Implementation and Comparison of Serial and Parallel algorithms of SAO in HEVC

THESIS UNDER THE GUIDANCE OF

DR. K. R. RAO

Thesis by

Harsha Nagathihalli Jagadish

DEPARTMENT OF ELECTRICAL ENGINEERING

UNIVERSITY OF TEXAS, ARLINGTON

Acknowledgements

I would like to express the deepest appreciation and gratitude to my Professor, Dr. K. R. Rao who has continually and convincingly conveyed a spirit of adventure in regard to research and an excitement in regard to teaching. Without his guidance and persistent help this thesis would not have been possible.

I thank Dr. Jonathan Bredow and Dr. Howard Russell for taking time to review my work and accepting to be a part of my thesis defense committee.

I also would like to extend my thanks to CalAmp where I interned for 6 months. This research was supported by my manager Somu Ramiah who is a Senior Principal Firmware Engineer at CalAmp. He was always willing to help and give his best suggestions. I thank him for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research.

I would like to thank my Multimedia Processing Lab mates- Ninad, Swaroop, Tuan and Shiba, for providing valuable suggestions during my research work. Also a special thanks to my friends Anuj, Anirudh, Anjana, Akhil who helped me be organized and gave moral support during the thesis defence.

Most importantly, I wish to thank my loving and supportive parents, family and friends. Special thanks to my parents, Mrs. B. S. Shantha Kumari and Mr. N. S. Jagadish whose love and guidance are with me in whatever I pursue. They are the ultimate role models.

November 16, 2017

Abstract

Implementation and Comparison of Serial and Parallel algorithm of Sample Adaptive Offset in HEVC

Harsha Nagathihalli Jagadish, M.S

The University of Texas at Arlington, 2014

Supervising Professor: K.R. Rao

The High Efficiency Video Coding (HEVC) standard is the latest video coding project developed by the Joint Collaborative Team on Video Coding (JCT-VC) which involves the International Telecommunication Union (ITU-T) Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations. HEVC also known as H.265 supports encoding videos with wide range of resolutions, starting from low resolution to beyond High Definition i.e. 4k or 8k. The HEVC standard is an optimization of the previous standard H.264/AVC (Advanced Video coding) which is a very well established and widely used standard in industry and finds its applications in broadcast TV and multimedia telephony. HEVC was preceded by H.264/AVC with the bit-rate reduction of about 50% at the same visual quality.

The in-loop filters are an important part of HEVC video coding standard. They attenuate discontinuities at the prediction and transform boundaries and also improves the quality by attenuating the ringing artifacts and changes in the sample intensity depending on the classification algorithm. The main advantage of these filters is it improves the subjective quality of reconstructed video.

In HEVC, the size of motion predicted blocks varies from 8x4 and 4x8, to 64x64 luma samples, while the size of block transforms and intra-predicted blocks varies from 4x4 to 32x32 samples.

These blocks can be coded independently from the neighboring blocks which allow scope for parallelism. Various methods have been implemented serially to reduce the computational complexity of sample adaptive offset. To improve the coding efficiency, an extra step is taken to implement the code in parallel since the blocks can be coded independent of each other. The technology is rapidly evolving and moving towards a world of parallelization so as to reduce the amount of time spent of computation. Multi core and many core based computation and design are the new trends in the market. As a result, in this thesis an attempt is made to map the video coding algorithm on the GPU cores to accelerate the speed at which the execution takes place. This is done using CUDA programming for SAO algorithm. SAO has many stages of implementation. Each of these stages is implemented in parallel using NVIDIA GPUs. A comparison of the results obtained in serial and parallel are evaluated using speedup metric and the subjective quality is measured using PSNR (Peak Signal to Noise Ratio).

Table of Contents

CHAPTER 1

Introduction

Uncompressed video signals generate a huge quantity of data, and video use has become more and more ubiquitous. There is also a constant hunger for higher quality video like in the form of higher resolutions, higher frame rates, and higher fidelity—as well as a hunger for greater access to video content. Moreover, the creation of video content has moved from the being the exclusive domain of professional studios toward individual authorship, real-time video chat, remote home surveillance, and even “always on” wearable cameras. As a result, video traffic is the biggest load on communication networks and data storage world-wide—a situation that is unlikely to fundamentally change; although anything that can help ease the burden is an important development. HEVC offers a major step forward in that regard.

Video content is produced daily through variety of electronic devices, however, storing and transmitting video signals in raw format are impractical due to its excessive resource requirement. Today popular video coding standards such as MPEG-4 visual [41], H.264/AVC [45] and HEVC [2] are used to compress the video signals before storing and transmitting. Accordingly, efficient video coding plays an important role in video communications. While video applications become wide-spread, there is a need for high compression and low complexity video coding algorithms that preserve the image quality.

There have been several video coding standards introduced by organizations like the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), Moving Picture Experts Group (MPEG) and the Joint Collaborative Team on Video Coding (JCT-VC) [49] . Each standard is an improvement over the previous standard. With every standard, the general thumb of rule has been to retain the same video quality by being able to reduce the bit rate by 50%. MPEG-2 basically created the world of digital video television as we know it, so while AVC was being developed, some people doubted that it could achieve a similar degree of ubiquity when so much infrastructure had been built around the use of MPEG-2. Although it was acknowledged that AVC might have better compression capability, some thought that the entrenched universality of MPEG-2 might not allow a new non-compatible coding format to achieve “critical mass”. AVC had about twice the compression capability of MPEG-2—i.e., one can code video using only about half the bit rate while still achieving the same level of quality—so that one can send twice as many TV channels through a communication link or store twice as much video on a disc without sacrificing quality. Alternatively, the improved compression capability can be used to provide higher quality or enable the use of higher picture resolution or higher frame rates than would otherwise be possible. AVC also emerged at around the same time that service providers and disc storage format designers were considering a transition to offer higher resolution “HDTV” rather than their prior “standard definition” television services. Once system developers realized that they needed to store and send twice as much data if they were going to use MPEG-2 instead of AVC for whatever video service they were trying to provide, most of them decided they needed to find a transition path to AVC. While MPEG-2 video remains a major presence today for legacy compatibility reasons, it is clearly fading away in terms of importance. HEVC offers the same basic value proposition today that AVC did when it emerged—i.e., a doubling of compression capability. It can compress video about twice as much as AVC without sacrificing quality, or it can alternatively be used to enable delivery of higher resolutions and frame rates—or other forms of higher quality, such as a higher dynamic range or higher precision for improved color quality. It also comes at another time when new video services are emerging—this time for UHDTV, higher dynamic range, and wider color gamut. Compression capability—also known as “coding efficiency” or “compression efficiency”—is the most fundamental driving force behind the adoption of modern digital video compression technology, and HEVC is exceptionally strong in that area. It is this meaning from which the High Efficiency Video Coding standard derives its name. However, it is also important to remember that the standard only provides encoders with the ability to compress video efficiently—it does not guarantee any particular level of quality, since it does not govern whether or not encoders will take full advantage of the capability of the syntax design.

1.1 EVOLUTION OF VIDEO CODING STANDARDS

Major video coding standards have been developed by the International Standardization organization / International Electro technical Commission (ISO/IEC) and the International Telecommunication Union – Telecommunication Standardization Sector (ITU-T) [49]. Figure 1.1 shows a historical perspective for video coding standards development since the very first ITU-T H.120. The emergence of H.264/AVC doubled the coding efficiency from that of the MPEG-4 simple profile and has therefore gained wide industrial acceptance recently [5]. Further extensions of H.264/AVC include high profiles, scalable video coding (SVC) extension, and multi view video coding (MVC) extension [49].

Back in 2005, the ITU-T Video Coding Experts Group (VCEG) considered the future work beyond H.264/AVC [3]. Possible targets and scope of the standard were brainstormed and a software known as Key Technology Area (KTA) was developed and released in 2008 [3]. In 2009, the ISO/IEC Moving Picture Experts Group (MPEG) began a similar call for High-Performance Video Coding (HVC) [49].

There have been several video coding standards introduced by organizations like the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), Moving Picture Experts Group (MPEG) and the Joint Collaborative Team on Video Coding (JCT-VC). Each standard is an improvement over the previous standard. With every standard, the general thumb of rule has been to retain the same video quality by being able to reduce the bit rate by 50%. Figure 1.1 shows the evolution of video coding standards over the years.

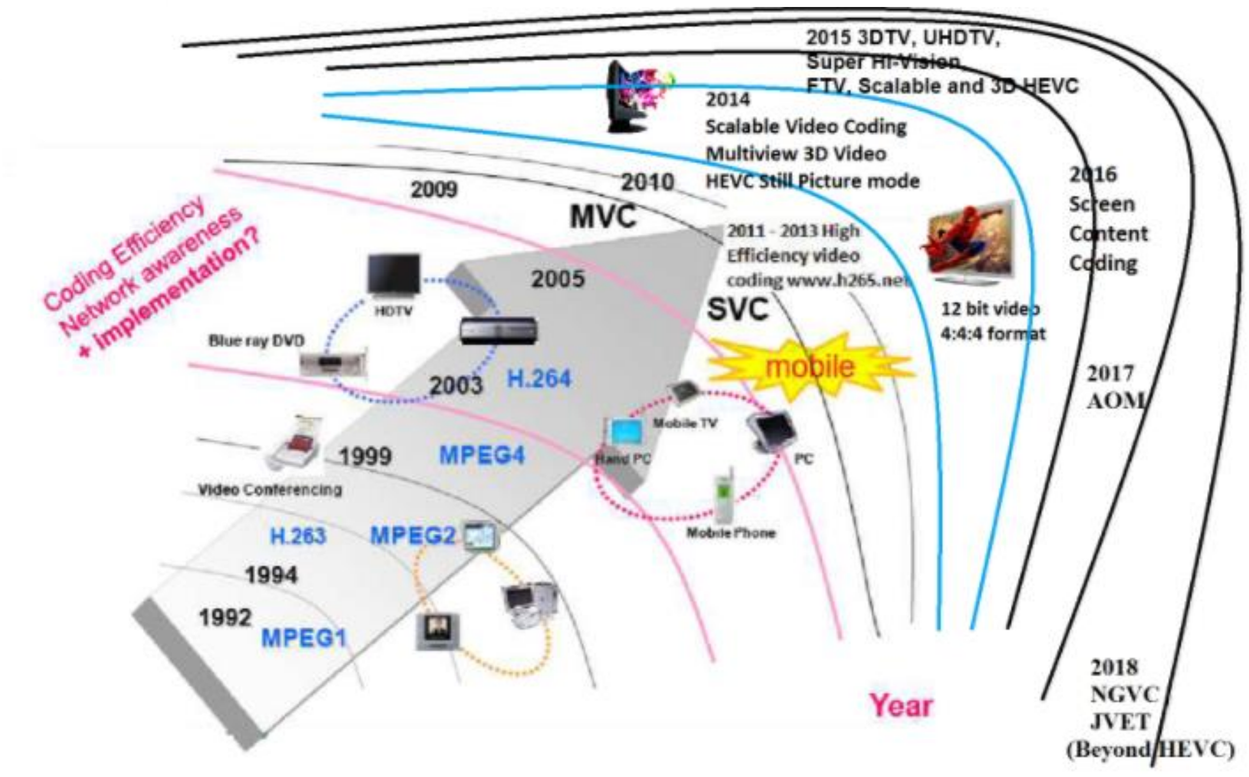

2015 3DTV, UHDTV, Super Hi-Vision FTV, Scalable and 3D-HEVC
2014 Scalable Video Coding Multiview 3D Video HEVC Still Picture mode
2016 Screen Content Coding
2011 - 2013 High Efficiency video coding www.h265.net
12 bit video 4:4:4 format
2017 AOM
2018 NGVC JVET (Beyond HEVC)
Coding Efficiency Network awareness + implementation?
2009
2010
MVC
2005
SVC
mobile
HDTV
Blue ray DVD
2003
H.264
Mobile TV
Hand PC
PC
Mobile Phone
1999
MPEG4
Video Conferencing
H.263
MPEG2
1994
1992
MPEG1
Year

Figure 1.1: Evolution of video coding standards [3] [4]

Many of the basic concepts of video coding such as transform coding, motion estimation and compensation and entropy coding were developed in the 1970s and 1980s. MPEG-1 was standardized on the early 1990s. MPEG-4 was developed in the late 1990s and soon after

H.263 was standardized. H.264/AVC was published in 2003 and Google released the VP8 video coding format in 2010. 2013 saw the publication of the High Efficiency Video Coding (HEVC) standard and Google's VP9 format.

1.2 NEED FOR COMPRESSION

Videos take up a lot of space—just how much varies widely depending on the video format, the spatial resolution and the number of frames per second you select. Uncompressed 1080 HD video footage takes up about 10.5 GB of space per minute of video. If you use a smartphone to shoot your video, 1080p footage takes up 130 MB per minute of footage, while 4K video takes up 375 MB of space for each minute film. Because it takes up so much space, the video must be compressed before it is put on the web. Compressed" just means that the information is packed into a smaller space. There are two kinds of compression*: lossy* and *lossless*.

Lossy compression means that the compressed file has fewer data in it than the original file. In some cases, this translates to lower quality files, because information has been "lost," hence the name. However, you can lose a relatively large amount of data before you start to notice a difference. Lossy compression makes up for the loss in quality by producing comparatively small files. For example, DVDs are compressed using the MPEG-2 [60] format, which can make files 15 to 30 times smaller, but viewers still tend to perceive DVDs as having high-quality pictures.

Most video that is uploaded to the internet uses lossy compression to keep the file size small while delivering a relatively high-quality product.

Lossless compression is exactly what it sounds like, compression where none of the information is lost. This is not nearly as useful as lossy compression because files often end up being the same size as they were before compression. This may seem pointless, as reducing the file size is the primary goal of compression. However, if the file size is not an issue, using lossless compression results in a perfect-quality picture. For example, a video editor transferring files from one computer to another using a hard drive might choose to use lossless compression to preserve quality while he is working.

Lossless compression preserves the original quality so that after decompression the original data is obtained, whereas, in lossy compression, while offering higher compression ratio, the decompressed data is not equal to the original data.

Real time video transmission requires huge amounts of data to be transmitted over a channel in a very short period of time. A video sequence with fairly small resolution such as 352x288 pixels, having 24 bits/pixel (8 bits for each color component) with frame rate of 30 frames/sec requires: 352x288x24x30 bits/sec, which is about 72.9 megabits/sec [1]. So, transmitting this video over a multimedia channel requires very high bandwidth.

The high bit rates that result from the various types of digital video make their transmission through their intended channels very difficult. High resolution videos captured by HD cameras and HD videos on the internet would require bandwidth and storage space if used in the raw format. It is true that both the storage and transmission capacities continue to increase. However, an efficient and well-designed video compression system gives very significant performance advantages for visual communication at both low and high transmission bandwidths.

Even after meeting the bandwidth requirements, significant amount of processing will be involved to exploit redundancies in the data. The advantage with data compression will be lost if we cannot meet the data processing requirements. For example, a 30 frames/second video requires a single frame to be processed in 33 milliseconds.

1.3 Basics of Video Coding Techniques

To reduce video bandwidth requirements compression methods are used. In general, compression is defined as encoding data to reduce the number of bits required to present the data. As discussed earlier, compression can be lossless or lossy. Video data is compressed and decompressed with the techniques discussed under the term video coding, with compressor often denoted as enCOder and decompressor as DECoder, which collectively form the term CODEC. Therefore, a CODEC is the collection of techniques used to compress and decompress digital videos. The general process of encoding and decoding of video signal in transmission chain is given in Figure 1.2.

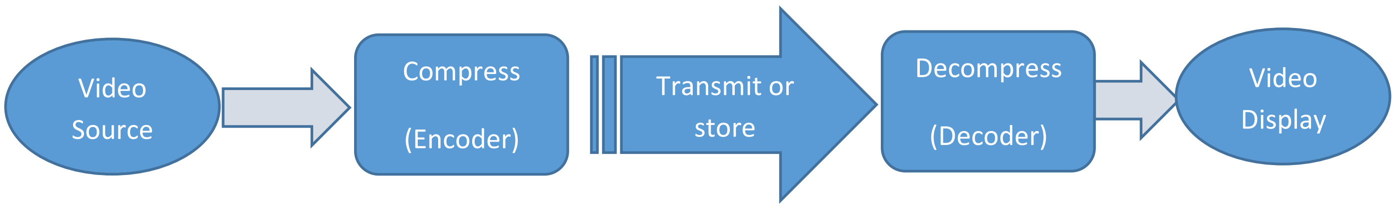


Figure 1.2: Flow of video coding process

The decoder inverses the operation performed by the encoder. The encoder and decoder are based on the same underlining techniques. The encoder maximizes compression efficiency by exploiting temporal, spatial, and statistical redundancies. A common encoder and decoder model is illustrated in Figure 1.3.

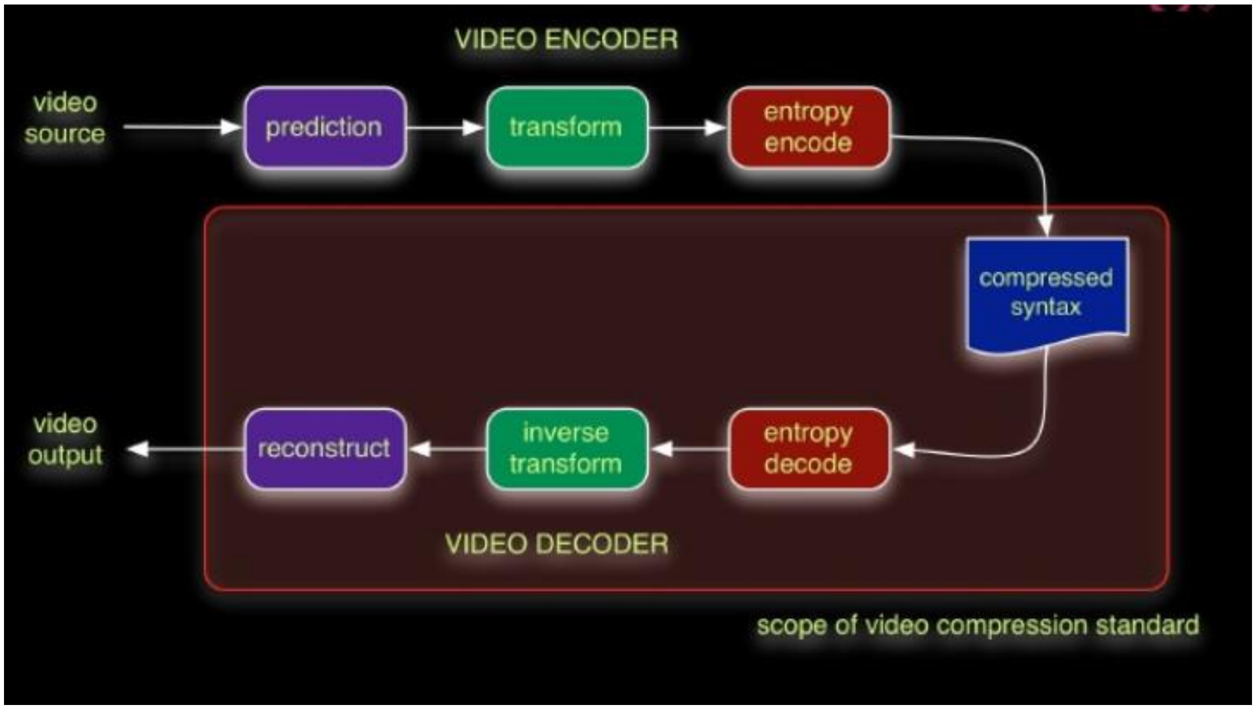


Figure 1.3: Basic flow of a Video Codec process [75]

Video compression is the ability to exploit the temporal and spatial redundancies while sending the images. The temporal redundancies are exploited by a technique called inter frame coding which generates all the P-frames or predictive frames and B-frames or bi-predictive frames. It compares the current frame with a reference frame and sends only the change in the images. The spatial redundancies are exploited by a technique called intra-frame coding. This technique takes advantage of the fact that pixels tend to have intensities that are very close to the neighboring pixels. Intra-frame technique generates all the I-frames. The three types of frames are shown in fig 1.4

I-Frames and P-Frames

In most of the videos, there is very little change from frame to another. Certain frames in a video are designated as key frames and these frames are not compressed. Such frames are called as “Intra frames” or simply “I-Frames”. For a macro block in the current frame, motion estimation tries to find the closest matching macro block in a previous reference frame (i.e. I-Frame or P-Frame). If the video does not contain any moving objects, then it is likely that the matching block also occur at the same position as the block in the current frame. Then the current block need not be stored fully, but only the difference in the pixel values (brightness and color), from the matching block in the IFrame is stored. Compression is achieved because difference numbers are much smaller than actual values. This difference vector that is stored is called as the “prediction vector”. Such frames that are stored as differences from a previous reference frame are called as “P-Frames”. Note that the reference frame can be I-Frame or a P-Frame also, as long as it occurs before the frame. Considering that the video contains moving objects, the matching block in the reference frame might be at some other location as compared to the block in the current frame. Therefore, we need to specify a “motion vector” also, that specifies where the current block, with reference to the matching block is. For example, suppose the current block is at location (0, 0) and the matching block in the reference frame is at location (2, 2), then the motion vector for the current block, with respect to the reference frame would be (-2,-2). Using the motion-vector and the prediction-vector of each block, we can reconstruct the entire P-Frame by knowing the pixel values in the reference frame [76].

B-Frames

Video compression algorithms commonly encode frames in an order that is different from the order in which frames are displayed. The encoder may skip several frames ahead and encode a future frame. Then, it may go backwards and encode the next frame in the sequence. By doing this, the encoder can use the encoded future frame as a reference frame. That is, it can perform motion estimation backward in time. "B-Frames" are frames that are constructed by using both a previous frame and/or a future frame as a reference. The typical sequence of I-Frame, P-Frame and B-Frame is shown in figure 1.4 [76]:

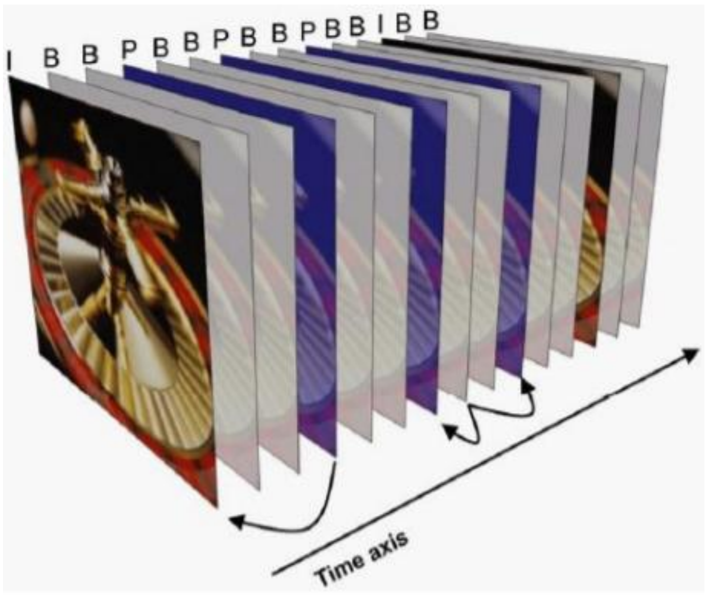


Figure 1.4: The I-frame, P-frame and B-frames [76]

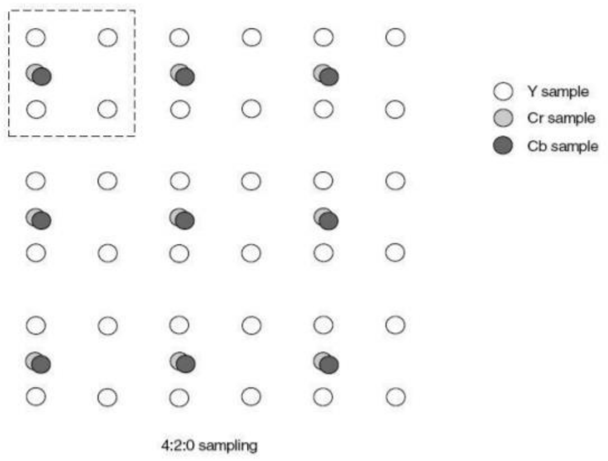
In this chapter, the general description and the need for video compression is discussed. It also shows the evolution of video coding standards and the basic steps involved in video coding. The following chapter explains about the latest video coding technology called High Efficiency Video Coding (HEVC).

CHAPTER 2

High Efficiency Video Coding

High Efficiency Video Coding (HEVC) [2] is an international standard for video compression developed by a working group of ISO/IEC MPEG (Moving Picture Experts Group) and ITU-T VCEG (Video Coding Experts Group). The main goal of HEVC standard is to significantly improve compression performance compared to existing standards (such as H.264/Advanced Video Coding [7]) in the range of 50% bit rate reduction at similar visual quality [2].

HEVC is designed to address existing applications of H.264/MPEG-4 AVC and to focus on two key issues: increased video resolution and increased use of parallel processing architectures [2]. It primarily targets consumer applications as pixel formats are limited to 4:2:0 8-bit and 4:2:0 10-bit. The next revision of the standard, finalized in 2014, enables new use-cases with the support of additional pixel formats such as 4:2:2 and 4:4:4 as shown in figure 2.1 and bit depth higher than 10-bit [8], embedded bit-stream scalability, 3D video [10] and multiview video [9]. HEVC was designed to double the compression ratios of its predecessor H.264/AVC with a higher computational complexity. After several years of developments, mature encoding and decoding, solutions are emerging accelerating the upgrade of the video coding standards of video contents from the legacy standards such as H.264/AVC [77]. With the increasing needs of ultra-high resolution videos, it can be foreseen that HEVC has already become the most important video coding standard.



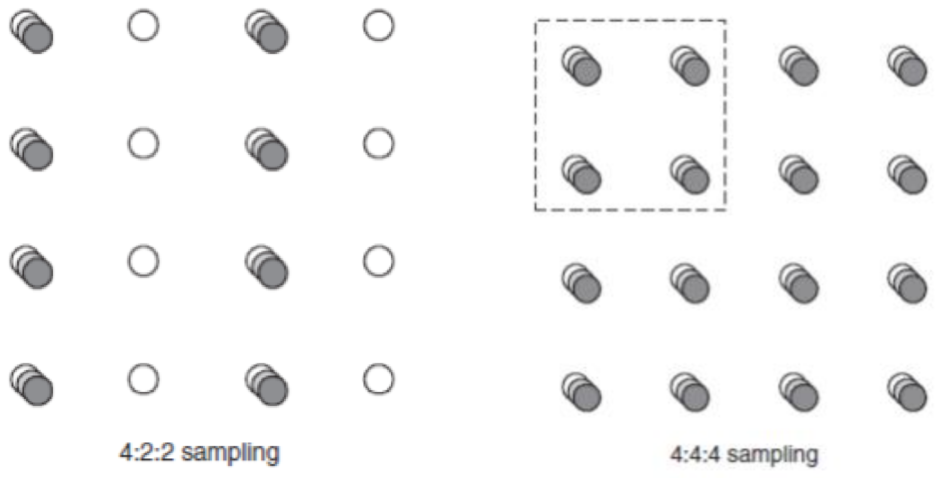


Figure 2.1: 4:2:0, 4:2:2: and 4:4:4 sampling patterns for luminance and chrominance [45]

HEVC, the High Efficiency Video Coding standard, is the most recent joint video project of the ITU-T VCEG and ISO/IEC MPEG standardization organizations, working together in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) [11]. The HEVC standard is designed to achieve multiple goals: coding efficiency, transport system integration and data loss resilience and as well as implementability using parallel processing architectures [12].The main goal of the HEVC standardization effort is to enable significantly improved compression performance relative to existing standards – in the range of 50% bit rate reduction for equal perceptual video quality [13] [14].

The block diagram of HEVC encoder is shown in figure 2.2 [12]. The corresponding decoder block diagram is shown in figure 2.3 [15].

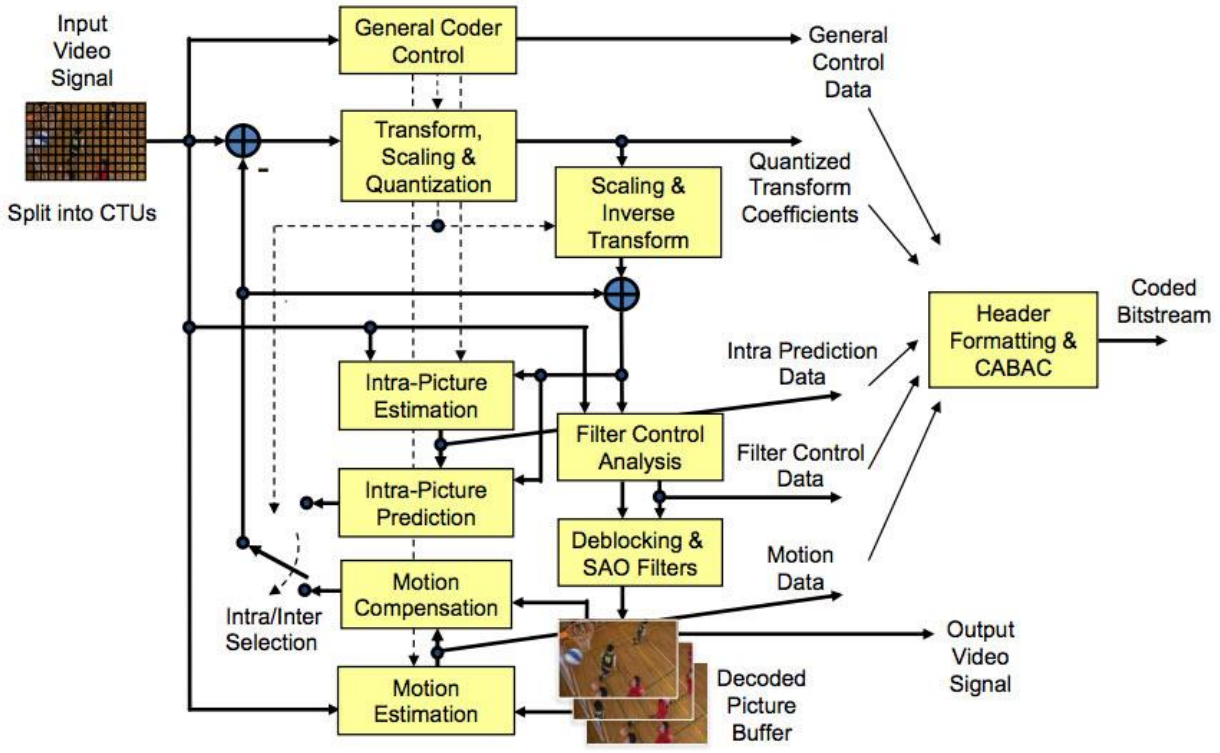


Figure 2.2. HEVC encoder block diagram [12]

The video coding layer of HEVC employs the same “hybrid” approach (inter-/intra-picture prediction and 2D transform coding) used in all video compression standards since H.261 [12]. Some differences in HEVC are coding tree units instead of macro blocks, single entropy coding methods-Context Adaptive Binary Arithmetic Coding (CABAC) and features like tiles , wave front parallel processing and dependent slices to enhance parallel processing.

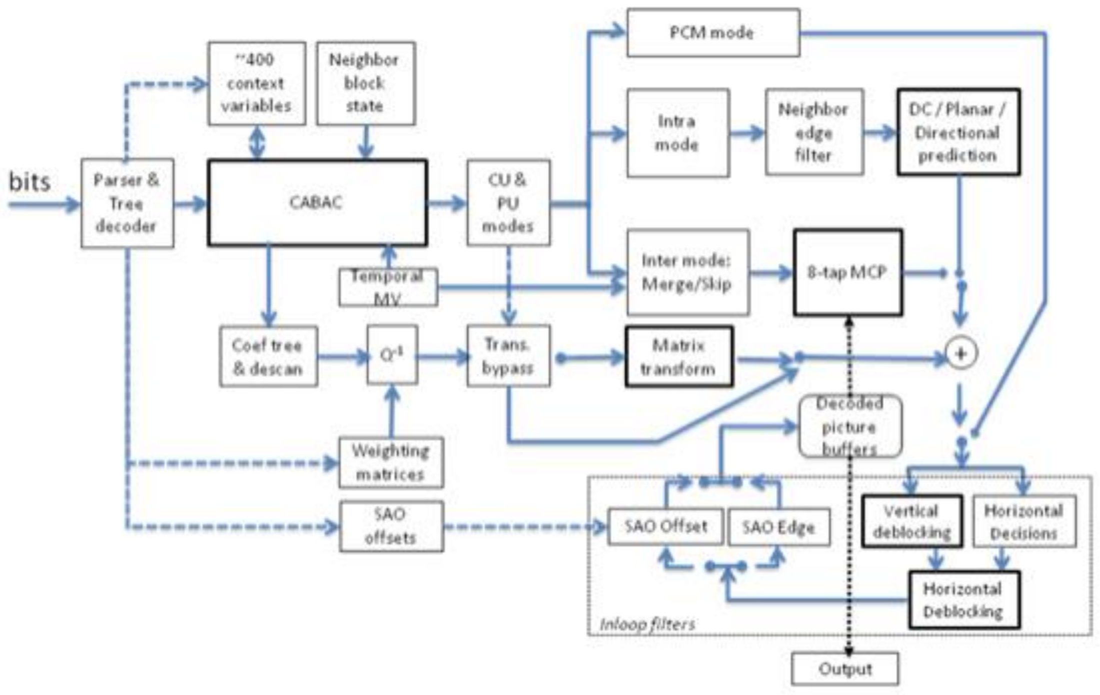


Figure 2.3. HEVC decoder block diagram [15]

The residual signal of the intra or inter prediction, which is the difference between the original block and its prediction, is transformed by a linear spatial transform [12]. The transform coefficients are then scaled, quantized, entropy coded, and transmitted together with the prediction information. The residual is then added to the prediction, and the result of that addition may then be fed into one or two loop filters to smooth out artifacts induced by the block-wise processing and quantization. The final picture representation (which is a duplicate of the output of the decoder) is stored in a decoded picture buffer to be used for the prediction of subsequent pictures [12].

2.1 Coding tree unit

HEVC has replaced the concept of macroblocks (MBs) by coding tree units. The coding tree unit has a size selected by the encoder and can be larger than the traditional macroblocks. It consists of luma coding tree blocks (CTB) and chroma CTBs. HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quad tree-like signalling [12].

The quadtree syntax of the CTU specifies the size and positions of its luma and chroma *coding blocks* (CBs). One luma CB and ordinarily two chroma CBs, together with associated syntax, form a coding unit (CU). Each CU has an associated partitioning into prediction units (PUs) and a tree of transform units (TUs). Similarly, each CB is split into prediction blocks (PB) and transform blocks (TB) [16].The decision whether to code a picture area using inter-picture or intra-picture prediction is made at the CU level.

Figure 2.4 shows the division of a CTB into CBs and transform blocks TB [12].

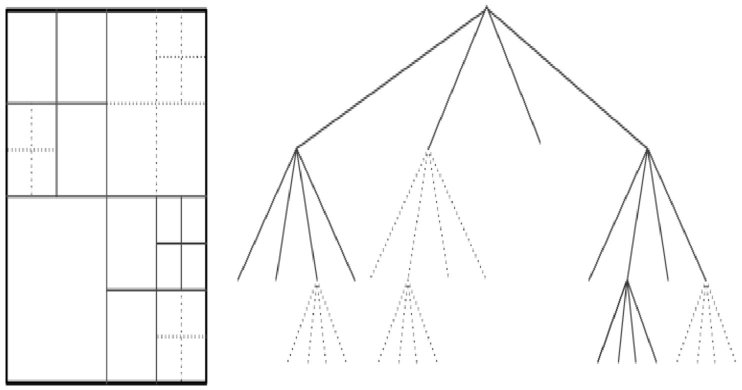


Figure 2.4. Subdivision of CTB into TB [12]

Figure 2.5 shows the sizes of CTU [17]

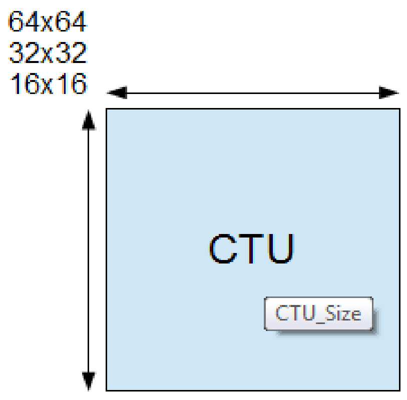


Figure 2.5. Different sizes of CTU [17]

2.2 Intra Picture Prediction

Intra prediction in HEVC [46] is quite similar to H.264/AVC [16]. Samples are predicted from reconstructed samples of neighboring blocks. The mode categories remain identical: DC, plane, horizontal/vertical, and directional; although the nomenclature has somewhat changed with planar and angular respectively corresponding to H.264/AVC's plane and directional modes [16].

For intra prediction, previously decoded boundary samples from adjacent PUs must be used. Directional intra prediction is applied in HEVC, which supports 17 modes for 4x4 block and 34 modes for larger blocks, inclusive of DC mode [18]. Directional intra prediction is based on the assumption that the texture in a region is directional, which means the pixel values will be smooth along a specific direction [18].

The increased number of directions improves the accuracy of intra prediction, however it increases the complexity and increased overhead to signal the mode [6].With the flexible structure of HEVC, more accurate prediction, and other coding tools, a significant improvement in coding efficiency is achieved over H.264/AVC [18]. HEVC supports various intra coding methods referred to as Intra_Angular, Intra_Planar and Intra_DC. In [19], an evaluation of HEVC coding efficiency compared with H.264/AVC is provided, which shows that the average bit rate saving for random access high efficiency (RA HE) case is 39%, while for all intra high efficiency (Intra HE) case this bit rate saving is 25%, which is also considerable but may not be as much as expected. It seems that the improvement of intra coding efficiency is still desirable. Figure 2.6 shows different intra prediction modes for HEVC [18].

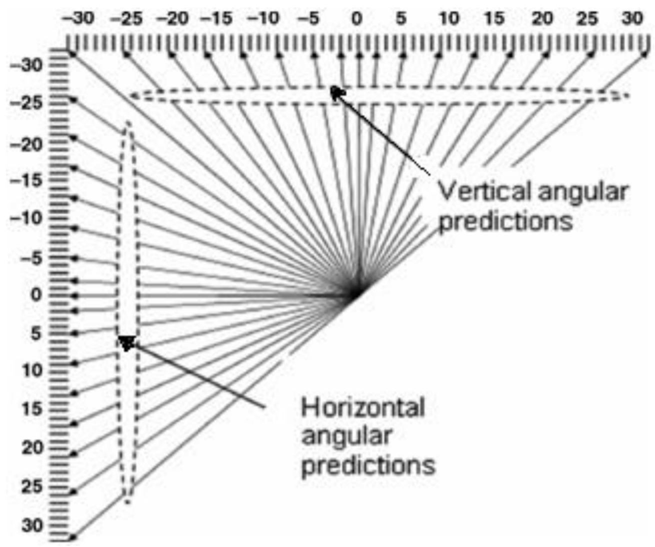


Figure 2.6. Intra prediction modes for HEVC [18]

Figure 2.7 shows different intra prediction modes for 4 x 4 blocks in H.264 [20].

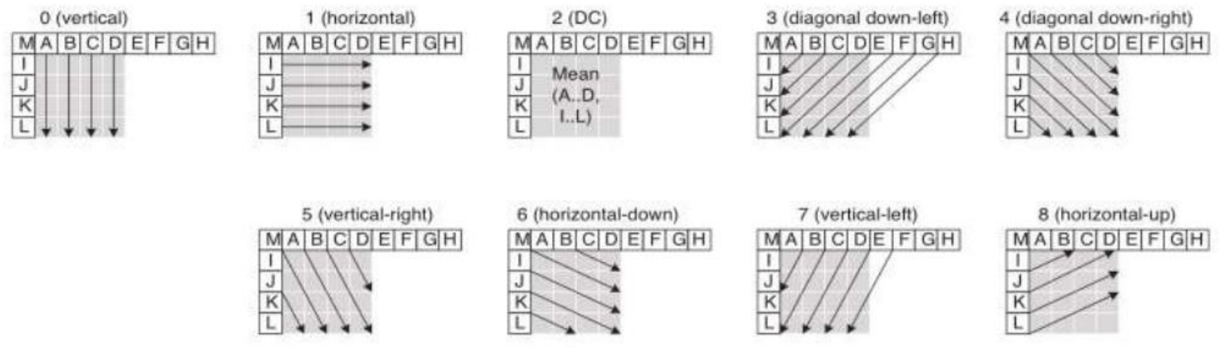


Figure 2.7. Intra prediction modes for 4 x 4 blocks in H.264 [20]

2.3 Inter Picture Prediction

Inter picture prediction in HEVC is divided into prediction block partitioning, fractional sample interpolation, motion vector prediction for merge and non-merge mode. HEVC supports more PB partition shapes for inter-coded CBs. The samples of the PB for an inter-coded CB are obtained from those of a corresponding block region in the reference picture identified by a reference picture index, which is at a position displaced by the horizontal and vertical components of the motion vector.

HEVC only allows a much lower number of candidates to be used in the motion vector prediction process for the non-merge case, since the encoder can send a coded difference to change motion vector. Further, the encoder needs to perform motion estimation, which is one of the most computationally expensive operations in the encoder, and complexity is reduced by allowing less number of candidates [2]. When the reference index of the neighboring PU is not equal to that of the current PU, a scaled version of the motion vector is used [2]. The neighboring motion vector is scaled according to the temporal distances between the current picture and the reference pictures indicated by the reference indices of the neighboring PU and the current PU, respectively [2]. When two spatial candidates have the same motion vector components, one redundant spatial candidate is excluded [2]. Figure 2.8 shows the motion estimation flow for HEVC. [40]

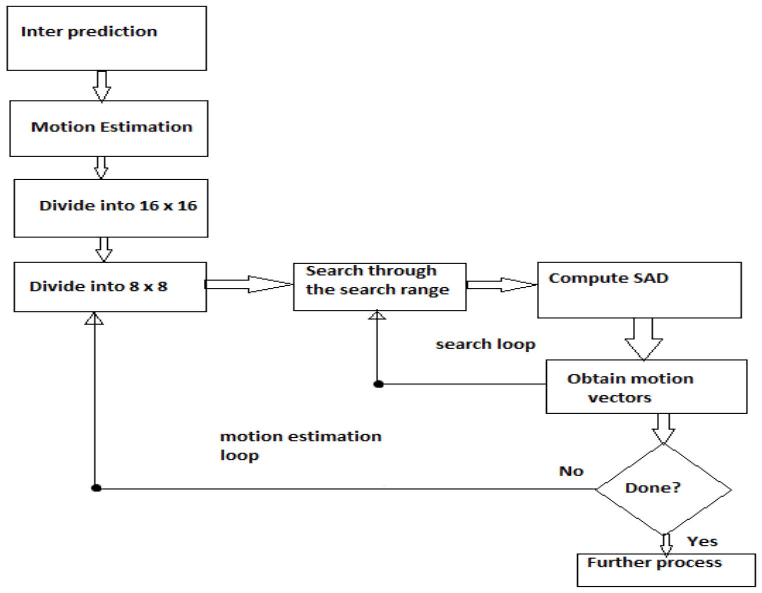


Figure 2.8. HEVC motion estimation flow [40]

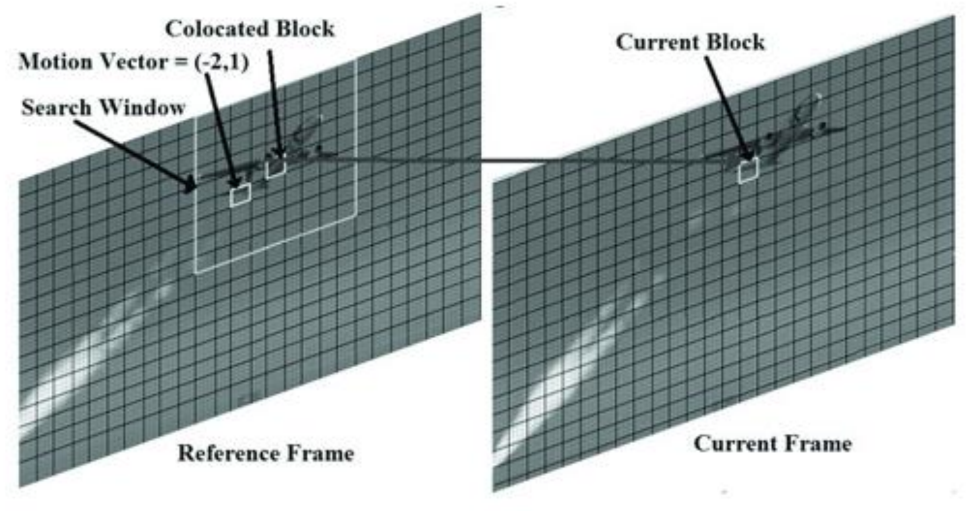


Figure 2.9: Illustration of motion estimation process [47]

In multi-view video coding, both temporal and interview redundancies can be exploited by using standard block based motion estimation (BBME) [21]. Due to its simplicity and efficiency [22], BBME [23] has been adopted in several international video coding standards such as MPEG-x, H.26x, AVS China and VC-1 [24] [25]. In BBME, the current frame is divided into NxN- pixel –size macroblocks (MBs) and for each MB a certain area of the reference frame is searched to minimize a block difference measure (BDM), which is usually a sum of absolute differences (SAD) between the current MB and the reference MB [21]. The displacement within the search area (SA) which gives the minimum BDM value is called a motion vector [21]. With the development of video coding standards, the basic BBME scheme was extended by several additional techniques such as sub-pixel, variable block size, and multiple reference frame motion estimation [23]. Figure 2.9 shows multiple frame reference frame motion [26].

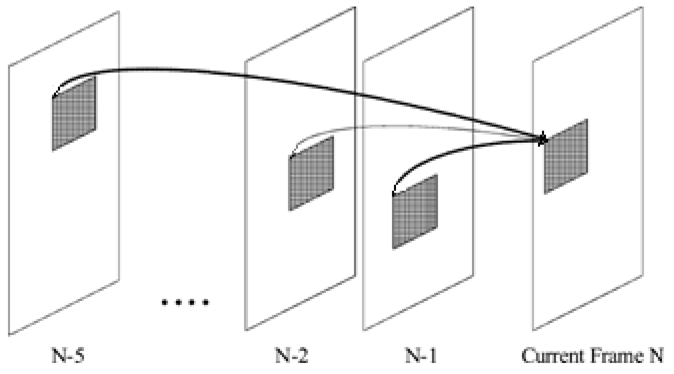


Figure 2.10. Multiple frame reference frame motion [26]

Figure 2.11 Variable block sizes in motion estimation [27].

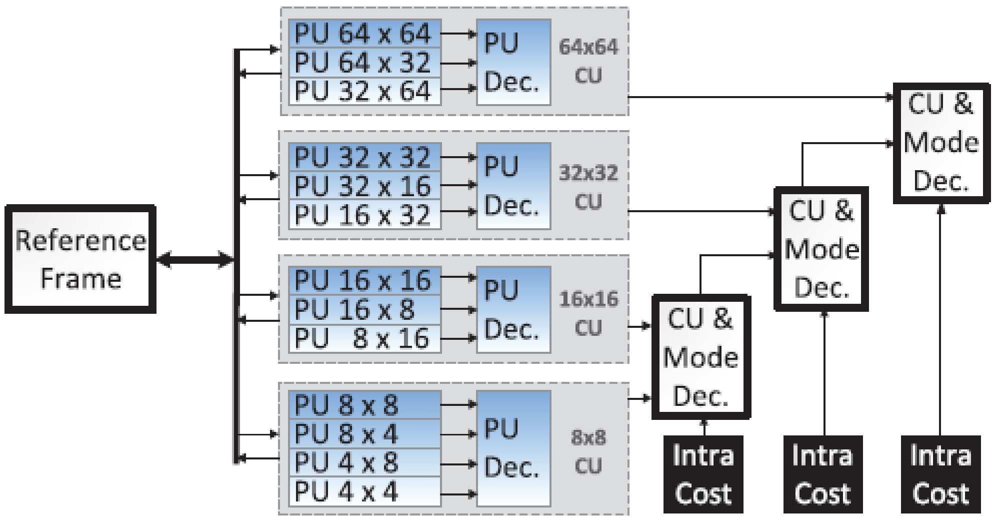


Figure 2.11. Variable block sizes in motion estimation HEVC [27]

2.4 Block Artifacts in Video Coding

In HEVC both the motion prediction and transform coding are block-based. The size of motion predicted blocks varies from 4x8 and 8x4 to 64x64 luma samples, while the size of block transforms and intra-predicted blocks varies from 4x4 to 32x32 samples. These blocks are coded relatively independently from their neighboring blocks and approximate the original signal with some degree of similarity. Since coded blocks only approximate the original signal, the difference between the approximations may cause discontinuities at the prediction and transform block boundaries. For example, motion prediction of the adjacent blocks may come from the non-adjacent areas of a reference picture (see Fig. 2.12) or even from different reference pictures. In case of non-overlapping block transforms, used in HEVC, coarse quantization can also create discontinuities at the block boundaries.

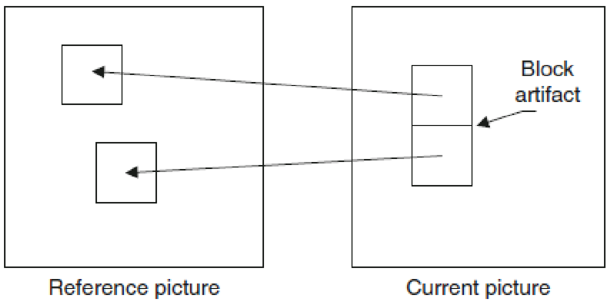


Figure 2.12: Block artifact may be created when adjacent blocks are predicted from non-adjacent areas in the reference picture [28]

The example of a block artifact in one dimension is shown in figure 2.13. The horizontal axis shows the sample positions along a horizontal or vertical 1-D line, and the vertical axis shows the sample values.

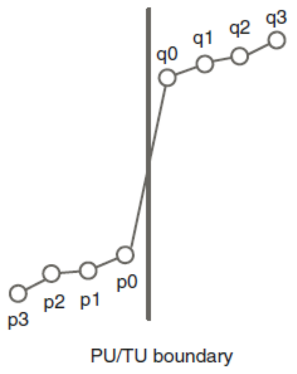


Figure 2.13. Example of block artifact in one dimension [29]

Deblocking filter attenuates the artifacts in the areas, where they are mostly visible, i.e. in the smooth, uniform areas. The excessive filtering in the highly detailed areas should be avoided since it can cause undesirable blurring. The artifacts in those areas are rarely noticed by the human eye, while it is also more difficult to determine whether the discontinuity is caused by a block boundary or belongs to the original signal. Therefore, an important part of the deblocking filter is the deblocking filtering decisions, which determine whether a particular part of a block boundary is to be filtered. In these decisions, the HEVC deblocking filter uses the mode and motion information from the decoded bit stream as well as analyzes of the reconstructed values samples on the sides of the block boundary. The strength of the deblocking filter can also be adjusted by the encoder on the picture and the slice basis.

Block-based intra/inter prediction and transform coding are applied in HEVC. Therefore, artifacts that are commonly seen in prior video coding standards at medium and low bitrates, such as blocking artifacts, ringing artifacts, color biases, and blurring artifacts may still exist in HEVC. In order to reduce these artifacts, HEVC also adopts in-loop filtering as in AVC. HEVC uses integer DCTs ranging from 4 × 4 to 32 × 32, while AVC uses transforms no larger than 8 × 8. A larger transform could introduce more artifacts including ringing artifacts that mainly come from quantization errors of transform coefficients. Besides, HEVC uses 8-tap fractional luma sample interpolation and 4-tap fractional chroma sample interpolation, while AVC uses 6-tap and 2-tap for luma and chroma, respectively. A higher number of interpolation taps can also lead to more serious ringing artifacts. Hence, it is necessary to incorporate new in-loop filtering techniques in HEVC.

Coding standard	**Number of taps for Luma sample interpolation**	**Number of taps for Chroma sample interpolation**
AVC	6	2
HEVC	8	4

Table 2.1: Interpolation taps for HEVC and AVC

First we have the deblocking filter which attenuates discontinuities at the transform block boundaries followed by SAO at the output section of this deblocking filter. There is an advantage of using the deblocking filter as an in-loop filter compared to a post filter because if it is used as a post filter, block artifacts can be copied by motion estimation inside the blocks, which can make the artifacts detection more difficult and increases deblocking complexity compared to in-loop filtering which needs to be applied only to the block boundaries.

The main function of Sample Adaptive Offset (SAO) is to attenuate ringing artifacts which are more likely to appear when larger transform sizes are used. The concept here is to classify reconstructed pixels into different categories and then reduce the distortion by simply adding an offset for each category of pixels. The pixel intensity and edge properties are used for pixel classification. To further improve the coding efficiency, a picture can be divided into regions for localization of offset parameters. A customized SAO encoder does not necessarily attempt to minimize mean sample distortion but can use another criterion to generate SAO parameters.

This chapter describes an overview of HEVC. The various steps and techniques involved in reconstructing the samples, difference between HEVC and previous standards, the reasons behind blocking artifacts, In-loop filters and functions of SAO. In the next chapter, proposal is discussed in detail that explains the use of CUDA programming to implement GPU algorithms for SAO filter.

CHAPTER 3

Proposal in Detail

In this thesis, parallel algorithm is designed for the SAO by exploiting GPU multi-core computing ability, including parallel computation of sample classification and statistics collection for each coding tree block (CTB), parallel calculation of the best offset values and minimum distortions for each class of edge offset (EO) and each band of band offset (BO), parallel processing of the SAO merging, and parallel implementation of SAO filtering in HEVC. All the parallel algorithms are implemented on GPU programmed with CUDA language.

With the development of high definition (HD) and ultra-high definition (UHD) videos, video applications have turned into the fields of digital video broadcast, mobile wireless video, remote monitoring and portable photography etc. The tendency of diversity and high definition of video applications brings up higher requirements to existing video coding standards. Nowadays, computers are heading towards multi-core direction [1]. Multi-core and many-core based parallel algorithm design and computation are the beginning stage. Mapping video encoding algorithm onto GPU has been an important direction for accelerating the execution speed [1]. The corresponding parallel algorithms for SAO designed in [1] is implemented through CUDA programming. This is implemented on GPU by fully exploiting GPU many-core computing ability to improve the encoding time.

Based on the proposals of JCT-VC, SAO was adopted into the working draft of HEVC in the 5th JCT-VC meeting [1]. SAO is an in-loop filtering tool which is newly adopted in HEVC and provides superior image reconstruction performance at the cost of high computational complexity. So far, some successful works have been conducted to reduce the computational complexity of SAO. However, despite the improved methods (based on the serial algorithm) have limits to improve the coding efficiency. So it is necessary to design efficient parallel algorithms to satisfy the requirement of high computational complexity of SAO.

This thesis focuses on designing the corresponding parallel algorithms for the SAO by exploiting GPU multi-core computing ability, including parallel computation of sample classification and statistics collection for each coding tree block (CTB), parallel calculation of the best offset values and minimum distortions for each class of edge offset (EO) and each band of band offset (BO), parallel processing of the SAO merging, and parallel implementation of SAO filtering. All the parallel algorithms are implemented on GPU programmed with CUDA language.

CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA [34] [48]. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming [48]. Also, CUDA supports programming frameworks such as OpenACC and OpenCL. When it was first introduced by NVIDIA, the name CUDA was an acronym for Compute Unified Device Architecture.

Driven by its power integrated with a massively-parallel and multithreaded many-core architecture, many areas of science and technology have benefitted by the use of the GPU in addressing their advanced computational problems, which were previously thought of as challenging and not feasible [33] [34]. The GPUs are not well suited to solve all types of problems [35], however there are many kinds of applications that have achieved quite significant speed-up depending on the hardware platform [33][34]. Using GPU in computational environment means combining multi-core CPU and many core GPU to form a computing environment [36]. Figure 3.1 shows problem decomposition for serial parts in a CPU and parallel parts in a GPU [33].

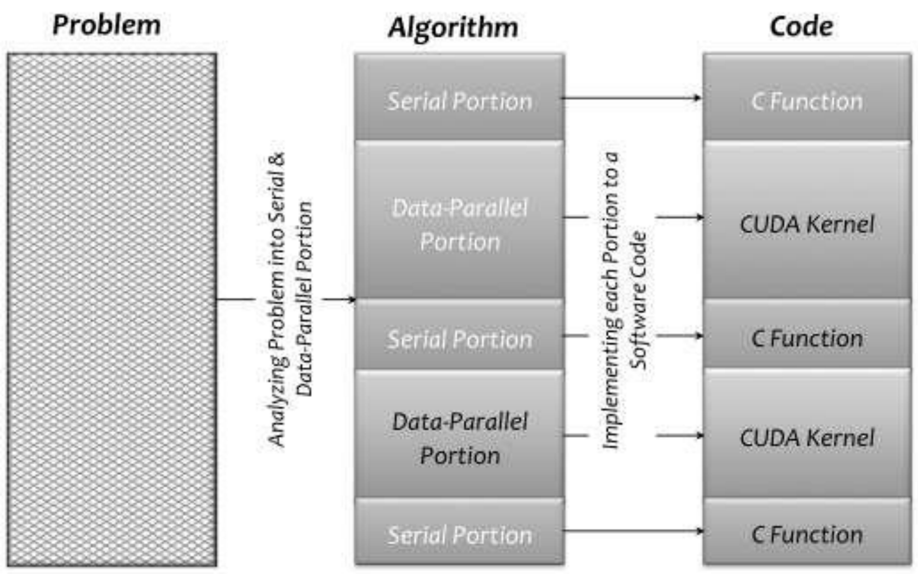


Figure 3.1: Decomposition of the problem for serial parts to be executed on CPU and parallel parts to be executed on the GPU [33]

CUDA is basically used to transfer the control to GPU from CPU where many threads are used which share the same code and start executing in parallel. But it is important that the threads should be organised such that they can execute independent of one another and the results do not depend upon one another as shown in figure 3.2 [37][38].

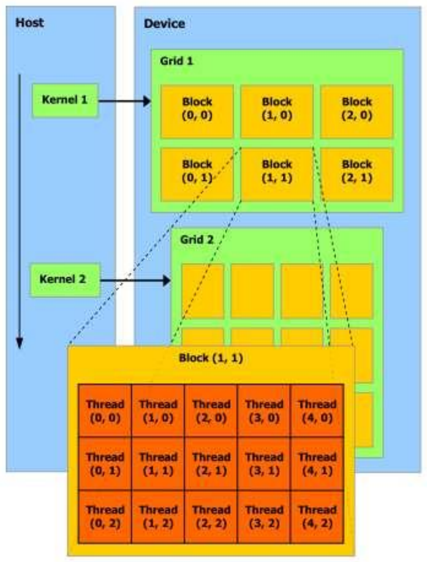


Figure 3.2: Threads grouped into block [38]

CUDA enhances the programmability and flexibility for general-purpose computation on GPU. Experimental results also show that, with the assistance of GPU, the processing time is several times faster than that of using CPU only [39].

The experimental results indicate that these parallel algorithms can provide a significant improvement in the computation efficiency with an overall speed up ratio for high definition video sequences, compared with the original serial algorithm implemented on CPU.

This chapter has the proposal in detail for the implementation of SAO with the help of CUDA programming. The next chapter throws light on the serial algorithm and explains the two different types of SAO in detail.

CHAPTER 4

Sample Adaptive Offset

SAO may use different offsets sample by sample in a region depending on the sample classification. SAO parameters are altered and adapted from region to region. Two SAO types that can satisfy the requirements of low complexity are adopted in HEVC. The two types are:

1. Edge Offset (EO) and
2. Band Offset (BO).

For EO, the sample classification is based on comparison between current samples and neighboring samples.
For BO, the sample classification is based on sample values. Please note that each color component may have its own SAO parameters [53]. To achieve low encoding latency and to reduce the buffer requirement, the region size is fixed to one CTB. To reduce side information, multiple CTUs can be merged together to share SAO parameters [54].

The best coding efficiency could be achieved by a picture-based region partitioning method [78], which would, however, introduce additional encoding latency. In order to achieve low encoding latency and reduce the buffer requirement, the size of the region can be fixed and set as small as one coding tree unit (CTU). Multiple CTUs can share the same SAO parameters by region merging [80] to reduce side information. In HEVC, a CTU consists of three coding tree blocks (CTBs) of color components, and each color component can have its own SAO offsets and share the same EO/BO type for chroma components [79]. Please also note that SAO not only is useful in HEVC but also can be applied on top of AVC and other prior video coding standards.

Let us discuss how offsets are calculated using each of these types.

4.1 Edge Offset

A good design is when there is a reasonable balance between complexity and coding efficiency. Edge offset uses four 1-D directional patterns for sample classification as shown in Fig. 4.1:

- Horizontal
- Vertical
- 135° diagonal
- 45° diagonal

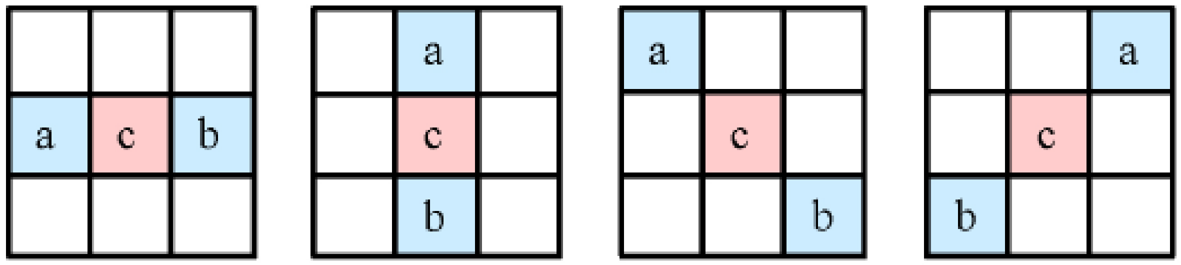


Fig. 4.1: Four 1-D directional patterns for EO sample classification: horizontal (EO class = 0), vertical (EO class = 1), 135° diagonal (EO class = 2), and 45° diagonal (EO class = 3). [29]

In Fig. 4.1 label “c” represents a current sample and the labels “a” and “b” represent two neighboring samples. According to these patterns, four EO classes are specified, and each EO class corresponds to one pattern. On the encoder side, only one EO class can be selected for each CTB that enables EO. Based on rate-distortion optimization, the best EO class is sent in the bit stream as side information. Since the patterns are 1-D, the results of the classifier do not exactly correspond to extreme samples. For a given EO class, each sample inside the CTB is classified into one of five categories. The current sample value, labeled as “c,” is compared with its two neighbors along the selected 1-D pattern. The classification rules for each sample are summarized in Table 4.1.

Category	Condition
1	c < a && c < b
2	(c < a && c==b) ||(c == a && c < b)
3	(c > a && c==b) ||(c == a && c > b)
4	c > a && c > b
0	None of the above

Table 4.1: Sample Classification Rules for Edge Offset [1]

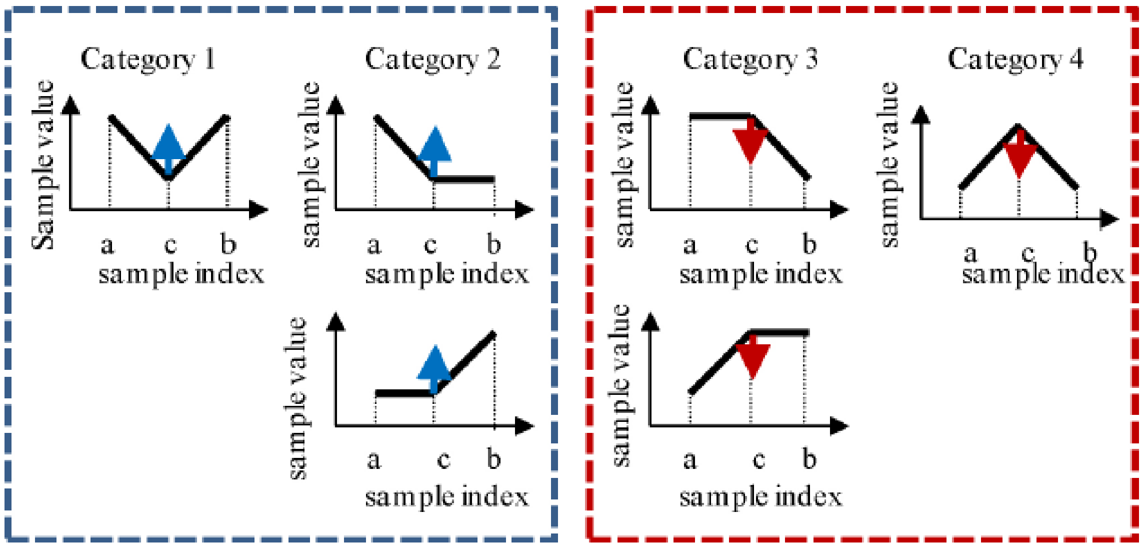


Fig. 4.2: Classification of EO based on table 4.1. Positive offsets for EO categories 1 and 2 and negative offsets for EO categories 3 and 4 result in smoothing [29]

Categories 1 and 4 are associated with a local valley and a local peak along the selected 1-D pattern, respectively. Categories 2 and 3 are associated with concave and convex corners along the selected 1-D pattern, respectively. If the current sample does not belong to EO categories

1–4, then it is category 0 and SAO is not applied. The meanings of edge offset signs are illustrated in Fig. 4.2 and explained as follows. Positive offsets for categories 1 and 2 result in smoothing since local valleys and concave corners become smoother, while negative offsets for these categories result in sharpening. On the other hand, the meanings are opposite for categories 3 and 4, where negative offsets result in smoothing and positive offsets result in sharpening.

Fig. 4.3 shows the well-known Gibbs phenomenon. It can be used to simulate a few video compression artifacts, especially the ringing artifacts. The horizontal axis and the vertical axis are not explicitly shown but are used to denote the sample position along a 1-D path and the sample value, respectively. The dotted curve represents the original samples, while the solid curve represents the reconstructed samples by discarding high frequencies of the original samples. The local peaks, convex corners, concave corners, and local valleys are painted as solid circles, while none-of-the-above samples are painted as empty circles. It can be seen that if we apply negative offsets to local peaks and convex corners, and positive offsets to concave corners and valleys, the distortion can be effectively reduced.

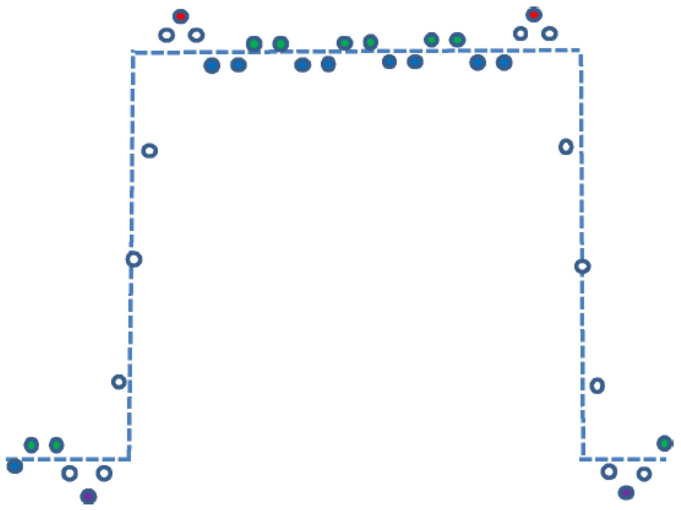


Figure 4.3: Gibbs Phenomenon where the dotted curve is the original samples and the solid curves is the reconstructed samples [29]

4.2 Band Offset

BO implies one offset is added to all samples of the same band. The sample value range is equally divided into 32 bands. For 8-bit samples ranging from 0 to 255, the width of a band is 8, and sample values from 8*k* to 8*k* + 7 belong to band *k*, where *k* ranges from 0 to 31. The average difference between the original samples and reconstructed samples in a band (i.e., offset of a band) is signaled to the decoder. There is no constraint on offset signs.

Only offsets of four consecutive bands and the starting band position are signaled to the decoder [56], [57]. The number of signaled offsets in BO was decided to be reduced from 16 to 4 and is now equal to the number of signaled offsets in EO for reducing the line buffer requirement, which will be described in Section 5.3. Another reason of selecting only four bands is that the sample range in a region can be quite limited after the regions are reduced from picture quad tree partitions to CTBs, especially for chroma CTBs, as an example shown in Fig. 4.4.

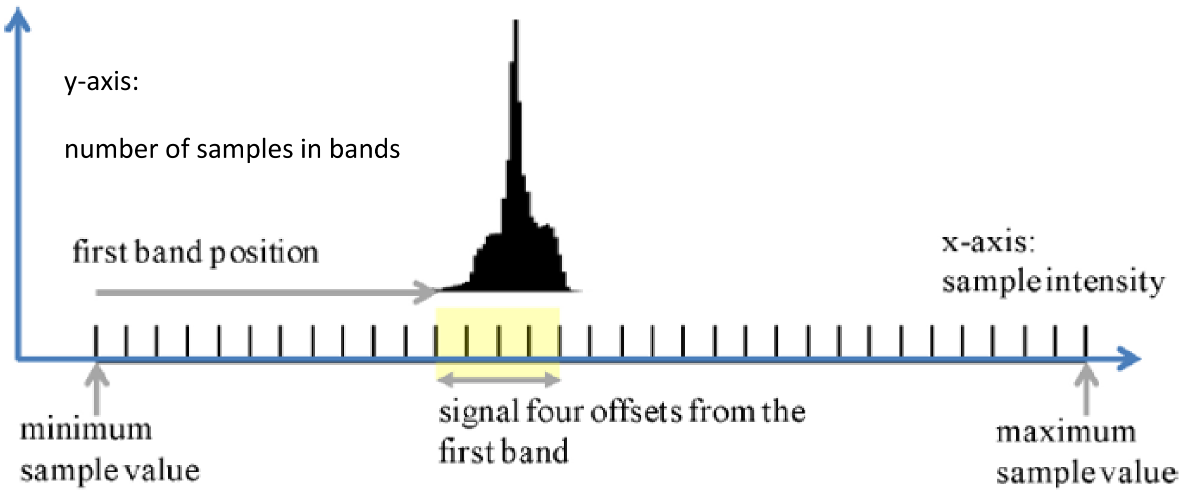


Figure 4.4: Example of sample distribution in a CTB, where BO sends the offsets of four consecutive bands. [29]

Figure 4.5 can be used to explain why BO works in a few circumstances. The horizontal axis and the vertical axis are not explicitly shown but are used to denote the sample position and the sample value, respectively.

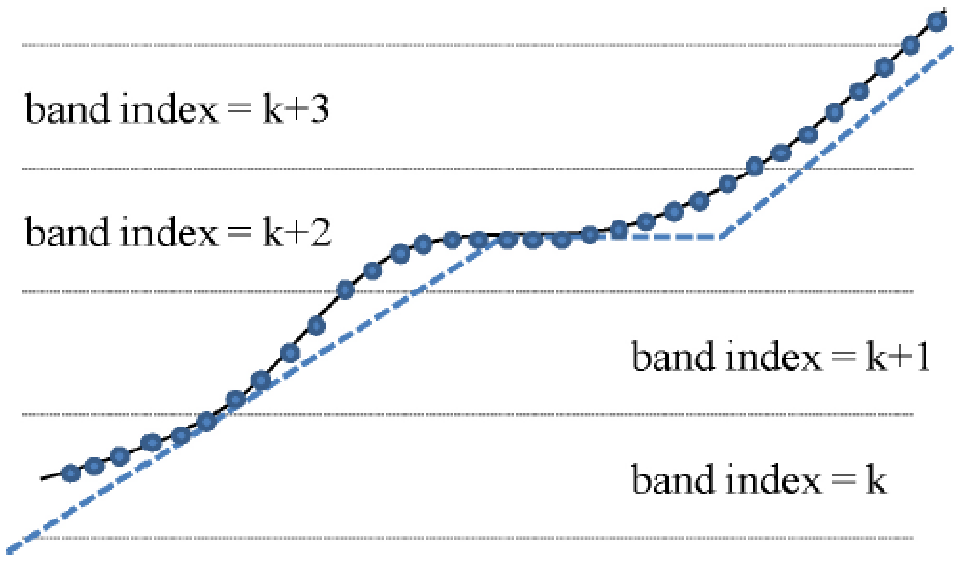


Figure 4.5. Example of BO, where the dotted curve is the original samples and the solid curve is the reconstructed samples. [29]

The dotted curve is the original samples, while the solid curve is the reconstructed samples, which may be corrupted by quantization errors of prediction residues and phase shifts due to coded motion vectors deviating from the true motions. In this example, the reconstructed samples are shifted to the left of the original samples, which systematically results in negative errors that can be corrected by BO for bands *k*, *k* + 1, *k* + 2, and *k* + 3.

4.3 Syntax Design

In the sequence parameter set (SPS), one syntax element, sample–adaptive–offset–enabled –flag, is used to indicate whether SAO is enabled in the current video sequence. In the slice header, two syntax elements, slice–sao–luma–flag and slice–sao–chroma–flag, indicate whether SAO is enabled for luma and chroma, respectively, in the current slice. The current SAO encoding algorithm can be configured as CTU-based for low-delay applications. Syntax-wise, the basic unit for adapting SAO parameters is always one CTU. If SAO is enabled in the current slice, SAO parameters of CTUs are inter leaved into the slice data. The SAO data of one CTU is placed at the beginning of the CTU in the bitstream. The CTU-level SAO parameters contain SAO merging information, SAO type information, and offset information.

4.3.1 SAO Merging

For each CTU, there are three options: reusing SAO parameters of the left CTU by setting the syntax element sao−merge−left−flag to true, reusing SAO parameters of the above CTU by setting the syntax element sao−merge−up−flag to true, or transmitting new SAO parameters. Please note that the SAO merging information is shared by three color components. As shown in Fig. 4.6, a CTU includes the corresponding luma CTB, C_b CTB, and C_r CTB. When the current CTU selects SAO merge-left or SAO merge-up, all SAO parameters of the left or above CTU are copied for reuse, so no more information is sent. This CTU-based SAO information sharing [19] can reduce side information effectively.

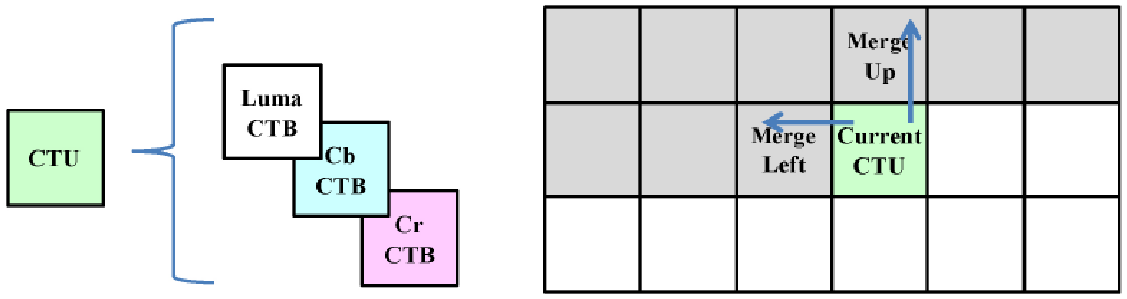


Figure 4.6: CTU consists of CTBs of three color components, and the current CTU can reuse SAO parameters of the left or above CTU.

4.3.2 SAO Type and Offsets

If a current CTU does not merge with a neighboring CTU, the rest SAO information of the current CTU will be signaled, as shown in fig. 4.7. All luma syntax elements are first sent and followed by all C_b syntax elements and then all C_r syntax elements. For each color component, the SAO type is first transmitted (sao−type−idx−luma or sao−type−idx−chroma) to indicate EO, BO, or OFF. If BO is selected, the starting band position (sao−band−position) is signaled; else if

EO is selected, the EO class (sao−eo−class−luma or sao−eo−class−chroma) is signaled. For both BO and EO, four offsets are transmitted. Please note that C_b and C_r share the SAO type (sao−type−idx−chroma) and EO class (sao−eo−class−chroma) to further reduce the side information and to speed up SAO by achieving more efficient memory access in certain platforms [20], so these syntax elements are coded for C_b only and do not have to be sent for C_r.

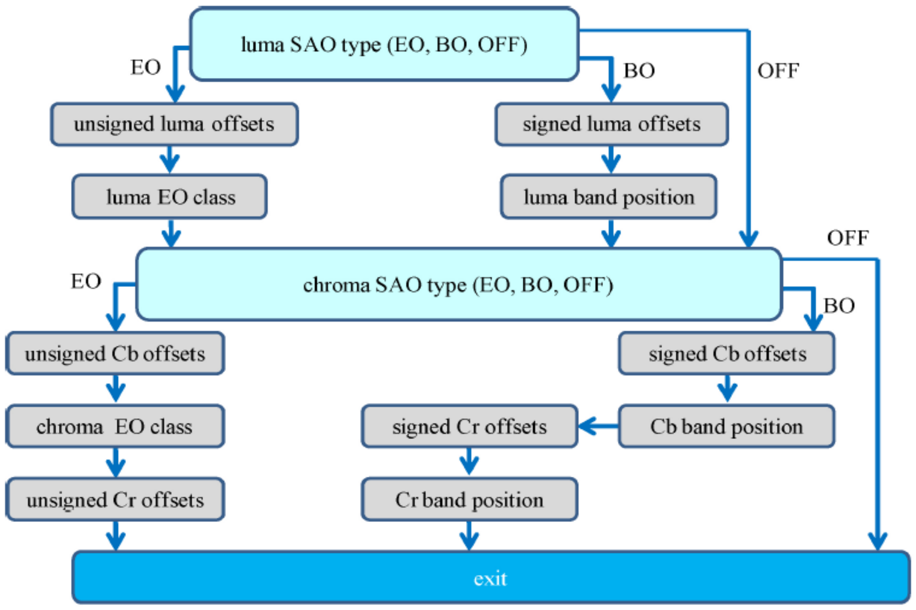


Figure 4.7: Illustration of coding the rest CTU-level SAO information when the current CTU is not merged with the left or above CTU.

This chapter describes in detail about the types of SAO- Edge Offset and Band Offset. It explains the categorization and classification of Edge Offset and Band Offset. In the next chapter, statistical estimation and finding the best offset for each of the categories and classification of EO and BO. These steps will be discussed in both serial and parallel implementation.

CHAPTER 5

Serial Implementation of SAO

There are six implementations to find the offset values. The first three are for both encoders and decoders, while the rest are encoder-only. [29]

5.1 *Fast Edge Offset Sample Classification*

Although the sample classification rules of EO in Table 4.1 seem nontrivial, Table 4.1 is mainly used for easy explanation of the concepts, and the EO sample classification does not need to be implemented in that way. A fast algorithm can be described as follows [29]:

$$\text{sign3}(x) = (x > 0)?(+1) : ((x == 0)?(0) : (-1)) \qquad (1)$$

$$\text{edge}Idx = 2 + \text{sign3}(c - a) + \text{sign3}(c - b) \qquad (2)$$

$$\text{edge}Idx\text{2category}[\,] = \{1, 2, 0, 3, 4\} \qquad (3)$$

$$\text{category} = \text{edge}Idx\text{2category}[\text{edge}Idx]. \qquad (4)$$

In (2), *c* is the current sample, and *a* and *b* are the two neighboring samples, as shown in Figure 4.1 and Table 4.1. Besides the fast calculations in (1)–(4), data reuse between samples can be further applied for the next sample classification. For example, assuming the EO class is 0 (i.e., using 1-D horizontal pattern) and the samples in the CTB are processed in the raster scan order, the sign3(c–a) of the current sample does not have to be calculated and can be directly set to the −sign3(*c*–*b*) of the neighboring sample to the left. Likewise, the sign3(*c*–*b*) of the current sample can be reused by the neighboring sample to the right.

5.2 *Fast Band Offset Sample Classification*

The sample range is equally divided into 32 bands in BO. Since 32 is equal to two to the power of five, the BO sample classification can be implemented as using the five most significant bits of each sample as the classification result. In this way, the complexity of BO becomes very low, especially in hardware that only needs wire connections without logic gates to obtain the classification result from the sample value.

5.3 *Line Buffer Requirement*

CTU-based processing is commonly adopted in practical implementations. CTUs are encoded or decoded one by one in raster scan order. In Deblocking Filter, vertical filtering across a horizontal boundary requires to read four luma samples, two C_b samples, and two C_r samples and write three luma samples, one C_b sample, and one C_r sample on both sides of the boundary. Hence, when a current CTU is being processed, DF has not finished processing the bottom sample rows of the above CTU. Since SAO is after DF, the bottom sample rows of the above CTU have not been processed by SAO either. In order to apply CTU-based processing, DF and SAO need line buffers. Assume that DF uses N sample line buffers to store horizontally deblocked samples of the bottom N rows in the above CTB, where N is four for luma and two for C_b and C_r. Since the Nth row above the horizontal CTB boundary will not be modified by the vertical deblocking, the SAO processes for the $(N + 1)_{th}$ row above the horizontal CTB boundary can be done before the current CTB comes. However, the bottom N rows of the above CTB still have to wait for the current CTB for applying DF and SAO. When the current CTB comes and if the above CTB selects EO with the EO class larger than zero, the SAO processes for the N_{th} row above the boundary needs to use the $(N + 1)_{th}$ row above the boundary. Intuitively, we can store samples of the $(N + 1)_{th}$ row in the SAO line buffer.

5.4 *Fast Distortion Estimation*

During the rate-distortion optimization [29] on the encoder side, distortions between original samples and reconstructed samples have to be calculated for many times. A straightforward implementation for SAO would need to add offsets to pre-SAO samples to generate post-SAO samples and then calculate the distortion between original samples and post-SAO samples. To reduce the memory access and operations, a fast distortion estimation method [29] can be implemented as follows. Let k, $s(k)$, and $x(k)$ be sample positions, original samples, and pre-SAO samples, respectively, where *k* belongs to *C* and *C* is the set of samples that are inside a CTB and belong to a specified SAO type (i.e., BO or EO), a specified starting band position or EO class, and a specified band or category. The distortion between original samples and pre-SAO samples can be described in the following equation [29]:

$$D_{\text{pre}} = \sum_{k \in C} (s(k) - x(k))^2 . \qquad (5)$$

The distortion between original samples and post-SAO samples can be described in the following equation:

$$D_{\text{post}} = \sum_{k \in C} (s(k) - (x(k) + h))^2 . \qquad (6)$$

In (6), *h* is the offset for the sample set. The delta distortion is defined as follows:

$$\Delta D = D_{\text{post}} - D_{\text{pre}} = \sum_{k \in C} \left(h^2 - 2h\,(s(k) - x(k))\right) = Nh^2 - 2hE. \qquad (7)$$

In (7), *N* is the number of samples in the set, and *E* is the sum of differences between original samples and pre-SAO samples as defined as follows [29]:

$$E = \sum_{k \in C} (s(k) - x(k)) . \qquad (8)$$

Please note that sample classification and (8) can be calculated right after pre-SAO samples are available during the DF processes. Thus, *N* and *E* can be calculated only once and stored.

Next, the delta rate-distortion cost is defined as follows [29]:

$$\Delta J = \Delta D + \lambda R \qquad (9)$$

In (9), λ is the Lagrange multiplier, and R represents the estimated bits of side information. For a given CTB with a specified SAO type (i.e., BO or EO), a specified starting band position or EO class, and a specified band or category, a few h values (i.e., offsets) close to the value of E/N are tested, and the offset that minimizes ΔJ will be chosen. After offsets of all bands or categories are chosen, we can add up the ΔJ of each of the 32 bands for BO or each of the five categories for EO to obtain the delta rate distortion cost of the entire CTB, where the distortions of the BO bands using zero offsets implicitly and the EO category 0 can be precalculated by (5) and stored for reuse. When the delta cost of the entire CTB is negative, SAO can be enabled for the CTB. Similarly, the best SAO type and the best starting position or EO class can be found using the fast distortion estimation.

5.5 *Slice-Level On/Off Control*

In the HEVC reference software common test conditions [30], hierarchical quantization parameter (QP) settings for each group of pictures (GOP) are often used. As an example, in the random access condition, the GOP size is eight. Any pictures with picture order count (POC, i.e., display order) equal to $8k$ belong to depth 0, any picture with POC equal to $(8k + 4)$belongs to depth 1, any picture with POC equal to $(8k +2)$ or $(8k + 6)$ belongs to depth 2,and any picture with POC equal to $(8k + 1)$, $(8k + 3)$, $(8k + 5)$, or $(8k + 7)$ belongs to depth 3,where k is a nonnegative integer. A picture with a larger depth will be given a higher QP. A slice-level on/off decision algorithm [31], [32] is provided as follows. For depth 0 pictures, SAO is always enabled in the slice header. Given a current processing picture with a nonzero depth N, the previous picture is set to the last picture of depth $(N -1)$ in the decoding order. If the previous picture disables SAO for more than 75% of CTBs, the current picture will early terminate the SAO encoding process and disable SAO in all slice headers. Please note that luma and chroma can be turned on or off independently in the slice header.

5.6 *Considerations for Right and Bottom Samples in the CTU*

In the reference software, SAO parameters are estimated for each CTU at the encoder. Since SAO is after DF, the SAO parameters cannot be precisely estimated until the deblocked samples are available. However, the deblocked samples of the right columns and the bottom rows in the current CTU may be unavailable because the right CTU and the below CTU may have not yet been reconstructed in a CTU-based encoder. In order to consider this fact for practical CTU-based encoders, two options are provided. The first option [19] avoids using the bottom three luma sample rows, the bottom one C_b sample row, the bottom one C_r sample row, the rightmost four luma sample columns, the rightmost two C_b sample columns, and the rightmost two C_r sample columns in the current CTU during SAO parameter estimation on the encoder side. It does not suffer noticeable coding efficiency loss when the CTU size is 64×64 in luma. Nevertheless, when the CTU size is smaller, the percentage of unused samples in the CTU becomes higher and might cause considerable coding efficiency loss. Hence, the second option [32] uses predeblocked samples to replace the unavailable deblocked samples in the current CTU during SAO parameter estimation, which can reduce the coding efficiency loss caused by the first option for smaller CTU sizes.

This chapter explained the classification and estimation of EO and BO. After classification the fast distortion estimation is explained that finds the best offset for every category of every class in EO or for every band in case of BO. This chapter also describes the line buffer requirement and slice on/off control. In the next chapter, the parallel implementation of the same process in explained in detail.

CHAPTER 6

Parallel Implementation of SAO

GPU Implementation of SAO:

The overall parallel scheme for realizing the SAO process is illustrated in figure 6.1 that includes five parallel processing modules corresponding to the five computational processes described in chapter 5. The parallel algorithm will be described in detail in the next few sections.

A. ***Parallel Algorithm for Sample Classification and Statistics Collection***
B. ***Parallel Algorithm for obtaining the Best EO Parameters.***
C. ***Parallel Algorithm for Obtaining the Best BO Parameters***
D. ***Parallel Algorithm for SAO Merging***
E. ***Parallel Algorithm for SAO filtering***

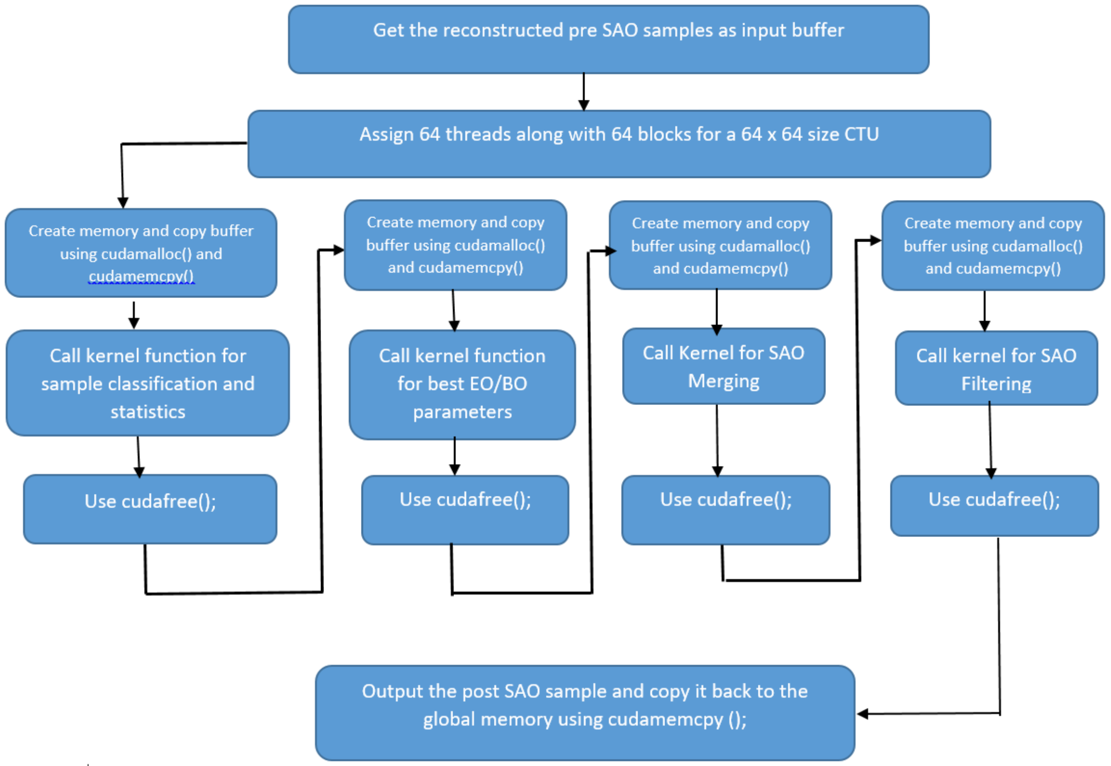

Get the reconstructed pre SAO samples as input buffer
Assign 64 threads along with 64 blocks for a 64 x 64 size CTU
Create memory and copy buffer using cudamalloc() and cudamemcpy()
Call kernel function for sample classification and statistics
Use cudafree();
Create memory and copy buffer using cudamalloc() and cudamemcpy()
Call kernel function for best EO/BO parameters
Use cudafree();
Create memory and copy buffer using cudamalloc() and cudamemcpy()
Call Kernel for SAO Merging
Use cudafree();
Create memory and copy buffer using cudamalloc() and cudamemcpy()
Call kernel for SAO Filtering
Use cudafree();
Output the post SAO sample and copy it back to the global memory using cudamemcpy ();

Figure 6.1: The overall parallel scheme of SAO

6.1 Parallel Algorithm for Sample Classification and Statistics Collection

As pointed by Choi and Joo [58], sample classification and statistics collection dominate the SAO encoding time. In order to significantly accelerate the process, a parallel sample classification and statistics collection method is designed in this paper.

The sample classification and statistics collection of EO are based on comparison between current and neighboring pre-SAO samples. This process can be operated on all CTBs simultaneously. And since all EO classes are independent of each other, we can process all the sample classifications on the EO class-level in parallel. In order to further accelerate this process, we divide each CTB into $N_{sub\text{-}CTB}$ sub-CTBs to perform the sample classification and statistics collection for all the sub-CTBs simultaneously and then accumulate the results to complete the process for each CTB. Let N_{CTB} be the total number of CTBs in an image, N_{EO} the number of EO classes, and *Num* the number of samples in one sub-CTB, respectively. The parallel algorithm for realizing the sample classification and statistics collection of EO is given below.

```
begin
for i=1to NCTB do in  parallel
    for j=1 to NEO do in  parallel
        for k=1 to Nsub-CTB do in parallel
            for m=1 to 5 do /* five categories */
                Eijkm     0; Nijkm    0;    /* initialize Eijkm and Nijkm */
            end for
            for n=1 to Num do
                (i) determine the category of rijkn;
                    /* rijkn is the n-th sample in the k-th sub-CTB of the
                    j-th class in the i-th CTB */
                (ii)    if (the category of rijkn is m) compute Eijkm based on (1); Nijkm     Nijkm+1;
                        end if
            end for
        end for
    end for

    for p=1 to NEO do in  parallel
        for q=1 to 5 do in parallel      /* five categories */
            Eipq     0; Nipq    0;  /* initialize Eipq and Nipq */
            for l=1 to  Nsub-CTB do
            (i) Eipq    Eipq+ Eiqpl ;   /* Eiqpl= Eijkm */
            (ii)    Nipq    Nipq+ Niqpl ;   /* Niqpl= Nijkm */
            end for
        end for
    end for
end for
end
```

Figure 6.2: Parallel Algorithm 6(a) [1]

In *Parallel Algorithm (fig. 6.2)*, *Nipq* and *Eipq* are the number of samples and the sum of differences of the *q-th* category for the *p-th* class in the *i-th* CTB, respectively. N_{ijkm} and E_{ijkm} are the statistics of the corresponding each sub-CTB. We can see that this parallel algorithm consists of two main steps. The first step has $N_{CTB}N_{sub\text{-}CTB}N_{EO}$ parallel branches with each one for computing the statistics for each EO class of each sub-CTB in each CTB and the parallel degree is $N_{CTB}N_{sub\text{-}CTB}N_{EO}$. The second step contains $5N_{CTB}N_{EO}$ parallel branches with each branch for completing the statistics computation for each CTB and the parallel degree is $5\ N_{CTB}N_{EO}$. For 1920×1080 (i.e. HD) video sequences, if we set the size of CTB to 16×16 and $N_{sub\text{-}CTB}$ to 16, N_{CTB} is 8160, the parallel degree of the first step is 522240 and the parallel degree of the second step is 163200.

In the same way as the design of *Parallel Algorithm* shown in *fig. 6.2*, the sample classification and statistics collection of BO is also performed in parallel for each CTB. The parallel algorithm for realizing this operation procedure is given below (where N_{band} denotes the number of BO bands).

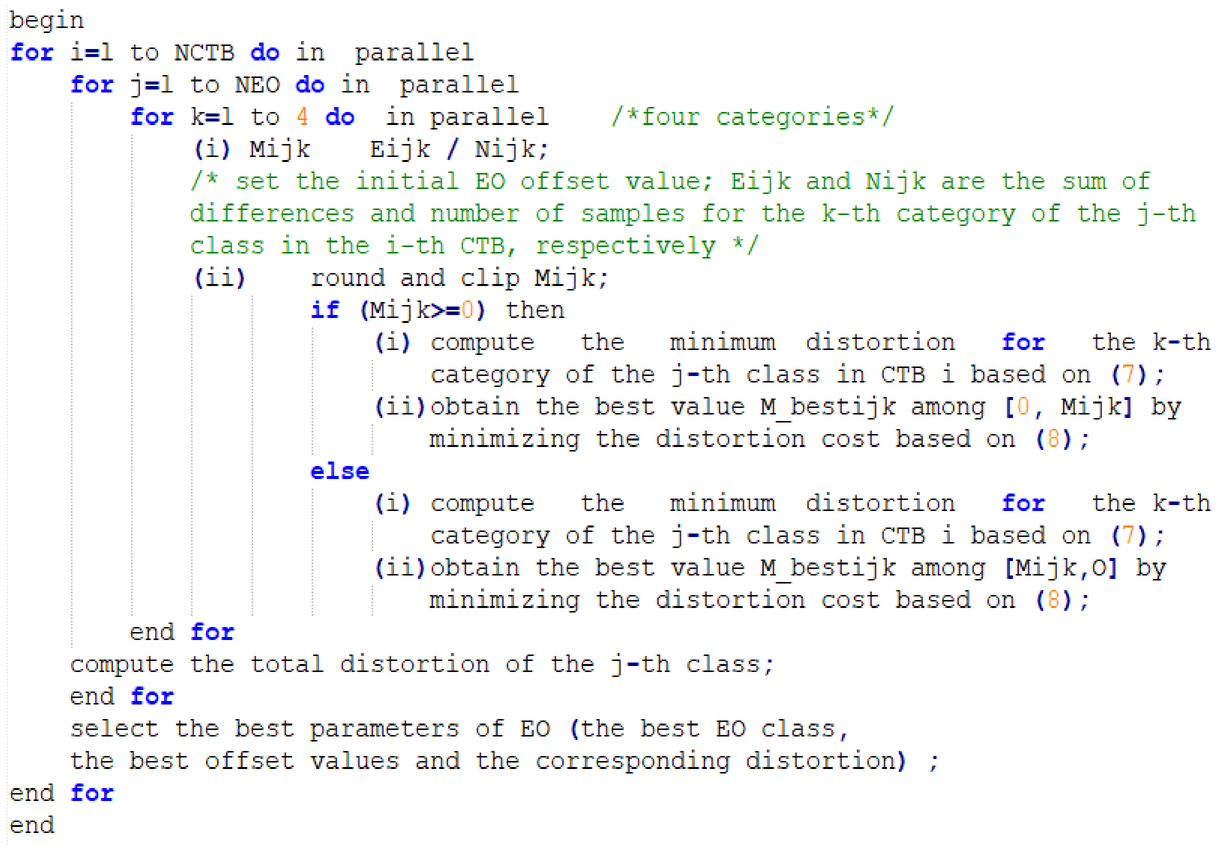
```
begin
for i=lto NCTB do in parallel
    for j=l to Nsub-CTB do in parallel
        for k=l to Nband do
            Eijk      0; Nijk     0;  /* initialize Eijk and Nijk */
        end for

        for m=l to Num do
        (i) determine the band of rijm;
            /* rijm is the m-th sample in the j-th sub-CTB in the
            i-th CTB */
        (ii)    if (the band of rijm is k)
                compute Eijk based on (1); /* accumulate Eijk */
                Nijk      Nijk+l;    /* accumulate Nijk */
                end if
        end for
    end for

    for p=l to Nband do in parallel
    Eip      0; Nip     0;     /* initialize Eip and Nip */
        for q=l to  Nsub-CTB do
        (i) Eip      Eip+ Eiqp ; /* Eiqp= Eijk */
        (ii)    Nip     Nip+ Niqp ; /* Niqp= Nijk */
        end for
    end for
end for
end
```

Figure 6.3: Parallel Algorithm 6(b) [1]

In *Parallel Algorithm fig. 6.3*, N_{ip} and E_{ip} are the number of samples and the sum of differences of the *p-th* band in the *i-th* CTB, respectively and N_{ijk} and E_{ijk} are the statistics of the corresponding each sub-CTB. We can see that this parallel algorithm also contains two main computation steps. The first step has $N_{CTB}N_{sub\text{-}CTB}$ parallel branches with each branch for computing the BO statistics for each sub-CTB in each CTB and the parallel degree is $N_{CTB}N_{sub\text{-}CTB}$. The second step contains $N_{CTB}N_{band}$ parallel branches with each one for completing the statistics computation for each CTB and the parallel degree is $N_{CTB}N_{band}$.

6.2 Parallel Algorithm for Obtaining the Best EO Parameters

The objective of this subsection is to design a parallel algorithm to compute the best EO parameters including the best offset values and corresponding distortions for each category in different EO classes of each CTB parallelly. The parallel algorithm is presented below.



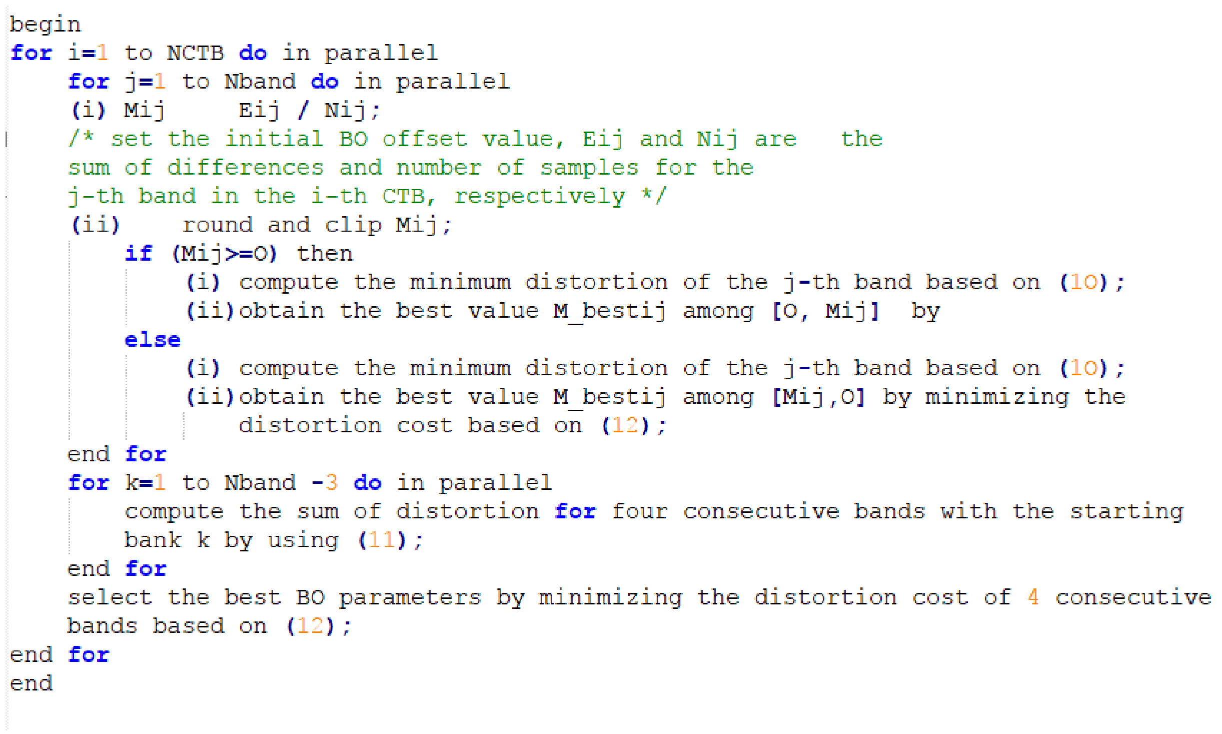
```
begin
for i=1 to NCTB do in  parallel
    for j=1 to NEO do in  parallel
        for k=1 to 4 do  in parallel    /*four categories*/
            (i) Mijk    Eijk / Nijk;
            /* set the initial EO offset value; Eijk and Nijk are the sum of
            differences and number of samples for the k-th category of the j-th
            class in the i-th CTB, respectively */
            (ii)    round and clip Mijk;
                    if (Mijk>=0) then
                        (i) compute   the   minimum  distortion   for   the k-th
                            category of the j-th class in CTB i based on (7);
                        (ii)obtain the best value M_bestijk among [0, Mijk] by
                            minimizing the distortion cost based on (8);
                    else
                        (i) compute   the   minimum  distortion   for   the k-th
                            category of the j-th class in CTB i based on (7);
                        (ii)obtain the best value M_bestijk among [Mijk,0] by
                            minimizing the distortion cost based on (8);
        end for
    compute the total distortion of the j-th class;
    end for
    select the best parameters of EO (the best EO class,
    the best offset values and the corresponding distortion) ;
end for
end
```

Figure 6.4: Parallel Algorithm 6(c) [1]

In *Parallel Algorithm (fig.6.4)*, M_{ijk} and M_best_{ijk} represent the candidate and the best offset value of the *k-th* category for the *j-th* EO class in the *i-th* CTB, respectively. We can see that this algorithm contains three parallel computation processes: the first process contains $4N_{CTB}N_{EO}$ parallel branches with each one for computing the best offset value and distortion for each category of an EO class in each CTB parallelly, the second process contains $N_{CTB}N_{EO}$ parallel branches with each one for calculating the total distortion of 4 categories for each EO class in each CTB, and the third process contains N_{CTB} branches with each one for obtaining the best EO parameters for each CTB parallelly. It is obvious that, for the three parallel processes, the parallel degrees are $4N_{CTB}N_{EO}$, $N_{CTB}N_{EO}$ and N_{CTB}, respectively. For 1920×1080 video sequences, if we set the size of CTB to 16×16, N_{CTB} is 8160, and the parallel degrees are 130560, 32640 and 8160, respectively.

6.3 Parallel Algorithm for Obtaining the Best BO Parameters

Let M_{ij} and M_best_{ij} be the candidate and the best offset value of the *j-th* band in the *i-th* CTB, respectively. The parallel algorithm for computing the best BO parameters is given below under the condition that the sample classification and statistics collection of each BO band in each CTB have been completed by using *Parallel Algorithm* as shown below.



```
begin
for i=1 to NCTB do in parallel
    for j=1 to Nband do in parallel
    (i) Mij     Eij / Nij;
    /* set the initial BO offset value, Eij and Nij are   the
    sum of differences and number of samples for the
    j-th band in the i-th CTB, respectively */
    (ii)    round and clip Mij;
        if (Mij>=0) then
            (i) compute the minimum distortion of the j-th band based on (10);
            (ii)obtain the best value M_bestij among [0, Mij]  by
        else
            (i) compute the minimum distortion of the j-th band based on (10);
            (ii)obtain the best value M_bestij among [Mij,0] by minimizing the
                distortion cost based on (12);
    end for
    for k=1 to Nband -3 do in parallel
        compute the sum of distortion for four consecutive bands with the starting
        bank k by using (11);
    end for
    select the best BO parameters by minimizing the distortion cost of 4 consecutive
    bands based on (12);
end for
end
```

Figure 6.5: Parallel Algorithm 6(d) [1]

It can be seen that there are three computation steps in *Parallel Algorithm fig. 6.5*. The parallel degrees are $N_{CTB}N_{band}$, $N_{CTB}(N_{band}$-3) and N_{CTB}, respectively.

6.4 Parallel Algorithm for SAO Merging

After having obtained the best parameters of EO and BO by *Parallel Algorithm 6(c) (fig. 6.4)* and *Parallel Algorithm 6(d) (fig. 6.5)*, we can select the best SAO parameters for each color component of the current CTU. Then the SAO merging can be adopted to allow the current CTU to reuse SAO parameters of the left or up CTU for selecting the final best SAO parameters. For performing the SAO merging, it is obvious that we cannot determine the final best SAO parameters of the current CTU until the SAO parameters of the left and up CTUs have been set. On the other hand, it can be seen that there is no dependency between the CTUs located in the same diagonal direction. Therefore we can process these CTUs in parallel. Here, a diagonal parallel scheme is designed to implement the SAO merging. As shown in Fig. 6.6, the parallel algorithm is performed to process the CTBs in the same diagonal direction iteratively as the label increases and the total iteration number is computed by (10). For each iteration, the CTUs with the same label are processed in parallel and the total number of CTUs processed at the *i-th* iteration is computed by (12), where *Wimage* and *Himage* are the width and height of the image; *WCTU* and *HCTU* are the width and height of the CTU. The *Parallel Algorithm (fig. 6.7) is* for realizing this parallel scheme as follows:

$$N_{diag} = \frac{W_{image}}{W_{CTU}} + \frac{H_{image}}{H_{CTU}} - 1$$

(10)

$$t = \min(\frac{W_{image}}{W_{CTU}}, \frac{H_{image}}{H_{CTU}})$$

(11)

$$N\_diag_{CTU,i} = \begin{cases} i+1, 0 \le i < t \\ t, t \le i < N_{diag} - t \\ N_{diag} - i, N_{diag} - t \le i < N_{diag} \end{cases} \quad \textbf{(12)}$$

1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	
4	5	6	7	8	9		
5	6	7	8	9			
6	7	8	9				
7	8	9					
8	9						

Figure 6.6: The diagonal parallel algorithm for SAO merging

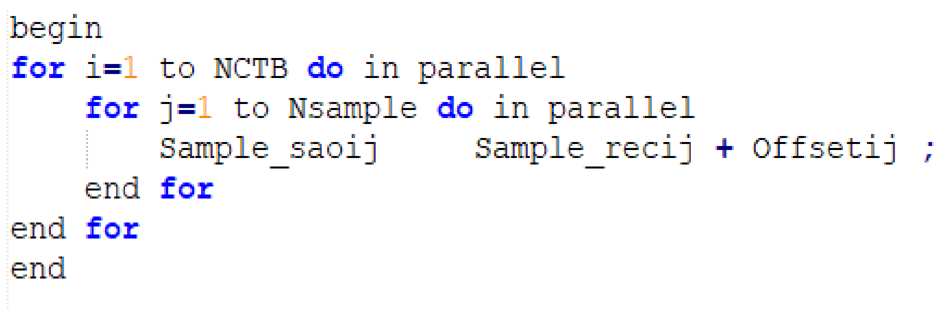
```
begin
for i=1 to Ndiag do
    compute  N_diagCTU,i by using (15);
    for j=1 to N_diagCTU,i do in parallel
    (i)  reuse the parameters of up CTU to compute the ΔJup;
         reuse the parameters of left CTU to compute the ΔJleft;
    (ii) select the best SAO parameters by obtaining the minimum one among ΔJup,
         ΔJleft and the distortion cost of the current CTU.
end for end for end
```

Figure 6.7: Parallel Algorithm 6(e) [1]

In Parallel Algorithm fig. 6.7, ΔJ_{up} and ΔJ_{left} are the corresponding distortions when we reuse the parameters of up and left CTUs on the current CTU. For the 1920×1080 video sequences, if we set the size of CTU to 16×16, N_{diag} is 187, the minimum parallel degree is 1 and the maximum parallel degree is 68. The average parallel degree is 43.64

6.5 Parallel Algorithm for SAO filtering

For each CTU, the best SAO parameters have been obtained by *Parallel Algorithm 6(e) (fig. 6.7)*. For a given luma or chroma CTB, the process of filtering for the current sample is irrelevant to neighboring samples and this computation process can be realized parallelly by using *Parallel Algorithm (fig. 6.8)* as described below.



```
begin
for i=1 to NCTB do in parallel
    for j=1 to Nsample do in parallel
        Sample_saoij     Sample_recij + Offsetij ;
    end for
end for
end
```

Figure 6.8: Parallel Algorithm 6(f)

In *Parallel Algorithm fig. 6.7*, *Nsample* is the total number of samples in luma or chroma CTB, *Sample_rec*$_{ij}$ and *Sample_sao*$_{ij}$ are the pre-SAO and post-SAO sample values, respectively, and *Offsetij* is the best offset value for the *j-th* sample in the *i-th* CTB. We can see that the parallel degree in this parallel algorithm is $N_{CTB}N_{sample}$ and the parallel granularity is in pixel-level for an image.

This chapter describes the parallel implementation of SAO. The steps mentioned in chapter 5 is rewritten in parallel using CUDA programming. The SAO filter has lot of computational steps that are independent of the neighboring samples which allow parallelism using GPUs. The pseudo code for these algorithms are explained in this chapter. In the following chapter, the results for serial and parallel implementation is compared and conclusions are drawn for the same.

CHAPTER 7

Results and Conclusions

7.1 Test Environment:

A. HM Code 16.9

Refer [82] for complete description of the HM test model. All the definitions of structures, enumerations and function declaration can be found in this documentation. Refer the user manual [51] that comes along with the software to run the HM code. The syntax to run the code in command prompt is given in readme files for both encoder and decoder.

B. CUDA Toolkit

This toolkit provides a development environment for creating high performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler and a runtime library to deploy your application [83].

GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning and graph analytics. For developing custom algorithms, we can use available integrations with commonly used languages and numerical packages as well as well-published development APIs. CUDA applications can be deployed across all NVIDIA GPU families available on the work station. Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to thousands of GPUs [83].

C. Visual Studio 2015

Microsoft Visual Studio is the Integrated Development environment (IDE) used to code, build and run the HEVC encoder and decoder. It provides a feature called Nsight which helps in running the code on GPUs. It provides profiling which helps us understand the memory usage, timing and tracking of data structures. This helps the developers to optimize the code easily

D. GPU hardware specification

4K Ultra HD Resolution Support	Yes
Cooling System	Air
Featured Technology	NVIDIA CUDA, NVIDIA G-SYNC, NVIDIA GameStream, NVIDIA SLI, DirectX 12
GPU Clock Speed	1708 megaHertz
Graphics Processing Unit (GPU)	NVIDIA GeForce GTX 1080
Video Memory Capacity	8 gigabytes
Video Memory Type	GDDR5X
GPU Boost Clock Speed	1847 megaHertz

E. Intel i-5 processor, 8GB RAM, Windows 10-64bit [88]

7.2 Test Sequences [87]

1. **BasketballPass_416x240_50.yuv**
2. **RaceHorses_832x480_30.yuv**
3. **SlideEditing_1280x720_30.yuv**

7.3 Metrics used

- PSNR=20 $\log_{10} \frac{MAX_f}{\sqrt{MSE}}$ (13)

  MAX_f is the maximum value of the sample.

- The formula used for $Y'C_bC_r$

$$\mathrm{PSNR}_{Y'C_bC_r} = \frac{6 \times \mathrm{PSNR}_{Y'} + \mathrm{PSNR}_{C_b} + \mathrm{PSNR}_{C_r}}{8}$$

(14)

- The format used is 4:2:0
- Speed up= (Encoding time in serial)/ (Encoding time in parallel) (15)

7.4 Serial code results

One of the reasons that HEVC provides better coding efficiency than prior standards is the extended transform size, up to 32 × 32. A large size transform provides better energy compaction; however, it tends to cause undesirable ringing artifacts. The following results show examples of test sequence to highlight these effects. As shown in the figure, SAO significantly improves the visual quality by suppressing the ringing artifacts near true edges.

The HM encoder is built based on a rate-distortion quantization process optimization. The JCT-VC common test conditions introduced a few configurations for encoder to be applied in tests. These configurations are [81]:

1. All **intra** (AI) All images are encoded with I slices.

2. **Random access** (RA): A pyramidal structure with a randomly selected picture approximately every one second is used as a picture reordering. Mentioned configuration simulates what we use in a broadcasting environment.

3. **Low-delay with B slices** (LB): In this mode, just the first frame is encoded using I slices and random selection of pictures and picture reordering are not used. This configuration simulates what we use in a videoconferencing environment.

7.4.1 BasketballPass_416x240_50.yuv

(Performed for 200 Frames)

QP	Encoding time with SAO Off (in seconds)	Encoding time with SAO On (in seconds)	YUV- PSNR with SAO Off (in dB)	YUV-PSNR with SAO On (in dB)
27	494.327	495.279	37.3066	38.5323
32	370.083	371.423	34.2664	35.7813
37	340.875	344.548	31.5472	32.7192

Table 7.1: Encoding time and PSNR of BasketballPass

Snapshot of frame 14

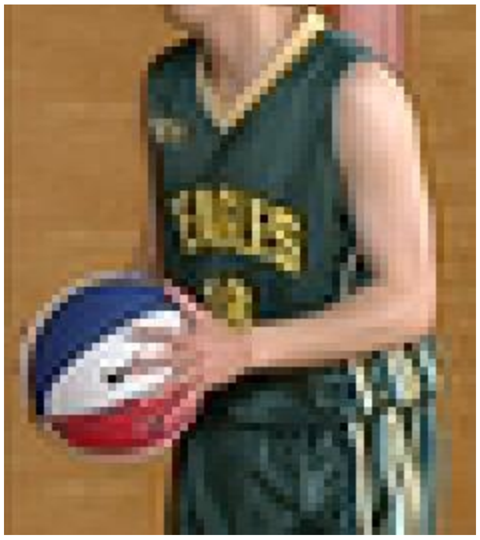


Fig. 7.1 BasketballPass_416x240_50.yuv Original Image

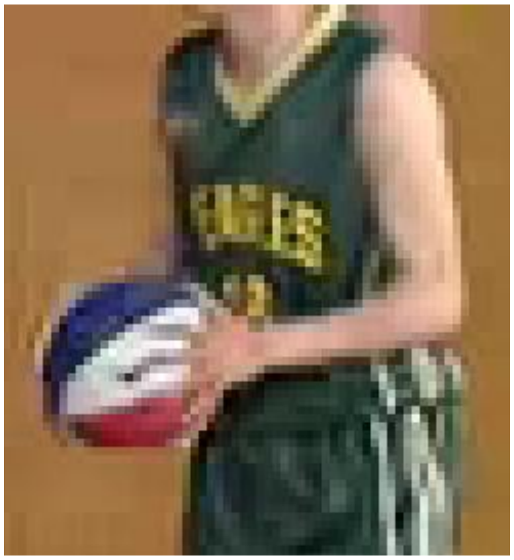


Fig. 7.2 BasketballPass_416x240_50.yuv SAO=0

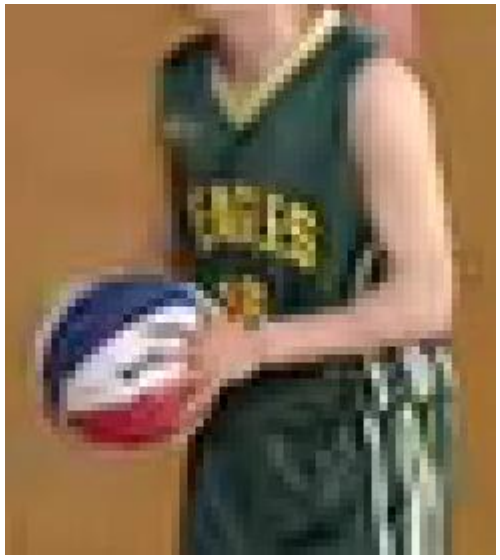


Fig.7.3 BasketballPass_416x240_50.yuv SAO=1

SAO=1 is SAO filter ON
SAO=0 is SAO filter OFF

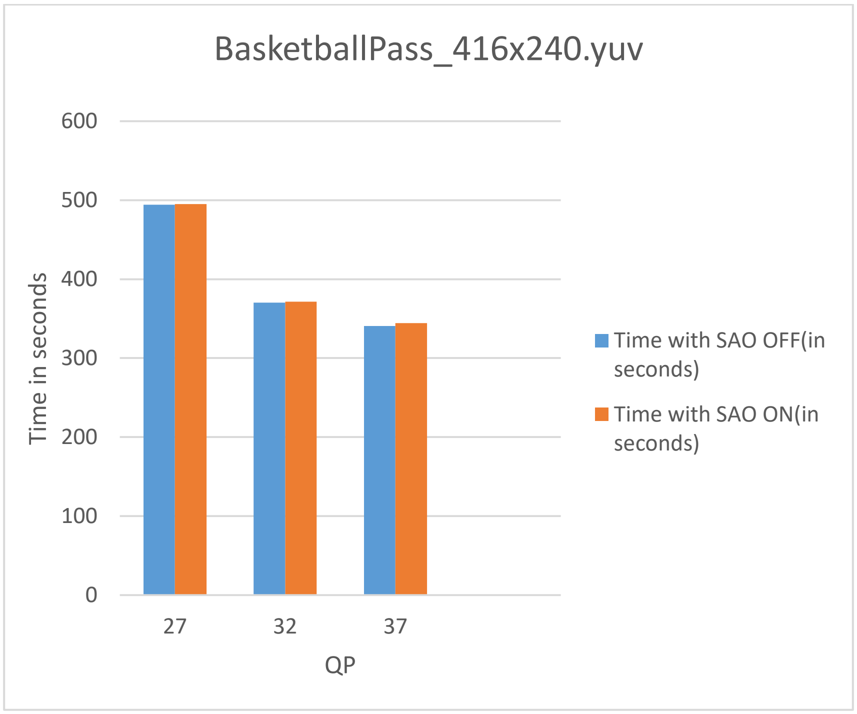


Figure 7.4: Graphical representation of Time vs QP for BasketballPass

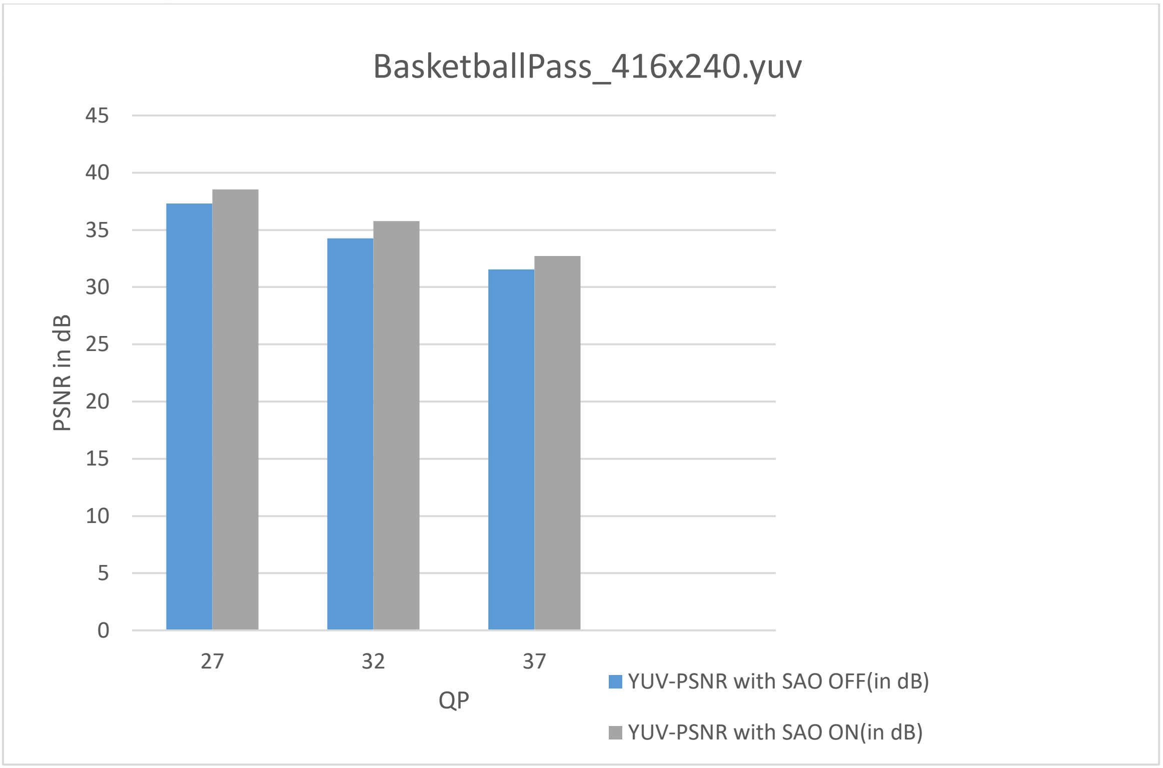


Figure 7.5: Graphical representation of PSNR vs QP for BasketballPass

7.4.2 RaceHorses_832x480_30.yuv

QP	Encoding time with SAO Off (in seconds)	Encoding time with SAO On (in seconds)	YUV- PSNR with SAO Off (in dB)	YUV-PSNR with SAO On (in dB)
27	2172.776	2207.137	34.7324	35.9245
32	1823.035	1904.508	32.8541	33.4520
37	1644.197	1675.346	29.9451	30.7209

Table 7.2: Encoding time and PSNR of RaceHorses

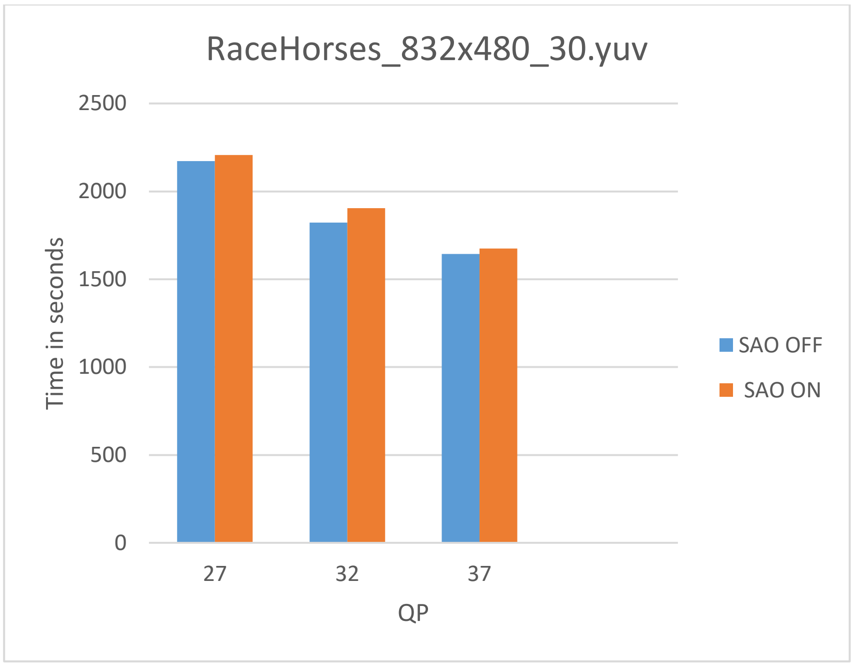


Figure 7.6: Graphical representation of Time vs QP for RaceHorses

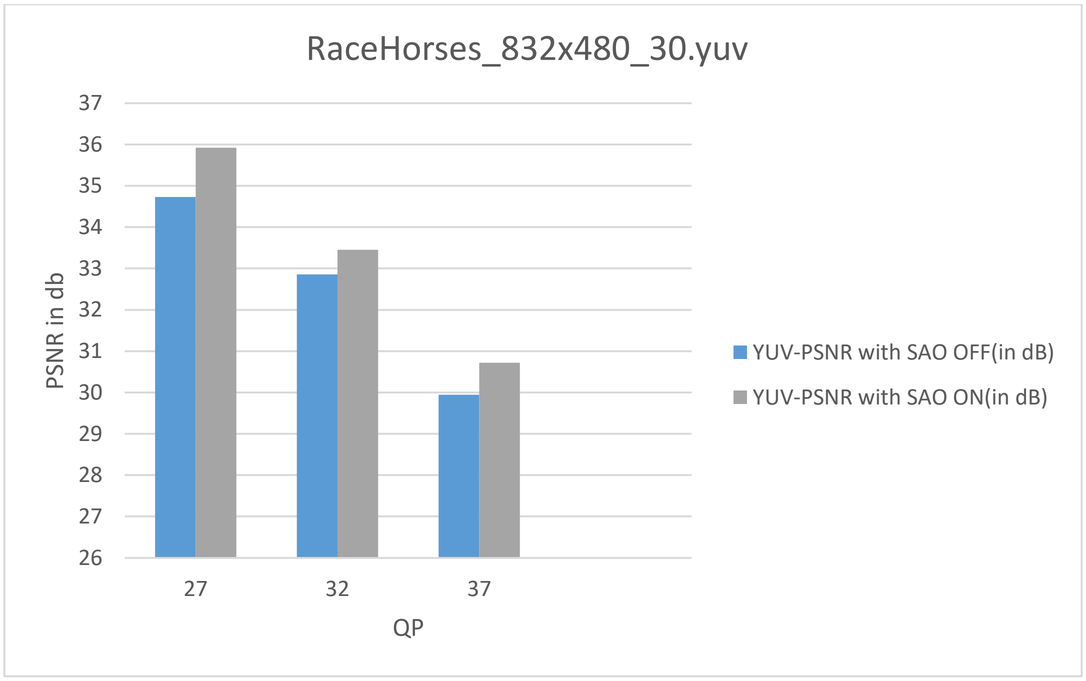


Figure 7.7: Graphical representation of PSNR vs QP for RaceHorses

RaceHorses_832x480_30.yuv sequences frame 1

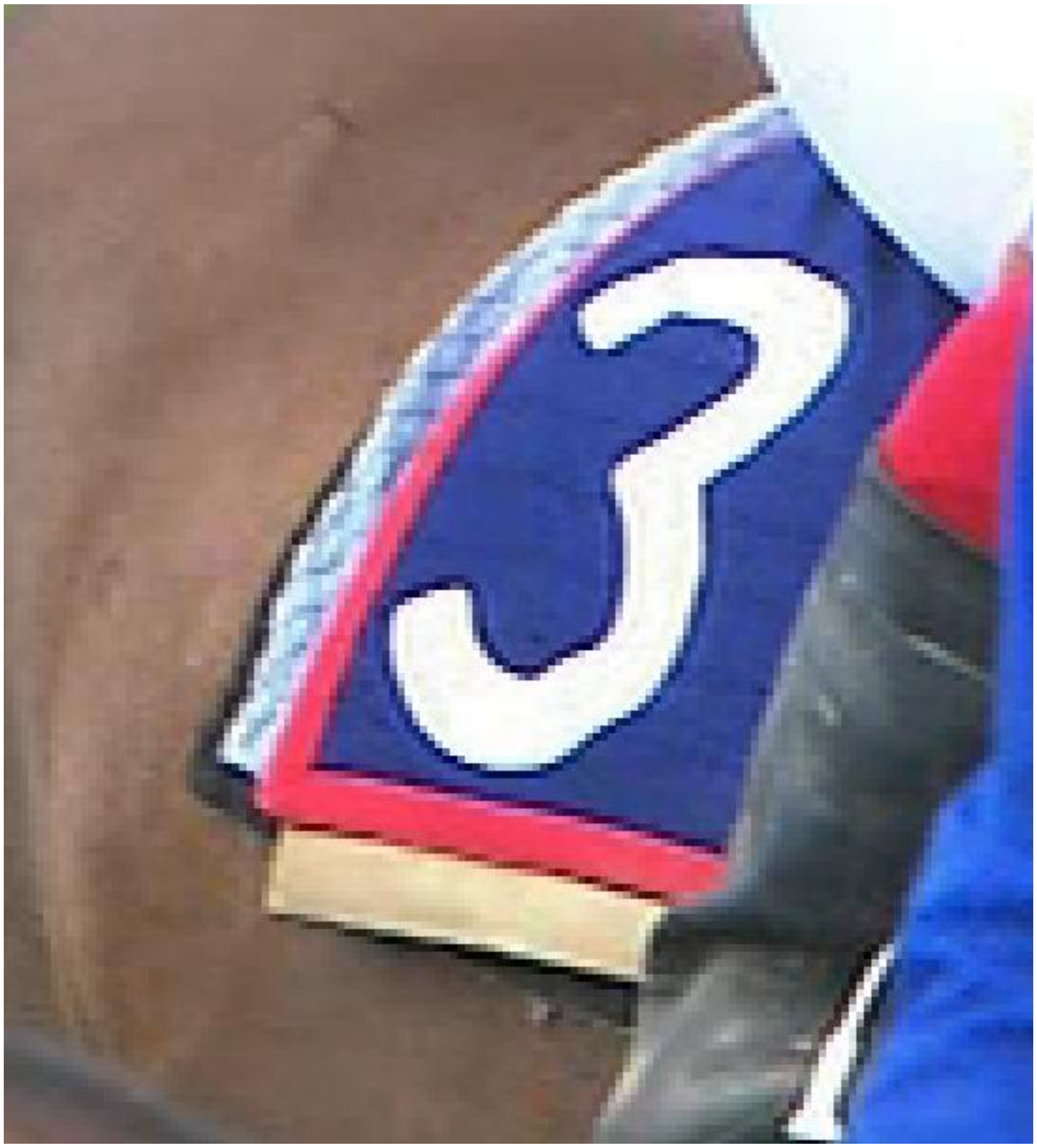


Fig.7.8 RaceHorses_832x480_30.yuv Original Image

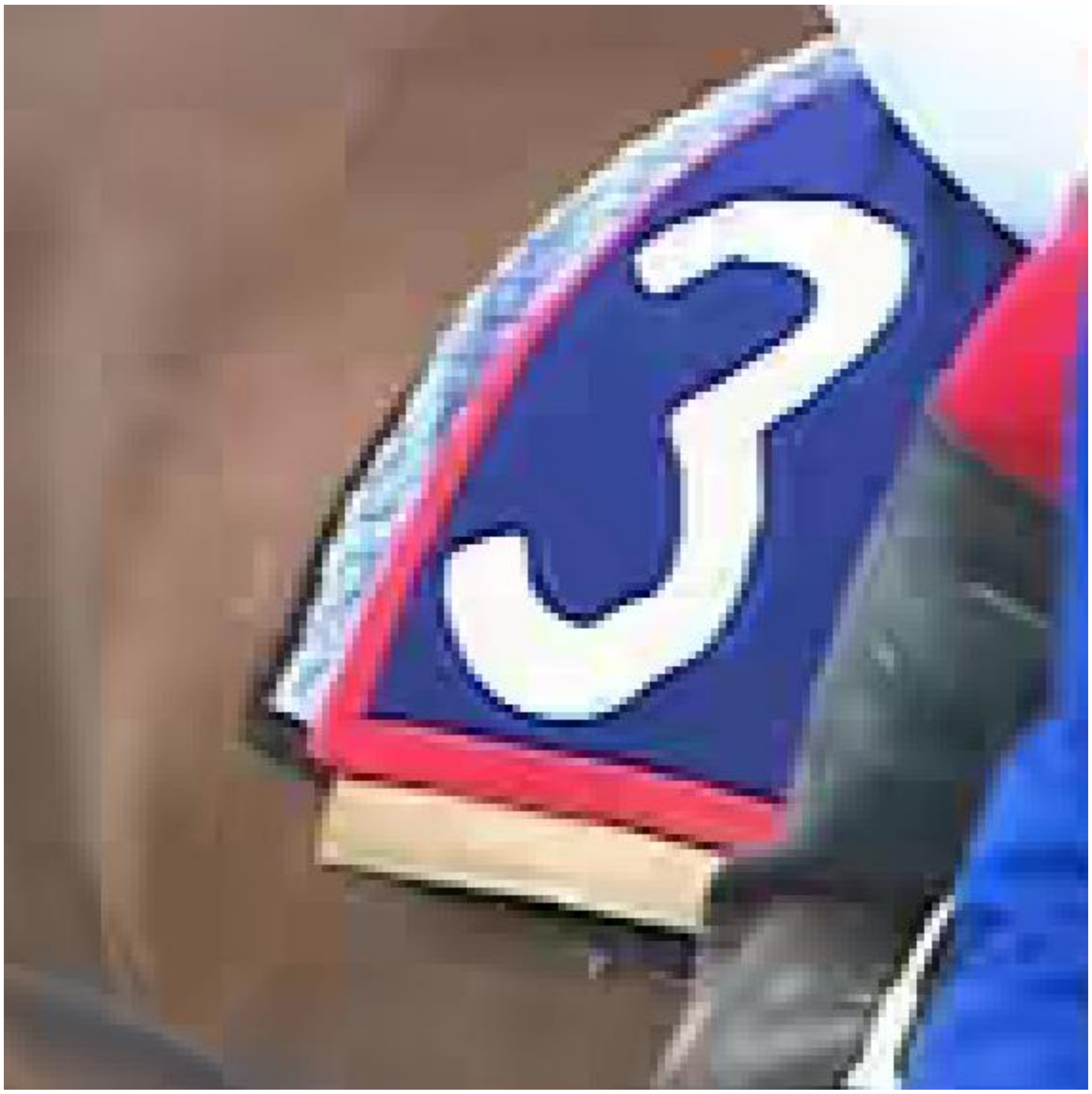


Fig.7.9 RaceHorses_832x480_30.yuv SAO=0

SAO=1 is SAO filter ON
SAO=0 is SAO filter OFF

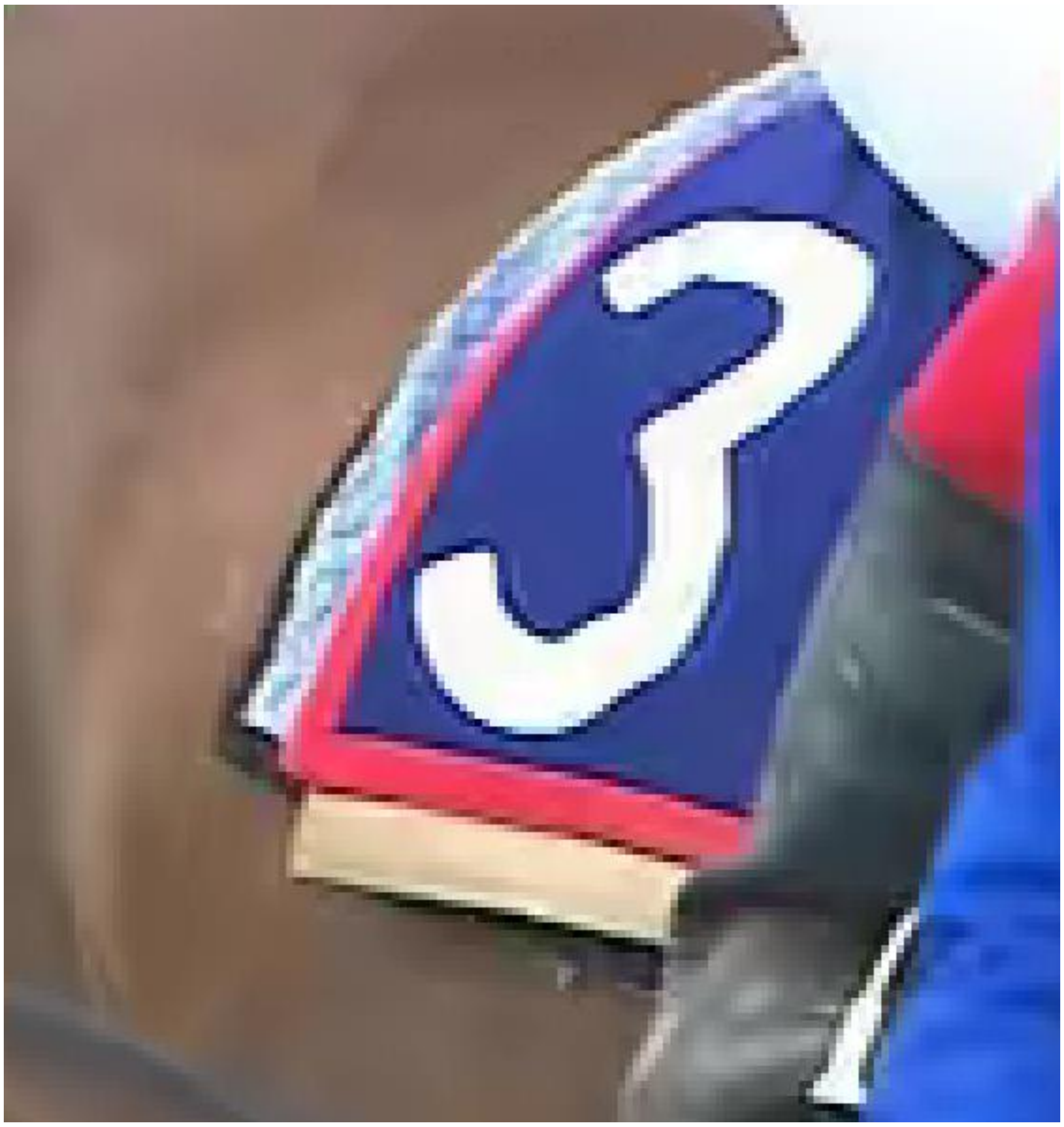


Fig.7.10 RaceHorses_832x480_30.yuv SAO=1

SAO=1 is SAO filter ON
SAO=0 is SAO filter OFF

7.4.3 SlideEditing_1280x720_30

(For 200 frames)

QP	Encoding time with SAO Off (in seconds)	Encoding time with SAO On (in seconds)	YUV- PSNR with SAO Off (in dB)	YUV-PSNR with SAO On (in dB)
27	2392.732	2625.679	42.3313	43.7979
32	2349.765	2462.828	37.9853	39.6725
37	2294.324	2423.639	33.4506	35.0188

Table 7.3 Encoding time and PSNR of SlideEditing

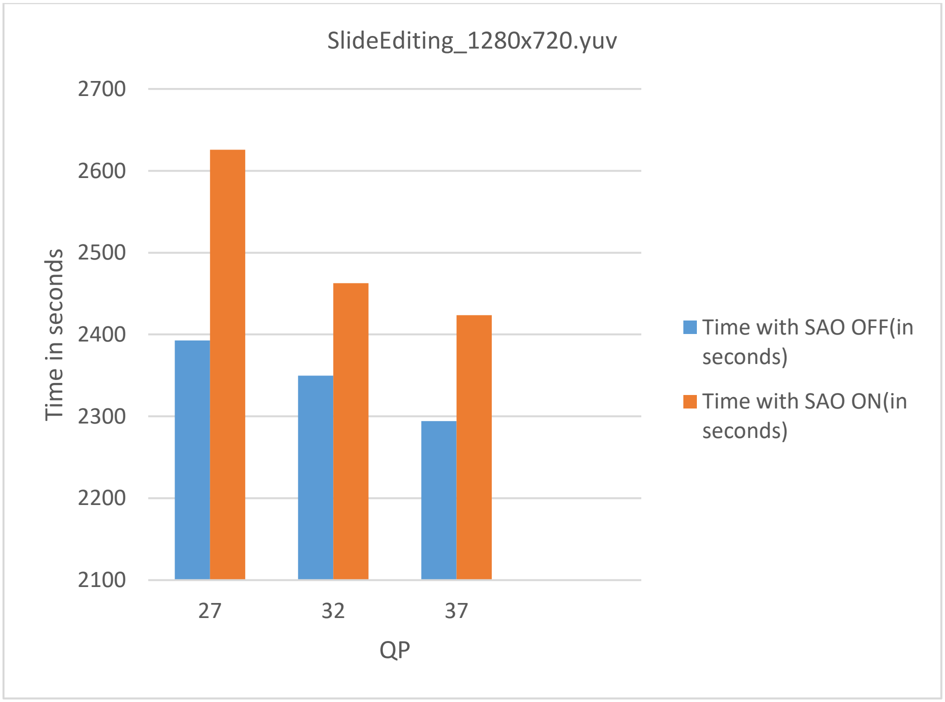


Figure 7.11: Graphical representation of Time vs QP for SlideEditing

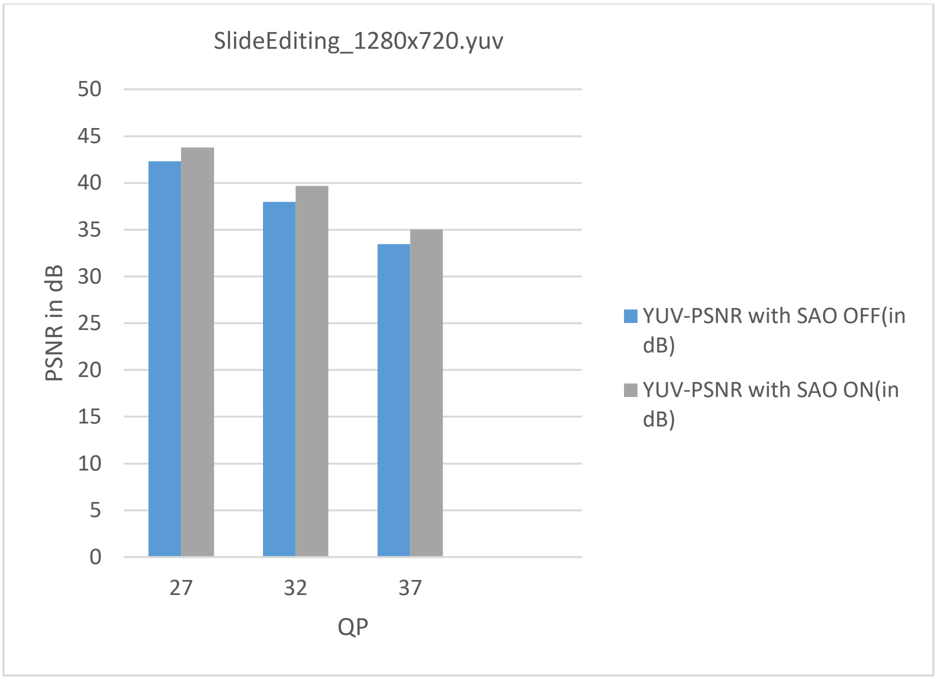
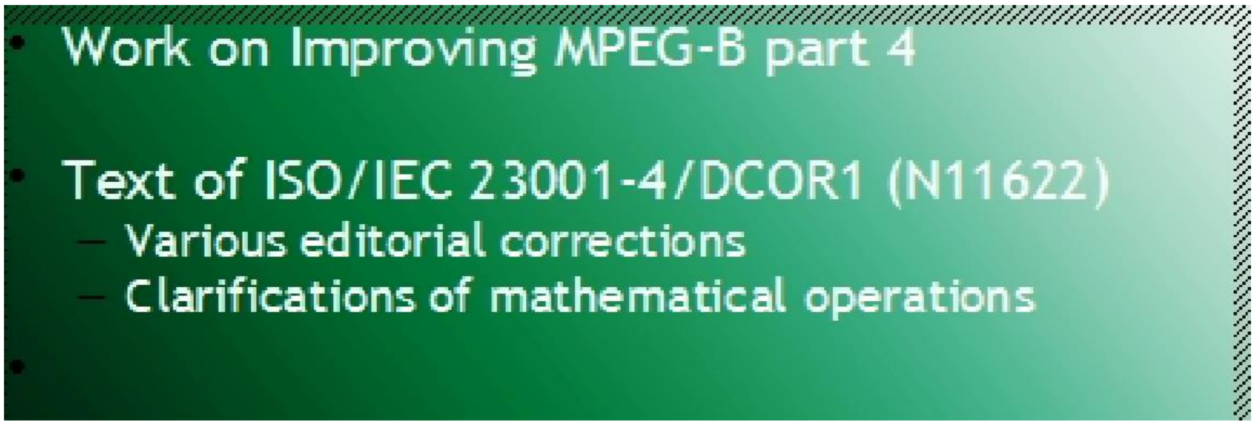


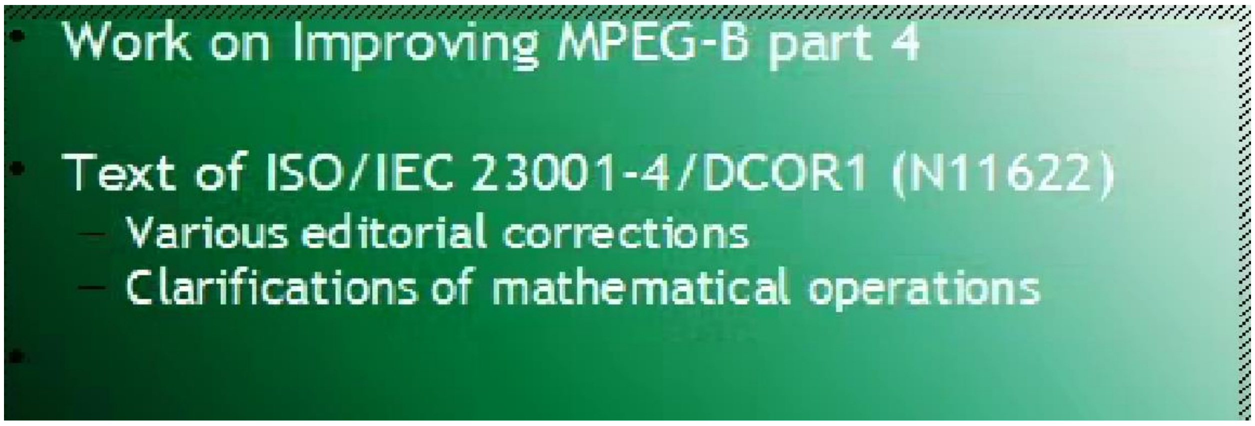
Figure 7.12: Graphical representation of PSNR vs QP for SlideEditing

SlideEditing_1280x720_30 frame 100



- Work on Improving MPEG-B part 4
- Text of ISO/IEC 23001-4/DCOR1 (N11622)
  - Various editorial corrections
  - Clarifications of mathematical operations

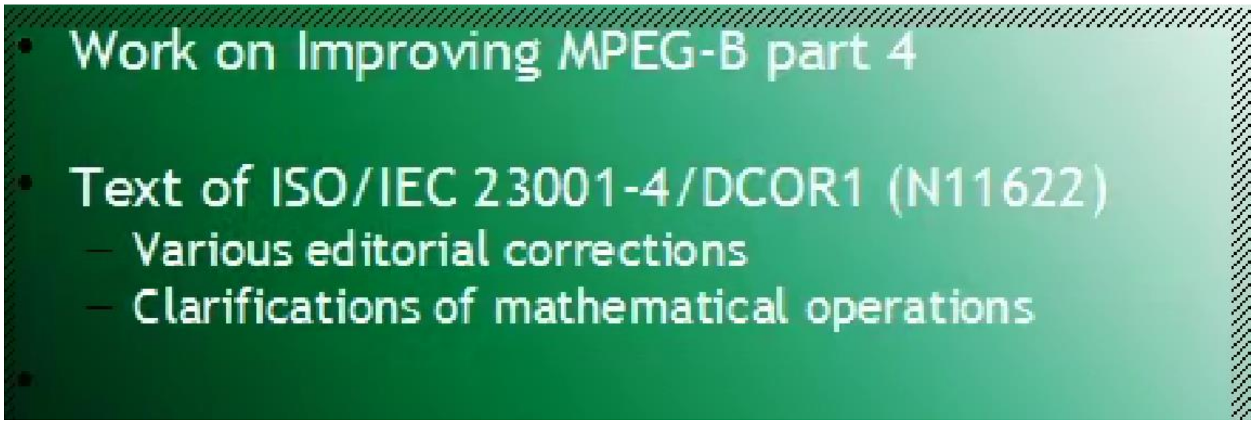
Fig. 7.13 SlideEditing_1280x720_30.yuv Original frame



Work on Improving MPEG-B part 4

- Text of ISO/IEC 23001-4/DCOR1 (N11622)
  - Various editorial corrections
  - Clarifications of mathematical operations

Fig. 7.14 SlideEditing_1280x720_30.yuv SAO=0



Work on Improving MPEG-B part 4

- Text of ISO/IEC 23001-4/DCOR1 (N11622)
  - Various editorial corrections
  - Clarifications of mathematical operations

Fig.7.15 SlideEditing_1280x720_30.yuv SAO=1

SAO=1 is SAO filter ON
SAO=0 is SAO filter OFF

7.5 Parallel code results

In order to test the performance of the parallel algorithms designed in the work, we use speedup (*SP*) defined in (16) to evaluate the computation efficiency, where T_s represents the running time of the original serial algorithm implemented on CPU and T_p represents the running time of the proposed parallel algorithm implemented on GPU

$$SP= T_s/ T_p \qquad (16)$$

7.5.1 BasketballPass_416x240_50.yuv

(Performed for 200 Frames)

QP	Encoding time with SAO On (in sec)	Encoding time with SAO On using GPU (in sec)	YUV-PSNR with SAO On using GPUs (in dB)	SP
27	495.279	18.689	38.3379	26.45
32	371.423	14.230	35.6873	26.10
37	344.548	13.090	32.7812	26.32

Table 7.4: Encoding time of BasketballPass using GPUs

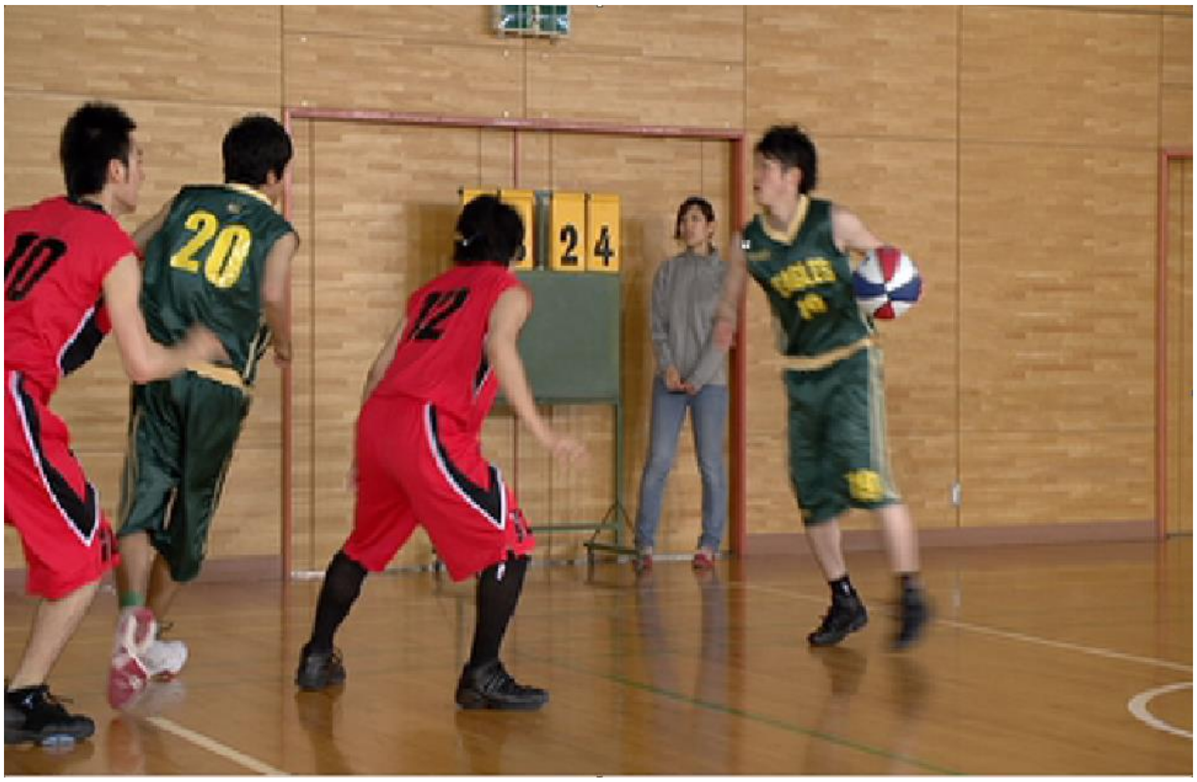


Figure 7.16 Frame of BasketballPass video sequence using serial algorithm

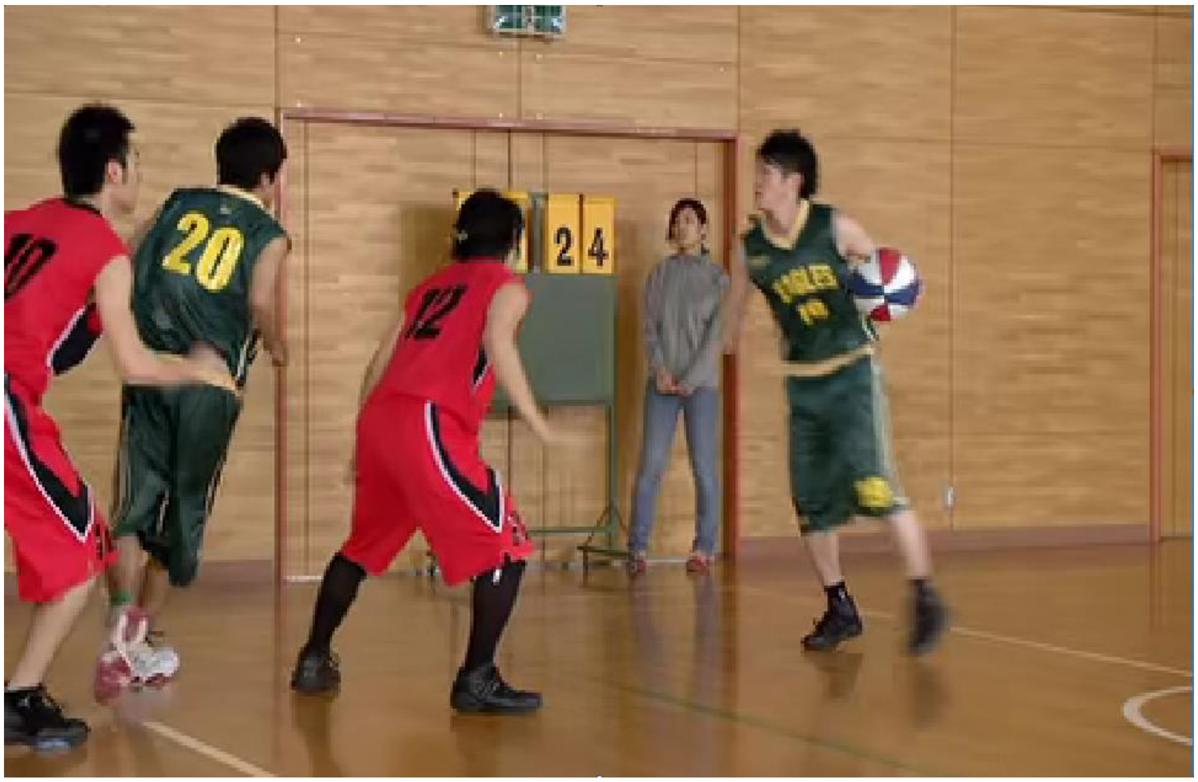


Figure 7.17 Frame of BasketballPass video sequence using parallel algorithm

7.5.2 RaceHorses_832x480_30.yuv

QP	Encoding time with SAO On (in sec)	Encoding time with SAO On using GPU(in sec)	GPU YUV-PSNR with SAO On using GPUs(in dB)	SP
27	2207.137	81.109	38.8920	27.21
32	1904.508	70.746	36.1813	26.92
37	1675.346	61.820	32.6192	27.10

Table 7.5: Encoding time of RaceHorses using GPUs

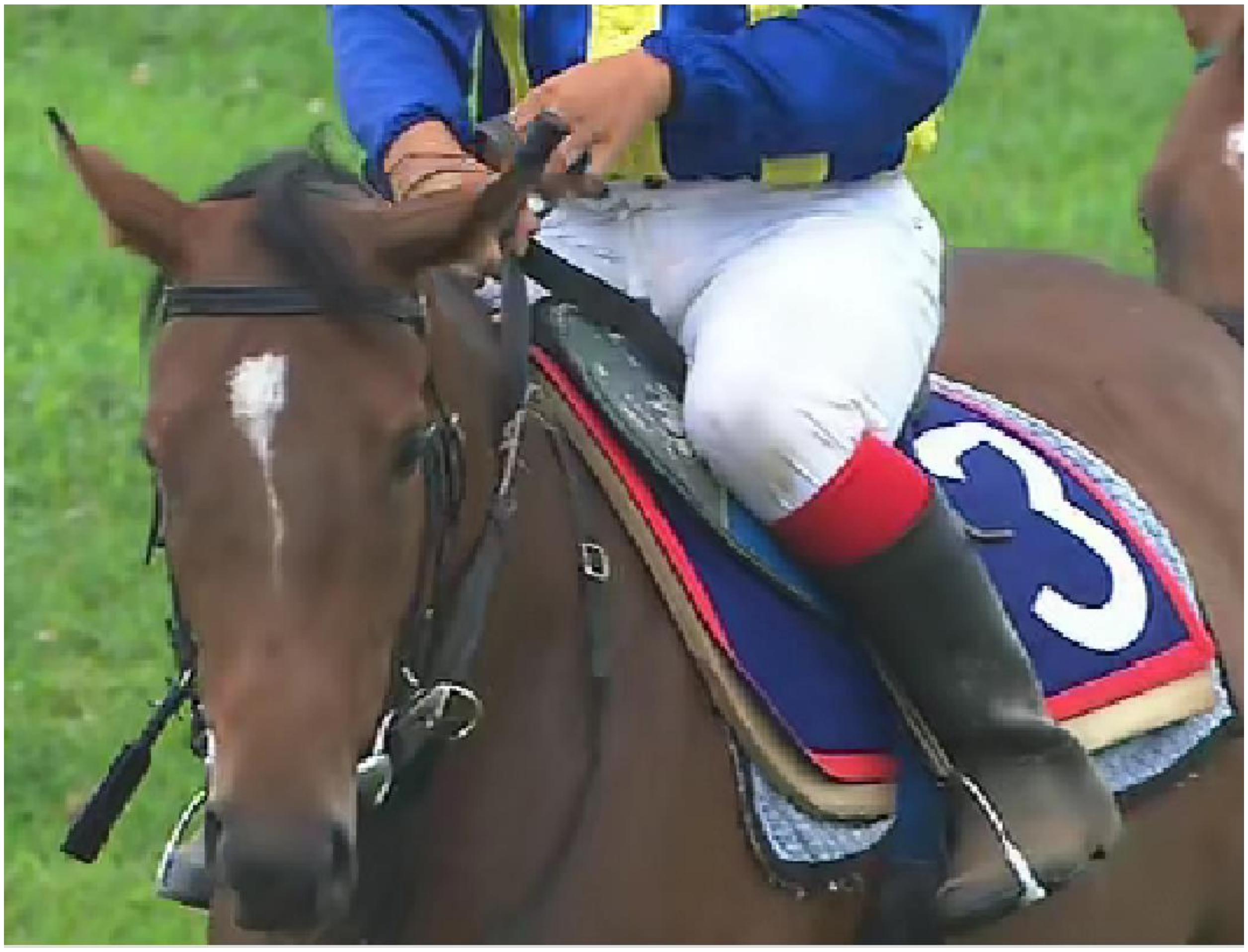


Figure 7.18: Frame of RaceHorses video sequence using serial algorithm

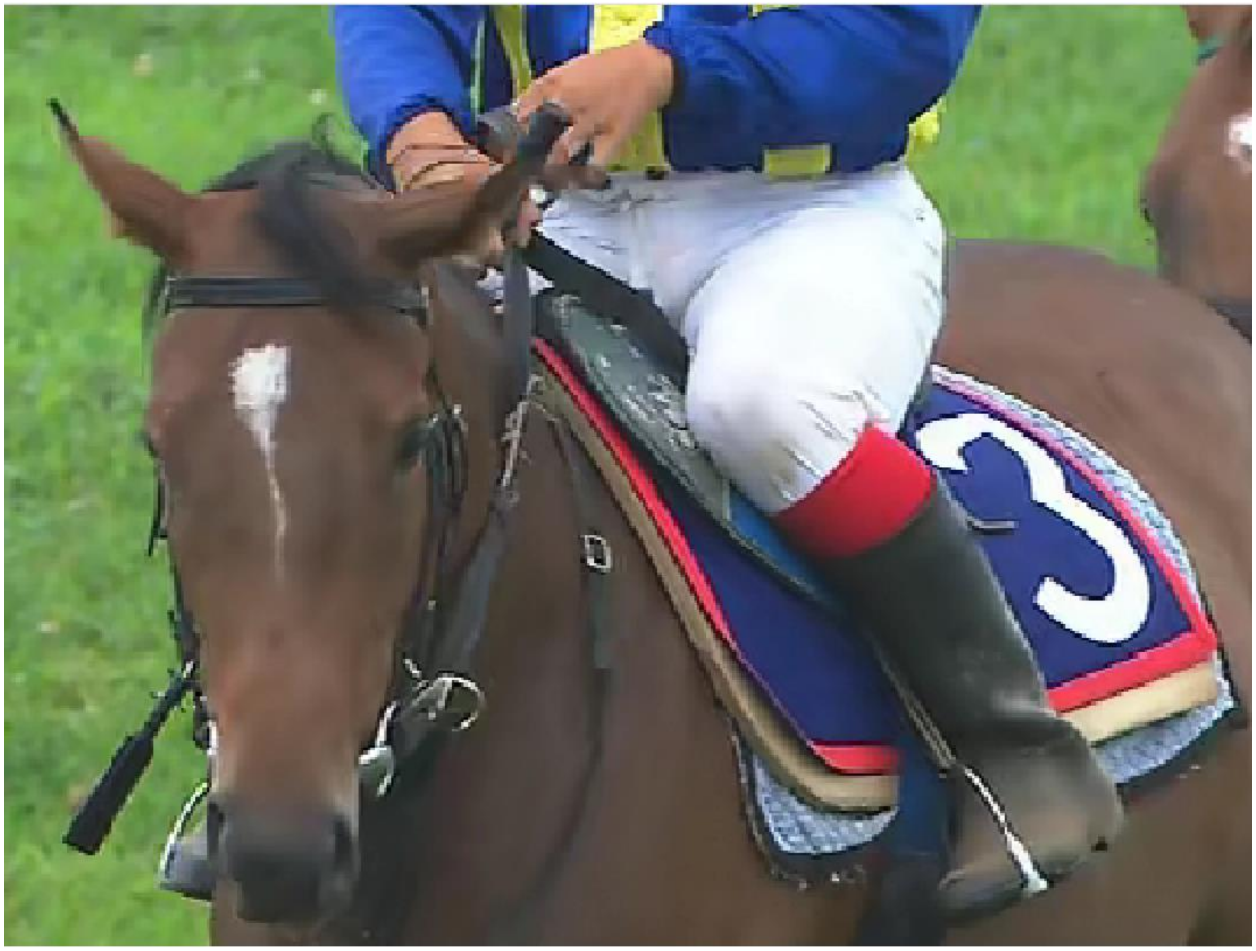


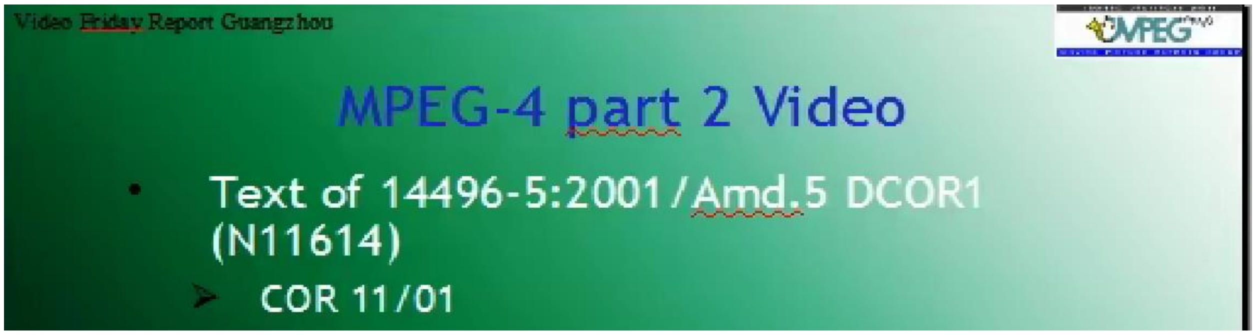
Figure 7.19: Frame of RaceHorses video sequence using parallel algorithm

7.5.3 SlideEditing_1280x720_30

(For 200 frames)

QP	Encoding time with SAO On (in seconds)	Encoding time with SAO On using GPU(in sec)	YUV- PSNR with SAO On using GPUs (in dB)	SP
27	2625.679	92.780	42.7313	28.3
32	2462.828	88.083	38.1043	27.96
37	2423.639	86.066	33.6406	28.16

Table 7.6: Encoding time of SideEditing using GPUs

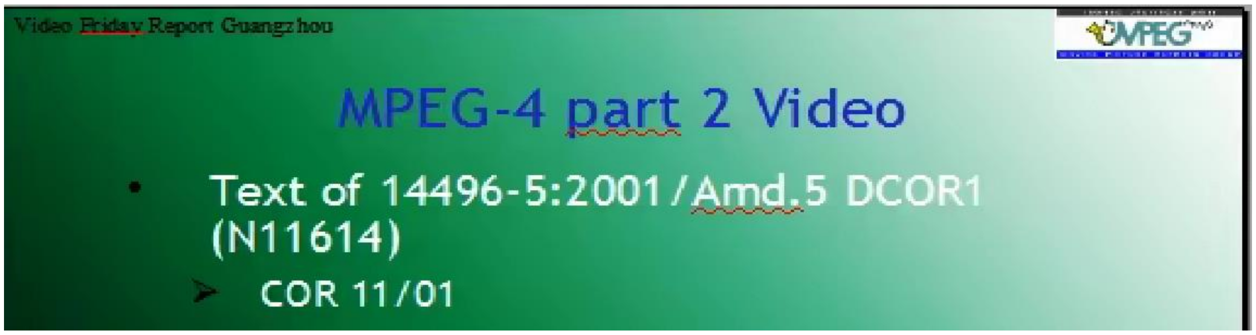


Video Friday Report Guangzhou

MPEG-4 part 2 Video

- Text of 14496-5:2001/Amd.5 DCOR1 (N11614)
  - ➢ COR 11/01

Figure 7.20: Frame from SlideEditing video sequence using serial algorithm



Video Friday Report Guangzhou

MPEG-4 part 2 Video

- Text of 14496-5:2001/Amd.5 DCOR1 (N11614)
  - ➢ COR 11/01

Figure 7.21: Frame from SlideEditing video sequence using parallel algorithm

Serial vs Parallel

Figure 7.22:

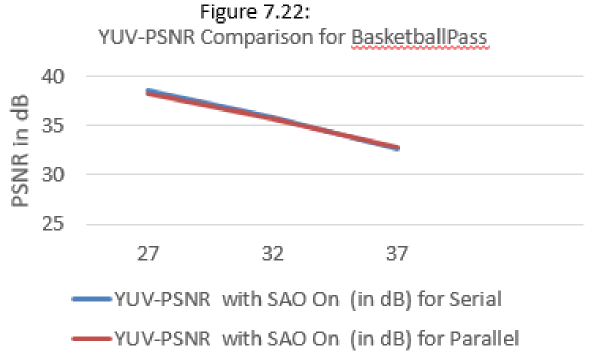


Figure 7.23:

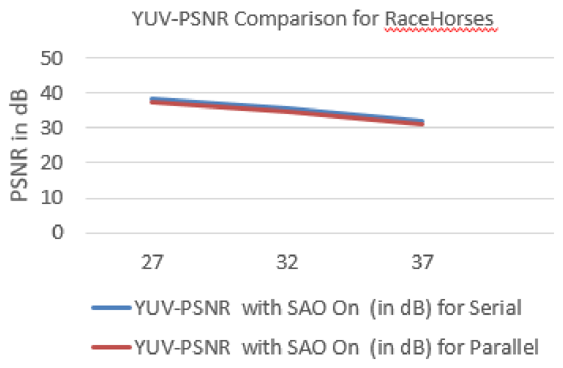
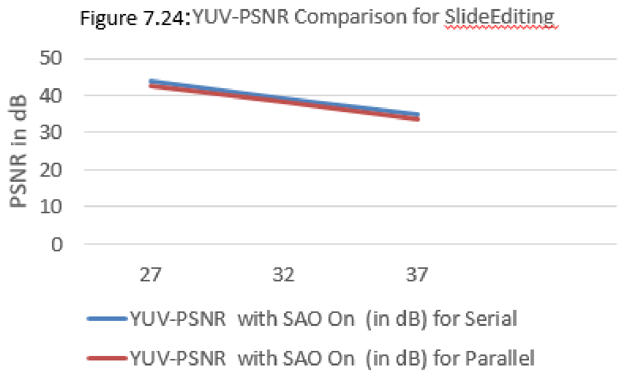
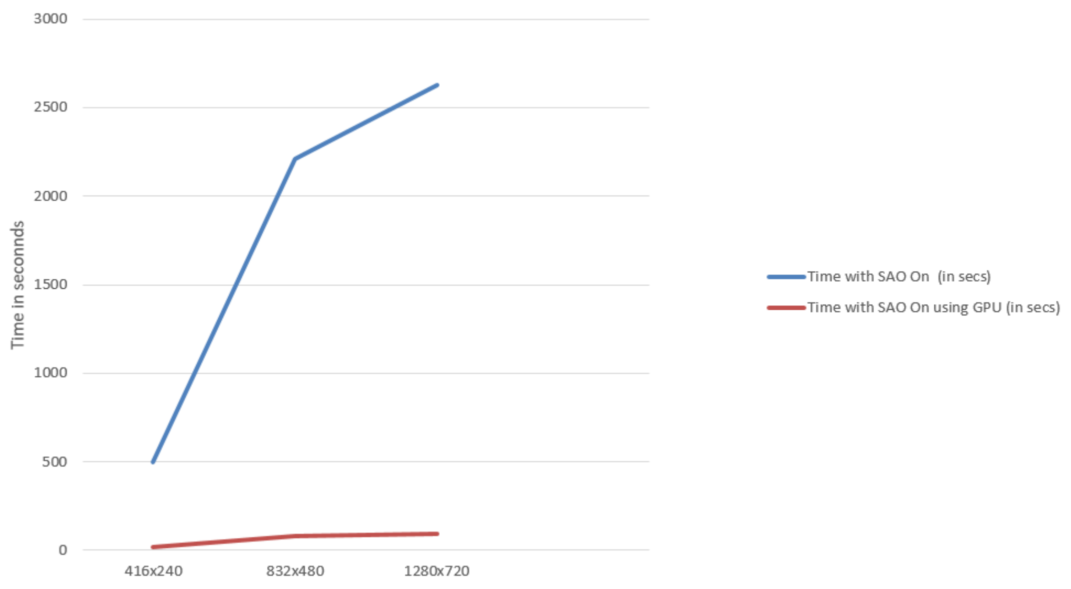


Figure 7.24:



Serial vs Parallel Encoding Time

Figure 7.25: Encoding time



7.6 Conclusions and future work

This thesis consists of implementation of serial and parallel algorithms to improve the computation efficiency of the SAO in HEVC. The algorithm is implemented in parallel using NVIDIA GPU platform programmed with CUDA language. Compared with the original serial algorithm implemented on CPU, the experimental results show that parallel algorithms can provide a significant improvement on the encoding time with an overall more than 25x speedup for high definition video sequences.

Post processing methods can be used to further improve the quality of the video. Implementing this post processing filter in parallel would be a much efficient way of computation. A significant amount of time is spent in transferring data from host to device and vice versa. In the future there is scope to decrease the overhead of pushing data to GPU and CPU back and forth. Research on increasing parallel data throughputs for further improving the encoding time can be a part of the future work.

8 References

1. W. Zhang and C. Guo, "Design and implementation of parallel algorithms for sample adaptive offset in HEVC based on GPU," *2016* Sixth International Conference on Information Science and Technology (ICIST)*,* Dalian, China, June 2016, pp. 181-187. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7483407&isnumber=7483373

2. G. J. Sullivan et al, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, No. 12, pp. 1649-1668, Dec. 2012.

3. N. Ling, "High efficiency video coding and its 3D extension: A research perspective," Keynote Speech, ICIEA, pp. 2150-2155, Singapore, July 2012- http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6361087
4. Multimedia course website: http://www.uta.edu/faculty/krrao/dip/

5. A. Norkin, et al, *CE12:* Ericsson's and MediaTek's Deblocking Filter, ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 document JCTVC-F118, Joint Collaborative Team on Video Coding (JCTVC), Turin, Italy, July 2011.

6. Design and implementation of next generation video coding systems ppt: http://www.uta.edu/faculty/krrao/dip/Courses/EE5359/budagaviiscas2014ppt.pdf
7. JVT Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264-ISO/IEC 14496-10 AVC), March 2003, JVT-G050 available on http://ip.hhi.de/imagecom_G1/assets/pdfs/JVT-G050.pdf

8. D. K. Kwon and M.Budagavi. "Combined scalable and multiview extension of High Efficiency Video Coding (HEVC)", IEEE Picture Coding Symposium, pp. 414 - 417, Dec. 2013.
9. Information on ringing artifacts : https://en.wikipedia.org/wiki/Ringing_artifacts

10. G. J. Sullivan et al, “Standardized Extensions of High Efficiency Video Coding (HEVC)”, IEEE Journal of selected topics in Signal Processing, vol. 7, pp.1001-1016, Dec. 2013.

11. B. Bross et al, “High efficiency video coding (HEVC) text specification draft 8”, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCTVC) document JCTVC-J1003, July 2012

12. F.Bossen, D.Flynn and K. Suhring, “HEVC reference software manual” http://phenix.int-evry.fr/jct/doc_end_user/documents/6_Torino/wg11/JCTVC-F634-v2.zip

13. P. Hanhart et al, “Subjective quality evaluation of the upcoming HEVC video compression standard”, SPIE Applications of digital image processing XXXV, vol. 8499, paper 8499-30, Aug. 2012.

14. M. Horowitz et al, “Informal subjective quality comparison of video compression performance of the HEVC and H.264/MPEG-4 AVC standards for low delay applications” , SPIE Applications of digital image processing XXXV , vol. 8499, paper 8499-31, Aug. 2012.

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

16. F. Bossen, et al, "HEVC complexity and implementation analysis," IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, pp.1685-1696, Dec. 2012.

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

18. X. Cao, C. Lai and Y. He, “Short distance intra coding scheme for HEVC”, IEEE Picture Coding Symposium, June 2012.

19. B. Li, G. J. Sullivan, and J. Xu, “Comparison of compression performance of HEVC working draft 4 with AVC high profile,” JCTVC-G399, Nov. 2011.

20. Thesis by S. Gangavati on “complexity reduction of H.264 using parallel programming “. This thesis describes the reduction of the motion estimation time in H.264 using CUDA language which includes the usage of GPUs and CPUs, 2012. http://www-ee.uta.edu/Dip/Courses/EE5359/Sudeep_Thesis_Draft_2.pdf

21. M. Jakubowski and G. Pastuszak , "Block –based motion estimation algorithms - a survey", Opto-Electronics Review vol. 21, no. 1, pp. 86-102, 2013.
22. F. Dufaux and F. Moscheni, "Motion estimation techniques for digital TV – a review and a new contribution", Proc. IEEE, vol.83, pp. 858 – 876, June1995.
23. J. R. Jain and A .K. Jain , " Displacement measurement and its application in interframe image-coding" IEEE Transactions on Communications, vol.com -29, pp. 1799-1808, Dec. 1981.
24. I.E.G. Richardson, Video codec design : Developing image and video compression systems , Wiley, Chichester, 2002.
25. J.B. Lee and H. Kalva , "The VC-1 and H.264 video compression standards for broadband video Services," Springer, New York, 2008.
26. Y. Su and M.-T. Sun, "Fast multiple reference frame motion estimation for H.264/AVC", IEEE Trans. on circuits and systems for video technology, vol. 16, pp. 447-452, March 2006.
27. M. E. Sinangil et al, "Memory cost vs coding efficiency trade-offs for HEVC motion estimation engine ", IEEE International conference on image processing, pp. 1533-1536, 2012.
28. V. Sze, M. Budagavi and G.J. Sullivan (Editors), "High Efficiency Video Coding (HEVC): Algorithms and Architectures", Springer, 2014.
29. C-M. Fu et al, "Sample Adaptive Offset in the HEVC Standard," IEEE Trans. on circuits and systems for video technology, vol. 22, pp. 1755-1764, Dec. 2012.
30. F. Bossen, Common HM Test Conditions and Software Reference Configurations, document JCTVC-J1100, July 2012.
31. N. Ahmed, R. Natarajan and K.R.Rao, "Discrete Cosine Transform", IEEE Trans. on computers, vol.C-23, pp.90-93, Jan.1974.
32. G. Laroche, T. Poirier, and P. Onno, *Non-CE1: Encoder Modification for SAO Interleaving Mode,* JCTVC-I0184, Joint Collaborative Team on Video Coding, Apr. 2012.

33. M. A. F. Rodriguez, “CUDA: Speeding up parallel computing”, International Journal of Computer Science and Security, Nov. 2010.

34. NVIDIA, NVIDIA CUDA Programming Guide, Version 3.2, NVIDIA, Sep. 2010. http://docs.nvidia.com/cuda/cuda-c-programming-guide/

35. “http://drdobbs.com/high-performance-computing/206900471” Jonathan Erickson, GPU Computing Isn’t Just About Graphics Anymore, Online Article, Feb.2008.

36. J. Nickolls and W. J. Dally, ” The GPU computing era” , IEEE Computer Society Micro-IEEE, vol. 30, Issue 2, pp . 56 - 69, April 2010.

37. J. Sanders and E. Kandrot, “CUDA by example: an introduction to general-purpose GPU programming” Addison-Wesley, 2010.

38. NVIDIA, NVIDIA’s Next Generation CUDA Compute Architecture: Fermi, White Paper, Version 1.1, NVIDIA 2009. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

39. W.-N. Chen, et al, “H.264/AVC motion estimation implementation on compute unified device architecture (CUDA)” , IEEE International Conference on Multimedia and Expo, pp. 697 – 700, 2008.

40. Thesis by J. Dubhashi on “Complexity reduction in H.265 Motion Estimation using Compute Unified Device Architecture “. This thesis describes the reduction of the motion estimation time in H.265 using CUDA language which includes the usage of GPUs and CPUs, 2014. Use the link below and scroll down to the thesis section to find J. Dubhashi’s thesis document http://www.uta.edu/faculty/krrao/dip/Courses/EE5359/index_tem.html

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

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

43. Video lectures from IITs and IISC: http://nptel.iitm.ac.in/

44. Useful tutorials on Video processing:

a. http://kohtoomaung.blogspot.com/p/blog-page_10.html

b. https://codesequoia.wordpress.com/2012/11/04/

45. I.E.G. Richardson, "The H.264 advanced video compression standard", 2nd Edition, Wiley, Hoboken, NJ, 2010.

46. J. Lainema et al, "Intra Coding of the HEVC Standard," in IEEE Trans. on Circuits and Systems for Video Technology, vol.22,no.12,pp.1792-1801,Dec.2012.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6317153&isnumber=6403920

47. N. Purnachand, L. N. Alves and A. Navarro, "Fast motion estimation algorithm for HEVC, " IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin), 2012.

48. Wikipedia URL: https://en.wikipedia.org/wiki/CUDA

49. T. Wiegand et al, "Overview of the H.264/AVC video coding standard," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560-576, July 2003.

50. K.R. Rao, D.N. Kim and J.J. Hwang, "High Efficiency Video Coding and other emerging standards," River publishers, 2017.

51. HEVC software manual-
https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/trunk/doc/software-manual.pdf

52. JCT-VC DOCUMENTS can be found in JCT-VC document management system http://phenix.int-evry.fr/jct. All JCT-VC documents can be accessed. [Online].

Available:
http://phenix.intevry.fr/jct/doc_end_user/current_meeting.php?id_meeting=154&type_order=&s ql_type=document_number

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

53. M. Wien, “High Efficiency Video Coding: Coding Tools and Specification”, Springer, 2014.
54. C.-M. Fu, et al, *Sample Adaptive Offset for Chroma*, document JCTVCF057, July 2011.
55. C.-M. Fu, et al, *E8.a.3: SAO with LCU-Based Syntax*, document JCTVC-H0273, Feb. 2012.
56. G. Laroche, T. Poirier, and P. Onno, *On Additional SAO Band Offset Classifications,* JCTVC-G246, Joint Collaborative Team on Video Coding, Nov. 2011.
57. E. Maani and O. Nakagami, *Flexible Band Offset Mode in SAO*, document JCTVC-H0406, Feb. 2012.
58. Y. Choi, J. Joo, “ Exploration of practical HEVC/H.265 sample adaptive offset encoding policies”, Signal Processing Letters, vol. 22, pp.465 – 468, Apr. 2015.
59. ISO/IEC 11172-2, “Information technology coding of moving pictures and associated audio for digital storage media at up to about 1. Mbit/s - part 2: Video”, 1993 [MPEG-1 Video]
60. ISO/IEC 13818-2, “Information technology: generic coding of moving pictures and associated audio information: Video”, 1995 [MPEG-2 Video]
61. ISO/IEC 14996-2, “Information technology - coding of audio-visual objects - part-2: Visual”, 1998 [MPEG- 4 Visual]
62. ISO/IEC 14996-10 and ITU-T Rec. H.264, “Advanced Video Coding”, 2003 [MPEG-4 part-10/H.264]
63. ITU-T Recommendation H.261, “Video CODEC for audiovisual services at px64 kbits/s”, 1988 [H.261]
64. http://www.itu.int/rec/T-REC-H.261-198811-S/en [H.261 document]
65. ITU-T Recommendation H.263, “Video coding for low-bit rate communications”, Version-2, 1998 [H.263]
66. h.265/hevc: ITU-T Recommendation H.265, “Infrastructure of audiovisual services – Coding of moving video: High efficiency video coding”, April 2013 [H.265]
67. Grois *et al*, “Performance Comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC Encoders”, IEEE Picture Coding Symposium 2013 (PCS 2013), San José, CA, USA, Dec 8-11, 2013.

68. Ohm *et al*, “Comparison of the Coding Efficiency of Video Coding Standards—Including High Efficiency Video Coding (HEVC)”, IEEE Trans. On Circuits and Systems for Video Technology, vol. 22, pp. 1669-1684, Dec. 2012SMPTE 421M-2006,

69. SMPTE 421M-2006, "VC-1 Compressed Video Bitstream Format and Decoding Process", 2006 [VC-1]

70. WebM project blog: http://blog.webmproject.org
71. http://www.theora.org [Theora]
72. RFC-6386: VP8 Data Format and Decoding Guide, 2011. [VP8]
73. A VP9 Bitstream Overview: draft-grange-vp9-bitstream-00, 2013.http://tools.ietf.org/id/draft-grange-vp9-bitstream-00.txt [VP9]
74. https://www.xiph.org/daala/ [Daala]
75. HEVC tutorials on -- https://www.vcodex.com
76. Introduction to video compression http://www.videsignline.com/howto/185301351;jsessionid=B2FYP22SRT0TEQSNDLOS KHSCJUNN2JV N?pgno=1
77. S. Kwon, A. Tamhankar, and K. R. Rao, "Overview of the H.264/MPEG-4 part 10," *Journal of Visual Communication and Image Representation*, vol. 17, is. 9, pp. 186-216, Apr. 2006.
78. Fu C-M, et al, Sample Adaptive Offset with LCU- independent decoding, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-E049, Geneva, Mar. 2011
79. Fu C-M, et al, Sample adaptive offset in the HEVC standard. IEEE Trans Circuits Systems Video Technology 22(12):1755–1764.
80. Fu C-M, et al, SAO with LCU-based syntax, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-H0273, San Jose, Feb. 2012
81. F. Bossen, B. Bross, K. Suhring, and D. Flynn, “HEVC complexity and implementation analysis,” Circuits and Systems for Video Technology, IEEE Transactions, vol. 22, no. 12, pp. 1685–1696, 2012.
82. HM Test Model documentation: https://hevc.hhi.fraunhofer.de/HM-doc/index.html
83. Downloading CUDA Toolkit https://developer.nvidia.com/cuda-toolkit

84. W.-Y. Wei, “Deblocking Algorithms in Video and Image Compression Coding”, National Taiwan University, Taipei, Taiwan, ROC
85. C-M Fu et al, “*Sample Adaptive Offset in the HEVC Standard*,” IEEE Trans. on Circuits and Systems for Video Technology, Vol. 22, pp. 1755-1764, Dec. 2012.
86. Tutorials: https://classroom.udacity.com/me
87. Video Sequences: http://trace.eas.asu.edu/yuv/index.html
https://media.xiph.org/video/derf/
88. Windows Operating System: https://www.microsoft.com/en-us/windows/

ACRONYMS

AVC:	Advanced Video Coding
API:	Application Programming Interface
BO:	Band Offset
BBME:	Block Based Motion Estimation
BDM:	Block Difference Measure
CABAC:	Context Adaptive Binary Arithmetic Coding
CBs:	coding blocks
CTB:	coding tree block
CUDA:	Compute Unified Device Architecture
DF:	Deblocking Filter
EO:	Edge Offset
GPU:	Graphics Processing Unit
GPGPU:	General-Purpose computing on Graphics Processing Units
GOP:	Group of Pictures
KTA:	Key Technology Area
HVC:	High Performance Video Coding
HEVC:	High Efficiency Video Coding
HD:	High Definition
ISO:	International Standardization Organization
IEC:	International Electro-technical Commission
Intra HE:	Intra high efficiency
IISc:	Indian Institute of Science

IIT: Indian Institute of Technology

ITU-T: International Telecommunication Union – Telecommunication Standardization Sector.

JCT-VC: Joint Collaborative Team on Video Coding

QP: Quantization Parameter

MBs: Macroblocks

MVC: Multi view video coding

MPEG: Moving Picture Experts Group

PB: Prediction blocks

PUs: Prediction units

SA: Search Area

SAD: Sum of Absolute Differences

SAO: Sample Adaptive Offset

SVC: Scalable Video Coding

TB: Transform blocks

TUs: Transform units

UHD: Ultra-High Definition.

VCEG: Video Coding Experts Group.

LIST OF ILLUSTRATIONS

List of Tables

BIOGRAPHICAL INFORMATION

Harsha Nagathihalli Jagadish was born in Bengaluru, Karnataka, India. His Schooling was done in Florence Public School, Bengaluru. He went on to pursue his Bachelor of Engineering in Electronics and Communications from B.M.S. Institute of Technology (Affiliated to Visvesvaraya Technological University).

Previously, he worked as a Programmer Analyst in Cognizant Technology Solutions, Chennai, India from 2014 to 2015. Harsha then enrolled in Masters in Electrical Engineering at the University of Texas at Arlington in fall 2015. During the course of the Masters he nurtured his skills in the field of Multimedia and Embedded Systems. He joined Multimedia Processing Lab under the guidance of Dr. K.R. Rao who is one of the founders of Discrete Cosine Transform (DCT). He also worked as a Graduate Teaching Assistant under Dr. Rao for Digital Image Processing and Digital Video Coding in Fall 2016 and Spring 2017. Harsha worked with CalAmp as a Software Engineer Intern in Summer 2017 and Fall 2017. His research activities include areas of both Multimedia and Embedded systems. His recent research activities include video coding standards, In-loop filters in HEVC, parallel algorithms, GPU based optimization, CUDA programming, Infotainment and Internet of Things.