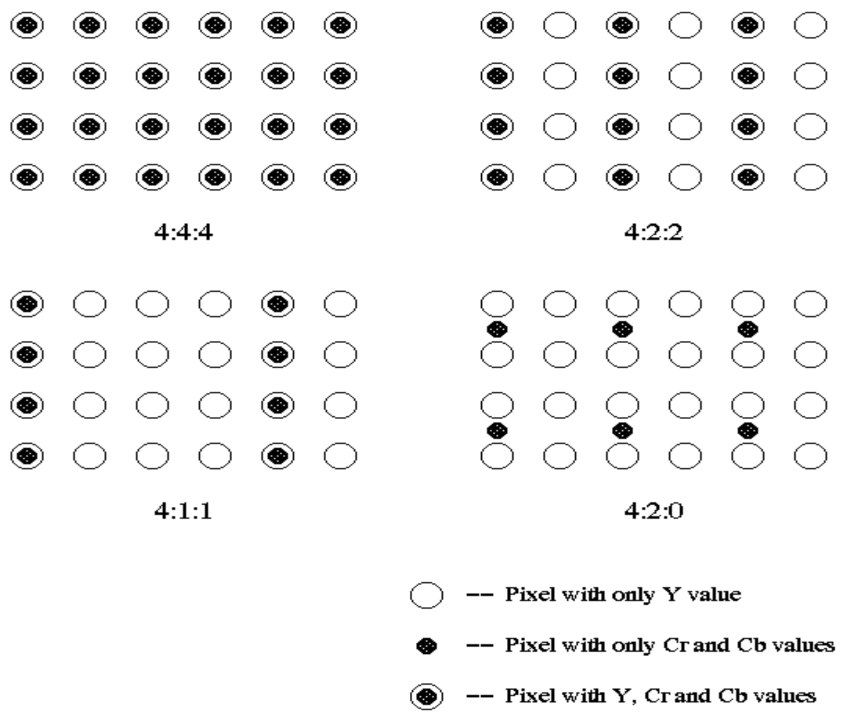

4:4:4
4:2:2
4:1:1
4:2:0
-- Pixel with only Y value
-- Pixel with only Cr and Cb values
-- Pixel with Y, Cr and Cb values

Figure 2-5: Formats for YUV components [57]

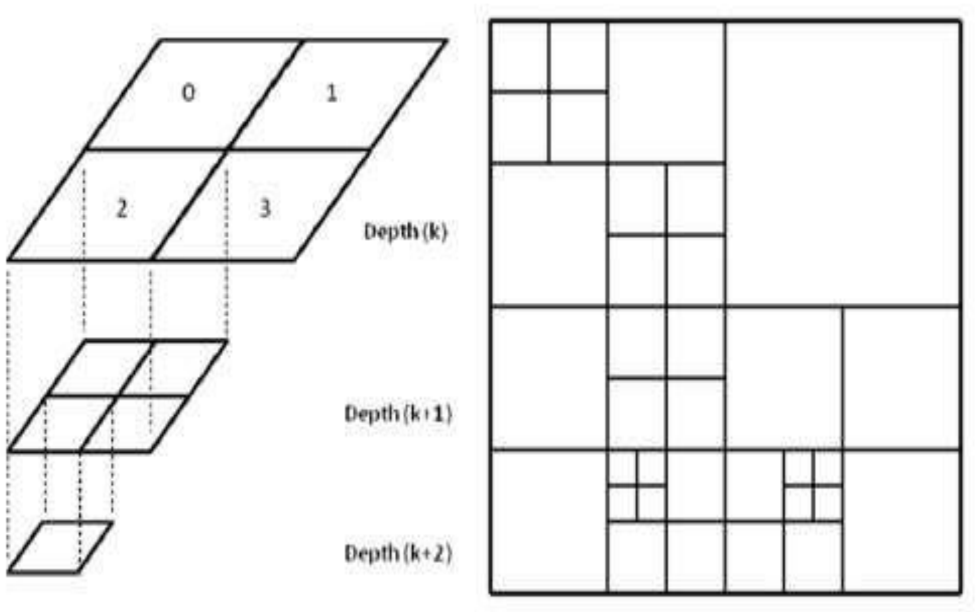


Figure 2-6: Quad tree CU structure in HEVC [1]

As the picture resolution of videos increase from standard definition to HD and beyond, the chances are that the picture will contain larger smooth regions, which can be encoded more effectively using large
block sizes. This is the reason that the HEVC standard supports encoding blocks larger than in H.264/AVC, while it also has a more flexible partitioning structure to allow smaller blocks to be used for more textured and in general uneven regions.

Each CU can be further split into smaller units, which form the basis for prediction. These units are called PUs. Each CU may contain one or more PUs, and each PU can be as large as its root CU or as small as 4x4 in luma block sizes. While an LCU can recursively split into smaller and smaller CUs, the splitting of a CU into PUs is nonrecursive. PUs can be symmetric or asymmetric. Symmetric PUs can be square or rectangular and are used in both intra prediction and inter prediction. In particular, a CU of size 2Nx2N can be split into two symmetric PUs of size Nx2N or 2NxN or four PUs of size NxN. Asymmetric PUs are used only for inter prediction. Starting at the level of a CU, a CB (coding block) can have one transform block (TB) of the same size as the CB or be split into smaller TBs as shown in figures 2-6, 2-7 and 2-8.

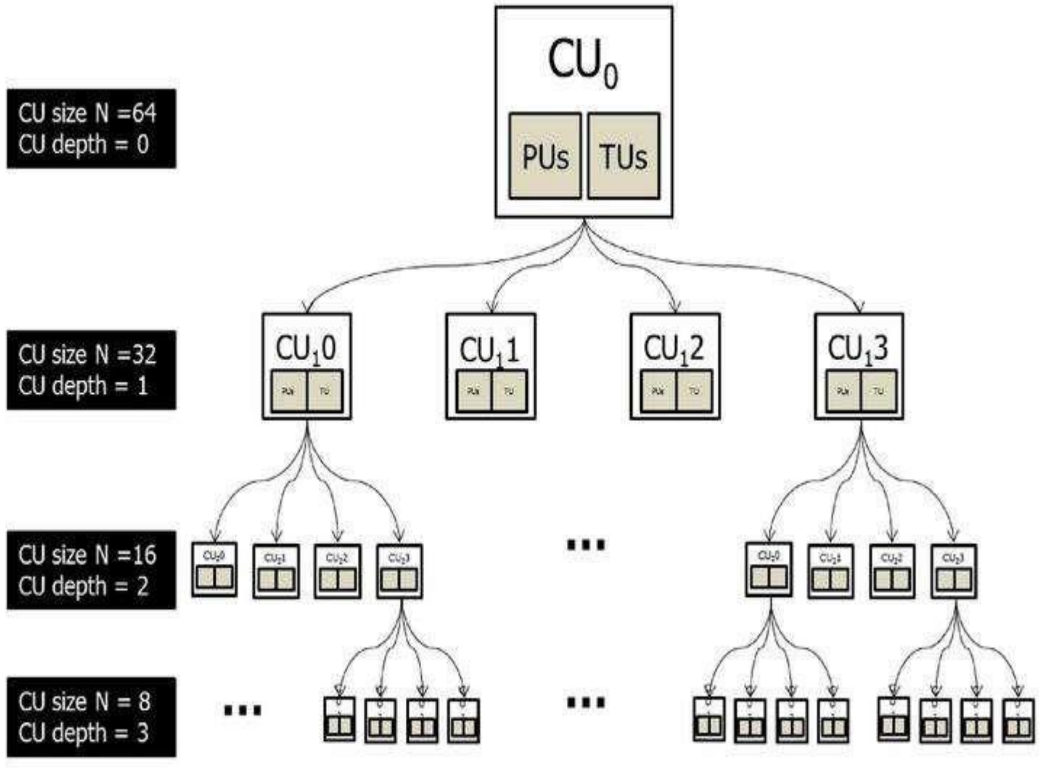


Figure 2-7: Splitting of coding unit into prediction units and transform units [59]

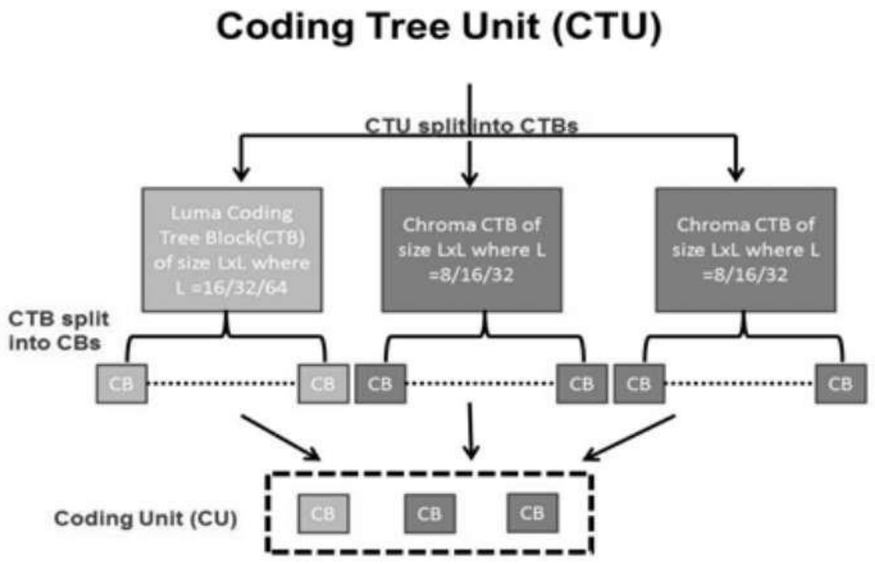

Coding Tree Unit (CTU)
CTU split into CTBs
Luma Coding Tree Block(CTB) of size LxL where L =16/32/64
Chroma CTB of size LxL where L =8/16/32
Chroma CTB of size LxL where L =8/16/32
CTB split into CBs
CB
Coding Unit (CU)

Figure 2-8: Splitting Coding tree units into Coding Blocks [1]

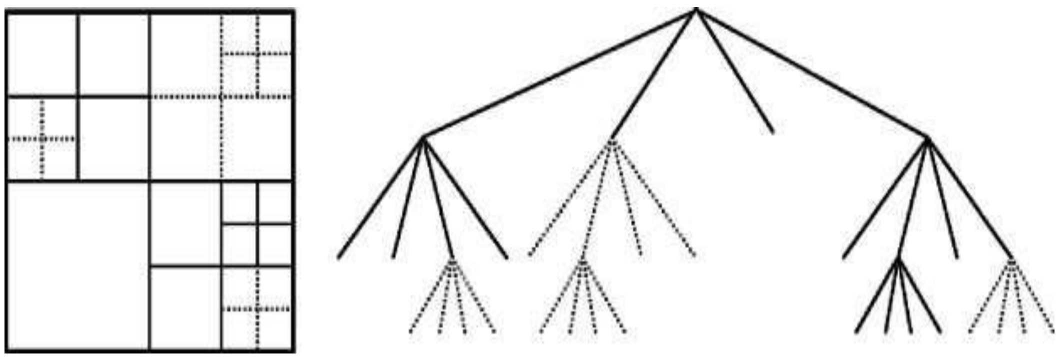


Figure 2-9: CTB with its partitioning and corresponding quad tree [1]

This allows partitioning, which matches the boundaries of the objects in the picture.

2.1.2 Tiles and Slices

The HEVC standard introduced tiles as a means to support parallel processing, with more flexibility than the normal slices in the H.264/AVC standard but considerably lower complexity than the flexible macro block ordering (FMO) standard. Tiles are specified by vertical and horizontal boundaries with intersections that partition a picture into rectangular regions. Figure 2-9 shows an example of tile partitions that contain slices. The spacing of the row and column boundaries of tiles need not be uniform. This offers greater flexibility and can be useful for error resilience applications. In each tile, LCUs are processed in a raster scan order. Similarly, the tiles themselves are processed in a raster scan order within a picture.

The HEVC standard also supports slices, similar to slices found in the H.264/AVC standard, but without FMO. Slices and tiles may be used together within the same picture. To support parallel processing, each slice in HEVC can be subdivided into smaller slices called entropy slices. Each entropy slice can be independently entropy decoded without reference to other entropy slices. Therefore, each core of a CPU can handle an entropy-decoding process in parallel.

The slices are processed in the order of a raster scan. A picture may be split into one or several slices as shown in figure 2-9 so that a picture is a collection of one or more slices. Slices are self-contained in the sense that, given the availability of the active sequence and picture parameter sets, their syntax elements can be parsed from the bit stream and the values of the samples in the area of the picture that the slice represents can be correctly decoded without the use of any data from other slices in the same picture.

Tiles are self-contained and independently decodable rectangular regions of the picture. The main purpose of tiles is to enable the use of parallel processing architectures for encoding and decoding. Multiple tiles may share header information by being contained in the same slice. Alternatively, a single tile may contain multiple slices. A tile consists of a rectangular arranged group of CTUs as shown in figure 2-10.

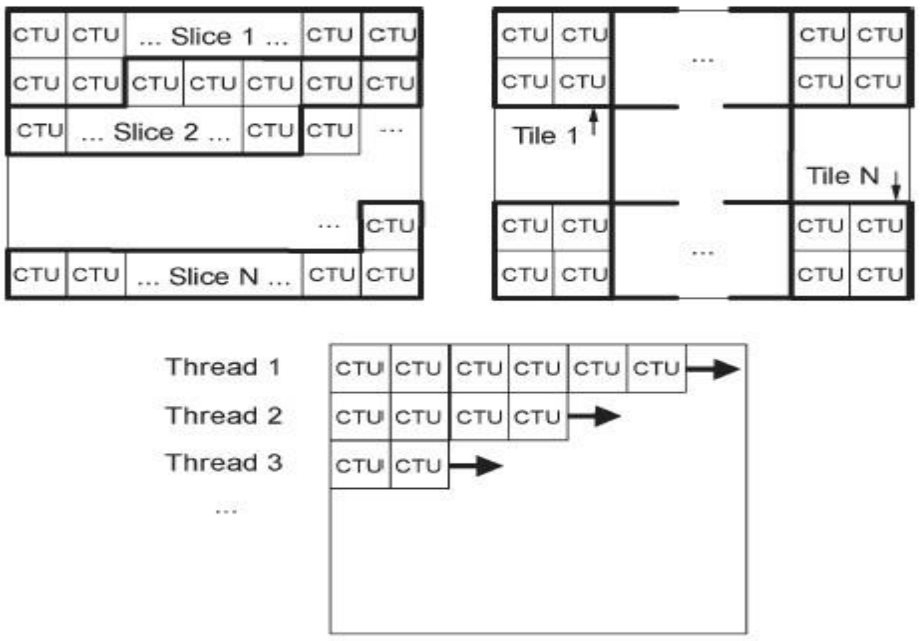


Figure 2-10: Subdivision of picture into tiles and slices [1]

2.2 HEVC Encoder Description

2.2.1 Intra-picture prediction

Intra-picture prediction [21] [23] [24] [25] operates according to the TB size and previously decoded boundary samples from spatially neighboring TBs which are used to form the prediction signal. Directional prediction with 33 different directional orientations is defined for (square) TB sizes from 4×4 up to 32×32. The possible prediction directions are show in figure 2-11. Alternatively, planar prediction and DC prediction can also be used. For chroma the horizontal, vertical, planar, and DC prediction modes can be explicitly signaled, or the chroma prediction mode can be indicated to be the same as the luma prediction mode.

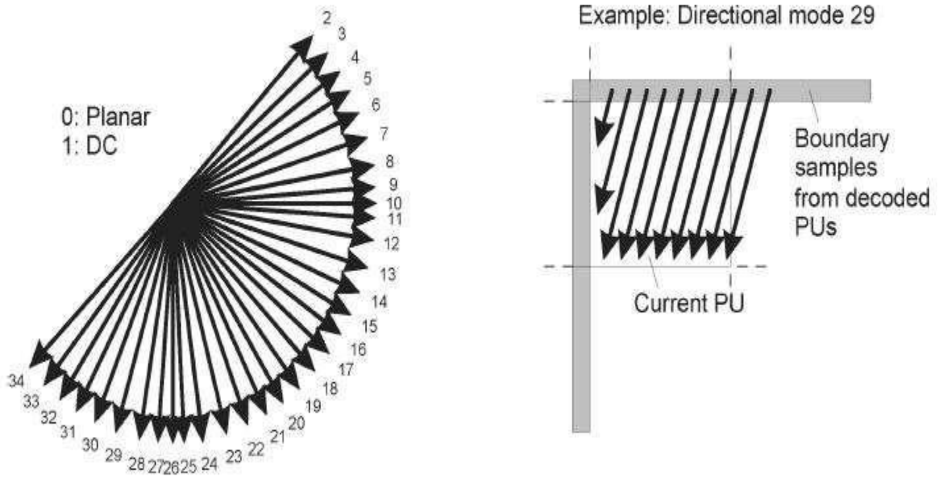


Figure 2-11: Mode decision for intra picture prediction [1]

The HEVC standard also includes a planar intra-prediction mode which is useful for predicting smooth picture regions. In planar mode, the prediction is generated from the average of two linear interpolations.

2.2.2 Inter-picture prediction

Compared to intra-picture predicted CBs, the HEVC standard supports more PB partition shapes for inter-picture predicted CBs. The partitioning modes of PART_2N×2N, PART_2N×N and PART_N×2N as shown in Figure 2-11 indicate the cases when the CB is not split, split into two equal-size PBs horizontally, and split into two equal-size PBs vertically, respectively. PART−N×N specifies that the CB is split into four equal size PBs, but this mode is only supported when the CB size is equal to the smallest allowed CB size. In addition, there are four partitioning types that support splitting the CB into two PBs having different sizes: PART−2N×nU, PART−2N×nD, PART−nL×2N, and PART−nR×2N (U=up, D=down, L=left and R=right) as shown in figure 2-12. These types are known as asymmetric motion partitions.

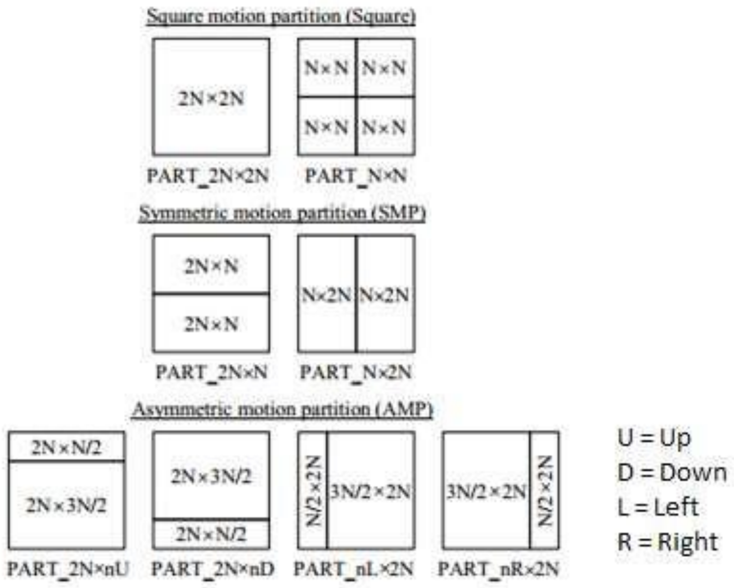


Figure 2-12: Partition modes in HEVC inter-prediction [1]

2.2.3 Entropy Coding

A new and improved CABAC (context adaptive binary arithmetic coding) is used for the entropy coding of the bitstreams. This coding has improved speed, compression and requires less memory then entropy coding used in the H.264/AVC standard (figure 2-13). Instead of implementing the normal CABAC re-initialization for every CTB row, the context state from the second CTU in the previous row is used to start the processing of a brand new CTB row Figure 2-14), and thus taking huge advantage of parallel processing.

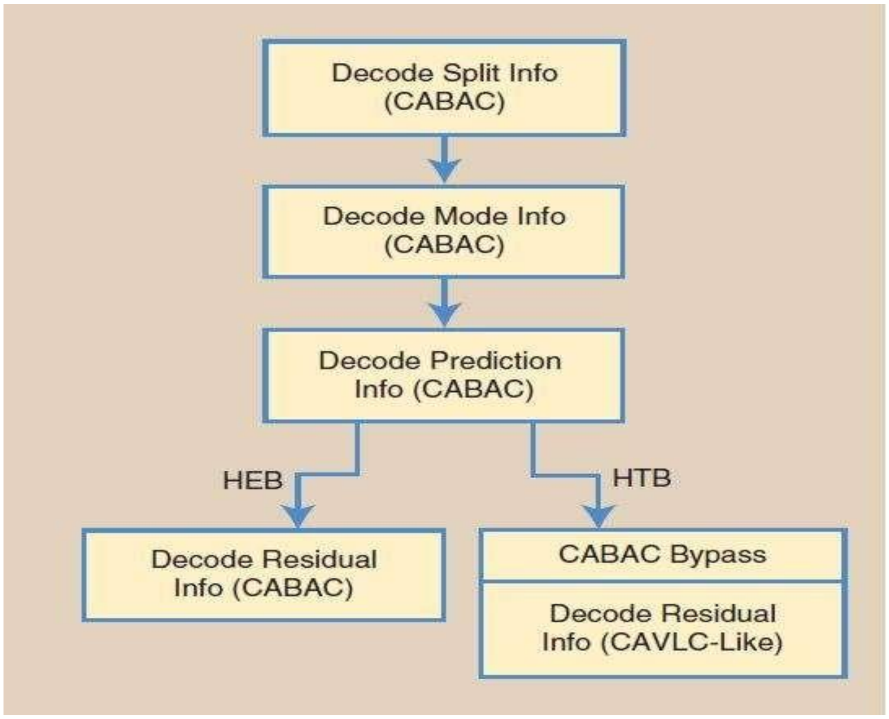

Decode Split Info (CABAC)
Decode Mode Info (CABAC)
Decode Prediction Info (CABAC)
HEB
HTB
Decode Residual Info (CABAC)
CABAC Bypass
Decode Residual Info (CAVLC-Like)

Figure 2-13: HEVC entropy coding [4]

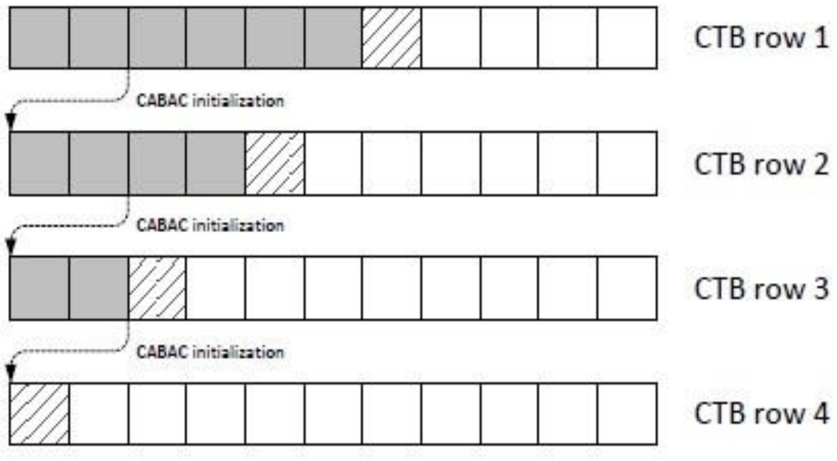


Figure 2-14: Example of waveform processing [4]

2.2.4 In-loop Filtering

In the HEVC standard, two processing steps, namely a deblocking filter (DBF) followed by a sample adaptive offset (SAO) filter are applied to the reconstructed samples before writing them into the decoded picture buffer in the decoder loop. The DBF is intended to reduce the blocking artifacts due to block-based coding. The deblocking filter is applied to all samples adjacent to a PU or TU boundary except the case when the boundary is also a picture boundary, or when deblocking is disabled across slice or tile boundaries. It should be noted that both PU and TU boundaries should be considered since PU boundaries are not always aligned with TU boundaries in some cases of interpicture-predicted CBs. Syntax elements in the SPS and slice headers control whether the deblocking filter is applied across the slice and tile boundaries. The SAO is a process that modifies the decoded samples by conditionally adding an offset value to each sample after the application of the deblocking filter. This is based on values in look-up tables transmitted by the encoder.

2.2.5 Transform, Scaling and Quantization

The HEVC standard uses transform coding of the prediction error residual in a similar manner as in prior standards [1]. The residual block is partitioned into multiple square TBs. The supported transform block sizes are 4×4, 8×8, 16×16, and 32×32. Pre-scaling operation is not needed when using HEVC code since the rows of the transform matrix are close approximations of values of uniformly scaled basis functions of the orthonormal DCT (discrete cosine transform) [1] [22]. Uniform reconstruction quantization (URQ) is used in the HEVC standard, with quantization scaling matrices supported for the various transform block sizes [1]. The range of the QP values is defined from 0 to 51, and an increase by 6 doubles the quantization step size such that the mapping of QP values to step sizes is approximately logarithmic.

2.3 Summary

This chapter gives an outline of the various coding tools of the HEVC codec. HEVC is meant to create a standard capable of providing good video quality at substantially lower bit rates than previous standards. Chapter 3 outlines the description of inter-prediction and the proposed usage of parallel programming to reduce the time taken for motion vector prediction. Chapter 3 gives the description of inter prediction and motion estimation.

Chapter 3

Inter prediction and motion estimation

3.1 Introduction to Inter Prediction

Each inter coded PU shall have a set of motion parameters consisting of motion vector, reference picture index, reference picture list usage flag to be used for inter prediction sample generation, in an explicit or implicit way of signaling. When a CU is coded with skip mode (i.e., PredMode == MODE_SKIP), the CU shall be represented as one PU that has no significant transform coefficients and motion vectors, reference picture index and reference picture list usage flag obtained by motion merge. The motion merge is to find neighboring inter coded PU such that its motion parameters (motion vector, reference picture index, and reference picture list usage flag) can be inferred as the ones for the current PU. Encoder can select the best inferred motion parameters from multiple candidates formed by spatial neighboring PUs and temporally neighboring PUs, and transmits corresponding index indicating chosen candidate. Not only for skipmode, the Motion Merge can be applied to any inter coded PU (i.e., PredMode == MODE_INTER). In any inter coded PUs, encoder can have freedom to use motion merge or explicit transmission of motion parameters, where motion vector, corresponding reference picture index for each reference picture list and reference picture list usage flag are signaled explicitly per each PU. For inter coded PU, significant transform coefficients are sent to the decoder.

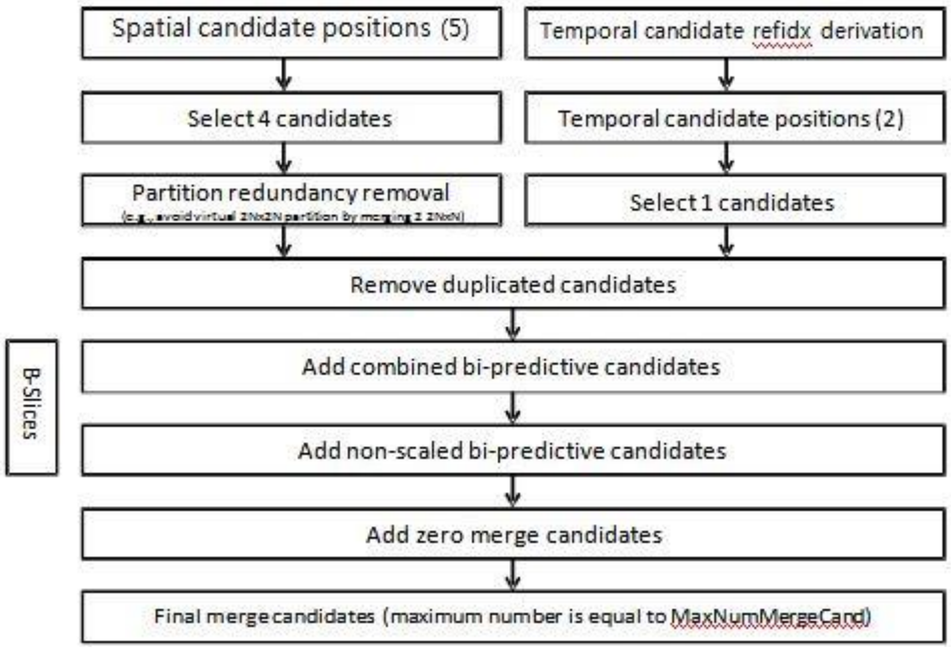

Spatial candidate positions (5)
Temporal candidate refidx derivation
Select 4 candidates
Temporal candidate positions (2)
Partition redundancy removal
(e.g., avoid virtual 2Nx2N partition by merging 2 2NxN)
Select 1 candidates
Remove duplicated candidates
B-Slices
Add combined bi-predictive candidates
Add non-scaled bi-predictive candidates
Add zero merge candidates
Final merge candidates (maximum number is equal to MaxNumMergeCand)

Figure 3-1: Derivation process for motion merge candidate [4]

Figure 3-1 summarizes derivation process for motion merge candidates. Two types of merge candidates are considered in motion merge: spatial merge candidate and temporal merge candidate. For spatial merge candidate derivation, four merge candidates are selected among candidates that are located in five different positions. In the process of candidate selection, redundant partition shape is avoided in order not to emulate virtual 2Nx2N partition by merging two 2NxN or two Nx2N partitions. For temporal merge candidate derivation, one merge candidate is selected among two candidates. After a list of spatio-temporal candidates is made, duplicated candidates which have same motion parameters in the list are removed in order to have distinctive candidates only. Since constant number of candidates for each PU is assumed at decoder, additional candidates are generated when the number of candidates does not reach to maximum number of merge candidate (MaxNumMergeCand) which is signaled in slice header. For B-Slices, combined bi-predictive and non-scaled bi-predictive candidates are generated utilizing the candidates from list of spatio-temporal candidates. For both P- and B- slices, zero merge candidates are added at the end of the list. Between each generation step, derivation process is stopped if the number of candidates reaches to MaxNumMergeCand. In current common test condition, MaxNumMergeCand is set equal to 5. Since the number of candidates is constant, index of best motion merge candidate is encoded using truncated unary binarization (TU).

3.2 Motion Vector Prediction

Motion vector prediction exploits spatio-temporal correlation of motion vector with neighboring PUs, which is used for explicit transmission of motion parameters. It constructs motion vector candidate list by firstly checking availability of left, above temporally neighboring PU positions, removing redundant candidates and adding zero vector to make the candidate list to be constant length as a normative process. Then, encoder can select the best predictor from the candidate list and transmits corresponding index indicating chosen candidate. Similarly with merge index signaling, index of the best motion vector candidate is encoded using truncated unary as maximum number is equal to 2. In the following sections, details about derivation process of motion vector prediction candidate are provided.

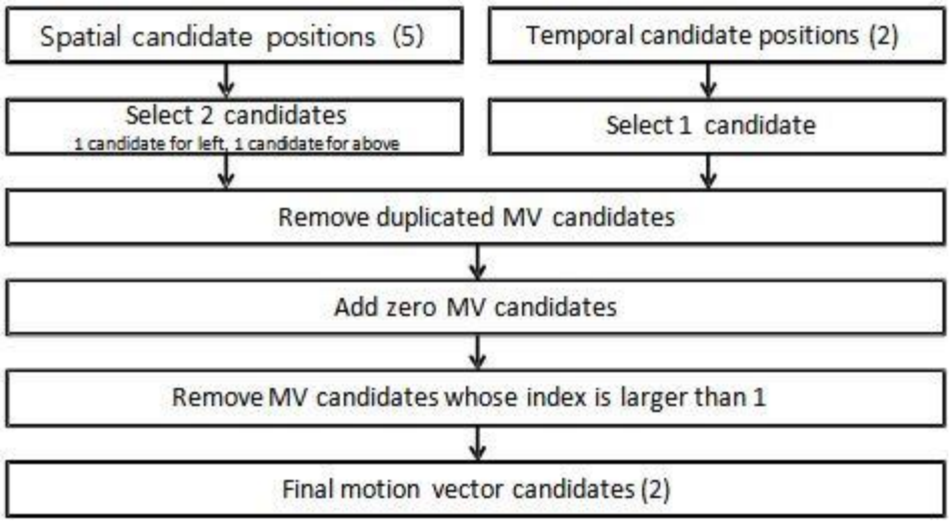

Spatial candidate positions (5)
Temporal candidate positions (2)
Select 2 candidates
1 candidate for left, 1 candidate for above
Select 1 candidate
Remove duplicated MV candidates
Add zero MV candidates
Remove MV candidates whose index is larger than 1
Final motion vector candidates (2)

Figure 3-2: Overall flow of deblocking filter process [4]

Figure 3-2 summarizes derivation process for motion vector prediction candidate. In motion vector prediction, 2 types of motion vector candidates are considered: spatial motion vector candidate and temporal motion vector candidate. For spatial motion vector candidate derivation, 2 motion vector candidates are derived based on motion vectors of each PU located in 5 different positions. In the process of derivation, 1 motion vector candidate is selected utilizing PUs in the left side of current PU and 1 motion vector candidate is derived utilizing Pus in the above side of current PU. For temporal motion vector candidate derivation, 1 motion vector candidate is selected between 2 candidates, which are derived based on 2 different co-located positions. After the first list of spatio-temporal candidates is made, duplicated motion vector candidates in the list are removed. If the number of candidates is larger than 2, motion vector candidates whose index is greater than 1 are removed from the list. If the number of spatio-temporal motion vector candidates is smaller than 2, additional zero motion vector candidates are added to the list.

3.2.1 Spatial Motion Vector Candidates

In the derivation of spatial motion vector candidates, maximum 2 candidates are considered among 5 candidates, which are derived from PUs located in positions. The candidate positions of motion vector prediction are same with those of motion merge. The order of derivation for left side of current PU is set as $A_0 \rightarrow A_1 \rightarrow$ scaled $A_0 \rightarrow$ scaled A_1. The order of derivation for above side of current PU is set as $B_0 \rightarrow B_1 \rightarrow B_2$ $\rightarrow$ scaled $B_0 \rightarrow$ scaled $B_1 \rightarrow$ scaled B_2. For each side, there are 4 cases which can be used for motion vector candidate. Even though two cases are not required to do spatial scaling, the other 2 cases are required to do spatial scaling. 4 different cases are summarized as follows.

No spatial scaling

(1) Same reference picture list, and same reference picture index (same POC)

(2) Different reference picture list, but same reference picture (same POC)

Spatial scaling

(3) Same reference picture list, but different reference picture (different POC)

(4) Different reference picture list, and different reference picture (different POC)

Spatial scaling is considered when POC is different between reference picture of neighboring PU and that of current PU regardless of reference picture list. If all PUs of left candidates is not available or intra coded, scaling for above motion vector is allowed to help parallel derivation of left and above MV candidates. Otherwise, spatial scaling is not allowed for above motion vector.

3.2.2 Interpolation Filter

For the luma interpolation filter, an 8-tap separable DCT-based interpolation filter is used, as shown in Table 3-1.

Table 3-1: Coefficients for DCT based luma interpolation filter [27]

Position	Filter coefficients
1/4	{ -1, 4, -10, 58, 17, -5, 1 }
2/4	{ -1, 4, -11, 40, 40, -11, 4, -1 }
3/4	{ 1, -5, 17, 58, -10, 4, -1 }

Similarly, a 4-tap separable DCT-based interpolation filter is used for the chroma interpolation filter, as shown in Table 3-2.

Table 3-2: Coefficients for DCT based chroma interpolation filter [60]

Position	Filter coefficients
1/8	{ -2, 58, 10, -2 }
2/8	{ -4, 54, 16, -2 }
3/8	{ -6, 46, 28, -4 }
4/8	{ -4, 36, 36, -4 }
5/8	{ -4, 28, 46, -6 }
6/8	{ -2, 16, 54, -4 }
7/8	{ -2, 10, 58, -2 }

For the bi-directional prediction, the bit-depth of the output of the interpolation filter is maintained to 14-bit accuracy, regardless of the source bit-depth, before the averaging of the two prediction signals

3.3 Summary

This chapter explains the inter prediction method in HEVC and its improvements compared to H.264 [26] inter prediction method. Chapter 4 will explain about parallel processing and OpenMP.

Chapter 4

Parallel Programming

4.1 Concept of parallel computing

In order to reduce the time taken by a computational process, operations are done in parallel so that more than one process is happening simultaneously. Parallel computing is a technique in which multiple computations are done simultaneously, either with the help of hardware or software. The basic approach for parallel processing is to break the task into multiple smaller tasks and further assign each task to each thread which performs required operations in parallel.

Earlier, programs were running on serial computing platforms, with a single processor running those instructions from start to finish. These sequences of instructions were executed one after the other.

In the present day scenario, in order to achieve maximum performance in the least time possible, serial computing is replaced with parallel computing. With the use of multi-core processors, GPUs and software, it is possible to implement parallel computing to a large extent. Parallelization can sometimes get complicated due to race conditions, data dependency, synchronization and communication among different threads.

4.2 OpenMP

OpenMP [40], short for Open Multi-processing, is an API (application program interface) which supports multi-platform parallel programming in C/C++. The API has a simple interface for developing parallel applications.

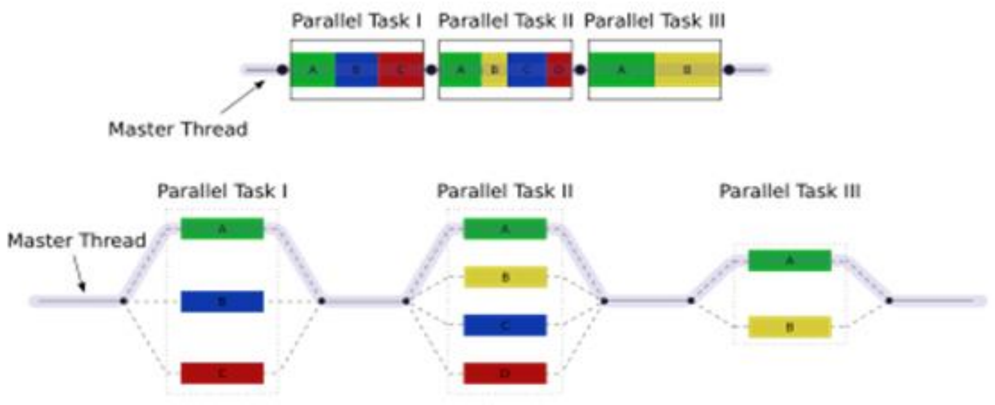


Figure 4-1: Fork-join model, multithreading in OpenMP [13] [44]

In OpenMP, fork-join model is used for parallel execution where taks are performed by multiple threads defined by OpenMP directives [15] as shown in Figure . The intention of the OpenMP API is to support programs that can run both in sequential and parallel executions.

4.2.1 OpenMP directives

4.2.1.1 OpenMP parallel pragma [56]

For creating the threads that execute the block of code following the directive in parallel, this directive is used, which also helps the programmer choose the partitioning of the code segment.

For example:

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


 …
 …
 }     //parallel work ends
}
```

The header file omp.h should be included in the program so that it includes and initiates the OpenMP functions. The pragma directive is used to execute the task enclosed within the curly braces, in parallel.

4.2.1.2 OpenMP barrier pragma [56]

This directive does the synchronization of all threads in a team. When the thread reaches a barrier, it will wait until all threads reach the barrier point. Then, it continues executing the code in parallel after the barrier.

For example:

```
#pragma omp barrier newline
```

4.2.1.3 OpenMP for pragma [56]

This directive is used to run multiple iterations of the for loop in parallel by assigning iterations to threads. Each of the threads will then execute one or more iterations simultaneously. This is helpful only when the iterations are independent of each other.

For example:

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

4.2.1.4 OpenMP sections pragma [56]

This directive identifies the code sections to be divided among all the threads. This way each of the threads will take up the task of executing a particular section simultaneously.

For example:

```
#pragma omp sections [clauses]
{
 #pragma omp section
  {
   //code
  }
 }
```

4.3 Threads

Thread is the basic unit of processing that happens in an operating system. The use of threads allows the user to improve performance significantly by allowing two or more activities to occur at the same time. However, managing simultaneous activities and their possible interaction will lead to the addition of complexity into the program. The concept of multithreading enables the running of processes and sub-processes concurrently, with most processes having multiple threads [47].

4.4 Approach

Parallelization can sometimes get complicated due to race conditions, data dependency, synchronization and communication among different threads.

The basic approach in deciding parallelization approach is to first analyze the part of the program that needs to be parallelized and then decide the type of parallel programming technique that needs to be implemented. It should also be determined whether or not the problem is one that can be parallelized. Parallel programming models are not limited to particular type of machinery but can be implemented on any underlying hardware.

4.5 Summary

Chapter 4 gives an introduction to parallel computing in general and OpenMP in particular. In this thesis, OpenMP API is used in the HEVC code to run certain code segments in parallel. Chapter 5 will show the results which compare various parameters observed before and after the usage of OpenMP in the HEVC [1] code.

Chapter 5

Implementation and results

5.1 Overview

One of the biggest advantages of HEVC [1] over H.264 [27] is the ability to provide very high compression ratios which are helpful in a variety of applications. But, this comes at the cost of increasing the complexity of the compression scheme. It also increases the time taken to encode a given video sequence. In the encoder, motion estimation by itself takes up to 70% of the total encoding time. The comparison and results provided below show that the use of OpenMP for running multiple segments of code in parallel will reduce the encoding time significantly [1] [31].

5.2 Proposed solution

The proposed approach will make use of the directive '#pragma omp sections' [56], '#pragma omp parallel for' [56] and '#pragma omp nowait' [56] along with certain code changes to account for the parallel execution of the code segments. Here, the code segments are chosen in such a way that there is no dependency on other threads that are running in parallel on the other segments. This way, the conditional segments take less time to run when multiple threads check for multiple conditions at the same time. The directive #pragma omp sections was used to create sections in particular segments of the code that can be run in parallel. The directive #pragma omp nowait will ensure that the implied barrier will be avoided at the end of the loop when there are multiple independent loops within a parallel region. The #pragma omp parallel for will create threads to run the 'for loop' in parallel. All these parallel executions lead to a reduction in the encoding time.

5.3 Test conditions

The performance of this implementation was evaluated using HEVC reference software HM13.0 [12]. A total of five standard video sequences [16] were used with different QP values of 22, 27, 32 and 37 as recommended by the JCT-VC with encoder random access main being the configuration for encoding.

Table 5-1: Standard test sequences used [16]

No.	Sequence	Resolution	Type	No. of frames	Frame rate(Hz)
1.	RaceHorses_416x240_30.yuv	416x240	WQVGA	50	30
2.	BasketballDrillText_832x480_50.yuv	832x480	WVGA	50	50
3.	BQMall_832x480_60.yuv	832x480	WVGA	50	60
4.	KristenAndSara_1280x720_60.yuv	1280x720	SD	50	60
5.	Kimono1_1920x1080_24.yuv	1920x1080	WQHD	50	24

5.4 Results

5.4.1 Encoding time reduction

The proposed approach gives a encoding time reduction in the range of 10-25% when encoding the standard test sequences for various QP when compared to the standard encoder in HM 13.0 [12]. The results are presented in Figure 5-1 through Figure 5-5.

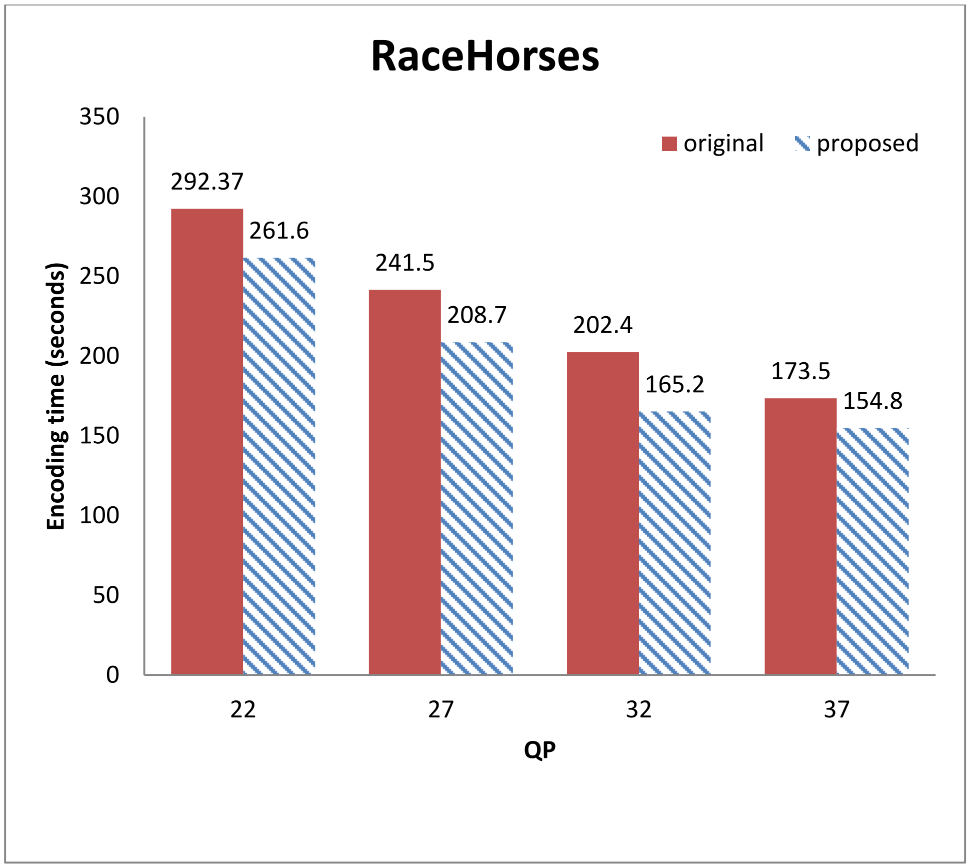


Figure 5-1: Encoding time vs QP for RaceHorses

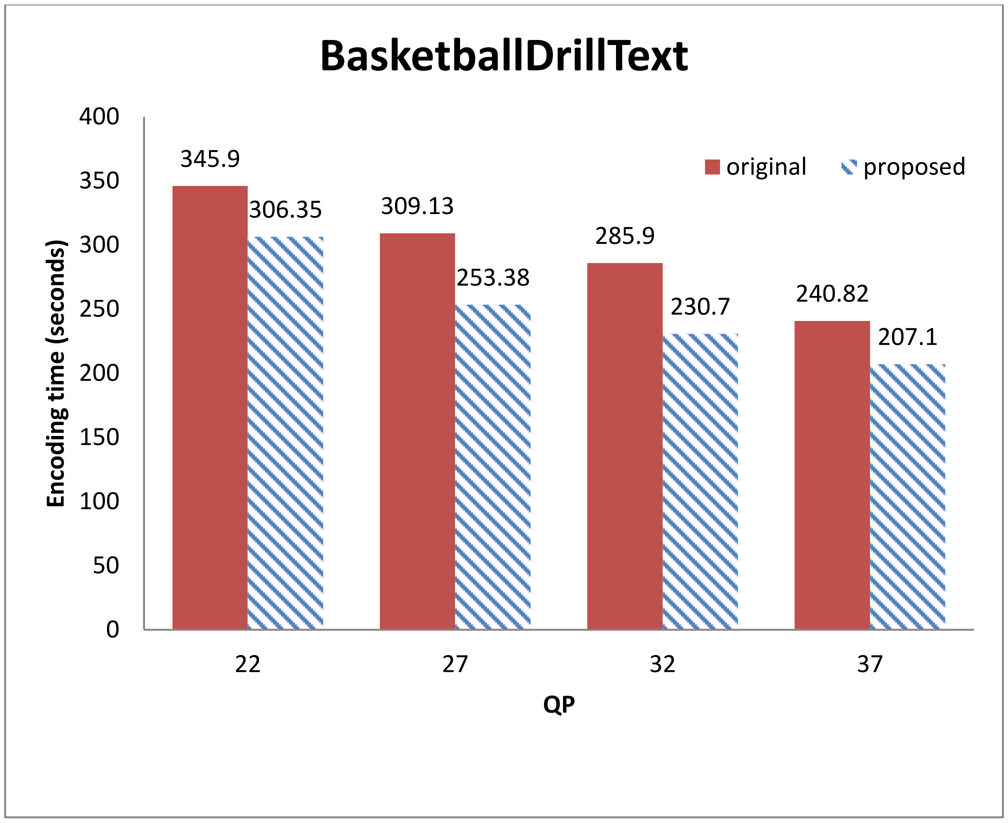


Figure 5-2: Encoding time vs QP for BasketballDrillText

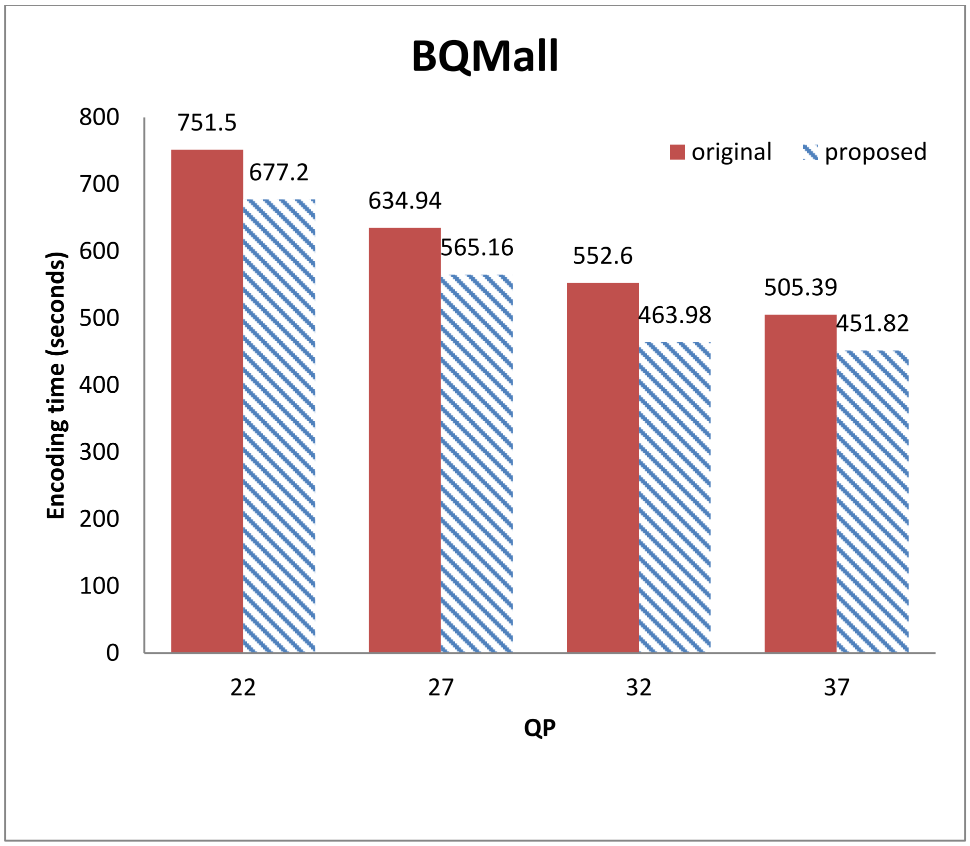


Figure 5-3: Encoding time vs QP for BQMall

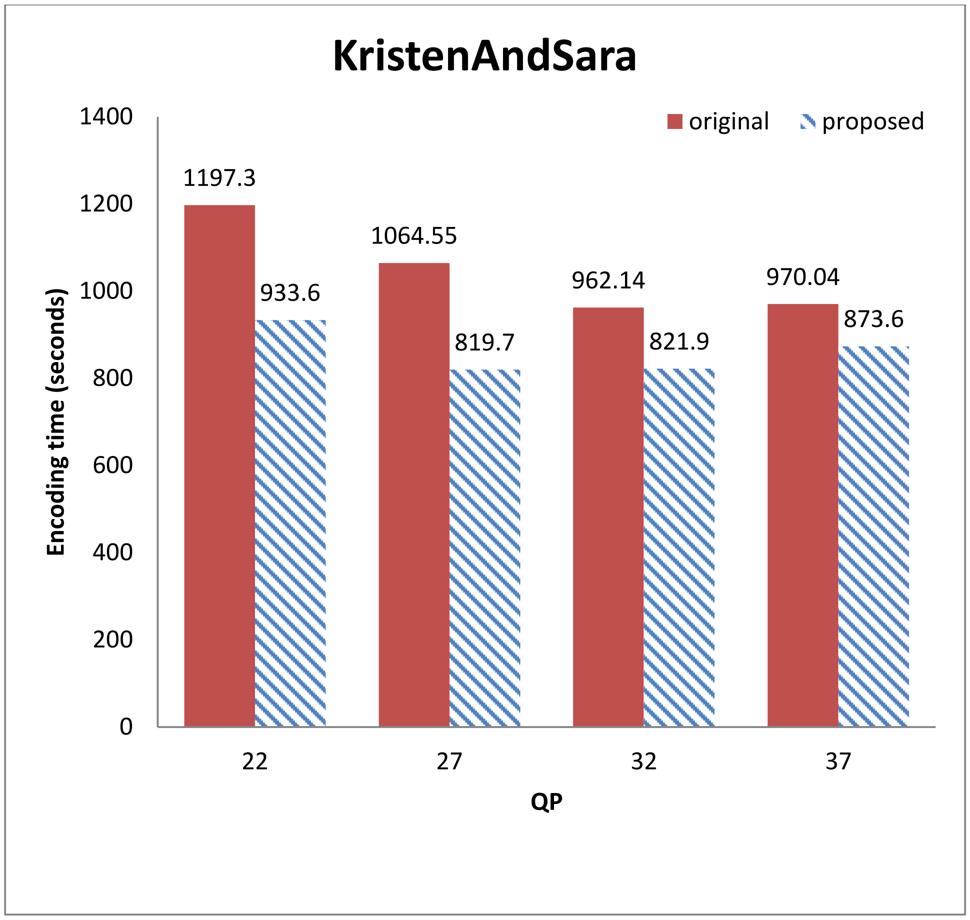


Figure 5-4: Encoding time vs QP for KristenAndSara

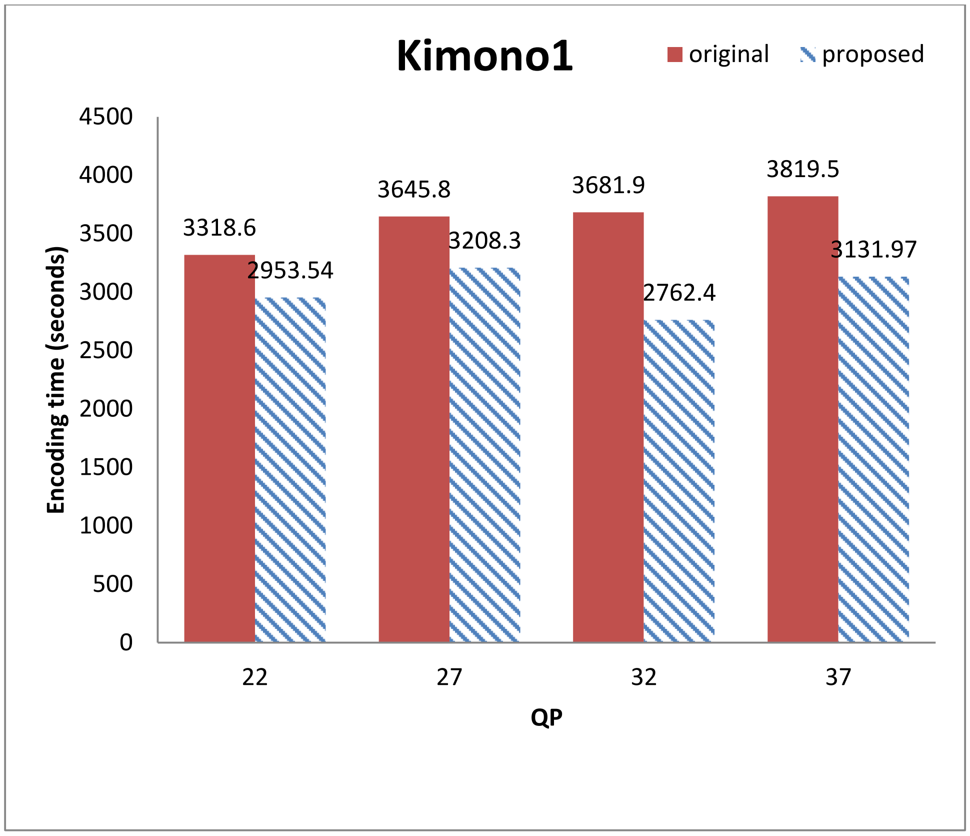


Figure 5-5: Encoding time vs QP for Kimono1

5.4.2 Percentage reduction in encoding time

Figures 5-6 through 5-10 shows the percentage reduction in encoding time for different values of QP by using the proposed method run on the standard test sequences.

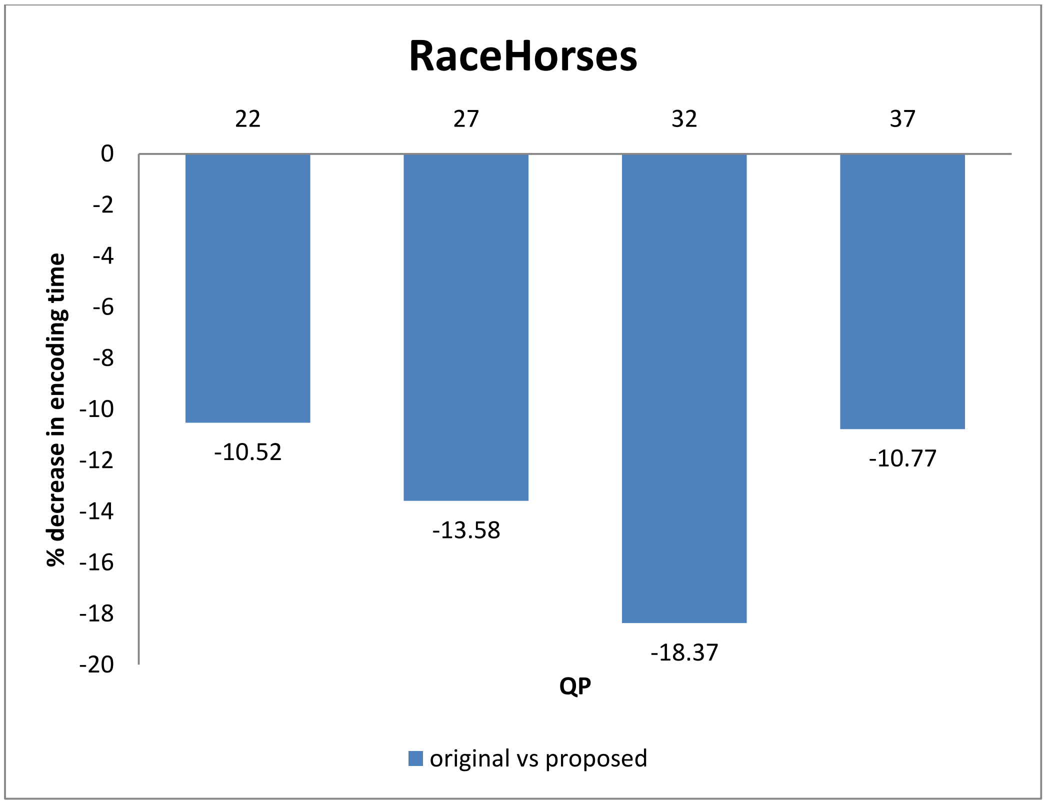


Figure 5-6: Percentage reduction in encoding time vs QP for Racehorses

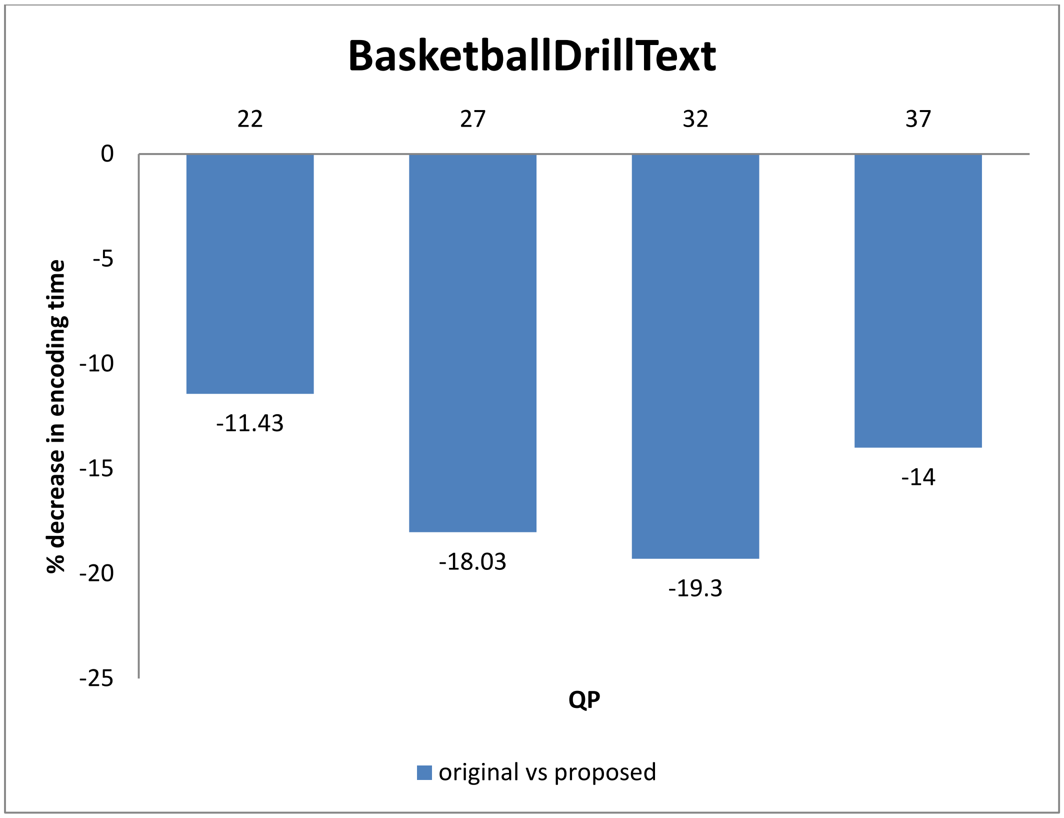


Figure 5-7: Percentage reduction in encoding time vs QP for BasketballDrillText

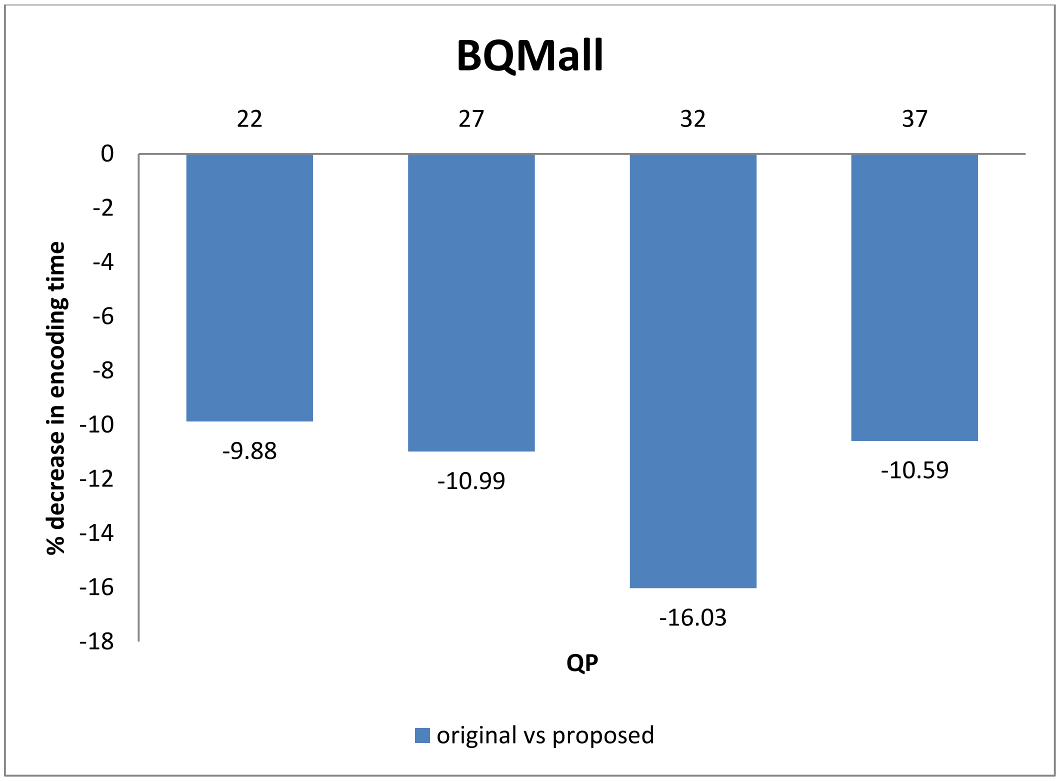


Figure 5-8: Percentage reduction in encoding time vs QP for BQMall

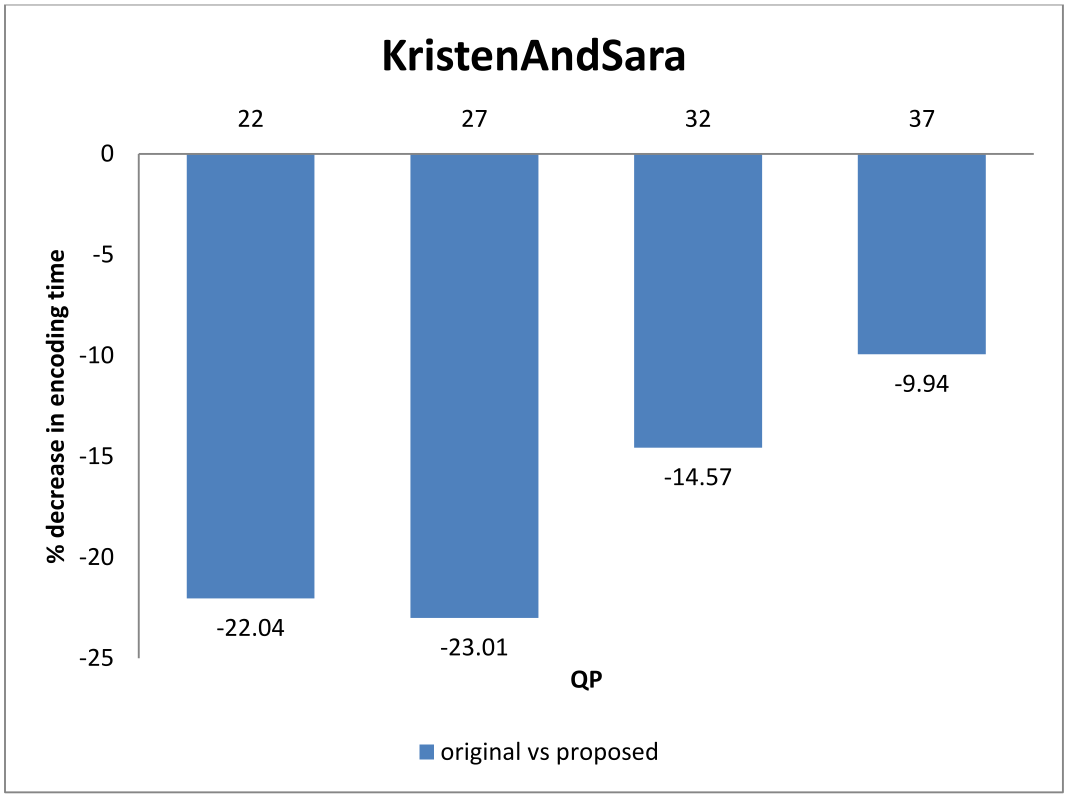


Figure 5-9: Percentage reduction in encoding time vs QP for KristenAndSara

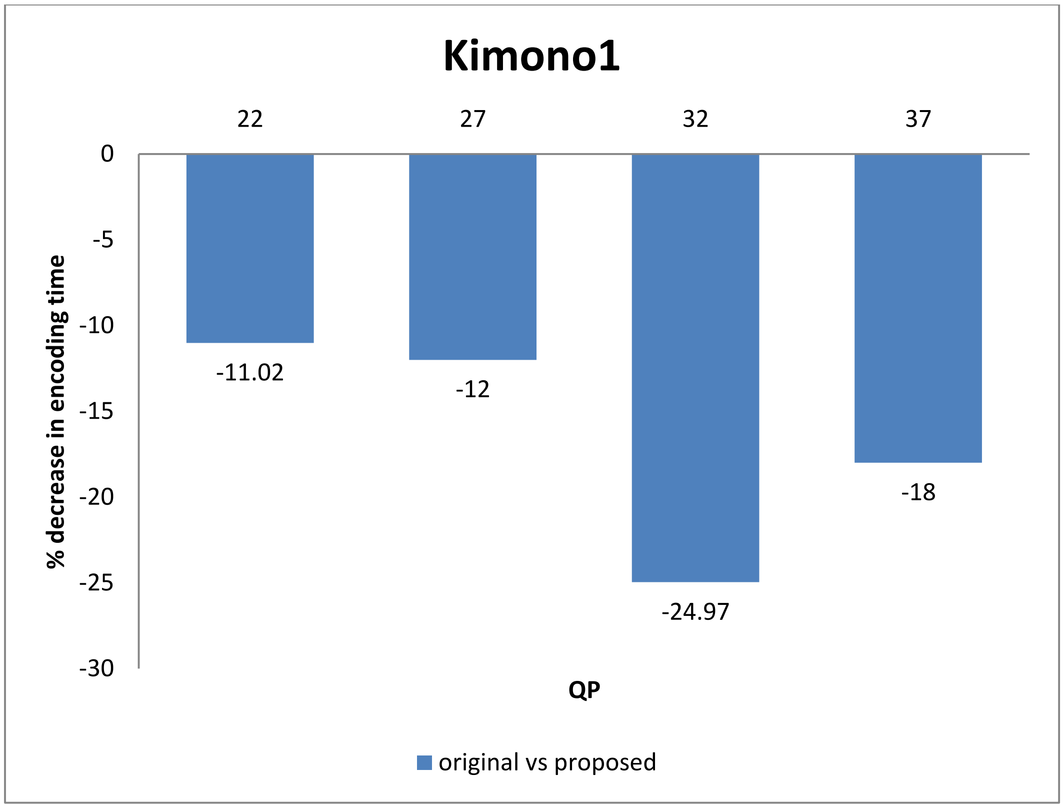


Figure 5-10: Percentage reduction in encoding time vs QP for Kimono1

5.4.3 BD-PSNR

BD-PSNR is a metric that will provide an evaluation of the rate distortion performance of the video sequence. But, coding complexity is not taken into account, which actually is a prime factor in the HEVC encoder. Figures 5-11 through 5-15 shows the BD-PSNR vs QP graphs for the standard test sequences.

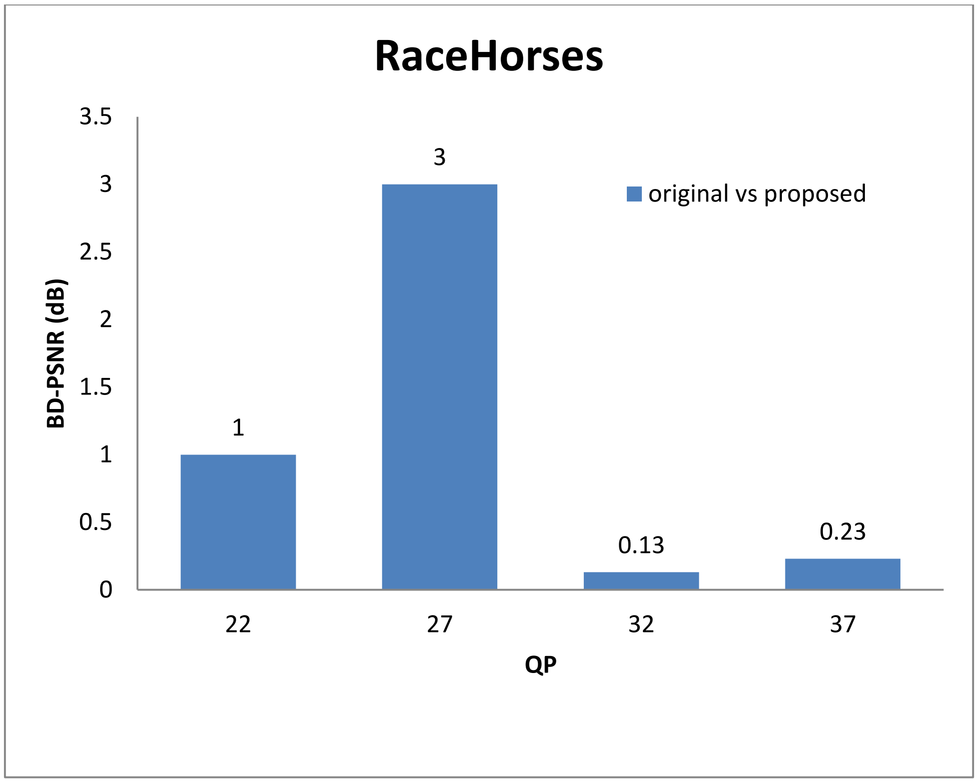


Figure 5-11: BD-PSNR vs QP for RaceHorses

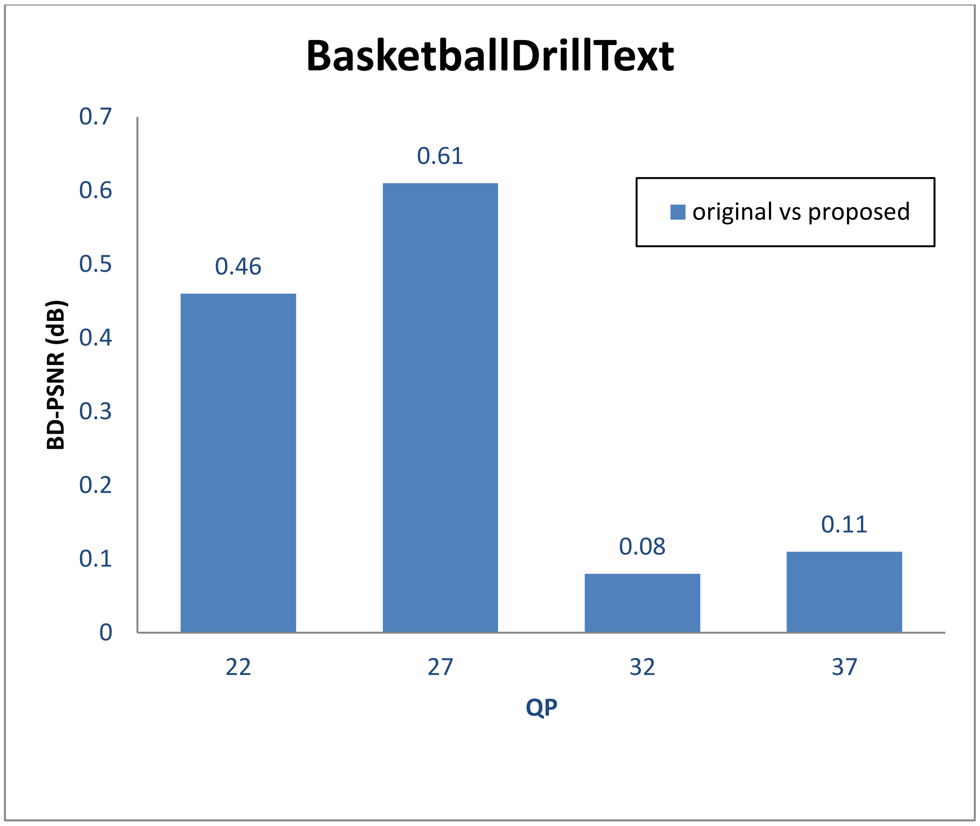


Figure 5-12: BD-PSNR vs QP for BasketballDrillText

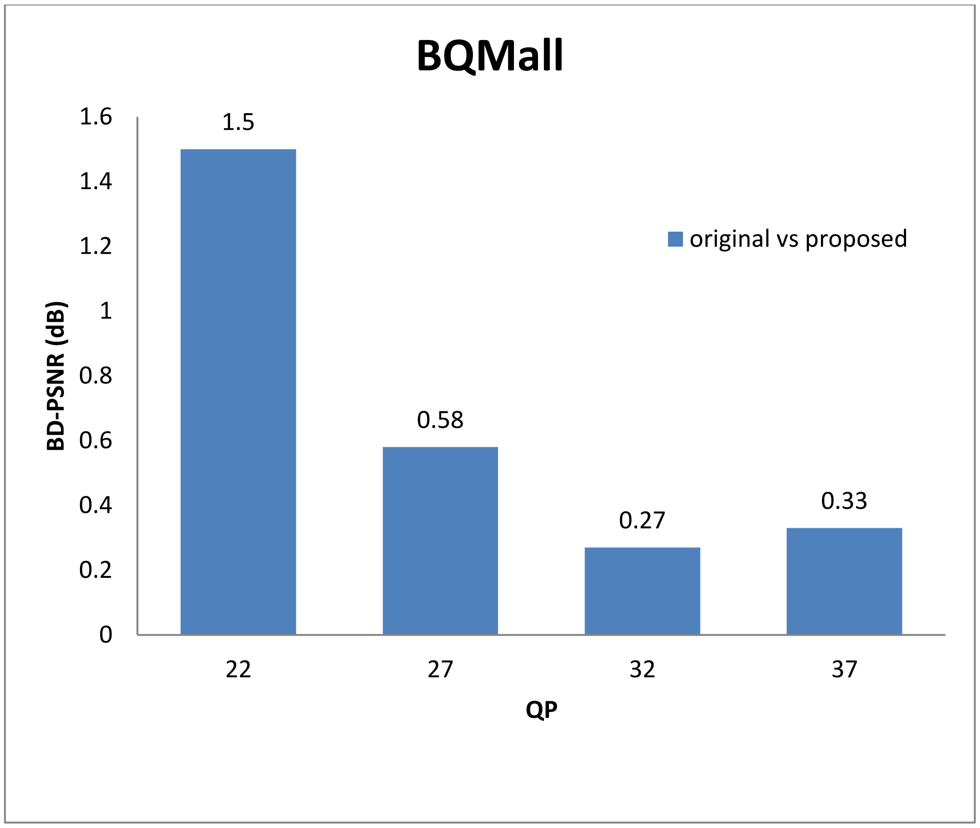


Figure 5-13: BD-PSNR vs QP for BQMall

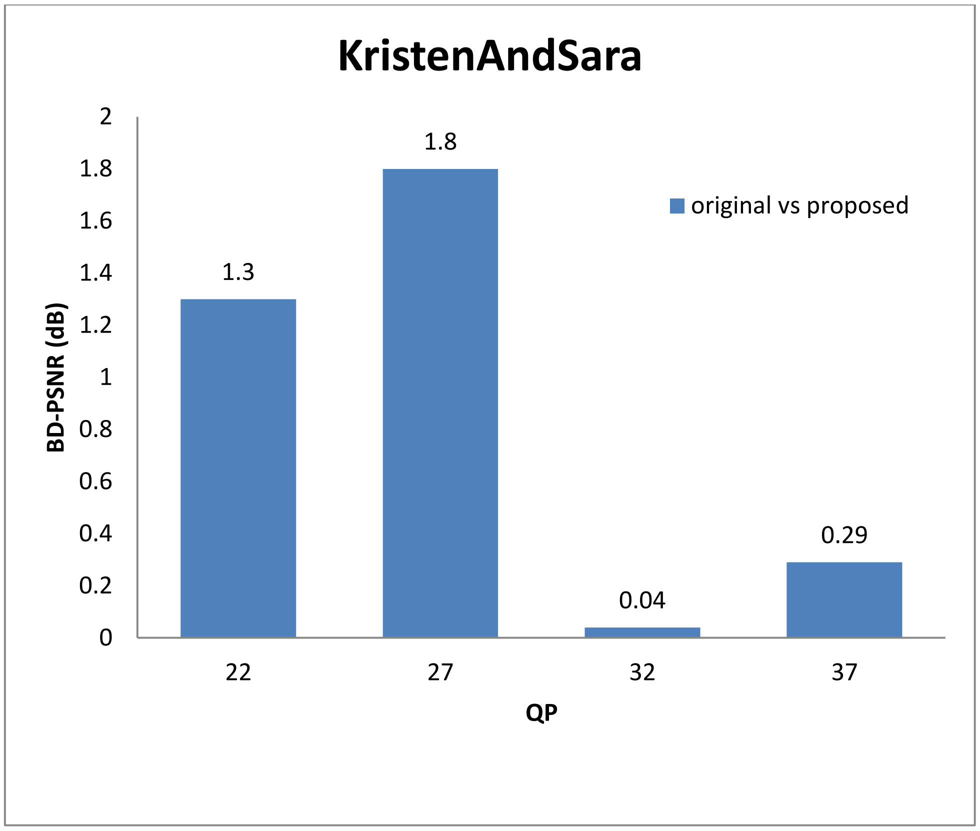


Figure 5-14: BD-PSNR vs QP for KristenAndSara

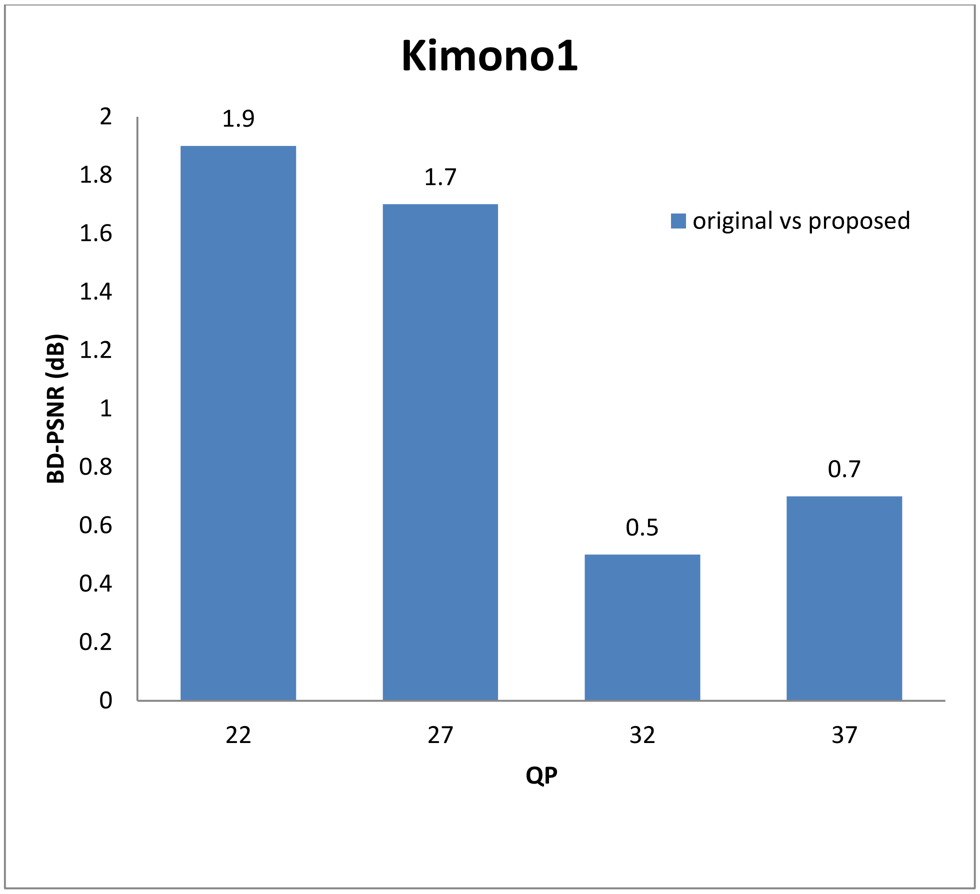


Figure 5-15: BD-PSNR vs QP for Kimono1

5.4.4 BD-bitrate [68]

It is a metric that determines the quality of an encoded video sequence. The bitrate changes for the proposed method are negligible increments and can be seen in Figure 5-16 through 5-20.

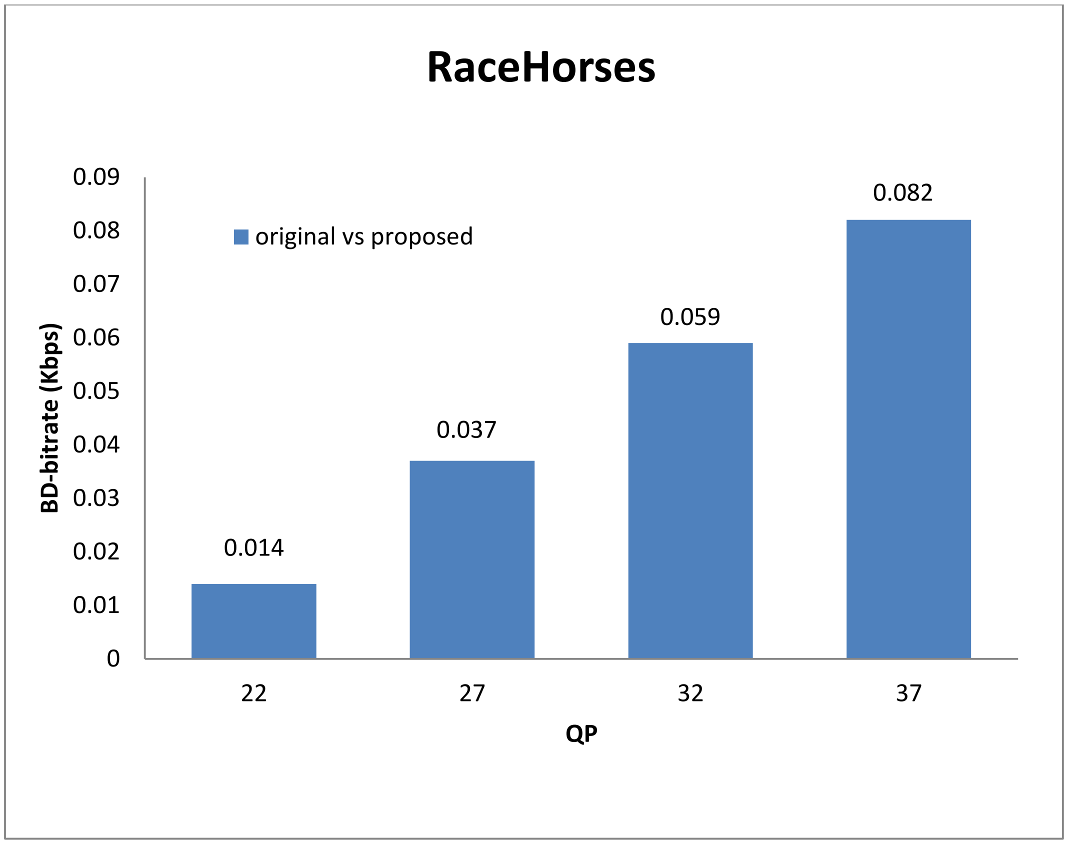


Figure 5-16: BD-bitrate vs QP for RaceHorses

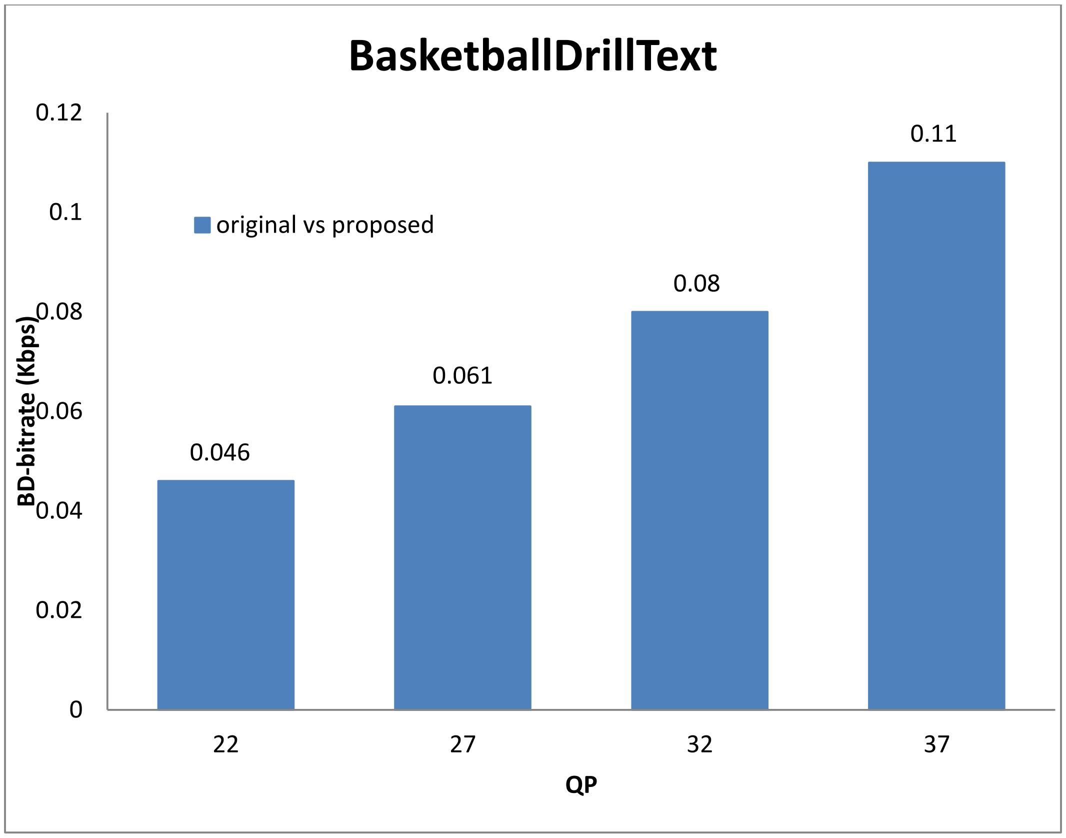


Figure 5-17: BD-bitrate vs QP for BasketballDrillText

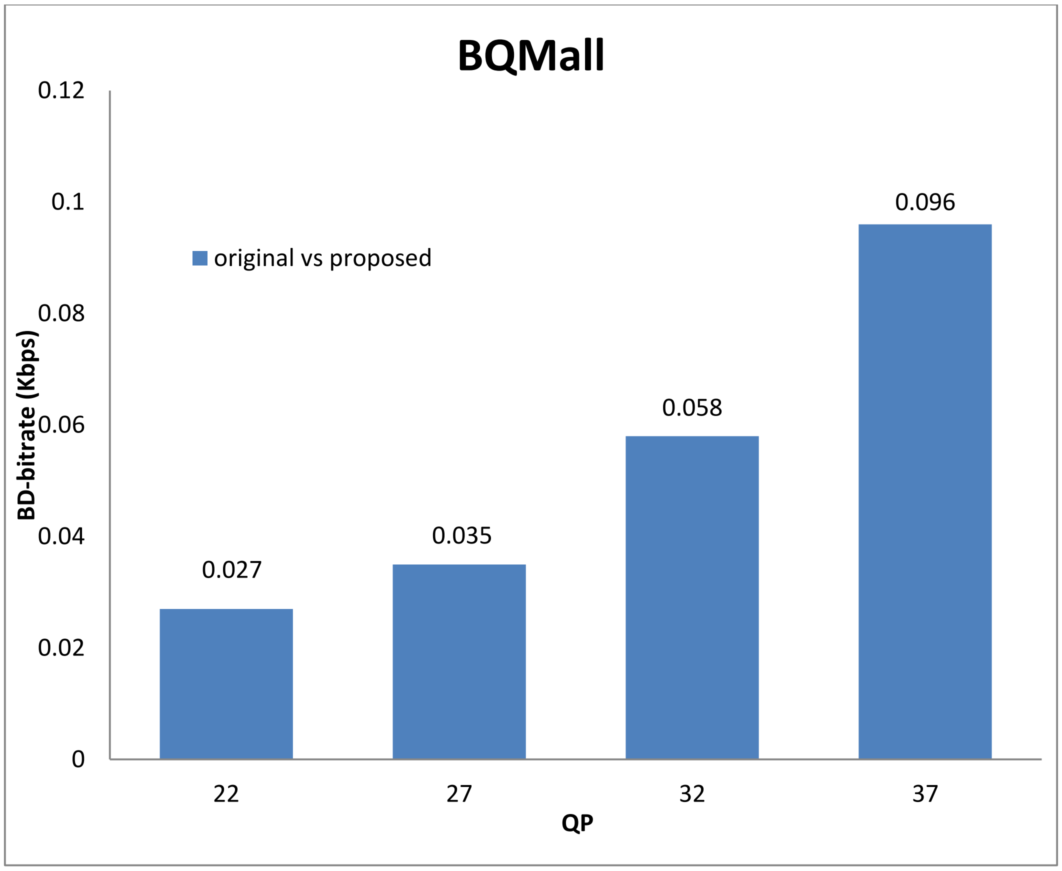


Figure 5-18: BD-bitrate vs QP for BQMall

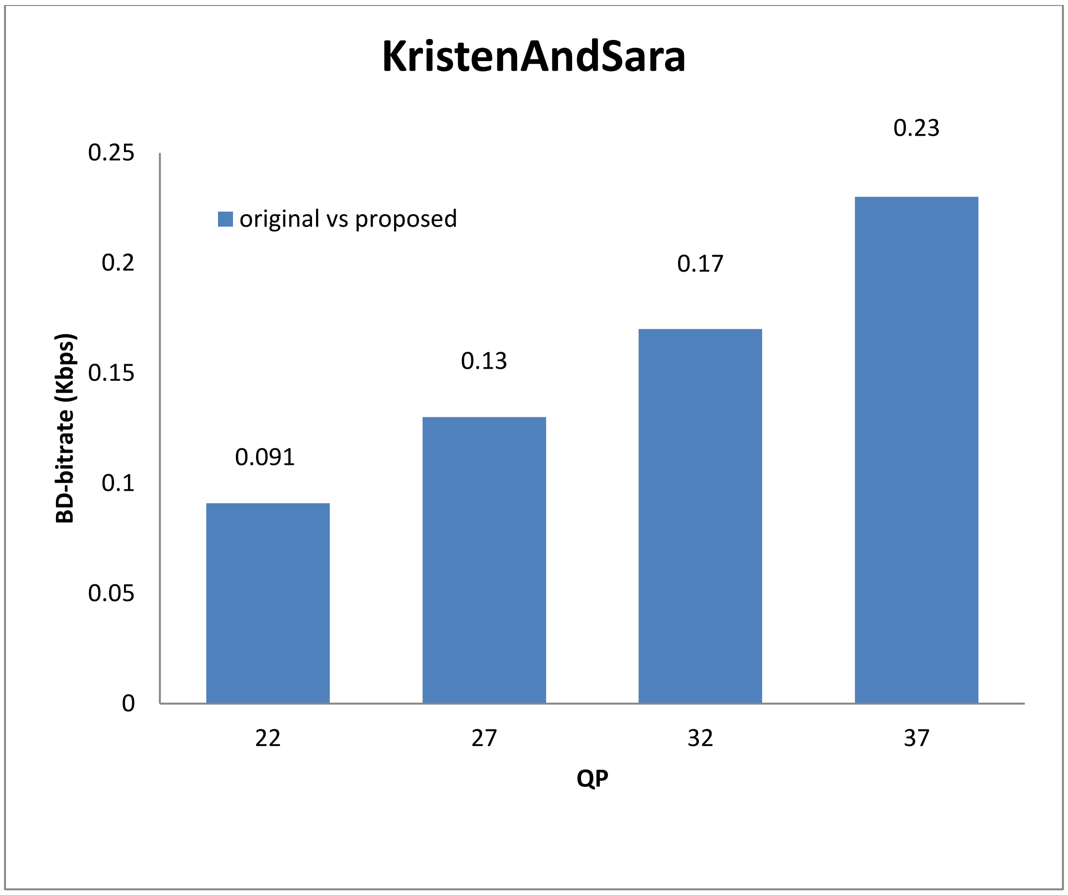


Figure 5-19: BD-bitrate vs QP for KristenAndSara

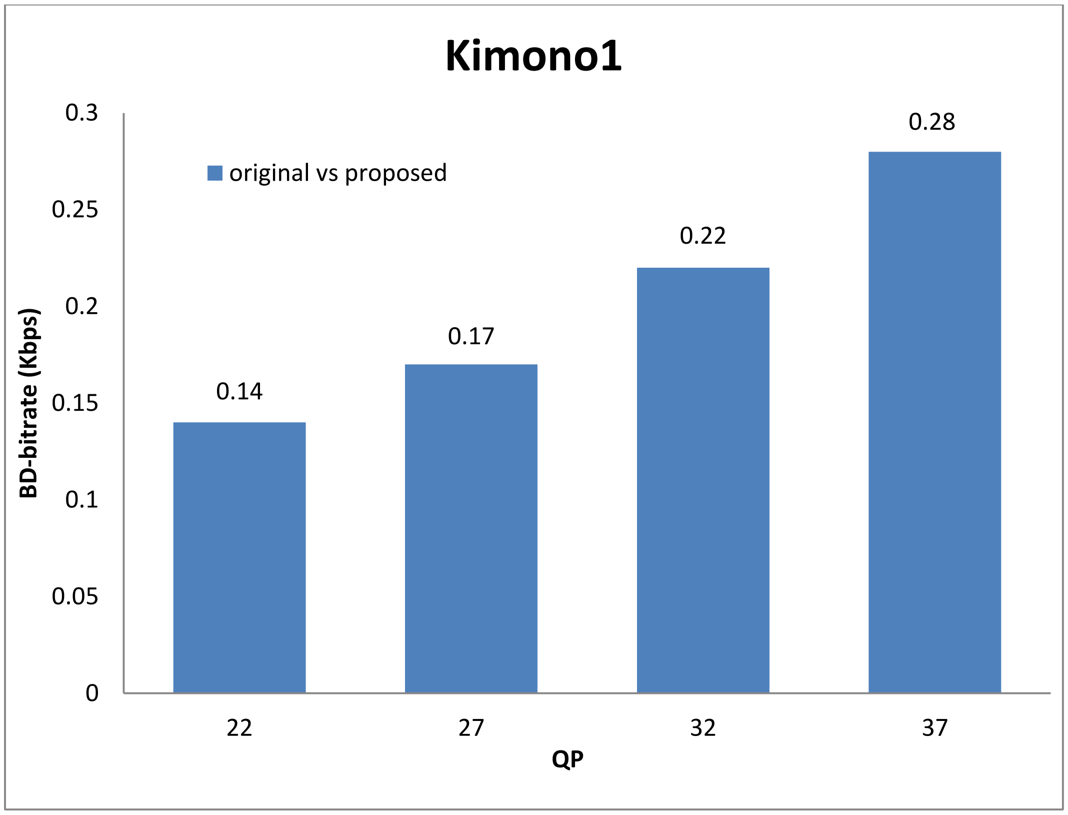


Figure 5-20: BD-bitrate vs QP for Kimono1

5.4.5 Rate Distortion plot (RD plot)

It is used to evaluate the performance of an algorithm based on the variation in PSNR value against bitrate. In the proposed method, there is negligible change in PSNR and bitrate which can be seen in Figures 5-21 through Figure 5-25.

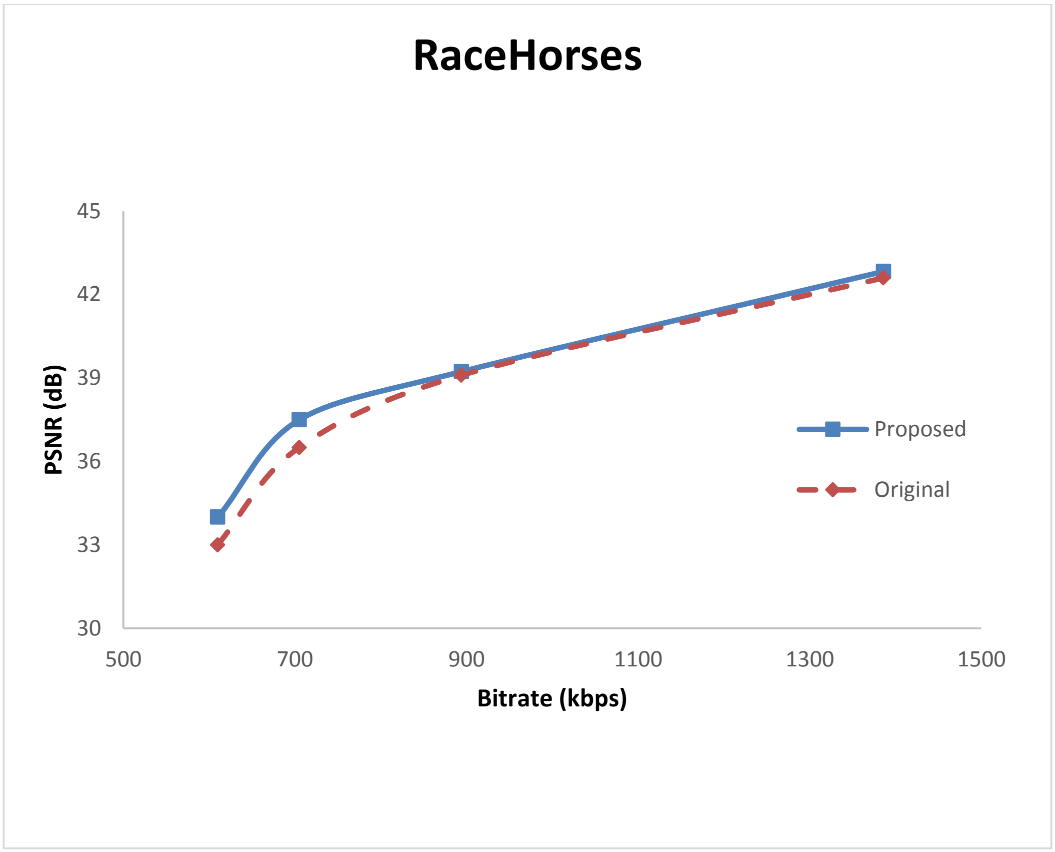


Figure 5-21: PSNR vs bitrate for RaceHorses

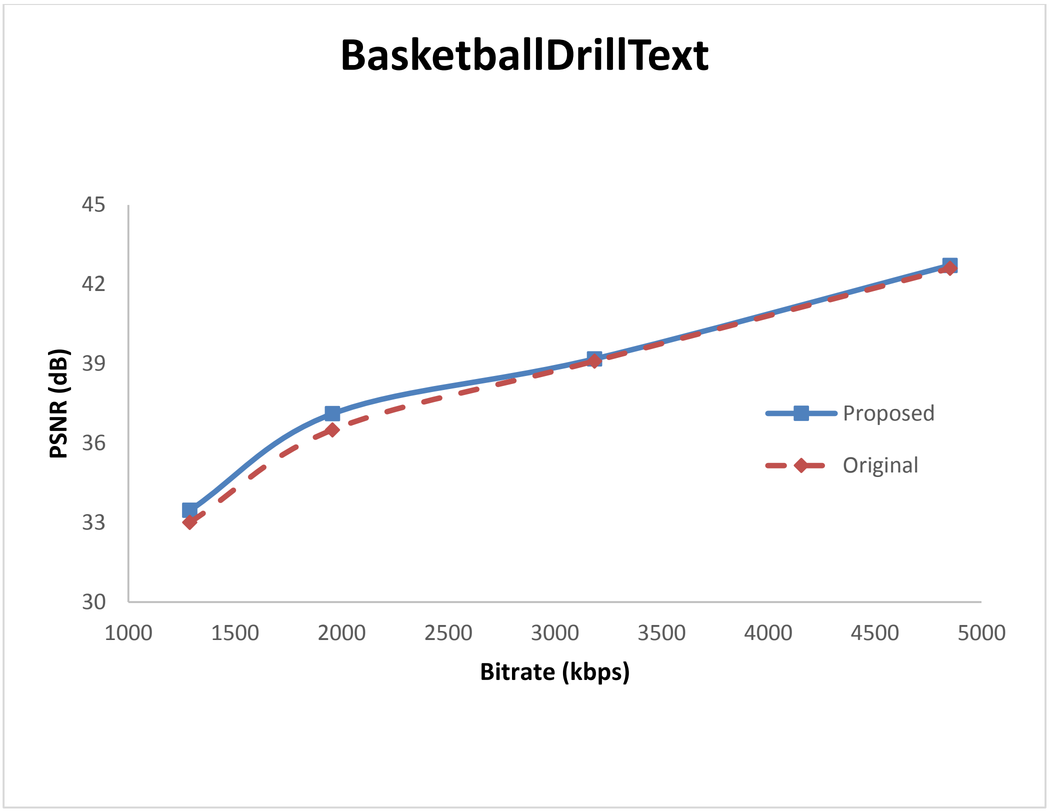


Figure 5-22: PSNR vs bitrate for BasketballDrillText

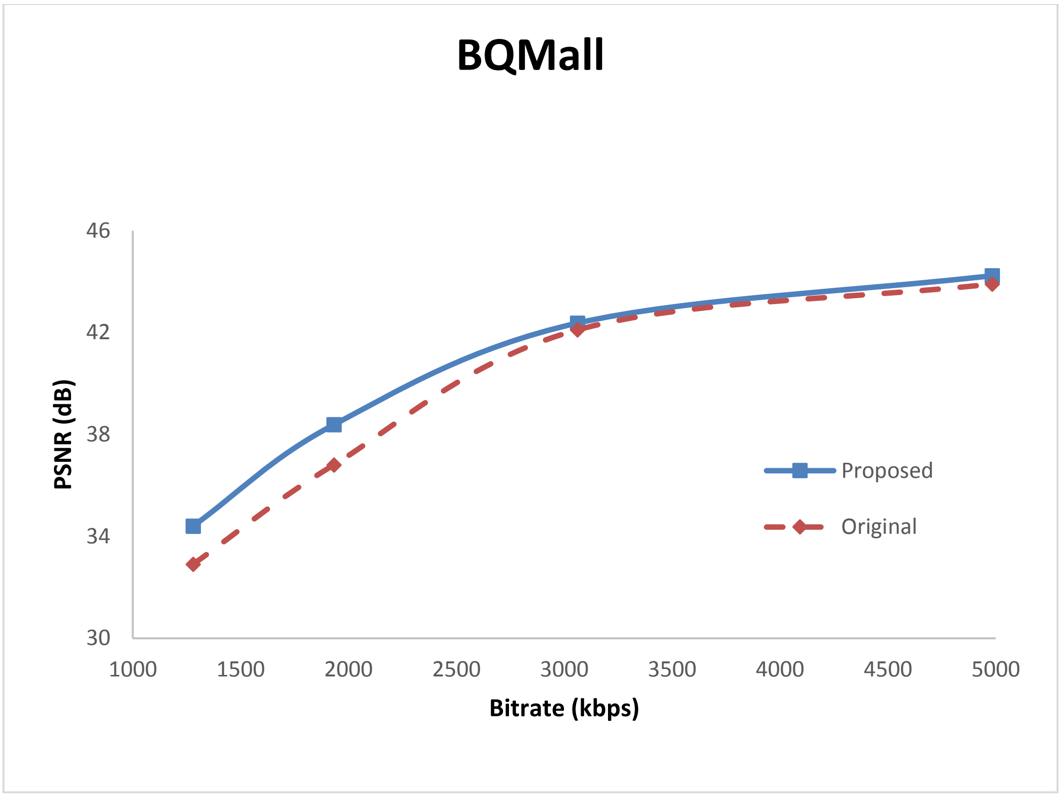


Figure 5-23: PSNR vs bitrate for BQMall

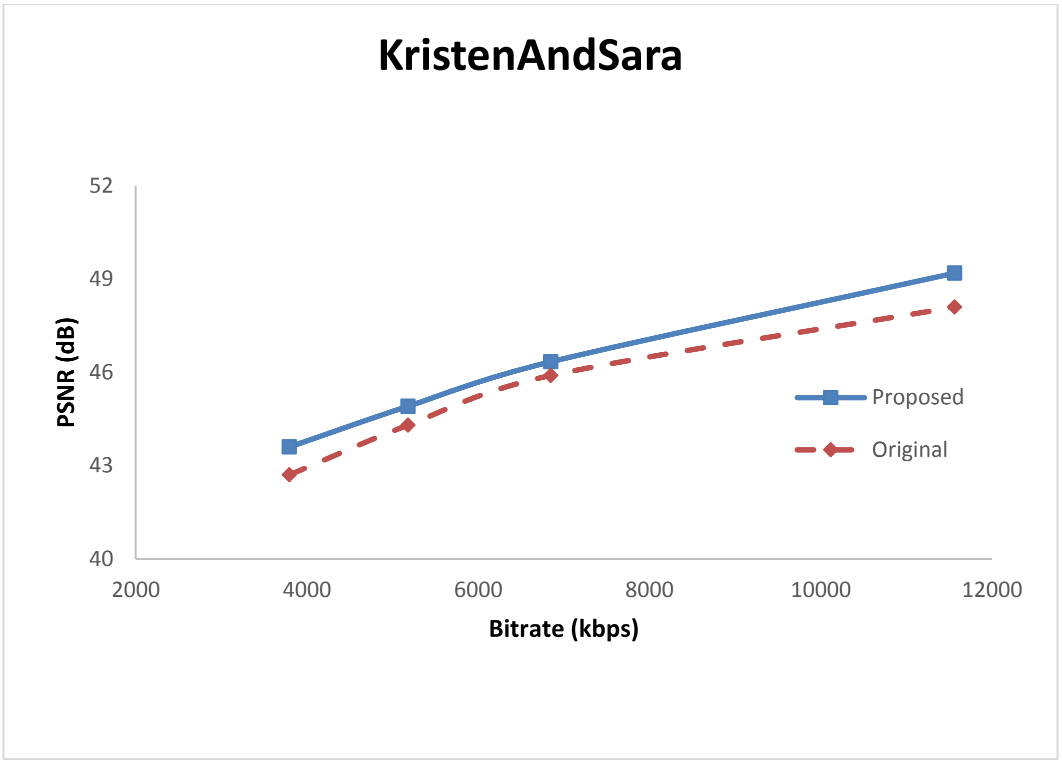


Figure 5-24: PSNR vs bitrate for KristenAndSara

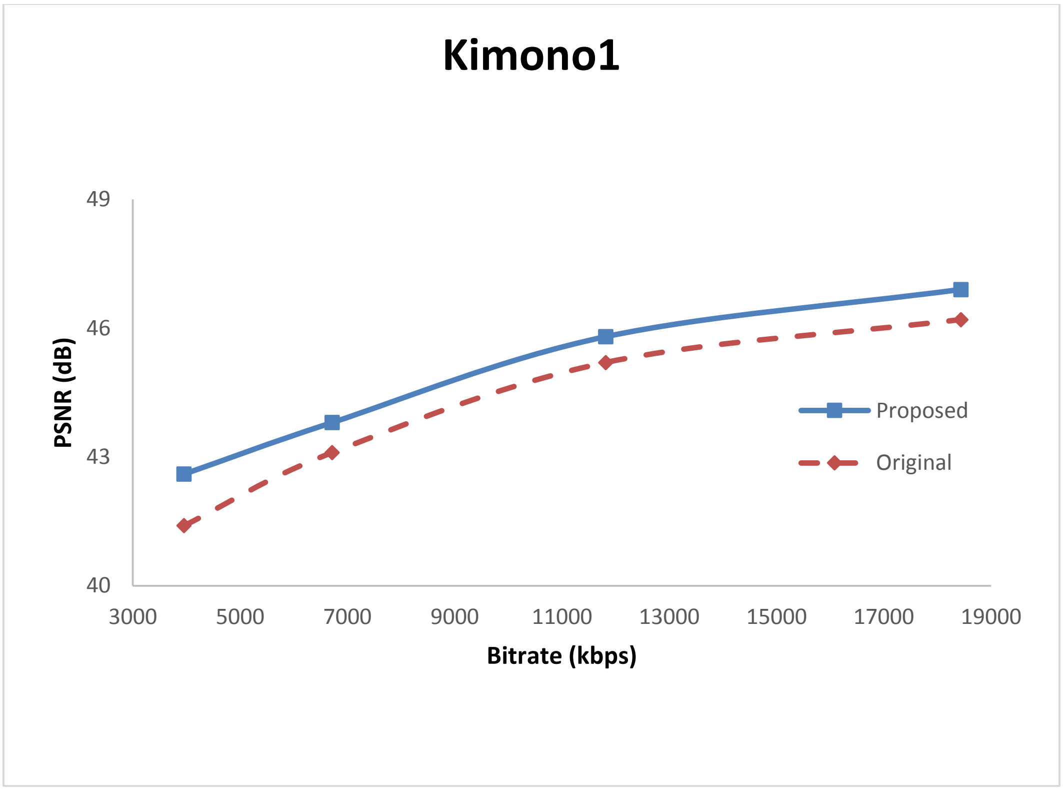


Figure 5-25: PSNR vs bitrate for Kimono1

5.4.6 Summary

In this chapter, numerous results and graphs show the differences observed in various parameters before and after the implementation of parallel programming in HEVC using OpenMP. The results shown in terms of encoding time, percentage decrease in encoding time, BD-PSNR, BD-bitrate and rate distortion plot for the standard test sequences show the improvement in the implemented method.

Chapter 6

Conclusions and Future work

6.1 Conclusions

This thesis gives an introduction to parallel programming and utilizes this concept to optimize motion estimation process for inter prediction in HEVC. Motion estimation in inter prediction has significantly improved performance at the cost of increasing the complexity and processing time. The proposed approach of using parallel programming shows the possible performance improvement in HEVC by reducing the encoding time with negligible reduction in image quality. Based on the results, it can be said that the encoding time can by reduced by approximately 10-25% on an average as compared to HM 13.0 [12] encoder. The image quality drops negligibly for various quantization parameters while using the various test sequences. The proposed technique is also evaluated by means of BD-PSNR, BD-bitrate, bitstream size and rate distortion plot.

6.2 Future Work

There are many different ways to achieve parallel computing. Through software, several APIs are used to obtain parallel processing. OpenMP is just one of the several APIs that are available to apply parallel programming. When more effective parallel programming is to be implemented, dedicated hardware like GPU is used and APIs like OpenCL and OpenCV are used to achieve parallel programming. Several software development kits are available from Intel and AMD which can be used along with these APIs. Also, NVIDIA invented a parallel computing model known as Computer Unified Device Architecture (CUDA) [41] where the overheads and thread creation times are minimized. When it comes to parallel programming using software, POSIX threads (p-threads) and Windows threads APIs are also used in place of OpenMP, which are known to reduce overhead in creation of threads and execute the program faster by pre-allocating the threads.

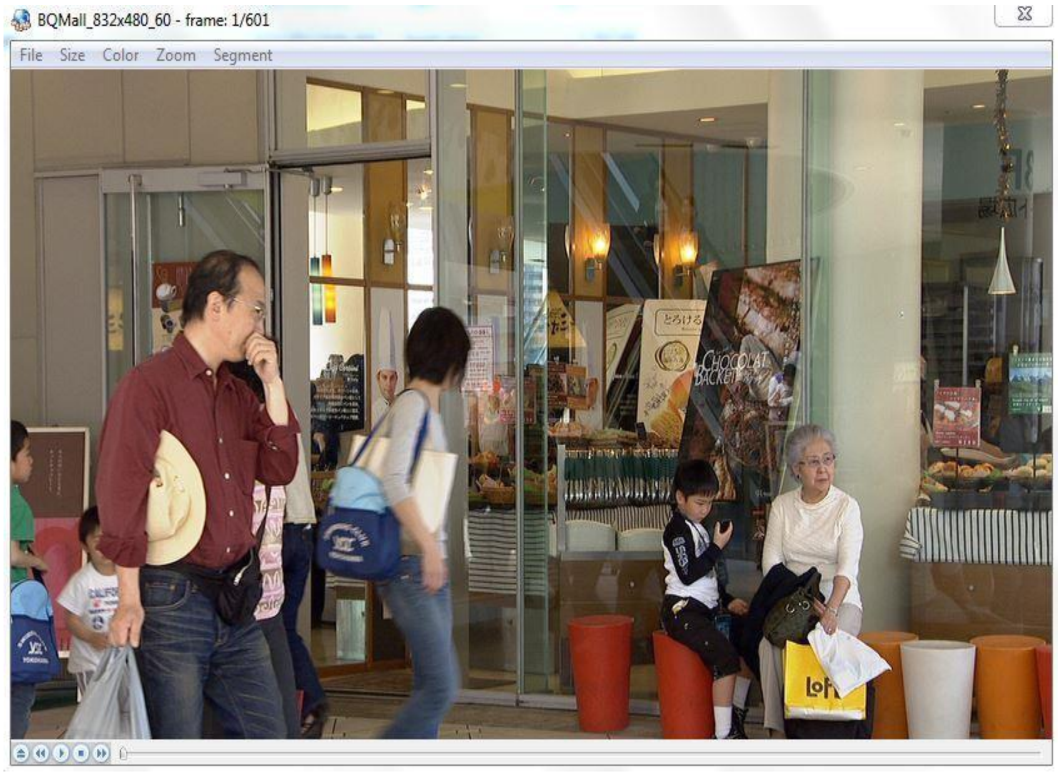
Appendix A

Test Sequences

In order to obtain the results for the proposed method, the following test sequences [16] have been used in this thesis.

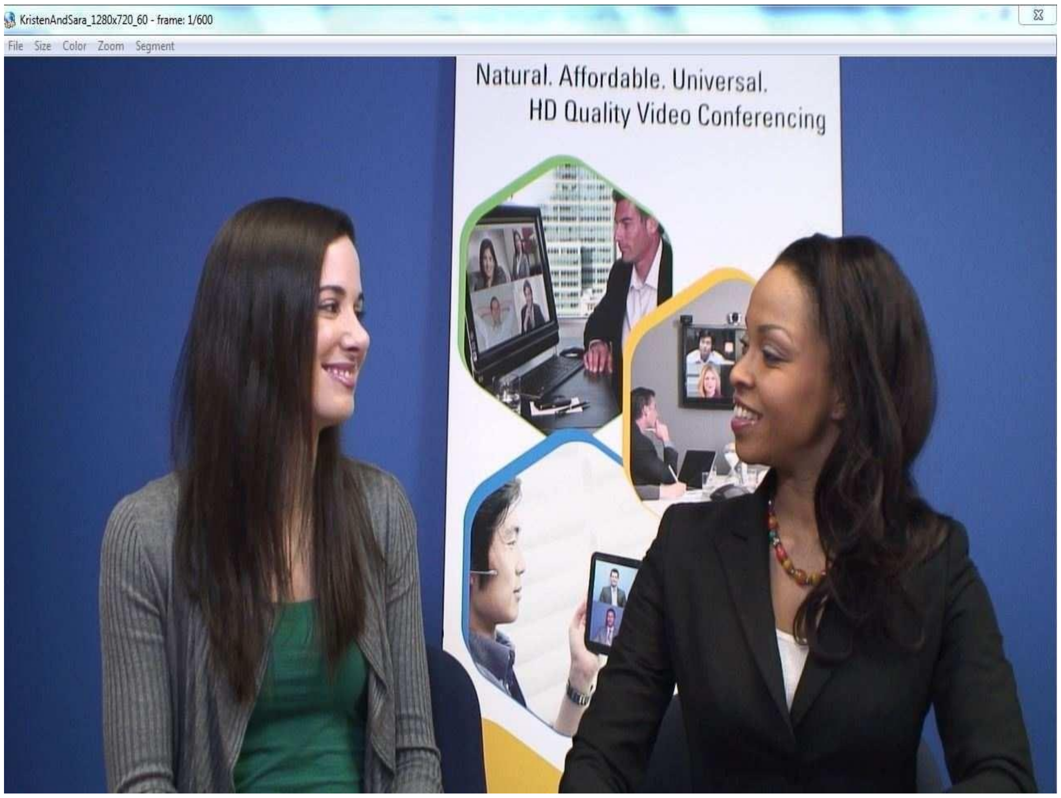
A.1 Basketball Drill Text

A.2 BQ Mall



A.3 Racehorses

A.4 KristenAndSara


KristenAndSara_1280x720_60 - frame: 1/600
File Size Color Zoom Segment
Natural. Affordable. Universal.
HD Quality Video Conferencing

A.5 Kimono1

Appendix B

Test conditions

The reference software used for the proposed method was HM 13.0 [11]. This thesis was implemented on a computer with Intel Core i7 processor running at 1.9 GHz and 8 GB memory and a 64 bit Windows 7 OS.

Appendix C

BD-PSNR and BD-bitrate

BD-PSNR (Bjontegaard – PSNR) and BD-bit rate (Bjontegaard – bit rate) metrics are used to compute the average gain in PSNR and the average per cent saving in bit rate between two rate-distortion graphs respectively and is an ITU-T approved metric [24]. This method was developed by Bjontegaard and is used to gauge compression algorithms from a visual aspect in media industry and referenced by many multimedia engineers. The MATLAB code is available online [25].

```
function avg_diff = bjontegaard(R1,PSNR1,R2,PSNR2,mode)
%BJONTEGAARD    Bjontegaard metric calculation
%   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
% 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_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
```

Appendix D

Acronyms

API: Application Programming Interface

AVC: Advanced Video Coding

BD: Bjontegaard Delta

CABAC: Context Adaptive Binary Arithmetic Coding

CB: Coding Block

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

FPGA: Field Programmable Gate Array

GOP: Group Of Pictures

GPU: Graphic Processing Unit

HEVC: High Efficiency Video Coding

ICIP: International Conference on Image Processing

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

MPL: Multimedia Processing Lab

NGVC: Next Generation Video Coding

OPENMP: Open Multiprocessing

PB: Prediction Block

POC: Phase-Only Correlation

PCM: Pulse Code Modulation

PU: Prediction Unit

SAO: Sample Adaptive Offset

SIMD: Single Instruction Multiple Data

TB: Transform Block

VCEG: Video Coding Experts Group

VCIP: Visual Communication and Image Processing

Appendix E

Code changes

```
/** \file     TEncSearch.cpp
 \brief    encoder search class

+ */


#include "TLibCommon/TypeDef.h"

#include "TLibCommon/TComRom.h"

#include "TLibCommon/TComMotionInfo.h"

#include "TEncSearch.h"

#include <math.h>

#include <omp.h>




//! \ingroup TLibEncoder

//! \{


int threadID;


static const TComMv s_acMvRefineH[9] =

{

  TComMv(  0,  0 ), // 0

  TComMv(  0, -1 ), // 1

  TComMv(  0,  1 ), // 2

  TComMv( -1,  0 ), // 3

  TComMv(  1,  0 ), // 4

  TComMv( -1, -1 ), // 5
  TComMv(  1, -1 ), // 6
  TComMv( -1,  1 ), // 7
  TComMv(  1,  1 )  // 8
};

static const TComMv s_acMvRefineQ[9] =
{
  TComMv(  0,  0 ), // 0
  TComMv(  0, -1 ), // 1
  TComMv(  0,  1 ), // 2
  TComMv( -1, -1 ), // 5
  TComMv(  1, -1 ), // 6
  TComMv( -1,  0 ), // 3
  TComMv(  1,  0 ), // 4
  TComMv( -1,  1 ), // 7
  TComMv(  1,  1 )  // 8
};

static const UInt s_auiDFilter[9] =
{
  0, 1, 0,
  2, 3, 2,
  0, 1, 0
};
```

```
TEncSearch::TEncSearch()
{
  m_ppcQTTempCoeffY  = NULL;
  m_ppcQTTempCoeffCb = NULL;
  m_ppcQTTempCoeffCr = NULL;
  m_pcQTTempCoeffY   = NULL;
  m_pcQTTempCoeffCb  = NULL;
  m_pcQTTempCoeffCr  = NULL;
#if ADAPTIVE_QP_SELECTION
  m_ppcQTTempArlCoeffY  = NULL;
  m_ppcQTTempArlCoeffCb = NULL;
  m_ppcQTTempArlCoeffCr = NULL;
  m_pcQTTempArlCoeffY   = NULL;
  m_pcQTTempArlCoeffCb  = NULL;
  m_pcQTTempArlCoeffCr  = NULL;
#endif
  m_puhQTTempTrIdx   = NULL;
  m_puhQTTempCbf[0] = m_puhQTTempCbf[1] = m_puhQTTempCbf[2] = NULL;
  m_pcQTTempTComYuv  = NULL;
  m_pcEncCfg = NULL;
  m_pcEntropyCoder = NULL;
  m_pTempPel = NULL;
  m_pSharedPredTransformSkip[0] = m_pSharedPredTransformSkip[1] =
m_pSharedPredTransformSkip[2] = NULL;
  m_pcQTTempTUCoeffY   = NULL;
  m_pcQTTempTUCoeffCb  = NULL;
  m_pcQTTempTUCoeffCr  = NULL;
#if ADAPTIVE_QP_SELECTION
  m_ppcQTTempTUArlCoeffY  = NULL;
  m_ppcQTTempTUArlCoeffCb = NULL;
  m_ppcQTTempTUArlCoeffCr = NULL;
#endif
  m_puhQTTempTransformSkipFlag[0] = NULL;
  m_puhQTTempTransformSkipFlag[1] = NULL;
  m_puhQTTempTransformSkipFlag[2] = NULL;
  setWpScalingDistParam( NULL, -1, REF_PIC_LIST_X );
}

TEncSearch::~TEncSearch()
{
  if ( m_pTempPel )
  {
    delete [] m_pTempPel;
    m_pTempPel = NULL;
  }

  if ( m_pcEncCfg )
  {
    const UInt uiNumLayersAllocated = m_pcEncCfg->getQuadtreeTULog2MaxSize()-
m_pcEncCfg->getQuadtreeTULog2MinSize()+1;
```

```
    for( UInt ui = 0; ui < uiNumLayersAllocated; ++ui )
    {
      delete[] m_ppcQTTempCoeffY[ui];
      delete[] m_ppcQTTempCoeffCb[ui];
      delete[] m_ppcQTTempCoeffCr[ui];
#if ADAPTIVE_QP_SELECTION
      delete[] m_ppcQTTempArlCoeffY[ui];
      delete[] m_ppcQTTempArlCoeffCb[ui];
      delete[] m_ppcQTTempArlCoeffCr[ui];
#endif
      m_pcQTTempTComYuv[ui].destroy();
    }
  }
  delete[] m_ppcQTTempCoeffY;
  delete[] m_ppcQTTempCoeffCb;
  delete[] m_ppcQTTempCoeffCr;
  delete[] m_pcQTTempCoeffY;
  delete[] m_pcQTTempCoeffCb;
  delete[] m_pcQTTempCoeffCr;
#if ADAPTIVE_QP_SELECTION
  delete[] m_ppcQTTempArlCoeffY;
  delete[] m_ppcQTTempArlCoeffCb;
  delete[] m_ppcQTTempArlCoeffCr;
  delete[] m_pcQTTempArlCoeffY;
  delete[] m_pcQTTempArlCoeffCb;
  delete[] m_pcQTTempArlCoeffCr;
#endif
  delete[] m_puhQTTempTrIdx;
  delete[] m_puhQTTempCbf[0];
  delete[] m_puhQTTempCbf[1];
  delete[] m_puhQTTempCbf[2];
  delete[] m_pcQTTempTComYuv;
  delete[] m_pSharedPredTransformSkip[0];
  delete[] m_pSharedPredTransformSkip[1];
  delete[] m_pSharedPredTransformSkip[2];
  delete[] m_pcQTTempTUCoeffY;
  delete[] m_pcQTTempTUCoeffCb;
  delete[] m_pcQTTempTUCoeffCr;
#if ADAPTIVE_QP_SELECTION
  delete[] m_ppcQTTempTUArlCoeffY;
  delete[] m_ppcQTTempTUArlCoeffCb;
  delete[] m_ppcQTTempTUArlCoeffCr;
#endif
  delete[] m_puhQTTempTransformSkipFlag[0];
  delete[] m_puhQTTempTransformSkipFlag[1];
  delete[] m_puhQTTempTransformSkipFlag[2];
  m_pcQTTempTransformSkipTComYuv.destroy();
  m_tmpYuvPred.destroy();
}
```

```
void TEncSearch::init(TEncCfg*      pcEncCfg,
                      TComTrQuant*  pcTrQuant,
                      Int           iSearchRange,
                      Int           bipredSearchRange,
                      Int           iFastSearch,
                      Int           iMaxDeltaQP,
                      TEncEntropy*  pcEntropyCoder,
                      TComRdCost*   pcRdCost,
                      TEncSbac*** pppcRDSbacCoder,
                      TEncSbac*   pcRDGoOnSbacCoder
                      )
{
  m_pcEncCfg            = pcEncCfg;
  m_pcTrQuant           = pcTrQuant;
  m_iSearchRange        = iSearchRange;
  m_bipredSearchRange    = bipredSearchRange;
  m_iFastSearch         = iFastSearch;
  m_iMaxDeltaQP         = iMaxDeltaQP;
  m_pcEntropyCoder      = pcEntropyCoder;
  m_pcRdCost            = pcRdCost;

  m_pppcRDSbacCoder     = pppcRDSbacCoder;
  m_pcRDGoOnSbacCoder   = pcRDGoOnSbacCoder;

  for (Int iDir = 0; iDir < 2; iDir++)
  {
    for (Int iRefIdx = 0; iRefIdx < 33; iRefIdx++)
    {
      m_aaiAdaptSR[iDir][iRefIdx] = iSearchRange;
    }
  }

  m_puiDFilter = s_auiDFilter + 4;

  // initialize motion cost
#if !FIX203
  m_pcRdCost->initRateDistortionModel( m_iSearchRange << 2 );
#endif

  for( Int iNum = 0; iNum < AMVP_MAX_NUM_CANDS+1; iNum++)
  {
    for( Int iIdx = 0; iIdx < AMVP_MAX_NUM_CANDS; iIdx++)
    {
      if (iIdx < iNum)
        m_auiMVPIdxCost[iIdx][iNum] = xGetMvpIdxBits(iIdx, iNum);
      else
        m_auiMVPIdxCost[iIdx][iNum] = MAX_INT;
    }
  }
  initTempBuff();

  m_pTempPel = new Pel[g_uiMaxCUWidth*g_uiMaxCUHeight];

  const UInt uiNumLayersToAllocate = pcEncCfg->getQuadtreeTULog2MaxSize()-
pcEncCfg->getQuadtreeTULog2MinSize()+1;
  m_ppcQTTempCoeffY  = new TCoeff*[uiNumLayersToAllocate];
  m_ppcQTTempCoeffCb = new TCoeff*[uiNumLayersToAllocate];
  m_ppcQTTempCoeffCr = new TCoeff*[uiNumLayersToAllocate];
  m_pcQTTempCoeffY   = new TCoeff [g_uiMaxCUWidth*g_uiMaxCUHeight   ];
  m_pcQTTempCoeffCb  = new TCoeff [g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
  m_pcQTTempCoeffCr  = new TCoeff [g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
#if ADAPTIVE_QP_SELECTION
  m_ppcQTTempArlCoeffY  = new Int*[uiNumLayersToAllocate];
  m_ppcQTTempArlCoeffCb = new Int*[uiNumLayersToAllocate];
  m_ppcQTTempArlCoeffCr = new Int*[uiNumLayersToAllocate];
  m_pcQTTempArlCoeffY   = new Int [g_uiMaxCUWidth*g_uiMaxCUHeight   ];
  m_pcQTTempArlCoeffCb  = new Int [g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
  m_pcQTTempArlCoeffCr  = new Int [g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
#endif

  const UInt uiNumPartitions = 1<<(g_uiMaxCUDepth<<1);
  m_puhQTTempTrIdx   = new UChar  [uiNumPartitions];
  m_puhQTTempCbf[0]  = new UChar  [uiNumPartitions];
  m_puhQTTempCbf[1]  = new UChar  [uiNumPartitions];
  m_puhQTTempCbf[2]  = new UChar  [uiNumPartitions];
  m_pcQTTempTComYuv  = new TComYuv[uiNumLayersToAllocate];
  for( UInt ui = 0; ui < uiNumLayersToAllocate; ++ui )
  {
    m_ppcQTTempCoeffY[ui]  = new TCoeff[g_uiMaxCUWidth*g_uiMaxCUHeight   ];
    m_ppcQTTempCoeffCb[ui] = new TCoeff[g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
    m_ppcQTTempCoeffCr[ui] = new TCoeff[g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
#if ADAPTIVE_QP_SELECTION
    m_ppcQTTempArlCoeffY[ui]  = new Int[g_uiMaxCUWidth*g_uiMaxCUHeight   ];
    m_ppcQTTempArlCoeffCb[ui] = new Int[g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
    m_ppcQTTempArlCoeffCr[ui] = new Int[g_uiMaxCUWidth*g_uiMaxCUHeight>>2];
#endif
    m_pcQTTempTComYuv[ui].create( g_uiMaxCUWidth, g_uiMaxCUHeight );
  }
  m_pSharedPredTransformSkip[0] = new Pel[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_pSharedPredTransformSkip[1] = new Pel[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_pSharedPredTransformSkip[2] = new Pel[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_pcQTTempTUCoeffY  = new TCoeff[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_pcQTTempTUCoeffCb = new TCoeff[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_pcQTTempTUCoeffCr = new TCoeff[MAX_TS_WIDTH*MAX_TS_HEIGHT];
#if ADAPTIVE_QP_SELECTION
  m_ppcQTTempTUArlCoeffY  = new Int[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_ppcQTTempTUArlCoeffCb = new Int[MAX_TS_WIDTH*MAX_TS_HEIGHT];
  m_ppcQTTempTUArlCoeffCr = new Int[MAX_TS_WIDTH*MAX_TS_HEIGHT];
#endif
```

```
  m_pcQTTempTransformSkipTComYuv.create( g_uiMaxCUWidth, g_uiMaxCUHeight );

  m_puhQTTempTransformSkipFlag[0] = new UChar  [uiNumPartitions];
  m_puhQTTempTransformSkipFlag[1] = new UChar  [uiNumPartitions];
  m_puhQTTempTransformSkipFlag[2] = new UChar  [uiNumPartitions];
  m_tmpYuvPred.create(MAX_CU_SIZE, MAX_CU_SIZE);
}

#if FASTME_SMOOTHER_MV
#define FIRSTSEARCHSTOP     1
#else
#define FIRSTSEARCHSTOP     0
#endif

#define TZ_SEARCH_CONFIGURATION
\
const Int  iRaster                  = 5;  /* TZ soll von aussen ?ergeben werden */
\
const Bool bTestOtherPredictedMV    = 0;                                              \
const Bool bTestZeroVector          = 1;                                              \
const Bool bTestZeroVectorStart     = 0;                                              \
const Bool bTestZeroVectorStop      = 0;                                              \
const Bool bFirstSearchDiamond      = 1;  /* 1 = xTZ8PointDiamondSearch   0 =
xTZ8PointSquareSearch */        \
const Bool bFirstSearchStop         = FIRSTSEARCHSTOP;
\
const UInt uiFirstSearchRounds      = 3;  /* first search stop X rounds after best match
(must be >=1) */     \
const Bool bEnableRasterSearch      = 1;                                             \
const Bool bAlwaysRasterSearch      = 0;  /* ===== 1: BETTER but factor 2 slower =====
*/                    \
const Bool bRasterRefinementEnable  = 0;  /* enable either raster refinement or star
refinement */            \
const Bool bRasterRefinementDiamond = 1;  /* 1 = xTZ8PointDiamondSearch   0 =
xTZ8PointSquareSearch */        \
const Bool bStarRefinementEnable    = 1;  /* enable either star refinement or raster
refinement */            \
const Bool bStarRefinementDiamond   = 1;  /* 1 = xTZ8PointDiamondSearch   0 =
xTZ8PointSquareSearch */        \
const Bool bStarRefinementStop      = 0;                                             \
const UInt uiStarRefinementRounds   = 2;  /* star refinement stop X rounds after best
match (must be >=1) */  \


__inline Void TEncSearch::xTZSearchHelp( TComPattern* pcPatternKey,
IntTZSearchStruct& rcStruct, const Int iSearchX, const Int iSearchY, const UChar
ucPointNr, const UInt uiDistance )
{
  UInt  uiSad;
  Pel*  piRefSrch;

  piRefSrch = rcStruct.piRefY + iSearchY * rcStruct.iYStride + iSearchX;

  //-- jclee for using the SAD function pointer
  m_pcRdCost->setDistParam( pcPatternKey, piRefSrch, rcStruct.iYStride,
m_cDistParam );

  // fast encoder decision: use subsampled SAD when rows > 8 for integer ME
  if ( m_pcEncCfg->getUseFastEnc() )
  {
    if ( m_cDistParam.iRows > 8 )
    {
      m_cDistParam.iSubShift = 1;
    }
  }

  setDistParamComp(0);  // Y component

  // distortion
  m_cDistParam.bitDepth = g_bitDepthY;
  uiSad = m_cDistParam.DistFunc( &m_cDistParam );

  // motion cost
  uiSad += m_pcRdCost->getCost( iSearchX, iSearchY );

  if( uiSad < rcStruct.uiBestSad )
  {
    rcStruct.uiBestSad      = uiSad;
    rcStruct.iBestX         = iSearchX;
    rcStruct.iBestY         = iSearchY;
    rcStruct.uiBestDistance = uiDistance;
    rcStruct.uiBestRound    = 0;
    rcStruct.ucPointNr      = ucPointNr;
  }
}

__inline Void TEncSearch::xTZ2PointSearch( TComPattern* pcPatternKey,
IntTZSearchStruct& rcStruct, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB )
{
  Int   iSrchRngHorLeft   = pcMvSrchRngLT->getHor();
  Int   iSrchRngHorRight  = pcMvSrchRngRB->getHor();
  Int   iSrchRngVerTop    = pcMvSrchRngLT->getVer();
  Int   iSrchRngVerBottom = pcMvSrchRngRB->getVer();

  // 2 point search,                   //   1 2 3
  // check only the 2 untested points  //   4 0 5
  // around the start point            //   6 7 8
  Int iStartX = rcStruct.iBestX;
  Int iStartY = rcStruct.iBestY;
  switch( rcStruct.ucPointNr )
  {
    case 1:
    {
      if ( (iStartX - 1) >= iSrchRngHorLeft )
      {
        xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY, 0, 2 );
      }
      if ( (iStartY - 1) >= iSrchRngVerTop )
      {
        xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iStartY - 1, 0, 2 );
      }
    }
      break;
    case 2:
    {
      if ( (iStartY - 1) >= iSrchRngVerTop )
      {
        if ( (iStartX - 1) >= iSrchRngHorLeft )
        {
          xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY - 1, 0, 2 );
        }
        if ( (iStartX + 1) <= iSrchRngHorRight )
        {
```

```
      xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY - 1, 0, 2 );
    }
  }
}
  break;
case 3:
{
  if ( (iStartY - 1) >= iSrchRngVerTop )
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iStartY - 1, 0, 2 );
  }
  if ( (iStartX + 1) <= iSrchRngHorRight )
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY, 0, 2 );
  }
}
  break;
case 4:
{
  if ( (iStartX - 1) >= iSrchRngHorLeft )
  {
    if ( (iStartY + 1) <= iSrchRngVerBottom )
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY + 1, 0, 2 );
    }
    if ( (iStartY - 1) >= iSrchRngVerTop )
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY - 1, 0, 2 );
    }
  }
}
  break;
case 5:
{
  if ( (iStartX + 1) <= iSrchRngHorRight )
  {
    if ( (iStartY - 1) >= iSrchRngVerTop )
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY - 1, 0, 2 );
    }
    if ( (iStartY + 1) <= iSrchRngVerBottom )
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY + 1, 0, 2 );
    }
  }
}
  break;
case 6:
{
  if ( (iStartX - 1) >= iSrchRngHorLeft )
```

```
 {
   xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY , 0, 2 );
 }
 if ( (iStartY + 1) <= iSrchRngVerBottom )
 {
   xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iStartY + 1, 0, 2 );
 }
}
 break;
case 7:
{
 if ( (iStartY + 1) <= iSrchRngVerBottom )
 {
   if ( (iStartX - 1) >= iSrchRngHorLeft )
   {
     xTZSearchHelp( pcPatternKey, rcStruct, iStartX - 1, iStartY + 1, 0, 2 );
   }
   if ( (iStartX + 1) <= iSrchRngHorRight )
   {
     xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY + 1, 0, 2 );
   }
 }
}
 break;
case 8:
    {
      if ( (iStartX + 1) <= iSrchRngHorRight )
      {
        xTZSearchHelp( pcPatternKey, rcStruct, iStartX + 1, iStartY, 0, 2 );
      }
      if ( (iStartY + 1) <= iSrchRngVerBottom )
      {
        xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iStartY + 1, 0, 2 );
      }
    }
      break;
    default:
    {
      assert( false );
    }
      break;
  } // switch( rcStruct.ucPointNr )
}

__inline Void TEncSearch::xTZ8PointSquareSearch( TComPattern* pcPatternKey,
IntTZSearchStruct& rcStruct, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB,
const Int iStartX, const Int iStartY, const Int iDist )
{
  Int   iSrchRngHorLeft   = pcMvSrchRngLT->getHor();
  Int   iSrchRngHorRight  = pcMvSrchRngRB->getHor();
Int   iSrchRngVerTop    = pcMvSrchRngLT->getVer();
Int   iSrchRngVerBottom = pcMvSrchRngRB->getVer();

// 8 point search,                    //   1 2 3
// search around the start point     //   4 0 5
// with the required  distance       //   6 7 8
assert( iDist != 0 );
const Int iTop        = iStartY - iDist;
const Int iBottom     = iStartY + iDist;
const Int iLeft       = iStartX - iDist;
const Int iRight      = iStartX + iDist;
rcStruct.uiBestRound += 1;

if ( iTop >= iSrchRngVerTop ) // check top
{
  if ( iLeft >= iSrchRngHorLeft ) // check top left
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iLeft, iTop, 1, iDist );
  }
  // top middle
  xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iTop, 2, iDist );

  if ( iRight <= iSrchRngHorRight ) // check top right
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iRight, iTop, 3, iDist );
    }
  } // check top
  if ( iLeft >= iSrchRngHorLeft ) // check middle left
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iLeft, iStartY, 4, iDist );
  }
  if ( iRight <= iSrchRngHorRight ) // check middle right
  {
    xTZSearchHelp( pcPatternKey, rcStruct, iRight, iStartY, 5, iDist );
  }
  if ( iBottom <= iSrchRngVerBottom ) // check bottom
  {
    if ( iLeft >= iSrchRngHorLeft ) // check bottom left
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iLeft, iBottom, 6, iDist );
    }
    // check bottom middle
    xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iBottom, 7, iDist );

    if ( iRight <= iSrchRngHorRight ) // check bottom right
    {
      xTZSearchHelp( pcPatternKey, rcStruct, iRight, iBottom, 8, iDist );
    }
  } // check bottom
}
```

```
__inline Void TEncSearch::xTZ8PointDiamondSearch( TComPattern* pcPatternKey,
IntTZSearchStruct& rcStruct, TComMv* pcMvSrchRngLT, TComMv* pcMvSrchRngRB,
const Int iStartX, const Int iStartY, const Int iDist )
{
  Int   iSrchRngHorLeft   = pcMvSrchRngLT->getHor();
  Int   iSrchRngHorRight  = pcMvSrchRngRB->getHor();
  Int   iSrchRngVerTop    = pcMvSrchRngLT->getVer();
  Int   iSrchRngVerBottom = pcMvSrchRngRB->getVer();

  // 8 point search,                  //   1 2 3
  // search around the start point     //   4 0 5
  // with the required  distance       //   6 7 8
  assert ( iDist != 0 );
  const Int iTop        = iStartY - iDist;
  const Int iBottom     = iStartY + iDist;
  const Int iLeft       = iStartX - iDist;
  const Int iRight      = iStartX + iDist;
  rcStruct.uiBestRound += 1;




          if ( iDist == 1 ) // iDist == 1
          {
                if ( iTop >= iSrchRngVerTop ) // check top
                {
                        xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iTop, 2, iDist );
                }
                if ( iLeft >= iSrchRngHorLeft ) // check middle left
                {
                        xTZSearchHelp( pcPatternKey, rcStruct, iLeft, iStartY, 4, iDist );
                }
                if ( iRight <= iSrchRngHorRight ) // check middle right
                {
                        xTZSearchHelp( pcPatternKey, rcStruct, iRight, iStartY, 5, iDist
);
                }
                if ( iBottom <= iSrchRngVerBottom ) // check bottom
                {
                        xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iBottom, 7,
iDist );
                }
        }
        else // if (iDist != 1)
        {
                if ( iDist <= 8 )
                {
                        const Int iTop_2      = iStartY - (iDist>>1);
                        const Int iBottom_2   = iStartY + (iDist>>1);
```

```
                    const Int iLeft_2     = iStartX - (iDist>>1);
                    const Int iRight_2    = iStartX + (iDist>>1);

                    if (  iTop >= iSrchRngVerTop && iLeft >= iSrchRngHorLeft &&
                          iRight <= iSrchRngHorRight && iBottom <=
iSrchRngVerBottom ) // check border
                    {
                          xTZSearchHelp( pcPatternKey, rcStruct, iStartX,  iTop,
2, iDist    );
                          xTZSearchHelp( pcPatternKey, rcStruct, iLeft_2,
iTop_2,    1, iDist>>1 );
                          xTZSearchHelp( pcPatternKey, rcStruct, iRight_2,
iTop_2,    3, iDist>>1 );
                          xTZSearchHelp( pcPatternKey, rcStruct, iLeft,
iStartY,   4, iDist    );
                          xTZSearchHelp( pcPatternKey, rcStruct, iRight,
iStartY,   5, iDist    );
                          xTZSearchHelp( pcPatternKey, rcStruct, iLeft_2,
iBottom_2, 6, iDist>>1 );
                          xTZSearchHelp( pcPatternKey, rcStruct, iRight_2,
iBottom_2, 8, iDist>>1 );
                          xTZSearchHelp( pcPatternKey, rcStruct, iStartX,
iBottom,   7, iDist    );
                    }
                    else // check border
```

```
                                        {
                                         // changes made here
                                         #pragma omp sections nowait
                    {
                                          #pragma omp section
                                                 {
                                                   if ( iTop >= iSrchRngVerTop ) // check top
                                                   {
                                                            xTZSearchHelp( pcPatternKey, rcStruct,
iStartX, iTop, 2, iDist );
                                                   }
                                                   if ( iTop_2 >= iSrchRngVerTop ) // check half top
                                                   {
                                                            if ( iLeft_2 >= iSrchRngHorLeft ) // check half
left
                                                            {
                                                                     xTZSearchHelp( pcPatternKey,
rcStruct, iLeft_2, iTop_2, 1, (iDist>>1) );
                                                            }
                                                            if ( iRight_2 <= iSrchRngHorRight ) // check
half right
                                                            {
                                                                     xTZSearchHelp( pcPatternKey,
rcStruct, iRight_2, iTop_2, 3, (iDist>>1) );
                                                            }
```

```
                                                    } // check half top
                                                  }
                                                  #pragma omp section
                                                  {
                                                    if ( iLeft >= iSrchRngHorLeft ) // check left
                                                    {
                                                            xTZSearchHelp( pcPatternKey, rcStruct, iLeft,
iStartY, 4, iDist );
                                                    }
                                                    if ( iRight <= iSrchRngHorRight ) // check right
                                                    {
                                                            xTZSearchHelp( pcPatternKey, rcStruct, iRight,
iStartY, 5, iDist );
                                                    }
                                                    if ( iBottom_2 <= iSrchRngVerBottom ) // check half
bottom
                                                    {
                                                            if ( iLeft_2 >= iSrchRngHorLeft ) // check half
left
                                                            {
                                                                    xTZSearchHelp( pcPatternKey,
rcStruct, iLeft_2, iBottom_2, 6, (iDist>>1) );
                                                            }
                                                            if ( iRight_2 <= iSrchRngHorRight ) // check
half right
```

```
                                                   {
                                                              xTZSearchHelp( pcPatternKey,
rcStruct, iRight_2, iBottom_2, 8, (iDist>>1) );
                                                   }
                                             } // check half bottom
                                            }

                                             #pragma omp section
        {
                                         if ( iBottom <= iSrchRngVerBottom ) // check bottom
                                             {
                                                   xTZSearchHelp( pcPatternKey, rcStruct,
iStartX, iBottom, 7, iDist );
                                             }
                                            }
                                            } // check border
                                       }
                                 }
                                 else // iDist > 8
                                 {
                                       if ( iTop >= iSrchRngVerTop && iLeft >= iSrchRngHorLeft &&
                                             iRight <= iSrchRngHorRight && iBottom <=
iSrchRngVerBottom ) // check border
                                       {
                                             xTZSearchHelp( pcPatternKey, rcStruct, iStartX, iTop,
0, iDist );
                                             xTZSearchHelp( pcPatternKey, rcStruct, iLeft,   iStartY,
0, iDist );
                                             xTZSearchHelp( pcPatternKey, rcStruct, iRight,
iStartY, 0, iDist );
                                             xTZSearchHelp( pcPatternKey, rcStruct, iStartX,
iBottom, 0, iDist );
                                             for ( Int index = 1; index < 4; index++ )
                                             {
                                                   Int iPosYT = iTop    + ((iDist>>2) * index);
                                                   Int iPosYB = iBottom - ((iDist>>2) * index);
                                                   Int iPosXL = iStartX - ((iDist>>2) * index);
                                                   Int iPosXR = iStartX + ((iDist>>2) * index);
                                                   xTZSearchHelp( pcPatternKey, rcStruct,
iPosXL, iPosYT, 0, iDist );
                                                   xTZSearchHelp( pcPatternKey, rcStruct,
iPosXR, iPosYT, 0, iDist );
                                                   xTZSearchHelp( pcPatternKey, rcStruct,
iPosXL, iPosYB, 0, iDist );
                                                   xTZSearchHelp( pcPatternKey, rcStruct,
iPosXR, iPosYB, 0, iDist );
                                             }
                                       }
                                       else // check border
```

```
                    {
            #pragma omp sections nowait
                        {
                         #pragma omp section
                         { if ( iTop >= iSrchRngVerTop ) // check top
                          {
                                  xTZSearchHelp( pcPatternKey, rcStruct,
iStartX, iTop, 0, iDist );
                          }
                         }
                         #pragma omp section
                         {
                          if ( iLeft >= iSrchRngHorLeft ) // check left
                          {
                                  xTZSearchHelp( pcPatternKey, rcStruct, iLeft,
iStartY, 0, iDist );
                          }
                         }
                         #pragma omp section
                         {
                          if ( iRight <= iSrchRngHorRight ) // check right
                          {
                                  xTZSearchHelp( pcPatternKey, rcStruct, iRight,
iStartY, 0, iDist );
                          }
```

```
                                        }
                                        #pragma omp section
                                        {
                                         if ( iBottom <= iSrchRngVerBottom ) // check bottom
                                         {
                                                xTZSearchHelp( pcPatternKey, rcStruct,
iStartX, iBottom, 0, iDist );
                                         }
        }
                                        }
                                       //changes made here
        #pragma omp parallel for

                                                for ( Int index = 1; index < 4; index++ )
                                                {
                                                        Int iPosYT = iTop    + ((iDist>>2) *
index);
                                                        Int iPosYB = iBottom - ((iDist>>2) *
index);
                                                        Int iPosXL = iStartX - ((iDist>>2) *
index);
                                                        Int iPosXR = iStartX + ((iDist>>2) *
index);
          #pragma omp sections nowait
                                                 {
```

```
                #pragma omp section
                                {
                                     if ( iPosYT >= iSrchRngVerTop ) //
check top
                                     {
                                          if ( iPosXL >= iSrchRngHorLeft
) // check left
                                          {
                                               xTZSearchHelp(
pcPatternKey, rcStruct, iPosXL, iPosYT, 0, iDist );
                                          }
                                          if ( iPosXR <=
iSrchRngHorRight ) // check right
                                          {
                                               xTZSearchHelp(
pcPatternKey, rcStruct, iPosXR, iPosYT, 0, iDist );
                                          }
                                     } // check top
                                    }


                #pragma omp section
                                {
                                    if ( iPosYB <= iSrchRngVerBottom ) //
check bottom
                                     {
```

```
                                                        if ( iPosXL >= iSrchRngHorLeft
) // check left
                                                        {
                                                                xTZSearchHelp(
pcPatternKey, rcStruct, iPosXL, iPosYB, 0, iDist );
                                                        }
                                                        if ( iPosXR <=
iSrchRngHorRight ) // check right
                                                        {
                                                                xTZSearchHelp(
pcPatternKey, rcStruct, iPosXR, iPosYB, 0, iDist );
                                                        }
                                                } // check bottom
                                            }
                                           }
                                          } // for ...

                                } // check border
                        } // iDist <= 8
                } // iDist == 1
 }
```

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, 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, March 2013.

[11] K.R.Rao, D.N.Kim and J.J.Hwang, "Video Coding Standards: AVS China, H.264/MPEG-4 Part10, HEVC, VP6, DIRAC and VC-1", Springer, 2014.

[12] HEVC open source software (encoder/decoder) https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-13.0

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

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

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

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

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

[18] H.265 standard finalized http://www.extremetech.com/extreme/147000-h-265-standard-finalized-could-finally-replace-mpeg-2-and-usher-in-uhdtv

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

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

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

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

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

[24] 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”, IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560-576, July 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] Il-Koo Kim et al, "HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description", JCTVC-K1002-vl, 11th Meeting: Shanghai, CN, 10-19 October, 2012.

[30] "Introduction to the issue on video coding: HDTV and beyond IEEE journal of selected topics in Signal Processing", vol 7, Dec 2013.

[31] G.J.Sullivan, et al, "HEVC Range Extensions Draft 5", JCT-VC, version 1, Geneva, Nov. 2013.

[32] Official website for information on OpenMP

http://openmp.org/

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

[34] J.Rehman and 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

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

[36] X. Li et al, "Rate-Complexity-Distortion evaluation for hybrid video coding", IEEE Transactions on CSVT, vol. 21, pp. 957- 970, July 2011.

[37] V.Sze, M.Budagavi and G.J. Sullivan, "High Efficiency Video Coding (HEVC) - Algorithms and Architectures", Springer, 2014.

[38] S.N.Agathos, P. Hadjidoukas and V. Dimakopoulos,"Task based execution of Nested OpenMP loops", Springer, 2012.

[39] R.Chandra et al, "Parallel programming in OpenMP", Academic Press, 2001.

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

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

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

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

[44] Introduction to parallel programming https://computing.llnl.gov/tutorials/parallel_comp/

[45] Introduction to parallel programming and MapReduce

https://courses.cs.washington.edu/courses/cse490h/07wi/readings/IntroductionToParallel

ProgrammingAndMapReduce.pdf

[46] Information about shared memory

http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/what-is-shm.html

[47] Definition of thread

http://www.techopedia.com/definition/27857/thread

[48] Information about message passing interface

http://www.hpcvl.org/faqs/programming/mpi-message-passing-interface

[49] Information about Data parallel programming model

http://insidehpc.com/2006/03/what-is-data-parallel-programming/

[50] Information about Load imbalance

https://software.intel.com/en-us/articles/load-balancing-between-threads

[51] Description about race around condition and deadlocks

http://support.microsoft.com/kb/317723

[52] Information and example for deadlock

http://www.roseindia.net/java/thread/deadlocks.shtml

[53] Open MP components

http://www.capsl.udel.edu/courses/cpeg421/2012/slides/openmp_tutorial_04_06_2012.pd

f

[54] 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

[55] OpenMP environment variable

http://msdn.microsoft.com/en-us/library/6sfk977f.aspx

[56] OpenMP tutorial

https://computing.llnl.gov/tutorials/openMP/#Abstract

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

[58] I. Richardson, “the H.264 advanced video compression standard”, Wiley, 2010.

[59] K. Choi et al, “Fast coding unit decision method based on coding tree pruning for high efficiency video coding”, Proc. SPIE Optical Engineering, vol. 51, 030502 , March 2012.

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

[61] Thesis by S.Gangavati on “Complexity reduction of H.264 using parallel programming” which describes significant speed-up in encoding time on GPU using CUDA and CPU combined than on CPU by data and task parallelization, 2012. Access from www.uta.edu/faculty/krrrao/dip

[62] Thesis by T.Saxena on “Reducing the encoding time of H.264 Baseline profile using parallel programming techniques” which describes task based parallelism using OpenMP software without any degradation of quality, 2012. Access from www.uta.edu/faculty/krrrao/dip

[63] Thesis by S.Muniyappa on “Implementation of complexity reduction algorithm for intra mode selection in H.264/AVC Video coding” which describes complexity reduction algorithm which is much faster than JM reference software, 2011. Access from www.uta.edu/faculty/krrrao/dip

[64] Thesis by T.Sathe on “Complexity reduction in H.264 encoder using OpenMP” which basically makes use of parallel processing approach using threads that are managed by OpenMP, 2012. Access from www.uta.edu/faculty/krrrao/dip

[65] Thesis by P.K.Gajalla on “Efficient HEVC lossless coding using sample based angular intra prediction” which describes use of sample based angular prediction approach which can be used for better intra prediction accuracy compared to HEVC block angular intra prediction, 2013. Access from www.uta.edu/faculty/krrrao/dip

[66] Thesis by Parashar Nayana Karunakar on Implementation of an out-of-the-loop post-processing technique for HEVC compressed depth maps, 2013. Access from www.uta.edu/faculty/krrrao/dip

[67] Thesis by Sapna Vasudevan on Implementation of ROT and fast intra prediction in HEVC, 2013. Access from www.uta.edu/faculty/krrrao/dip

[68] G. Bjontegaard, “Calculation of average PSNR differences between RD-curves”, Q6/SG16, VCEG, April 2013.

Biographical Information

Karthik Suresh was born in Chikkamagaluru, Karnataka, India in1990. After completing his schooling at Sadvidya School, Mysuru in 2008, he went on to obtain his Bachelor's degree in Electrical and Electronics Engineering from The National Institute of Engineering, Mysuru in the year 2012.

He joined University of Texas at Arlington to pursue his M.S in Electrical Engineering in Fall 2012. This was around the time he joined the Multimedia Processing Lab. He is currently working as an intern in Intel Corp.