REDUCING THE ENCODING TIME FOR H.264 BASELINE PROFILE USING PARALLEL PROGRAMMING TECHNIQUES

by

TUSHAR ASHOK SAXENA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

June 2012

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my professor Dr. K. R. Rao for his constant support and encouragement throughout the course of my research work. I have always admired his teaching methodologies and he has been a great source of inspiration in completing my thesis. I thank him for introducing me to the world of multimedia processing. It is a great honor for me to have worked under him.

I want to thank Dr. A. Davis and Dr. S. Tjuatja for taking time of their busy schedule in being a part of my thesis committee.

I want to thank Dr. Roger Walker for offering the course 'Real Time Data Acquisition and Control' and introducing me to the field of parallel programming using OpenMP.

I want to thank my friends Tejas, Urmi and Eugene for their comments and suggestions at various stages of my research work.

Finally, I am grateful to my grandmother Rajkumari Saxena, my mother Kiran Saxena, my father Ashok Saxena and my brother Kunal Saxena for their continuous support and motivations in all my endeavors.

July 2, 2012

ABSTRACT

REDUCING THE ENCODING TIME FOR H.264 BASELINE PROFILE USING PARALLEL PROGRAMMING TECHNIQUES

Tushar Saxena, M.S.

The University of Texas at Arlington, 2012

Supervising Professor: K. R. Rao

This thesis is aimed at reducing the encoding time for a group of pictures (GOP) for h.264 baseline profile on a general purpose CPU. This thesis also aims at drawing a comparison between task based parallelism and data based parallelism. The reduction is achieved by encoding the frames in parallel rather than serially. Task based parallelism is achieved by equally dividing the GOP's in two different threads and running them on the underlying hardware at the same time using Open MP software. Data based parallelism is achieved by finding the hot spots in the software and then parallelizing it using task based approach as stated above. JM 18.0 [1] is the reference software used to obtain the results. By adopting task based parallelism methodology the encoding time for a group of pictures is reduced approximately by 50%. But this reduction comes at a cost of increased CPU power consumption by approximately 50% and a marginal 2% increase in the physical memory usage. Data based parallelism does not show any improvement in the time complexity reduction as thread creation for every hot spot and every frame adds up to the time complexity overhead.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

ABSTRACT iv

LIST OF ILLUSTRATIONS xii

LIST OF TABLES xvi

LIST OF ACRONYMS xvii

Chapter Page

1. INTRODUCTION 1

1.1 H.264 2

1.2 SUMMARY 4

2. OVERVIEW OF H.264 / AVC STANDARD 5

2.1 H.264 profiles 5

2.1.1 Baseline profile 5

2.1.2 Extended profile 6

2.1.3 Main profile 7

2.1.4 High profile 7

2.2 H.264 block diagram 8

2.3 H.264 encoder 9

2.3.1 Intra prediction 9

2.3.2 Inter prediction 12

2.3.3 Transform coding 14

2.3.4 Quantization 15

2.3.5 De-blocking filter 15
2.4 H.264 decoder 16
2.5 Summary 18
3. INTRODUCTION TO PARALLEL PROGRAMMING 19
3.1Serial programming basics 19
3.2 Parallel programming basics 19
3.3 Limitation for serial computing 20
3.4 Classification of parallel computers 21
3.4.1 Single Instruction Single Data (SISD) 21
3.4.2 Single Instruction, Multiple Data (SIMD) 22
3.4.3 Multiple Instructions, Single Data (MISD) 22
3.4.4 Multiple Instructions, Multiple Data (MIMD) 23
3.5 Parallel programming models 24
3.5.1 Shared Memory 24
3.5.2 Threads 24
3.5.3 Distributed Memory / Message Passing 25
3.5.4 Data Parallel 25
3.6 Points to consider before writing a parallel program 26
3.6.1 Program hotspots 26
3.6.2 Bottlenecks 26
3.6.3 Inhibitors to parallelism 27
3.7 Summary 27
4. OPENMP: API SPECIFICATION FOR PARALLEL PROGRAMMING 28
4.1 OpenMP programming model 28
4.2 Goals of OpenMP 29

4.3 OpenMP core elements……..……30

4.4 OpenMP Runtime Library Routines…….……..30

4.5 OpenMP Directives……..……32

4.5.1 Parallel construct…….....…….32

4.5.2 Loop construct…….……32

4.5.3 Sections construct……..…….32

4.5.4 Single construct…….……...……33

4.5.5 Parallel loop construct……...…….33

4.5.6 Parallel sections construct…….……33

4.5.7 Task construct……..33

4.5.8 Critical construct……..33

4.5.9 Master construct……..33

4.5.10 Barrier construct……....…….33

4.5.11 Task wait construct…….…….. 34

4.5.12 Atomic construct……..…….…….34

4.6 OpenMP Clauses…….…….34

4.6.1 Default……..34

4.6.2 Shared……..…….……34

4.6.3 Private……...…….……34

4.6.4 First private……..…..34

4.6.5 Last private……..35

4.7 OpenMP Environment Variables……..…35

4.7.1 OMP_SCHEDULE……..…….……35

4.7.2 OMP_NUM_THREADS……..…….35

4.7.3 OMP_DYNAMIC…….…….35

4.7.4 OMP_NESTED 35
4.7.5 OMP_THREAD_LIMIT 35
4.8 Race Conditions 36
4.9 Summary 37
5. RESULTS OF COMPLEXITY REDUCTION USING OPENMP 39
5.1 Strategy adopted 39
5.2 Prediction Structure 40
5.3 Performance metric 41
5.3.1 %T reduction 41
5.3.2 Delta bit rate 41
5.3.3 PSNR 42
5.3.4 SSIM 42
5.3.5 CPU and physical memory usage 42
5.3.6 CPU power usage 43
5.4 Preview of test sequences 44
5.5 Encoding specifications 47
5.5.1 Configuration Parameters 47
5.6 Results with CIF and QCIF sequences 47
5.6.1 Results with CIF sequences 47
5.6.2 Results with QCIF sequences 48
5.7 Graphs 49
5.7.1 Average encoding time for all CIF sequences 49
5.7.2 Average encoding time for all QCIF sequences 50
5.7.3 PSNR graphs for CIF sequences 51
5.7.3.1 Foreman_qcif.yuv 51

5.7.3.2 Coastguard_qcif.yuv 51
5.7.3.3 Hall_qcif.yuv 52
5.7.3.4 Bridge-close_qcif.yuv 52
5.7.3.5 Mobile_qcif.yuv 53
5.7.3.6 News_qcif.yuv 53
5.7.3.7 Suzie_qcif.yuv 54
5.7.3.8 Highway_qcif.yuv 54
5.7.3.9 Mother-daughter_qcif.yuv 55
5.7.3.10 Salesman_qcif.yuv 55
5.7.3.11 Miss-america_qcif.yuv 56
5.7.3.12 Container-qcif.yuv 56
5.7.4 SSIM graphs for CIF sequences 57
5.7.4.1 Foreman_qcif.yuv 57
5.7.4.2 Bridge-close_qcif.yuv 57
5.7.4.3 Hall_qcif.yuv 58
5.7.4.4 Coastguard_qcif.yuv 58
5.7.4.5 Mobile-qcif.yuv 59
5.7.4.6 News-qcif.yuv 59
5.7.4.7 Suzie_qcif.yuv 60
5.7.4.8 Highway_qcif.yuv 60
5.7.4.9 Mother-daughter_qcif.yuv 61
5.7.4.10 Salesman_qcif.yuv 61
5.7.4.11 Miss-america_qcif.yuv 62
5.7.4.12 Container_qcif.yuv 62
5.7.5 PSNR graphs for CIF sequences 63

5.7.5.1 Foreman_cif.yuv 63
5.7.5.2 Coastguard_cif.yuv 63
5.7.5.3 Hall_cif.yuv 64
5.7.5.4 Bridge-close _cif.yuv 64
5.7.5.5 Mobile_cif.yuv 65
5.7.5.6 News_cif.yuv 65
5.7.5.7 Suzie_cif.yuv 66
5.7.5.8 Highway_cif.yuv 66
5.7.5.9 Mother-daughter_cif.yuv 67
5.7.5.10 Salesman_cif.yuv 67
5.7.5.11 Miss-america_cif.yuv 68
5.7.5.12 Container_cif.yuv 68
5.7.6 SSIM graphs for CIF sequences 69
5.7.6.1 Foreman_cif.yuv 69
5.7.6.2 Coastguard_cif.yuv 69
5.7.6.3 Hall_cif.yuv 70
5.7.6.4 Bridge-close _cif.yuv 70
5.7.6.5 Mobile_cif.yuv 71
5.7.6.6 News_cif.yuv 71
5.7.6.7 Bus_cif.yuv 72
5.7.6.8 Highway_cif.yuv 72
5.7.6.9 Mother-daughter_cif.yuv 73
5.7.6.10 Flower_cif.yuv 73
5.7.6.11 Paris_cif.yuv 74
5.7.6.12 Container_cif.yuv 74

5.7.7 CPU power usage graphs for QCIF sequences 75
5.7.8 CPU power usage graphs for CIF sequences 75
5.7.9 Physical memory usage graphs for CIF sequences 76
5.7.10 Physical memory usage graphs for QCIF sequences 76
5.8 Analysis of task based parallelism 77
5.9 Summary 78
6. CONCLUSIONS AND FUTURE WORK 80
6.1 Conclusions 80
6.2 Future work 80
REFERENCES 82
BIO GRAPHICAL INFORMATION 87

LIST OF ILLUSTRATIONS

Figure

1.1 Inter-frame predictions in modern video compression algorithms 2

1.2 Illustration of block-based motion compensation 3

2.1 Profiles for H.264 5

2.2 Interlaced video sequence 6

2.3 4:2:0, 4:2:2 and 4:4:4 sampling patterns (progressive) 8

2.4 Block diagram for H.264 video encoding and decoding 8

2.5 H.264 video encoder block diagram 9

2.6 Mode decision process of intra prediction 10

2.7 Nine intra prediction modes for 4x4 block sizes 11

2.8 Four intra prediction modes for 16x16 block sizes 12

2.9 Multi frame motion estimation 12

2.10 Example of macro block size and prediction sources 13

2.11 Macro block partitions and sub-macro block partitions 14

2.12 H.264 decoder block diagram 16

2.13 Inverse transform combining
weighted basis patterns to create a 4 × 4 image block 17

2.14 Reconstruction flow diagrams 18

3.1 Serial execution of a program 19

3.2 Parallel execution of a program 20

3.3 Classification of parallel computers 21

3.4 Instruction pipeline for a SISD system 22

3.5 Instruction pipeline for a SIMD system 22

3.6 Instruction pipeline for a MISD system 23

3.7 Instruction pipeline for a MIMD system 23

3.8 Thread based parallel programming mode 24

3.9 Message passing parallel programming model 25

3.10 Data parallel programming model 26

4.1 Fork/join model in OpenMP 28

4.2 An illustration of multithreading in OpenMP 29

4.3 Fork/join programming model in OpenMP 30

4.4 Chart of OpenMP constructs 30

5.1 Parallel encoding of 30 frames 40

5.2 Low delay, minimal storage prediction structure in H.264 video codec 41

5.3 Task manager snapshot 43

5.4 Joule-meter snapshot 44

5.5 CIF and QCIF formats 45

5.6 Preview of CIF and QCIF video sequences for testing 46

5.7 Configuration parameters for container_cif.yuv video sequence 47

5.8 Comparison of average encoding time for all CIF sequences 50

5.9 Comparison of average encoding time for all QCIF sequences 50

5.10 PSNR graph for foreman_qcif.yuv 51

5.11 PSNR graph for coastguard_qcif.yuv 51

5.12 PSNR graph for hall_qcif.yuv 52

5.13 PSNR graph for bridge-close_qcif.yuv 52

5.14 PSNR graph for mobile_qcif.yuv 53

5.15 PSNR graph for news_qcif.yuv 53

5.16 PSNR graph for suzie_qcif.yuv 54
5.17 PSNR graph for highway_qcif.yuv 54
5.18 PSNR graph for mother-daughter_qcif.yuv 55
5.19 PSNR graph for salesman_qcif.yuv 55
5.20 PSNR graph for miss-america_qcif.yuv 56
5.21 PSNR graph for container_qcif.yuv 56
5.22 SSIM graph for foreman_qcif.yuv 57
5.23 SSIM graph for bridge-close _qcif.yuv 57
5.24 SSIM graph for hall_qcif.yuv 58
5.25 SSIM graph for coastguard _qcif.yuv 58
5.26 SSIM graph for mobile_qcif.yuv 59
5.27 SSIM graph for news_qcif.yuv 59
5.28 SSIM graph for suzie_qcif.yuv 60
5.29 SSIM graph for highway_qcif.yuv 60
5.30 SSIM graph for mother-daughter_qcif.yuv 61
5.31 SSIM graph for salesman_qcif.yuv 61
5.32 SSIM graph for miss-america_qcif.yuv 62
5.33 SSIM graph for container_qcif.yuv 62
5.34 PSNR graph for foreman_cif.yuv 63
5.35 PSNR graph for coastguard_cif.yuv 63
5.36 PSNR graph for hall_cif.yuv 64
5.37 PSNR graph for bridge-close_cif.yuv 64
5.38 PSNR graph for mobile_cif.yuv 65
5.39 PSNR graph for news_cif.yuv 65
5.40 PSNR graph for suzie_cif.yuv 66

5.41 PSNR graph for highway_cif.yuv 66
5.42 PSNR graph for mother-daughter_cif.yuv 67
5.43 PSNR graph for salesman_cif.yuv 67
5.44 PSNR graph for miss-america_cif.yuv 68
5.45 PSNR graph for container_cif.yuv 68
5.46 SSIM graph for foreman_cif.yuv 69
5.47 SSIM graph for bridge-close _cif.yuv 69
5.48 SSIM graph for hall_cif.yuv 70
5.49 SSIM graph for coastguard _cif.yuv 70
5.50 SSIM graph for mobile_cif.yuv 71
5.51 SSIM graph for news_cif.yuv 71
5.52 SSIM graph for suzie_cif.yuv 72
5.53 SSIM graph for highway_cif.yuv 72
5.54 SSIM graph for mother-daughter_cif.yuv 73
5.55 SSIM graph for salesman_cif.yuv 73
5.56 SSIM graph for miss-america_cif.yuv 74
5.57 SSIM graph for container_cif.yuv 74
5.58 Comparison of CPU power usage for all QCIF sequences 75
5.59 Comparison of CPU power usage for all CIF sequences 75
5.60 Comparison of physical memory usage for all CIF sequences 76
5.61 Comparison of physical memory usage for all QCIF sequences 76

LIST OF TABLES

Table Page

4.1 Runtime library routines in OpenMP 31

4.2 Timeline for two threads in OpenMP 37

5.1 Simulation results for
CIF video sequences: QP 10 and 25 47

5.2 Simulation results for CIF
CIF video sequences: QP 35 and 45 48

5.3 Simulation results for QCIF
CIF video sequences: QP 10 and 25 48

5.4 Simulation results for QCIF
CIF video sequences: QP 35 and 45 49

LIST OF ACRONYMS

SISD - Single Instruction Single Data

SIMD - Single Instruction Multiple Data

MISD - Multiple Instructions Single Data

MIMD - Multiple Instructions Multiple Data

FPS - Frames per Second

AVC - Advanced Video Coding

VCEG - Video Coding Experts Group

MPEG - Moving Picture Experts Group

JPEG - Joint Picture Experts Group

JVT - Joint Video Team

CAVLC - Context- Adaptive Variable Length Coding

FMO - Flexible Macro-block Order

ASO - Arbitrary Slice Order

CAVLC - Context Adaptive Variable Length Coding

CABAC - Context Adaptive Binary Arithmetic Coding

HP - High Profile

H420P - High 4:2:0 profile

H422P - High 4:2:2 profile

H444P - High 4:4:4 profile

RDO - Rate Distortion Optimization

QP - Quantization Parameter

ME - Motion estimation

MV - Motion Vector

I MB - I Macro Block

P MB - P Macro Block

B MB - B Macro Block

B slice – Bi-directional predictive slice

I slice – Intra predictive slice

P slice – Inter predictive slice

DCT - Discrete Cosine Transform

MB - Macro Blocks

IDCT - Inverse Discrete Cosine Transform

CPU - Central Processing Unit

OpenMP - Open Multiprocessing

API - Application Programming Interface

GOP - Groups of Pictures

PSNR - Peak Signal to Noise Ratio

CIF - Common Intermediate Format

QCIF - Quarter Common Intermediate Format

SSIM - Structural Similarity Index Metric

CHAPTER 1

INTRODUCTION

Digital video has become the main stream and is being used in a wide range of applications including pocket PCs, handheld PCs, i-phones and i-pods [15] [17] [18]. These devices are either used as a source of entertainment for capturing and saving real time data or for visual communication purposes as video telephony and teleconferencing. As a result of which more and more users are seeking real-time video communication services with the rapid development of wireless networks. The emergence of digital cameras for mobile devices also provides conditions for real-time video communication. In order to meet the requirements of real-time video communication, the encoder as well as the decoder of the video codec needs to reduce the processing time of the frames. While doing so, care should also be taken that the qualities of the image as well as the bit rate of the multimedia file have not changed drastically.

Consider a digital video sequence having a picture resolution of 720x480 and a frame rate of 30 frames per second (FPS). If a picture is represented using the YUV color space with 8 bits per component or 3 bytes per pixel, size of each frame is 720x480x3 bytes. The disk space required to store one second of video is 720x480x3x30 = 31.1 MB. A one hour video would thus require 112 GB. With the number of devices inside household increasing, the bandwidth requirement is also increasing [16]. In addition to these extremely high storage and bandwidth requirements, using uncompressed video will add significant cost to the hardware and systems that process digital video. Digital video compression is thus necessary even with exponentially increasing bandwidth and storage capacities. Fortunately, digital video has significant redundancies and eliminating or reducing those redundancies results in compression [13].

Video compression is typically achieved by exploiting [3].

1. Spatial
2. Temporal
3. Statistical redundancies

1.1 H.264

H.264 or AVC (Advanced Video Coding) [3] [4] [5] [17] [18] is a digital video codec standard which is noted for achieving very high data compression [2]. It was developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) [4] as the product of a collective partnership effort known as the Joint Video Team (JVT) [2]. H.264 is an algorithm used for video compression. It is the latest industry standard for video compression and is the first choice of preference for most of the companies for its products and services [6]. For compression H.264 uses a technique known as difference coding where only the first image is coded entirely in a group of pictures.

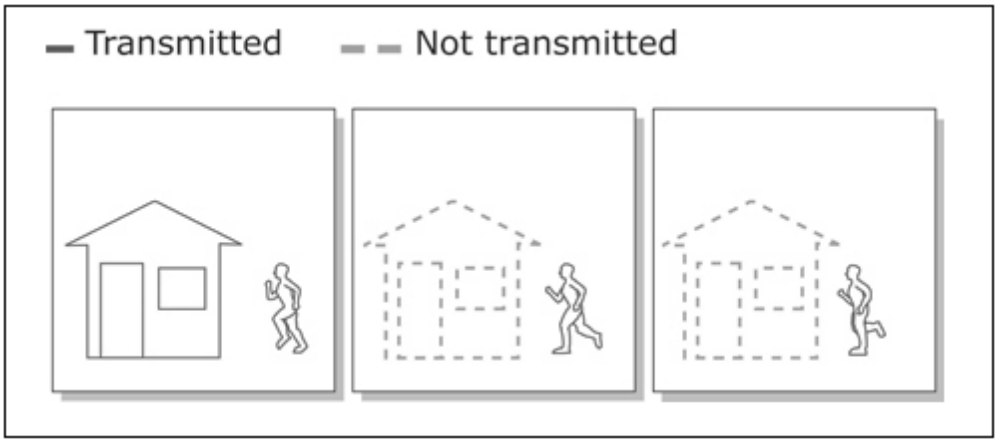

Figure 1.1 Inter-frame predictions in modern video compression algorithms [7]

As seen from Figure 1.1, only the moving parts of the picture i.e. the motion of the man are coded using motion vectors, thus reducing the amount of information that is sent and stored. Also, techniques like block-based motion compensation are included to further reduce the data.

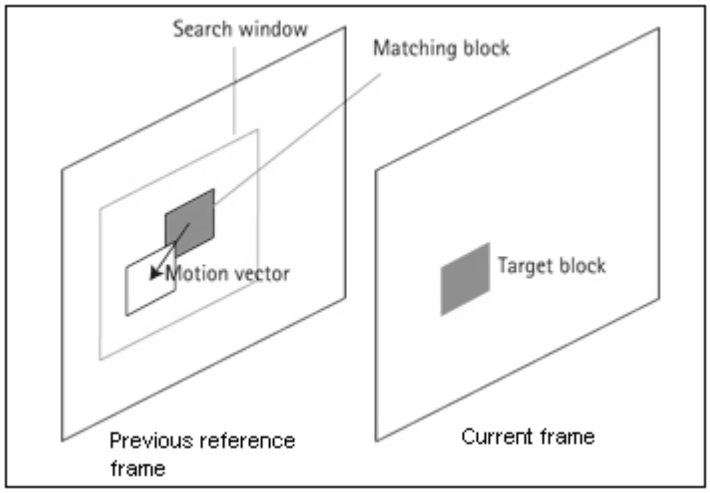

Figure 1.2 Illustration of block-based motion compensation [7]

In block based motion compensation, the entire frame is divided into blocks. Each block in the current frame is then matched to find a similar block in the reference around a search window. Referring to Figure 1.2, the target block in the current frame is matched with different blocks in the reference frame within the search window. Instead of transmitting the entire target block the motion vectors for the most appropriate matched blocks are coded and transmitted thus reducing the amount of information sent [8].

Thus difference coding and block-based motion compensation helps in reducing the bit rate [7] but the encoding time using the above two techniques is quite large. This acts as a hindrance for real time capability. To reduce this encoding time many different Intra mode approaches like in [9], [10], [11] and [12] have been proposed. These approaches bring about a change in the intra mode algorithm thus sacrificing the quality of the original video clip in return. It is important to maintain a tradeoff between encoding time and quality. Hence an efficient algorithm to reduce the encoding time for an H.264 video codec is proposed here having the same quality level as the original.

1.2 Summary

The thesis proposes an efficient way of reducing the encoding time for H.264 baseline profiles by incorporating parallel programming paradigms. It exploits the fact that the encoding algorithm for every frame is the same; hence by incorporating task level parallelism to encode two or more than two frames simultaneously the encoding time can be reduced drastically. The reference software used in this thesis is JM18.0 [13] and OpenMP [14], set of parallel programming library and was used to incorporate task level parallelism. Chapter 2 describes about the video coding standards and video formats.

CHAPTER 2

OVERVIEW OF H.264 / AVC STANDARD

INTRODUCTION

H.264 [39] Advanced Video Coding standard [19] [20] significantly improves video compression ratio and enhances the power of the video coding algorithms. It enhances and adds up new features for the next generation internet-based wire-line and wireless video applications such as cellular camera-phone, video-based web browser, smart set-top box with video-on-demand and game and packet-based video broadcast/on-demand [21].

2.1 H.264 profiles

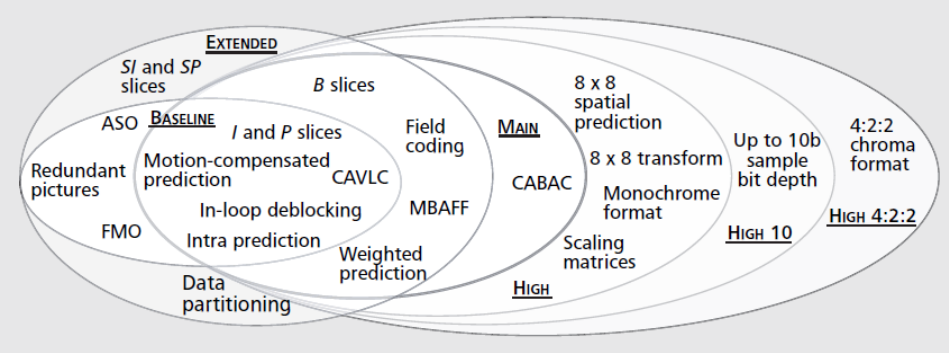

Figure 2.1 Profiles for H.264 [3]

As seen from Figure 2.1, H.264 [39] has the following profiles – Baseline, Extended, Main and High. The properties for the following profiles are as discussed below [22] –

2.1.1 Baseline profile

Baseline profile consists of the following features -

1. I, P slices only
2. CAVLC (Context- Adaptive Variable Length Coding) for entropy coding.

3. Flexible Macro-block Order (FMO): Macro-blocks may not necessarily be in the raster scan order. The map assigns macro-blocks to a slice group. Arbitrary slice order (ASO): The ordering of the slices in the bit stream may not be in raster scan order. Figure 2.2 shows the scanning of the frames as complete frames or as sequence of interlaced fields.

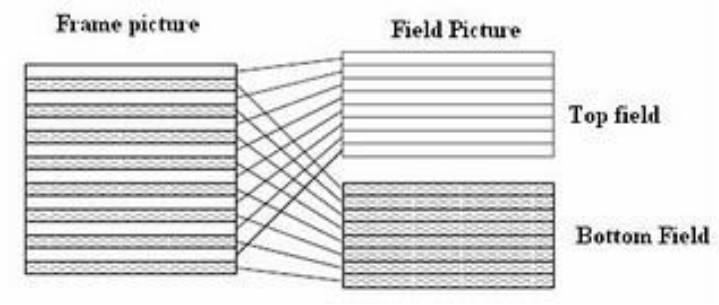

Figure 2.2 Interlaced video sequence [3]

A video signal can be progressively sampled (series of complete frames) or interlaced (sequence of interlaced fields). In an interlaced video sequence two fields comprise one video frame (figure 2.2) and a field consists of either the odd-numbered or even-numbered lines 13 within a complete video frame. The advantage of this sampling method is that it is possible to send twice as many fields in an equivalent progressive sequence with the same data rate, giving the appearance of smoother motion [3], [8].

2.1.2 Extended profile

Extended profile consists of the following features -

1. All the features of baseline profile
2. SP slices – SP frames make use of motion compensated predictive coding to exploit temporal redundancy in the sequence similar to P frames. It allows identical frames to be reconstructed even when they are predicted using different reference frames [40].

3. SI slices – SI frames are used in conjunction with SP frames. An SI frame uses only spatial prediction as an I frame and still reconstructs identically the corresponding SP frame which uses motion compensated prediction [40].
4. B slices
5. Weighted prediction

2.1.3 Main profile

Main profile consists of the following features -

1. I,P,B slices
2. Weighted prediction
3. CABAC (context adaptive binary arithmetic coding) / CAVLC (context adaptive variable length coding) for entropy coding.

2.1.4 High profile [23], [24]

High profile consists of the following features -

1. Includes high profile (HP) supporting 8-bit video with 4:2:0 sampling, addressing high end consumer use and other applications using high resolution video. The 4:2:0 sampling shown in Figure 2.3.
2. 8x8 intra prediction.
3. High 4:2:2 profile (H422P), supporting up to 4:2:2 sampling and up to 10-bits per sample. The 4:2:2 sampling shown in Figure 2.3.
4. High 4:4:4 profile (H444P), supporting up to 4:4:4 sampling, up to 12 bits per sample, lossless region coding and integer residual color transform for coding RGB video. The 4:4:4 sampling is as shown in Figure 2.3.
5. All features of main profile
6. Adaptive block size transform (introduction of 8x8 integer DCT)
7. Perceptual quantization matrices.

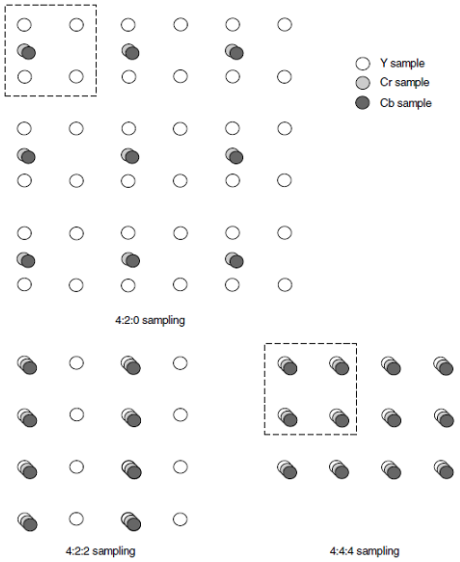

Figure 2.3 4:2:0, 4:2:2 and 4:4:4 sampling patterns (progressive) [5]

2.2 H.264 block diagram

Basic block diagram for H.264 [39] video encoding and decoding is shown in Figure 2.2.

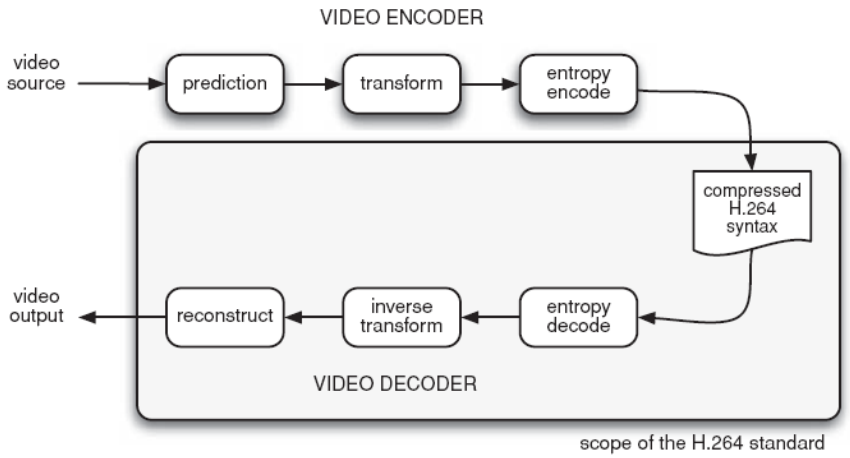

Figure 2.4 Block diagram for H.264 video encoding and decoding [3]

H.264 [39] takes the advantages of many kinds of coding methods and blends them into one progress, in which there is intra prediction; inter prediction, integer DCT, Run-Level scan and entropy encoding, which can reduce the redundancy of spatial, temporal and statistical

respectively. Figure 2.4 gives a brief view of the coding methods incorporated in this video codec. Compared with the former video coding standards, such as MPEG-4 [41] and H.263 [42], H.264 [39] has many obvious advantages for e.g. more intra prediction modes, quarter pixel's motion estimation and compensation, integer DCT, higher compression ratio, better picture quality and internet adaptability. For these advantages, H.264 [39] video coding standard has been and will be widely used in many telecommunication domains such as digital television broadcast, satellite broadcast, mobile phone television and internet based streaming media technique [27].

2.3 H.264 encoder

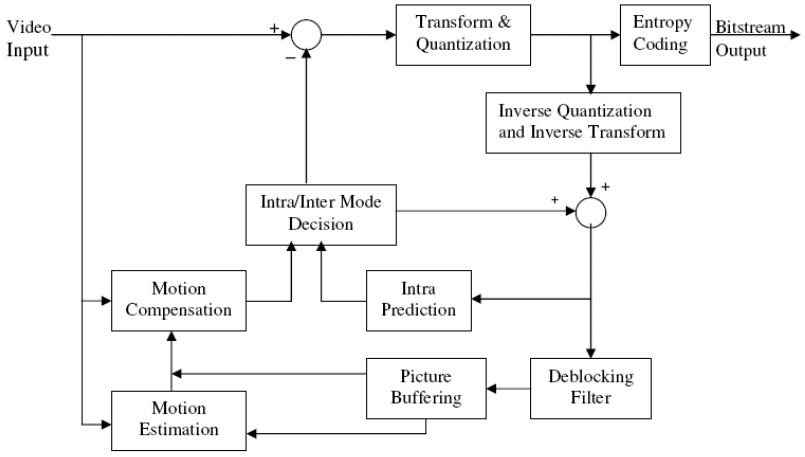

Figure 2.5 H.264 video encoder block diagram [22]

Figure 2.5 shows the detailed encoder block diagram for H.264 video codec. Different blocks can be explained as below –

2.3.1 Intra prediction

H.264/AVC introduced new intra-prediction methods which offer a still image coding performance that is comparable or superior to the JPEG and JPEG2000 image coding standards [26]. The H.264/AVC intra-prediction methods are directional in nature. There are different block sizes, each having a different number of modes and making use of up to 4

neighboring blocks for prediction. Despite this great encoding flexibility, which adapts itself to the image characteristics, only the block's surrounding pixels are used for prediction [25].

H.264 also provides various intra-prediction modes for both gray and color videos. For a gray signal, there are intra_4×4 prediction modes whose block is more suitable for detail information and intra_16×16 prediction modes suitable for smooth changes in the image. For color components, prediction mode is only intra_8×8 [28][29][30]. Selection for an intra prediction mode in H.264 usually adopts the rate distortion optimization (RDO) technology which intends to achieve the best encoding effect under the least bit rate. The rate distortion cost (*RD_Cost*) can be expressed as follows [28] –

$$RD\_Cost = SSD + \lambda * R$$

where *SSD* is the sum of squared differences between the actual pixels and the predicted pixels, *R* is the coding rate, and λ is a Lagrange parameter connected with the quantization parameter (*QP*) as

$$\lambda = 0.85 * 2^{(QP-12)/3}$$

Conventionally, when processing a macro block for intra-coding, all prediction modes are needed to be traversed and a mode decision process is used to select the optimal mode as the final coding mode [28]. Figure 2.4 shows the mode decision process of intra-prediction.

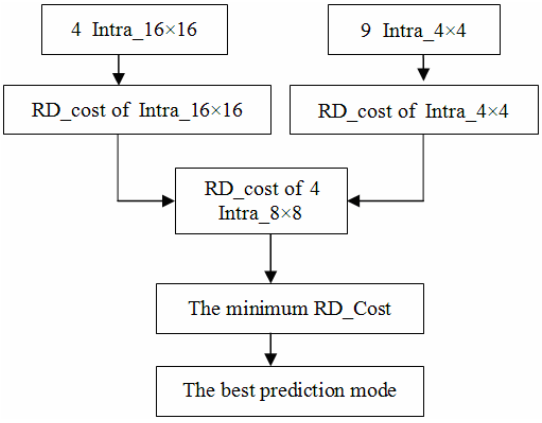

Figure 2.6 Mode decision process of intra prediction [28]

As shown in Figure 2.6, the mode decision algorithm for intra prediction mode selection for a macro-block can be depicted as follows [28] –

Step 1 - The macro-block is divided into sixteen 4×4 small blocks, then nine different intra-prediction modes are adopted respectively in each block and an optimal one is selected by the *RD_Cost* of the mode in each 4×4 block.

Step 2 - Similarly four kinds of intra-prediction modes for 16×16 blocks are tested for the macro-block and the *RD_Costs* of every mode is calculated. Also an optimal mode is selected by the minimum *RD_Cost*. Comparing the minimum *RD_Cost* of 4×4 block and that of 16×16 block, intra_4×4 or intra_16×16 block dividing intra-prediction mode is decided.

Step 3 - For color video, testing four prediction modes of 8×8 mode for chroma macro block, repeating above steps to select a intra-prediction mode with the minimum *RD_Cost* for chroma block.

Step 4 – Mode with minimum *RD_Cost is chosen as the best intra predicted mode.*

The different modes available to exploit spatial redundancy using intra prediction are as shown in Figures 2.7and 2.8.

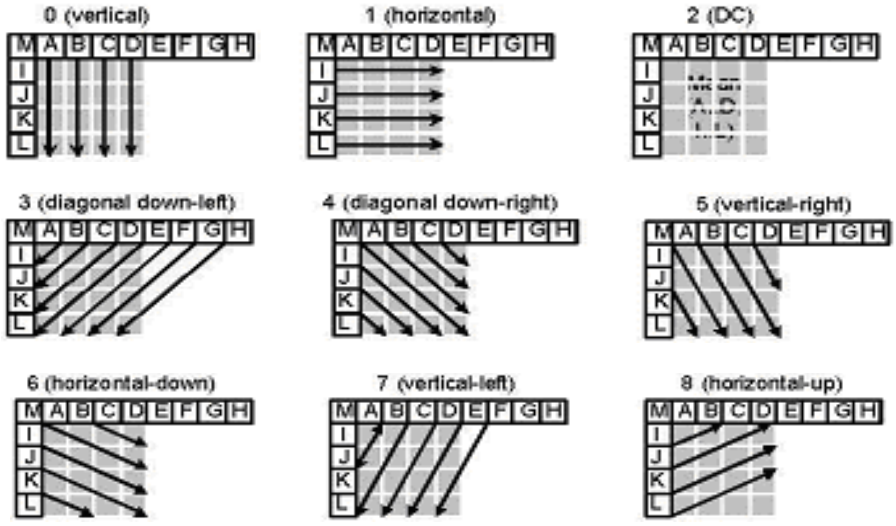

Figure 2.7 Nine intra prediction modes for 4x4 block sizes [31]

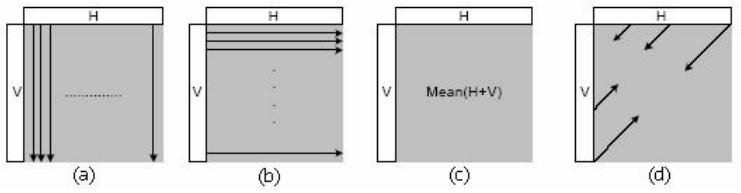

Figure 2.8 Four intra prediction modes for 16x16 block sizes [31]

Nine intra prediction modes for 8x8 block sizes as same as that for 4x4 block sizes [31]

2.3.2 Inter prediction

Inter prediction is the process of predicting a block of luma and chroma samples from a picture that has been previously coded and transmitted. This involves selecting a prediction region, generating a prediction block and subtracting this from the original block of samples to form a residual that goes through transform, quantization and entropy coding. The block of samples to be predicted, a macro-block partition or sub-macro block partition, can range in size from a complete macro-block, i.e. 16 × 16 luma samples and corresponding chroma samples, down to a 4 × 4 block of luma samples and corresponding chroma samples.

Motion estimation (ME) is an important part of inter-picture prediction. It is used to reduce temporal redundancy. It is a process of determining the best motion vectors that describe the transformation from one frame to another. Motion vector (MV) described as (dx, dy) is displacement vector of a moving object. Figure 2.9 shows a diagram depicting ME [3].

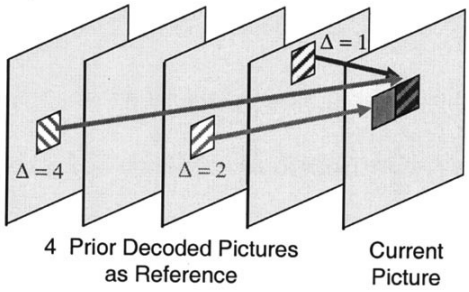

Figure 2.9 Multi frame motion estimation [1]

As shown in Figure 2.9, the current frame is composed of blocks. These blocks are predicted either from the last previously encoded frame alone or from a group of previously encoded frames.

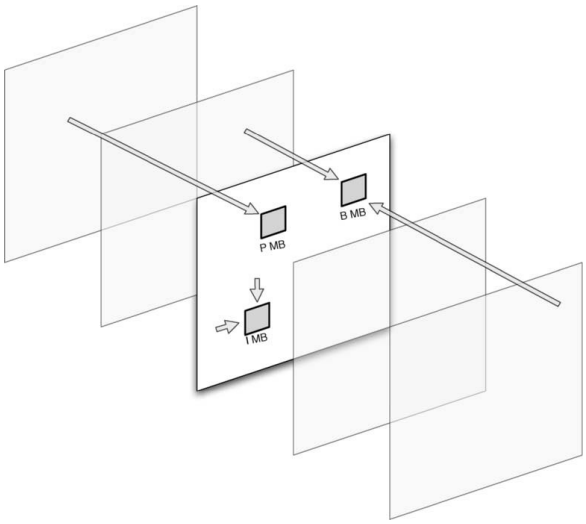

Figure 2.10 Example of macro block size and prediction sources [3]

Figure 2.10 shows the prediction sources for three macro-blocks, an I macro block, a P macro block and a B macro block. An I macro block (I MB) is predicted using intra prediction from neighboring samples in the current frame. A P macro block (P MB) is predicted from samples in a previously-coded frame. Different rectangular sections in a P MB may be predicted from different reference frames. Each partition in a B macro block (B MB) is predicted from samples in one or two previously-coded frames, for example, one ‘past’ and one ‘future’ as shown in the Figure 2.10 [3].

Each 16×16 P or B macro block may be predicted using a range of block sizes. The macro block is split into one, two or four macro block partitions as shown in Figure 2.12 [3] -

1. one 16 × 16 macro block partition (covering the whole MB)
2. two 8 × 16 partitions
3. two 16 × 8 partitions

4. four 8 × 8 partitions

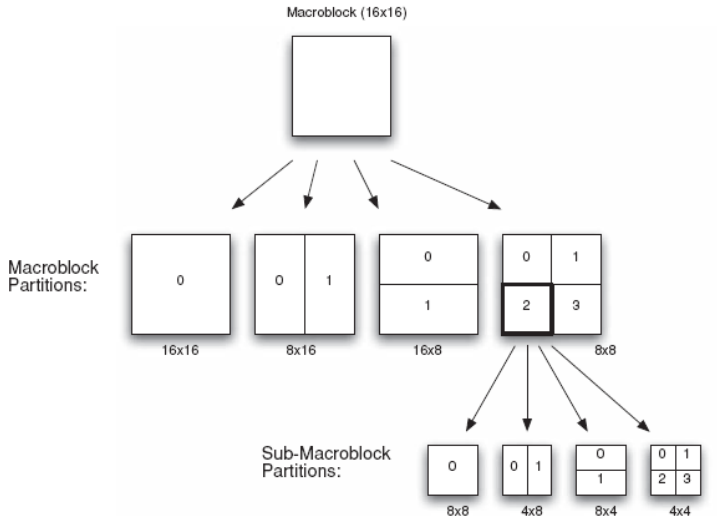

Figure 2.11 Macro block partitions and sub-macro block partitions [3]

If 8 × 8 partition size is chosen, then each 8 × 8 block of luma samples and associated chroma samples, a sub-macro block, is split into one, two or four sub-macro block partitions, one 8 × 8, two 4 × 8, two 8 × 4 or four 4 × 4 sub-MB partitions. Each macro block partition and sub-macro block partition has one or two motion vectors (x, y), each pointing to an area of the same size in a reference frame that is used to predict the current partition. A partition in a P macro block has one reference frame and one motion vector, a partition in a B macro block has one or two reference frames and one or two corresponding motion vectors [3].

2.3.3 Transform coding

The discrete cosine transform (DCT) is a prevalent transform method, widely-used in signal processing and multimedia applications. It is a real orthogonal computation, the performance of which is close to the K-L transform [35].An improvement of DCT-integer transform [33] inherits the advantages of DCT but avoids the mismatch between encoder and decoder. Integer transform is adopted by H.264 video coding standard to provide high

compression efficiency. The H.264/AVC uses a 2D-DCT based integer transform "C" for all 4x4 block of residual pixel data "X" and is given as follows [32][35] –

$$Z = C \; X \; C^{T}$$

where superscript T implies transpose of a matrix and matrix C is as given below

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

H.264 employs a purely integer spatial transform (a rough approximation of the DCT [36]) which is primarily 4x4 in shape, as opposed to the usual floating-point 8x8 DCT specified with rounding-error tolerances as used in earlier standards. The small shape helps to reduce blocking and ringing artifacts, while the precise integer specification eliminates any mismatch issues between the encoder and decoder in the inverse transform. It is also computationally less expensive as the number of multiplications is reduced [8].

2.3.4 Quantization

H.264 assumes a scalar quantizer. It can be represented by the following equation -

$$Z_{(i\,j)} = round(Y_{(i\,j)} / Qstep)$$

where $Y_{(i\,j)}$ is a transform coefficient, Qstep is a quantizer step size and $Z_{(i\,j)}$ is a quantized coefficient. A total of 52 values of Qstep are supported by the quantization parameter (QP). The wide range of quantizer step sizes makes it possible for an encoder to control the tradeoff accurately and flexibly between bit rate and quality [22].

2.3.5 Deblocking filter

Reducing the temporal and spatial redundancies of video sequences during coding can cause blocking artifacts. H.264/AVC standard defines methods of block-based prediction,

transformation and quantization for encoding/decoding. A frame or a slice is divided into many macro blocks (MBs) and each MB contains 16 × 16 pixels. Furthermore, a MB may be divided into 16 sub blocks and each consists of 4 × 4 pixels as shown in Figure 2.11. Operations based on blocks are one of the main reasons for blocking artifacts. Further, quantization process for transform coefficients in DCT (discrete cosine transform) transformation is relatively rough and it causes errors of transform coefficients which once pass through IDCT (inverse discrete cosine transform) process and cause blocking effect. One of the solutions to remove blocking artifacts is the in-loop deblocking algorithm used in H.264/AVC.

2.4 H.264 decoder

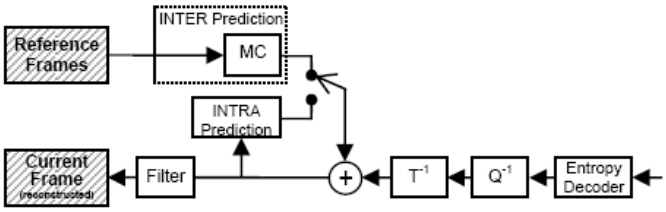

Figure 2.12 H.264 decoder block diagram [38]

A video decoder receives the compressed H.264 bit stream, decodes each of the syntax elements and extracts the information described above as shown in Figure 2.12, i.e. quantized transform coefficients, prediction information, etc. This information is then used to reverse the coding process and recreate a sequence of video images [3].

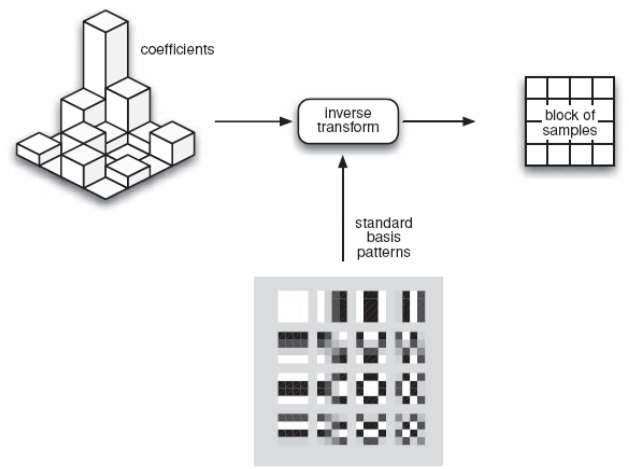

Figure 2.13 Inverse transform combining weighted basis patterns to create a 4 × 4 image block [3]

The quantized transform coefficients are re-scaled. Each coefficient is multiplied by an integer value to restore its original scale. The quantized coefficients are each multiplied by a QP. An inverse transform combines the standard basis patterns, weighted by the re-scaled coefficients, to re-create each block of residual data. Figure 2.13 shows how the inverse DCT or integer transform creates an image block by weighting each basis pattern according to a coefficient value and combining the weighted basis patterns [3].

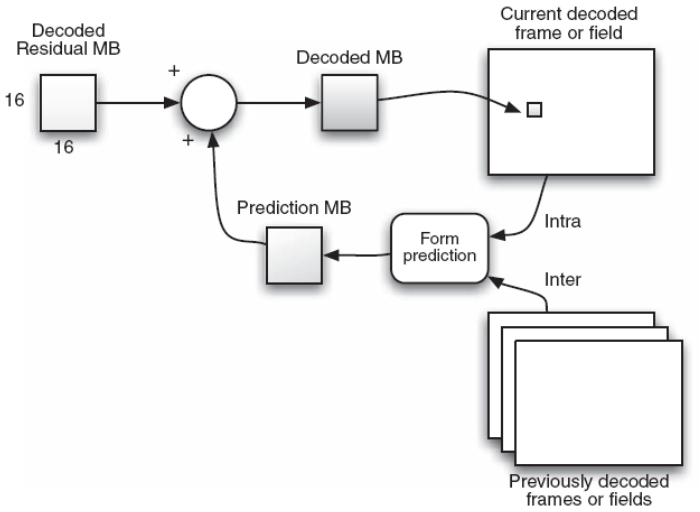

Figure 2.14 Reconstruction flow diagrams [3]

Figure 2.14 shows the reconstruction process at the decoder. For each macro block, the decoder forms an identical prediction to the one created by the encoder using inter prediction from previously-decoded frames or intra prediction from previously-decoded samples in the current frame. The decoder adds the prediction to the decoded residual to reconstruct a decoded macro block which can then be displayed as part of a video frame [3].

2.5 Summary

This chapter has presented a basic introduction to the H.264 video coding standard. H.264 performs better than the previous video coding standards by introducing new innovative algorithms and improving some previously used algorithms to provide superior visual quality at lower bitrates with better error resilience [22]. Chapter 3 describes about the hyper-threading technology.

CHAPTER 3

INTRODUCTION TO PARALLEL PROGRAMMING

3.1 Serial programming basics [53], [54]

Traditionally, software has been written for serial computation and has the following properties:

1. To be run on a single computer having a single Central Processing Unit (CPU).
2. A problem is broken into a discrete series of instructions.
3. Instructions are executed one after another.
4. Only one instruction may execute at any moment in time.

A typical scenario for a serial computation can be as shown in figure 3.1 where the problem to be solved is divided into instructions and executed one at a time.

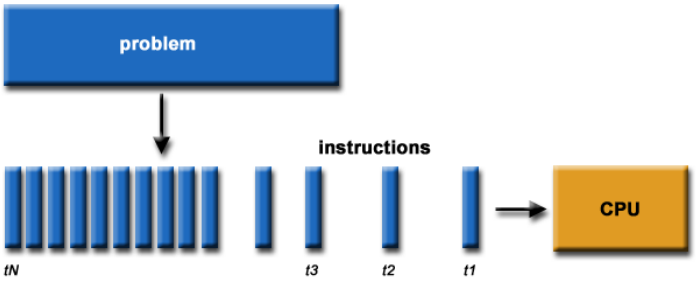

Figure 3.1 Serial execution of a program [53]

3.2 Parallel programming basics [53], [54]

In the simplest sense, *parallel computing* is the simultaneous use of multiple computer resources to solve a computational problem. The problem is broken into discrete parts that can be solved concurrently using multiple CPU's. Each part is further broken down into a series of instructions which part execute simultaneously on different CPUs.

A typical scenario for a serial computation can is shown in figure 3.2 where the problem to be solved is divided into instructions and multiple instructions are executed simultaneously depending upon the number of available cores.

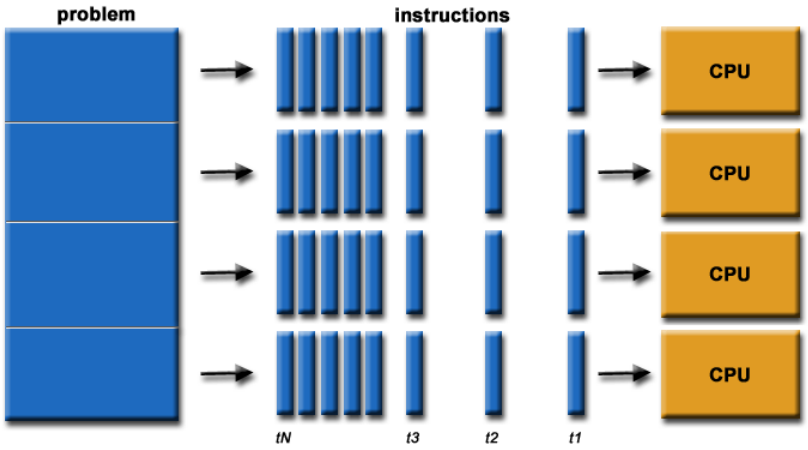

Figure 3.2 Parallel execution of a program [53]

3.2 Limitation for serial computing [53], [54]

Both physical and practical reasons pose significant constraints to simply building ever faster serial computers:

1. Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
2. Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
3. Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

4. Current computer architectures are increasingly relying upon hardware level parallelism to improve performance using multiple execution units, pipelined instructions and multi-core processors.

3.4 Classification of parallel computers [53], [54]

There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy. It distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of instruction and data. Each of these dimensions can have only one of two possible states: Single or Multiple. The matrix shown below in figure 3.3 defines the 4 possible classifications according to Flynn's Taxonomy [20] -

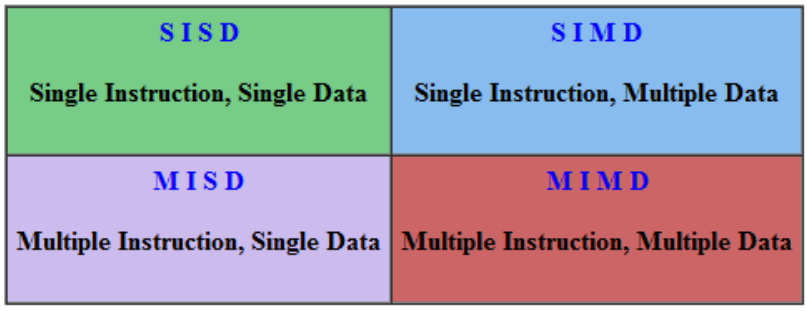

Figure 3.3 Classification of parallel computers [53]

3.4.1 Single Instruction Single Data (SISD)

In this set of parallel computers only one instruction and data stream is being acted on by the CPU during any one clock cycle. It shows a deterministic behavior. It is the oldest and even today, the most common type of computer. Figure 3.4 shows the instruction pipeline for a SISD system where only one instruction stream is being acted on by the CPU and only one data stream is being used as input in a clock cycle.

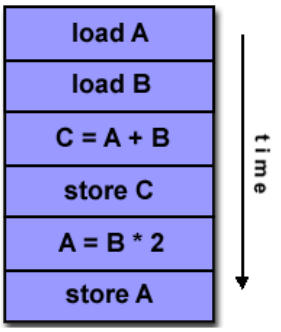

Figure 3.4 Instruction pipeline for a SISD system [53]

3.4.2 Single Instruction, Multiple Data (SIMD)

In this set of parallel computers all processing units execute the same instruction at any given clock cycle. Each processing unit can operate on a different data element. It is best suited for graphics/image processing. Modern computers with graphics processor units (GPUs) employ SIMD instructions and execution units. Figure 3.5 shows the instruction pipeline for a SIMD system where all processing units execute the same instruction and each processing unit operates on a different data element.

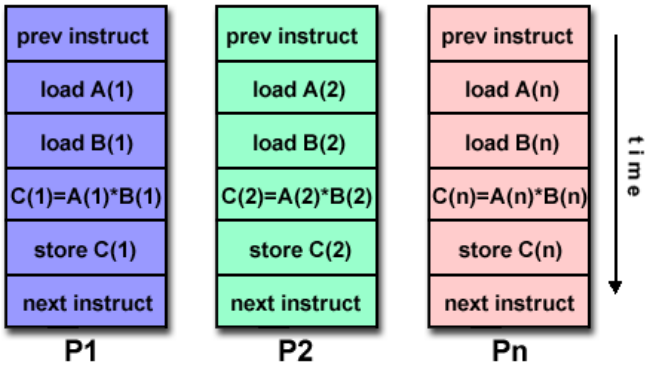

Figure 3.5 Instruction pipeline for a SIMD system [53]

3.4.3 Multiple Instructions, Single Data (MISD)

In this set of parallel computers each processing unit operates on the data independently via separate instruction streams. A single data stream is fed into multiple

processing units. Figure 3.6 shows the instruction pipeline for a MISD system where each processing unit operates on the data independently via separate instruction streams and a single data stream is fed into multiple processing units.

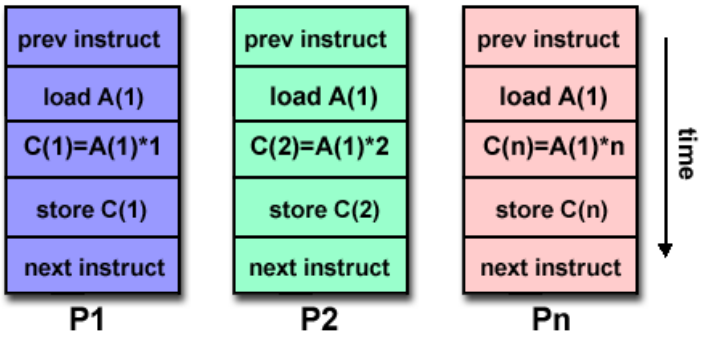

Figure 3.6 Instruction pipeline for a MISD system [53]

3.4.4 Multiple Instructions, Multiple Data (MIMD)

In this set of parallel computers every processor may be executing a different instruction from different data stream. Execution can be synchronous or asynchronous, deterministic or non-deterministic. Currently, the most common type of parallel computer - most modern supercomputers fall into this category. Figure 3.7 shows the instruction pipeline for a MIMD system where each processing unit is executing a different instruction stream working on different sets of data.

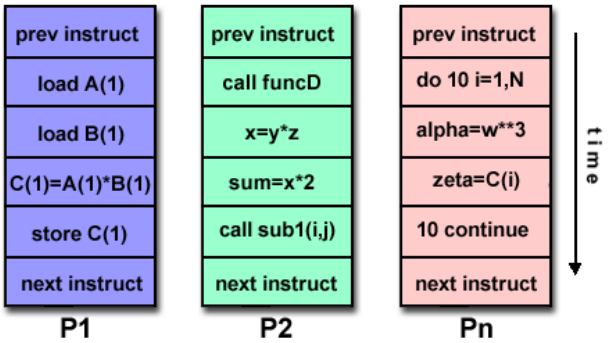

Figure 3.7 Instruction pipeline for a MIMD system [53]

3.5 Parallel programming models [53], [54]

Parallel programming models exist as an abstraction above hardware and memory architectures. These models are not specific to a particular type of machine or memory architecture. Any of these models can (theoretically) be implemented on any underlying hardware. Some of the parallel programming models are described as below:

3.5.1 Shared Memory

In this programming model, tasks share a common address space. Mechanisms such as locks / semaphores are used to control access to the shared memory. An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks. An important disadvantage in terms of performance is that it becomes more difficult to understand and manage data locality.

3.5.2 Threads

This programming model a single process can have multiple, concurrent execution paths. It is a type of shared memory programming. A typical scenario for a thread based parallel programming model is shown in figure 3.7. Here each function is executed in a different thread in a multi-threaded environment.

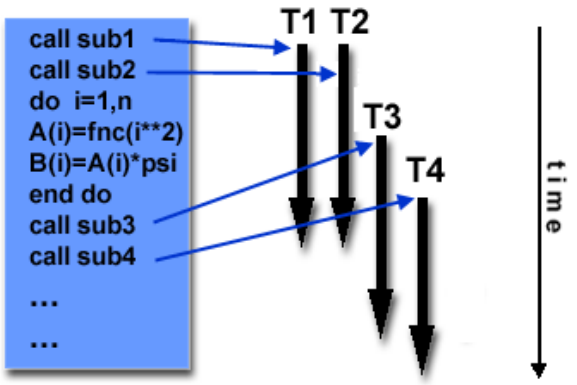

Figure 3.8 Thread based parallel programming model [53]

3.5.3 Distributed Memory / Message Passing

This programming model multiple tasks reside on the same physical machine and/or across an arbitrary number of machines. Tasks exchange data through communications by sending and receiving messages. Data transfer usually requires cooperative operations to be performed by each process. A typical scenario for a message passing parallel programming model is shown in figure 3.9. In this model, threads on different machines operate on a data and send the processed data to other machines via a network.

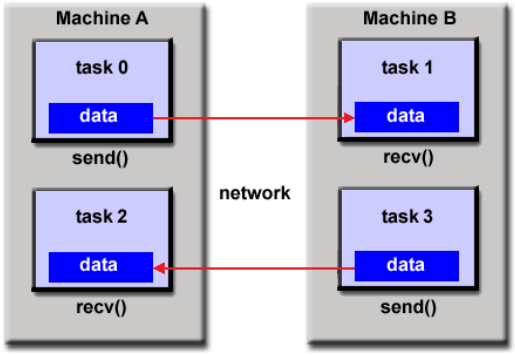

Figure 3.9 Message passing parallel programming model [53]

3.5.4 Data Parallel

This programming model focuses on performing operations on a data set. The data set is organized into a common structure, such as an array or cube. A set of tasks, work collectively on different partition of the same data structure. A typical scenario for a data parallel programming model is shown in figure 3.10. In this all tasks have access to the same data structure through global memory but operate on different indexes of the data structure.

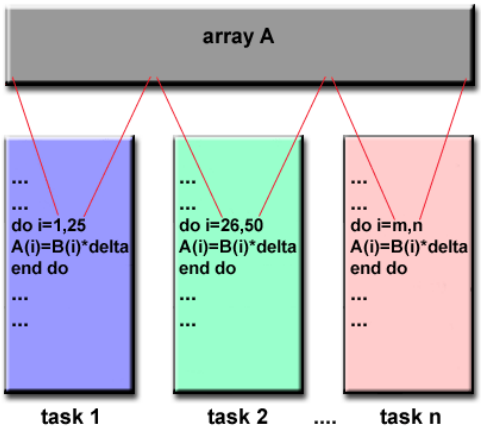

Figure 3.10 Data parallel programming model [53]

3.6 Points to consider before writing a parallel program [53], [54]

The first step in developing parallel software is to first understand the problem that one wishes to solve in parallel. Starting with a serial program necessitates understanding the existing code also. One also needs to determine whether or not the problem is one that can actually be parallelized. Few points that one needs to be considered are the program's hotspots, bottlenecks in the program and inhibitors to parallelism.

3.6.1 Program hotspots

The majority of scientific and technical programs usually accomplish most of their work in a few places. Profilers and performance analysis tools can help to identify these hot spots. Hence the focus should be in parallelizing the hotspots and ignore those sections of the program that account for little CPU usage.

3.6.2 Bottlenecks

There are areas that are disproportionately slow and cause parallelizable work to halt and be deferred. For example, I/O is usually something that slows a program down. In such a scenario a different algorithm should be used to reduce or eliminate unnecessary slow areas.

3.6.3 Inhibitors to parallelism

As demonstrated in the following example - Calculation of the Fibonacci series (0, 1,1,2,3,5,8,13,21...) by use of the formula

$$F(n) = F(n-1) + F(n-2)$$

This is a non-parallelizable problem because the calculation of the Fibonacci sequence as shown would entail dependent calculations rather than independent ones. The calculation of the F(n) value uses those of both F(n-1) and F(n-2). These three terms cannot be calculated independently and therefore, not in parallel. This common class of inhibitor is termed as *data dependence*.

3.7 Summary

This chapter gives an introduction to parallel programming paradigm, parallel programming models, limitation of serial programming and also the steps to consider while writing a parallel program.

The next chapter gives a detailed explanation to implement a parallel application on a CPU (Central Processing Unit) using a parallel programming library OpenMP (Open Multi Programming). It details all the directives and the environment variables which one needs to incorporate in the application to make it run parallel.

CHAPTER 4

OPENMP: API SPECIFICATION FOR PARALLEL PROGRAMMING

4.1 OpenMP programming model

OpenMP (Open Multiprocessing) is an API (Application Programming Interface) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran on most processor architectures and operating systems, including Solaris and Windows platforms. It consists of a set of compiler directives, library routines and environment variables that influence run-time behavior [14]. [43], [44], [45], [46], [50]. OpenMP is an implementation of multithreading. As shown in Figure 4.1 it is a method of parallelizing whereby a master *thread* (a series of instructions executed consecutively) *forks* a specified number of slave *threads* and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors [43], [50].

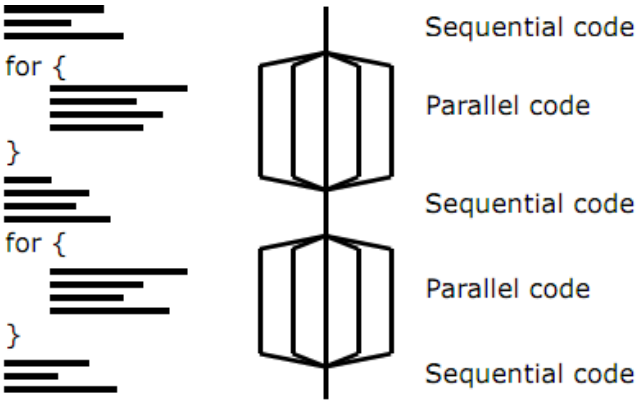

Figure 4.1 Fork/join model in OpenMP [50], [51]

The section of code that is meant to run in parallel is marked accordingly with a preprocessor directive that will cause the threads to form before the section is executed. Each

thread has an *id* attached to it. An illustration of multithreading can be seen in figure 4.2 where the master thread forks into many slave threads in a parallel region to perform the task in parallel. The thread id is an integer and the master thread has an id of *0*. After the execution of the parallelized code, the threads *join* back into the master thread, which continues onward to the end of the program. By default, each thread executes the parallelized section of code independently. *Work-sharing constructs* can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both task parallelism and data parallelism can be achieved using OpenMP in this way [43], [50].

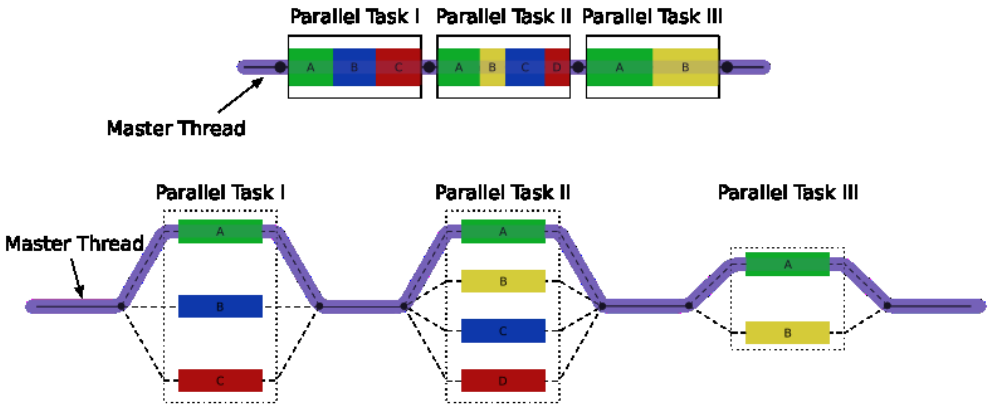

Figure 4.2 An illustration of multithreading in OpenMP [43]

The runtime environment allocates threads to processors depending on usage, machine load and other factors. The number of threads can be assigned by the runtime environment based on environment variables or in code using functions.

4.2 Goals of OpenMP [49], [50]

Fork/join programming model in OpenMP provides a standard for shared memory architectures/platforms. It establishes simple and limited set of directives for programming shared memory machines. Significant parallelism can be implemented by using just 3 or 4 directives. It provides support for portability of C and C++. Diagrammatic representation for fork/join model of OpenMP is shown in figure 4.3.

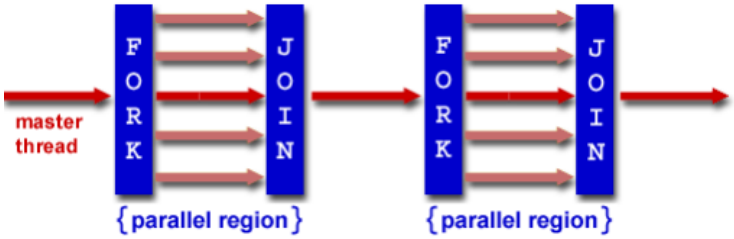

Figure 4.3 Fork/join programming model in OpenMP [49]

4.3 OpenMP core elements [43], [50]

The core elements of OpenMP are shown in figure 4.4. They can be categorized as follows - constructs for thread creation, workload distribution (work sharing), data-environment management, thread synchronization, user-level runtime routines and environment variables.

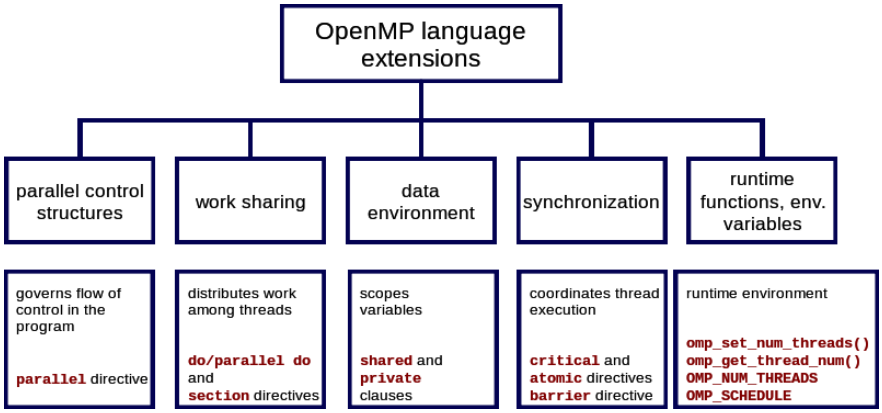

Figure 4.4 Chart of OpenMP constructs [43]

4.4 OpenMP Runtime Library Routines [50], [52]

Execution environment routines affect and monitor threads, processors and the parallel environment. The routines are explained in table

Table 4.1 Runtime library routines in OpenMP

Environment Routines	Description
omp_set_num_threads	Sets the number of threads used for subsequent parallel regions that do not specify a num_threads clause.
omp_get_num_threads	Returns the number of threads in the current team.
omp_get_max_threads	Returns maximum number of threads that could be used to form a new team using a parallel construct without a num_threads clause.
omp_get_thread_num	Returns the ID of the encountering thread where ID ranges from zero to the size of the team minus 1.
omp_get_num_procs	Returns the number of processors available in the program.
omp_in_parallel	Returns true if the call to the routine is enclosed by an active parallel region; otherwise, it returns false.
omp_get_team_size	Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.
omp_init_lock	This routine initializes an OpenMP lock.
omp_destroy_lock	This routine ensures that the OpenMP lock is uninitialized.
omp_set_lock	This routine provides a means of setting an OpenMP lock.
omp_unset_lock	This routine provides a means of unsetting an OpenMP lock.
omp_test_lock	These routines attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

4.5 OpenMP Directives [50], [52]

This section gives a brief overview of directives typically used in OpenMP. An OpenMP executable directive applies to the succeeding structured block. A structured-block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

4.5.1 Parallel construct

It is one of the basic constructs that initiates a parallel execution. The parallel construct forms a team of threads and starts parallel execution.

4.5.2 Loop construct

The loop construct specifies that the iterations of loops will be distributed among teams and executed by the encountering team of threads.

There can be various kinds in the loop as explained below:

1. Static: Iterations are divided into chunks of size chunk_size. Chunks are assigned to threads in the team in round-robin fashion in order of thread number.
2. Dynamic: Each thread executes a chunk of iterations and then requests another chunk until no chunks remain to be distributed.
3. Guided: Each thread executes a chunk of iterations and then requests another chunk until no chunks remain to be assigned. The chunk sizes start large and shrink to the indicated chunk_size as chunks are scheduled.
4. Auto: The decision regarding scheduling is delegated to the compiler and/or runtime system.
5. Runtime: The schedule and chunk size are taken from the run-sched-var ICV.

4.5.3 Sections construct

The sections construct contains a set of structured blocks that are to be distributed and executed by the encountering team of threads. The single construct specifies that the

associated structured block is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.

4.5.4 Single construct

The single construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread) in the context of its implicit task.

4.5.5 Parallel loop construct

The parallel loop construct is a shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

4.5.6 Parallel sections construct

The parallel sections construct is a shortcut for specifying a parallel construct containing one sections construct and no other statements.

4.5.7 Task construct

The task construct defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply.

4.5.8 Critical construct

The critical construct restricts execution of the associated structured block to a single thread at a time.

4.5.9 Master construct

The master construct specifies a structured block that is executed by the master thread of the team. There is no implied barrier either on entry to or exit from the master constructs.

4.5.10 Barrier construct

The barrier construct specifies an explicit barrier at the point at which the construct appears. Threads wait at this barrier till all threads have reached this point.

4.5.11 Taskwait construct

The taskwait construct specifies a wait on the completion of child tasks of the current task.

4.5.12 Atomic construct

The atomic construct ensures that a specific storage location is updated atomically rather than exposing it to the possibility of multiple simultaneous writing threads.

4.6 OpenMP Clauses [50], [52]

The set of clauses that is valid on a particular directive is described with the directive. Most clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible.

Data Sharing Attribute Clauses:

Data-sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

4.6.1 Default

Controls the default data-sharing attributes of variables that are referenced in a parallel or task construct.

4.6.2 Shared

Declares one or more list items to be shared by tasks generated by a parallel or task construct.

4.6.3 Private

Declares one or more of the list items to be private to a task.

4.6.4 Firstprivate

Declares one or more list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.

4.6.5 Lastprivate

Declares one or more list items to be private to an implicit task, and causes the corresponding original item to be updated after the end of the region.

4.7 OpenMP Environment Variables [50], [52]

Environment variable is a method to alter the execution features of OpenMP applications. It is used to control loop iterations scheduling, default number of threads, etc. in an application which in run using OpenMP software.

4.7.1 OMP_SCHEDULE

Sets the runtime environment variable that sets the schedule type and chunk size for the threads that run in the parallel region. Valid OpenMP schedule types are static, dynamic, guided or auto. Chunk is a positive integer that specifies chunk size.

4.7.2 OMP_NUM_THREADS

Sets the runtime environment variable to initialize the number of threads for use in a parallel region.

4.7.3 OMP_DYNAMIC

Sets the runtime environment variable for dynamic adjustment of threads in a parallel region. Valid values for dynamic are true or false.

4.7.4 OMP_NESTED

Sets the runtime environment variable to enable or disable nested parallelism. Valid values are true or false.

4.7.5 OMP_THREAD_LIMIT

Sets the runtime environment variable to control the maximum number of threads participating in the OpenMP program.

4.8 Race Conditions [50]

A race condition exists when two unsynchronized threads access the same shared variable with at least one thread modifying the variable. The outcome may be unpredictable and depends on the timing of the threads in the team. Race conditions are an insidious problem because they can remain undetected for many thousands of executions, and it is not always obvious that the program has generated incorrect results. Because communications and synchronizations are often implicit in shared memory programming, race conditions can arise unexpectedly. It is the programmer's responsibility to ensure that the code is free from situations that could give rise to race conditions that corrupt the computational results. Following simple example demonstrates the race condition -

```
int i=0;
#pragma omp parallel
{
:
i++;
:
}
```

Consider a possible time-line for a two-thread example as shown in table 4.2.

Table 4.2 Timeline for two threads in OpenMP

Clock	Thread 0	Thread 1
1	load i (i = 0)	
2	incr i (i = 1)	
3	swapped out	load i (i = 0)
4		incr i (i = 1)
5		store i (i = 1)
6	store i (i = 1)	swapped out

In this case, the result in i is 1 and not 2, as one would expect. Because the increment (++) operation is not atomic, it can be interrupted before completion and can cause incorrect results. A simple increment on a shared variable like this is a prime candidate for the use of the OpenMP atomic directive, as shown below, which eliminates the possibility of a race condition. Finally, the following two-step process goes a long way towards eliminating race conditions from the code:

1. Identify all shared variables within an OpenMP region.
2. Guard all modifications of those variables with critical regions or atomic directives, even when they look innocuous.

Even though it is easy to write shared memory programs, it is not easy to write correct shared memory programs.

4.9 Summary

This chapter gives a detailed description of key concepts in the OpenMP program such as programming model, directives, constructs, environmental variables etc. In the end, it explains race conditions that can occur while processing in parallel.

The next chapter describes how the OpenMP program is incorporated in this thesis to achieve task based parallelism. Finally, results, especially time complexity reduction are clearly illustrated with various graphs.

CHAPTER 5

RESULTS OF COMPLEXITY REDUCTION USING OPENMP

5.1 Strategy adopted

This thesis aims at drawing a conclusion based on the reduction in the encoding time of a video sequence by using task based and data based parallelism. The task based parallelism in JM 18.0 [13] reference software is achieved by dividing the GOP's into two equal parts and encoding each part in two threads independently on the underlying hardware. These threads are created by making use of OpenMP software. OpenMP software makes it possible to run multiple threads at the same time depending on the number of available cores. In this thesis a maximum of two threads are run in parallel on two different cores i.e. one thread on one dedicated core to achieve task level parallelism. Hence the two threads become independent of each other with no data dependency. The above design can be explained in detail with the help of figure 5.1 as follows:

Step 1] Divide the GOP's into two equal sub GOP's. For example, if the total number of frames to encode is 30, then GOP1 contains frame numbers from 1 to 15 to be encoded in thread 1 and GOP2 contains frame numbers from 16 to 30 to be encoded in thread 2.

Step 2] Perform intra coding on two different frames i.e. frame 1 and frame 16. Frame 1 can be used as a reference frame for frame 2 and frame 16 can be used as a reference frame for frame 17.

Step 3] Perform inter coding on frame 2 and frame 17 in two separate threads i.e. thread 1 and thread 2 respectively. Frame 1 and frame 16 acts as reference frames for frame 2 and frame 17 respectively. Similarly frame 2 and frame 17 acts as reference frames for frame 3 and frame 18 respectively. The two threads thus become independent with no data dependency and can run in parallel.

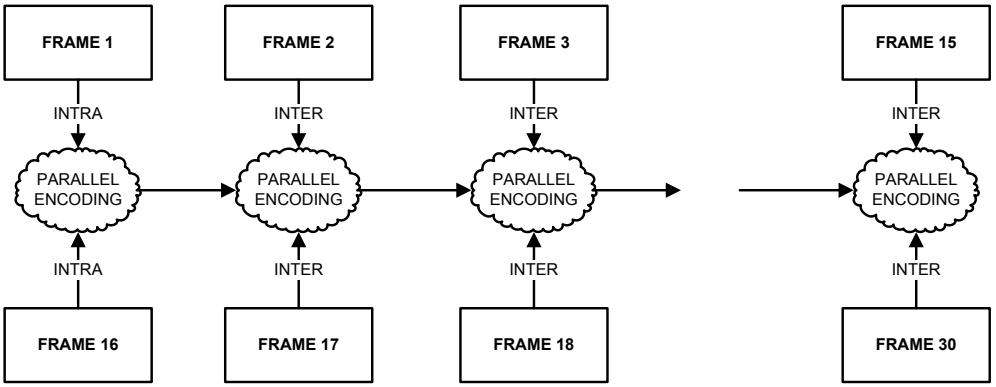

Figure 5.1 Parallel encoding of 30 frames.

Diagrammatic representation of step 1 to step 3 are shown in figure 5.1.

Data based parallelism is achieved by first finding the hot spots in the JM 18.0 [13] reference software. These hot spots are then made to run in parallel by dividing the total work of a hot spot into two different threads equally. These threads are created by making use of OpenMP software. The division of the work is done by changing the code of the hot spot under consideration. While dividing the work care needs to be taken to ensure that there is no data dependency between the two threads. Also it is preferable to maintain the load balancing between the two threads for optimum results. Thus software complexity increases and also there is an extra overhead of thread creation for each hot spot in the reference software.

5.2 Prediction Structure [3], [4], [5], [7] used

Prediction structures offer different options for choosing reference pictures for inter prediction. H.264 video codec offers four basic prediction structures as explained in chapter 2. To obtain results, low delay minimal storage prediction structure was used since it is compatible with the Baseline Profile of H.264 [3]. Figure 5.1 shows the low delay minimal storage prediction structure. It uses only I and P slices. It does not allow B slices. The first frame is coded as an I

slice and subsequent frames are coded as P slices for a GOP predicted from the previous frame.

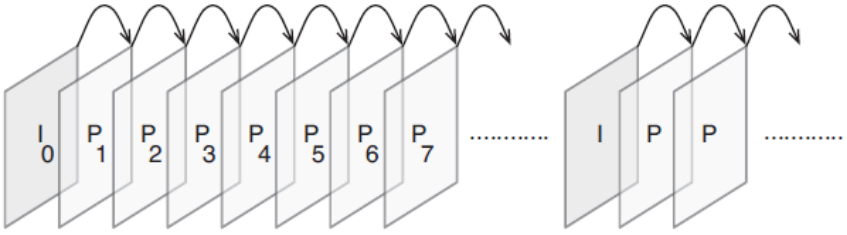

Figure 5.2 Low delay, minimal storage prediction structure in H.264 video codec [3]

5.3 Performance metric [3], [50]

The results are compared in terms of change of PSNR (ΔPSNR), bit-rate (Δ bit rate), SSIM (ΔSSIM), encoding time (Δ Time), physical memory usage and CPU power usage. A conclusion is drawn finally based on results obtained from the reduction of the encoding time using task based and data based parallelism.

5.3.1 %T reduction

Computational efficiency is measured by the amount of time reduction, which is computed as follows:

5.3.2 Delta bit rate

5.3.3 PSNR (Peak Signal to Noise Ratio) [59] is computed as follows:

5.3.4 SSIM [57], [58] (%) can be measured on similar lines, as follows:

5.3.5 Physical memory usage

Task manager can be used to find the Physical memory usage. The task manager can be viewed in a pop up window by pressing Ctrl + Alt + Delete key's together from the keyboard. Click on the performance tab to view the Physical memory usage for the JM 18.0 [13] software. A screen shot for the task manager is as shown in figure 5.3.

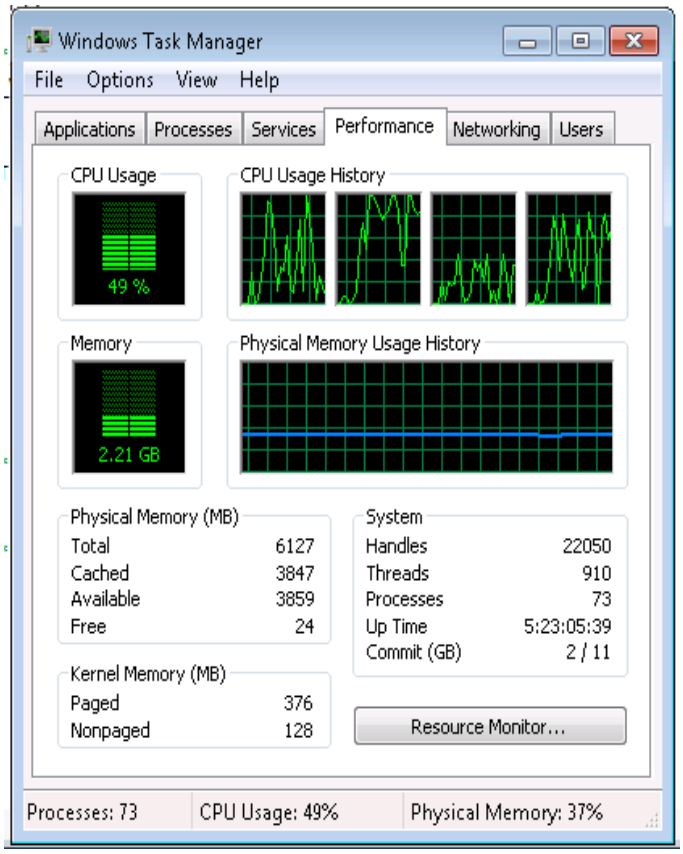

Figure 5.3 Task manager snapshot

5.3.6 CPU power usage

CPU power usage to run the JM software can be obtained by using third party software called joule-meter [62], [63]. This software gives the CPU power usage in watts for any application in the running state. A snap shot for the joule meter is as shown in figure 5.4.

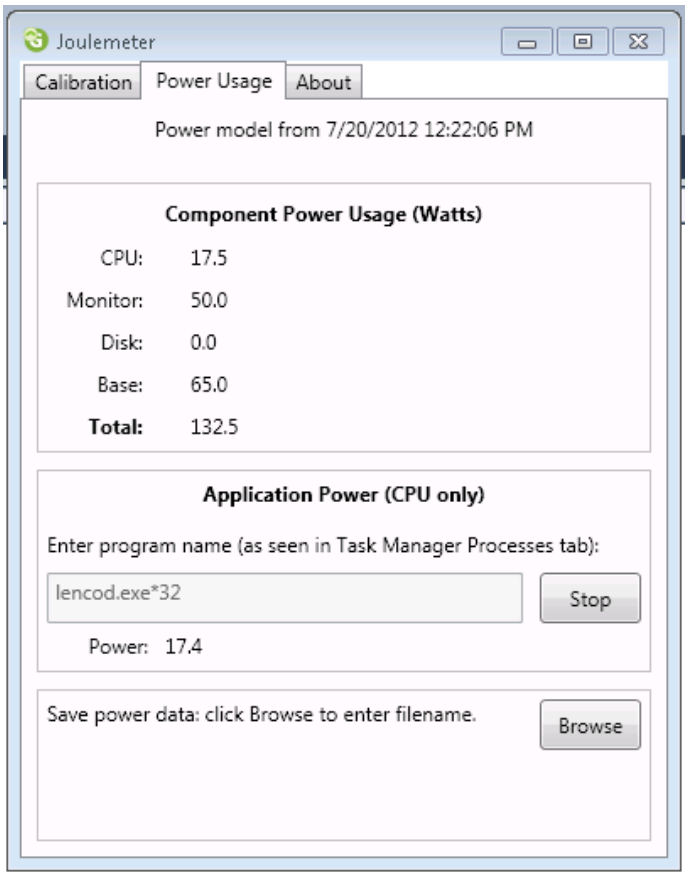

Figure 5.4 Joule-meter snapshot [62], [63]

5.4 Preview of test sequences [50], [54]

CIF (Common Intermediate Format) is a format used to standardize the horizontal and vertical resolutions in pixels of Y, C_b, C_r sequences in video signals, commonly used in video teleconferencing systems. QCIF means "Quarter CIF". To have one fourth of the area as "quarter" implies the height and width of the frame are halved. The differences in Y, C_b, C_r of CIF and QCIF are shown in figure 5.5.

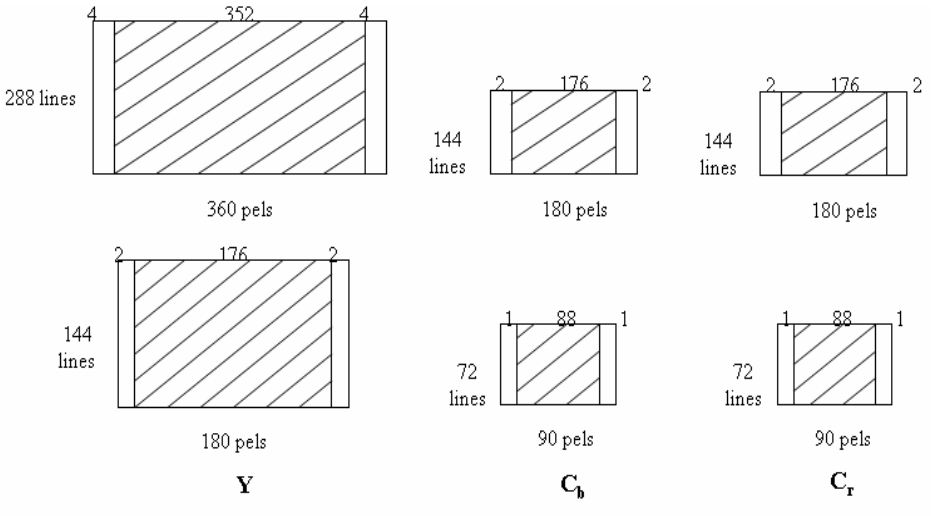

Figure 5.5 CIF and QCIF formats [55]

CIF (352 × 288) and QCIF (176 × 144) [55] sequences have been used to test the results of proposed technique. Following 12 CIF and QCIF video sequences have been used (figure 5.6) to draw a conclusion for this thesis.

Figure 5.6 Preview of CIF and QCIF video sequences for testing [60]

CIF and QCIF sequences have been used with frame rate selected as 25 Hz. Compared to original JM 18.0 reference software [13], results obtained by optimizing the software are shown based on PSNR, bit rate, SSIM (Structural Similarity Index Metric) [56], total encoding time [50], CPU power and memory usage.

SSIM, a recently proposed approach to image fidelity measurement has proven to be highly effective for measuring the fidelity of coded images. The human visual system is highly adapted to extract structural information from visual scenes; this is the basis for SSIM. For image fidelity measurement, the retention of signal structure should be an important ingredient [50].

5.5 Encoding specifications

5.5.1 Configuration Parameters

The following configuration parameters (Figure 5.7) are given as input before starting the encoding procedure. Figure 5.7 states the configuration parameters for container_cif.yuv video sequence.

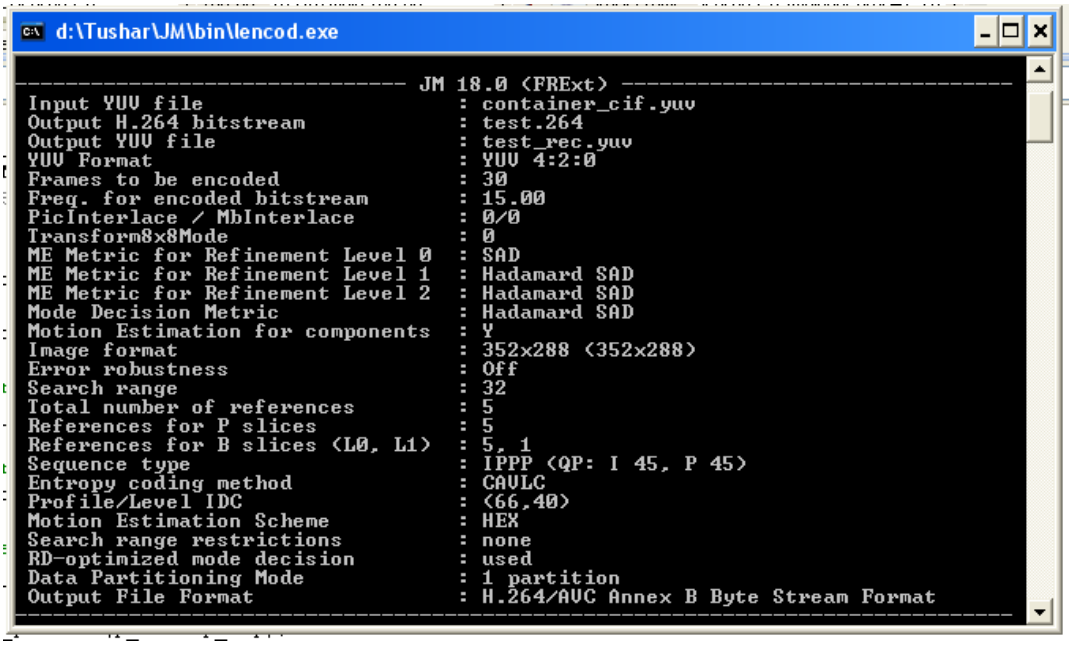

Figure 5.7 Configuration parameters for container_cif.yuv video sequence.

5.6 Results with CIF and QCIF sequences

5.6.1 Results with CIF sequences

Table 5.1 Simulation results for CIF video sequences at QP 10 and 25.

Test Sequence (QCIF)	QP = 10				QP = 25			
	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)
Foreman	47.12	0.034	-0.012	0.002	46.06	-0.012	0.001	0.001
Coastguard	46.75	-0.123	0.011	0.000	45.94	0.031	0.003	-0.002
Hall	44.92	0.002	0.013	0.001	44.21	-0.012	-0.002	0.103
Mobile	45.24	0.021	0.013	0.002	44.69	0.016	0.061	0.008
News	46.98	0.019	0.005	-0.007	46.23	0.010	0.021	0.004
Flower	45.12	0.034	0.007	-0.001	44.72	0.032	0.041	-0.016
Highway	45.92	0.034	0.018	0.001	45.11	0.046	0.022	-0.008

Table 5.1 continued

Mother-Daughter	45.56	0.031	0.004	-0.007	46.18	-0.050	0.052	0.006
Container	45.32	0.029	0.026	-0.005	45.20	-0.034	0.041	0.006
Bus	45.09	0.002	0.017	0.050	45.07	0.051	0.045	0.008
Paris	44.35	0.011	0.014	-0.011	43.57	0.043	0.033	0.003

Table 5.2 Simulation results for CIF video sequences at QP 35 and 45.

Test Sequence (QCIF)	**QP = 35**				**QP = 45**			
	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)
Foreman	47.45	0.018	-0.001	0.001	48.12	0.001	-0.002	0.101
Coastguard	45.89	-0.001	0.021	0.000	45.98	0.024	0.006	0.005
Hall	45.68	0.012	0.008	-0.003	45.73	-0.012	0.011	-0.012
Mobile	44.53	0.032	0.043	0.012	43.91	0.018	0.012	0.013
News	46.56	0.027	0.032	0.007	46.98	0.035	0.013	-0.012
Flower	45.98	0.062	0.012	0.005	46.12	0.026	0.032	0.012
Highway	46.23	0.062	0.062	0.003	45.89	0.020	-0.094	0.009
Mother-daughter	44.78	-0.028	0.001	0.004	45.18	0.038	0.009	0.011
Container	44.98	0.051	0.045	0.008	43.52	0.016	0.061	0.000
Bus	44.67	0.026	0.001	0.013	45.12	-0.122	0.009	0.163
Paris	45.12	0.014	0.009	0.005	45.51	0.025	0.043	0.007

5.6.2 Results with QCIF sequences

Table 5.3 Simulation results for QCIF video sequences at QP 10 and 25.

Test Sequence (QCIF)	**QP = 10**				**QP = 25**			
	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)
Foreman	47.34	0.062	-0.013	0.00	46.73	-0.002	0.089	-0.03
Coastguard	48.12	0.000	-0.014	0.005	47.56	0.002	-0.032	0.002
Hall	49.86	0.003	0.006	0.001	50.58	-0.003	0.001	0.101
Mobile	47.12	-0.053	-0.021	0.012	46.67	-0.001	-0.001	-0.012
News	48.56	0.005	-0.002	-0.006	47.53	-0.022	-0.129	-0.004
Suzie	50.82	0.012	-0.001	0.015	49.11	-0.010	-0.010	0.015
Highway	45.95	-0.089	-0.008	-0.012	46.18	-0.004	-0.041	0.001
Mother-daughter	46.91	0.001	-0.003	0.001	46.50	-0.001	-0.012	0.001
Salesman	48.10	0.003	-0.001	0.032	48.31	-0.062	-0.103	-0.002
Miss-america	48.12	-0.041	-0.012	0.014	49.18	-0.012	-0.055	0.004
Container	48.26	0.009	-0.031	-0.015	47.80	-0.007	-0.108	-0.001

Table 5.3 continued

Bridge-close	47.45	0.001	-0.001	0.001	46.38	0.00	-0.138	0.005

Table 5.4 Simulation results for QCIF video sequences at QP at 35 and 45.

Test Sequence (QCIF)	**QP = 35**				**QP = 45**			
	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)	%T Reduction	ΔPSNR (%)	Δ Bit-rate (%)	Δ SSIM (%)
Foreman	48.90	-0.197	-0.122	0.002	48.92	-0.011	-0.232	-0.012
Coastguard	46.98	-0.007	-0.196	-0.012	47.12	-0.016	-0.259	-0.018
Hall	48.59	0.001	-0.185	0.021	49.23	0.021	-0.324	-0.007
Mobile	46.34	0.008	-0.069	0.001	47.12	0.014	-0.214	0.001
News	46.89	-0.063	-0.013	0.091	46.12	-0.013	-0.198	0.012
Suzie	48.12	0.089	-0.021	-0.002	47.34	0.045	-0.109	-0.021
Highway	48.96	0.034	-0.154	0.036	47.98	0.006	-0.236	-0.009
Mother-daughter	46.78	-0.108	-0.121	0.001	45.92	-0.056	-0.126	0.089
Salesman	50.52	0.001	-0.109	-0.006	49.45	0.008	-0.214	0.056
Miss-america	47.51	0.005	-0.001	0.002	48.34	0.001	-0.167	-0.012
Container	47.12	-0.002	-0.115	0.014	48.56	-0.001	-0.091	0.002
Bridge-close	46.89	-0.001	-0.201	0.000	47.39	0.000	-0.506	-0.005

5.7 Graphs

5.7.1 Average encoding time for all CIF sequences

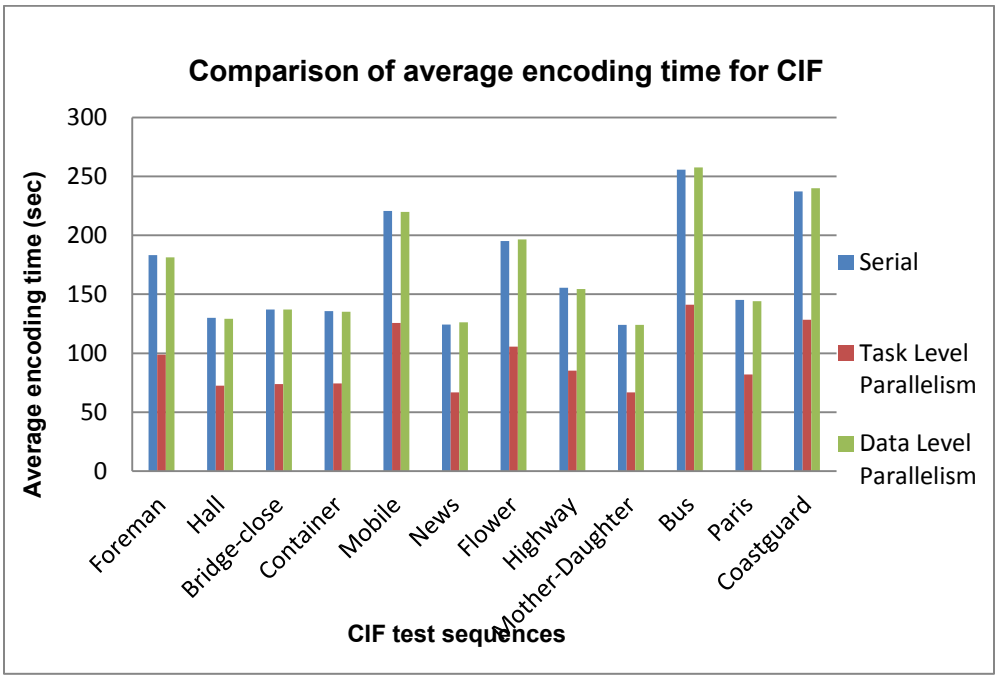

Figure 5.8 Comparison of average encoding time for all CIF sequences

5.7.2 Average encoding time for all QCIF sequences

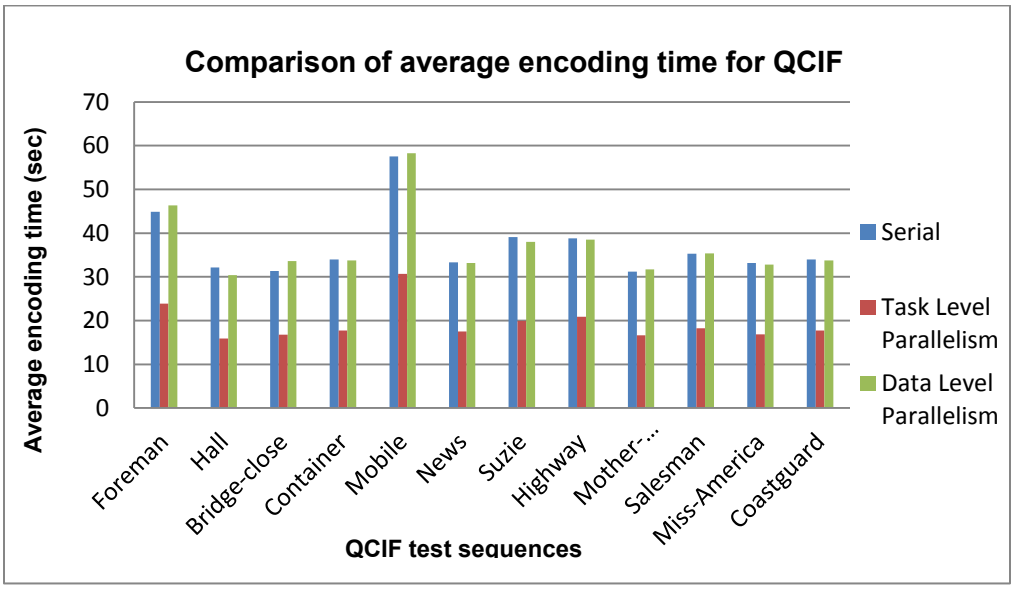

Figure 5.9 Comparison of average encoding time for all QCIF sequences

5.7.3 PSNR graphs for QCIF sequences

5.7.3.1 Foreman_qcif.yuv

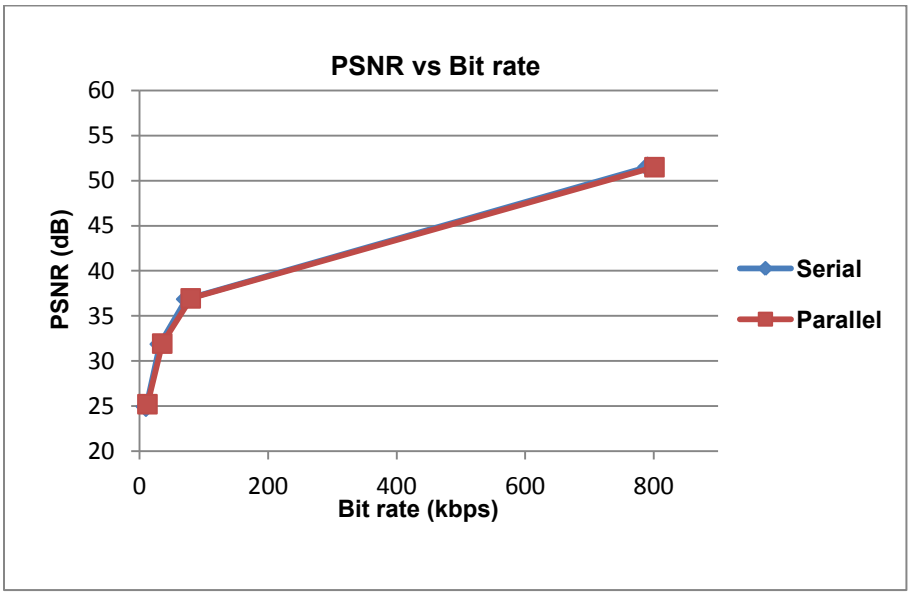

Figure 5.10 PSNR graph for foreman_qcif.yuv

5.7.3.2 Coastguard_qcif.yuv

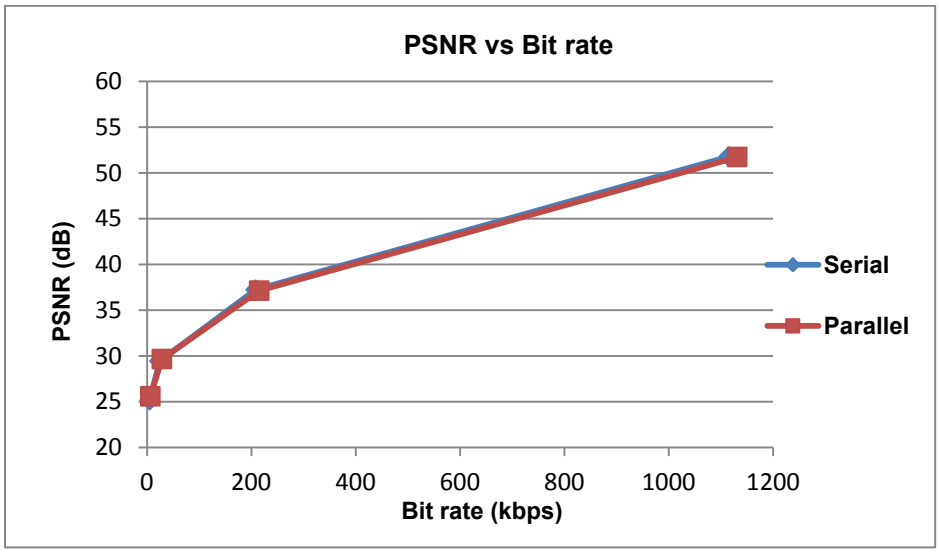

Figure 5.11 PSNR graph for coastguard_qcif.yuv

5.7.3.3 Hall_qcif.yuv

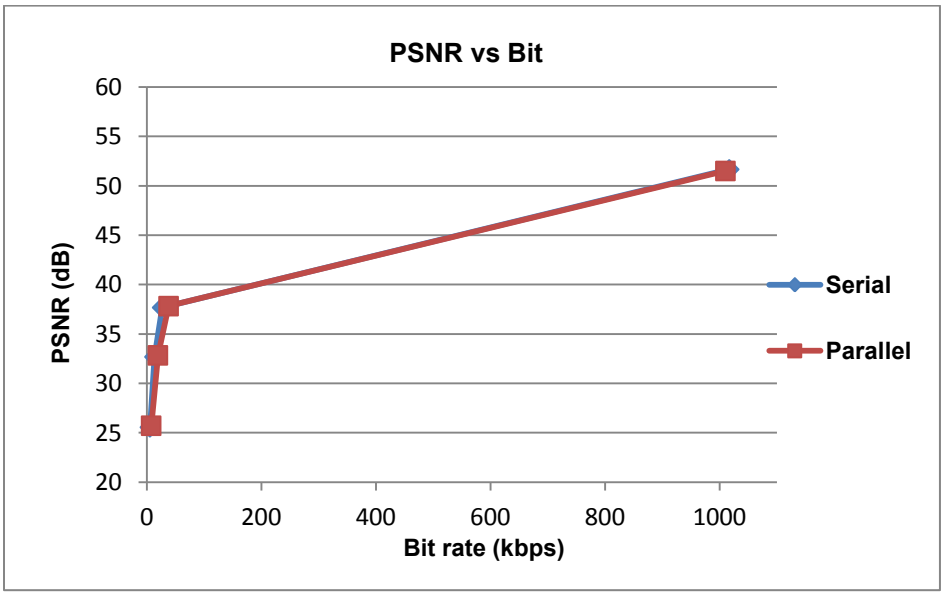

Figure 5.12 PSNR graph for hall_qcif.yuv

5.7.3.4 Bridge-close_qcif.yuv

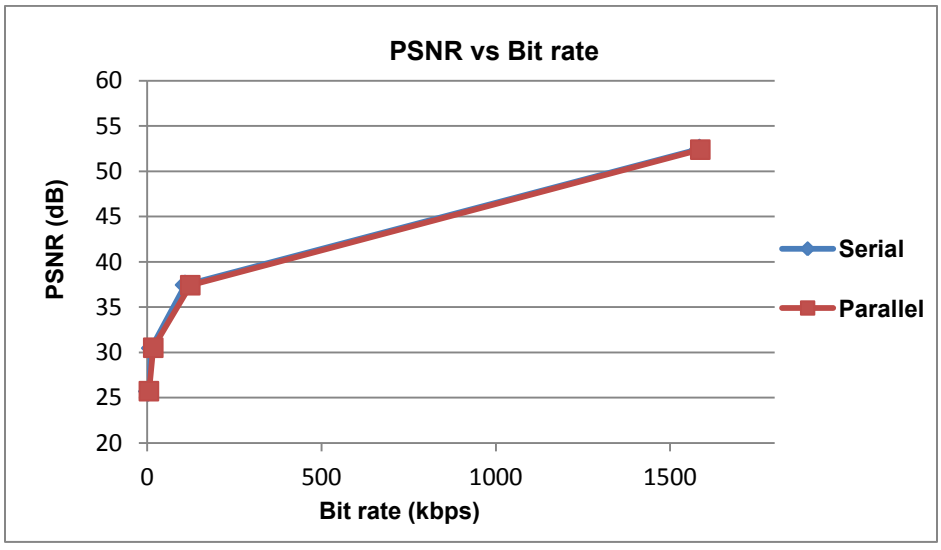

Figure 5.13 PSNR graph for bridge-close_qcif.yuv

5.7.3.5 Mobile_qcif.yuv

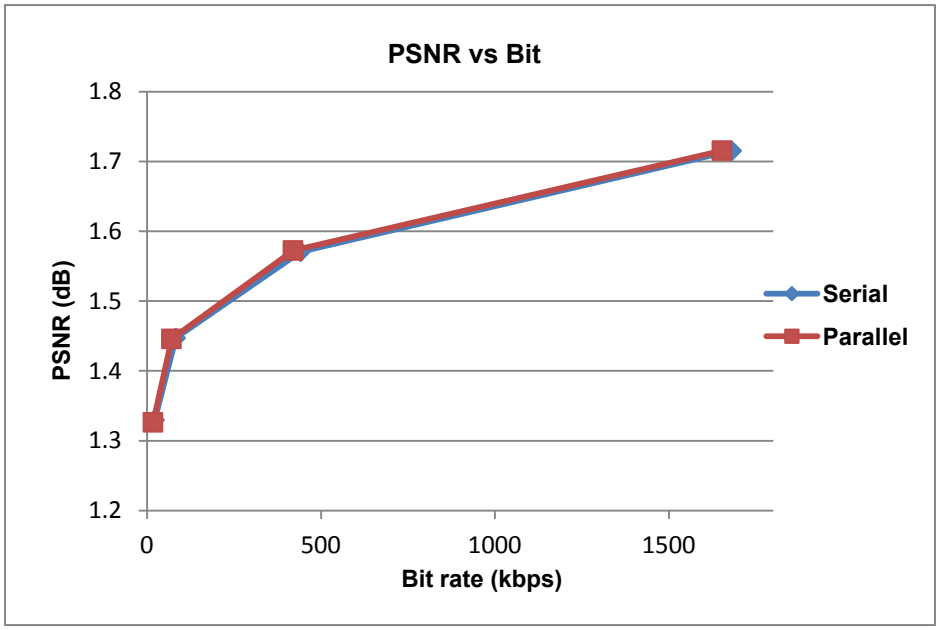

Figure 5.14 PSNR graph for mobile_qcif.yuv

5.7.3.6 News_qcif.yuv

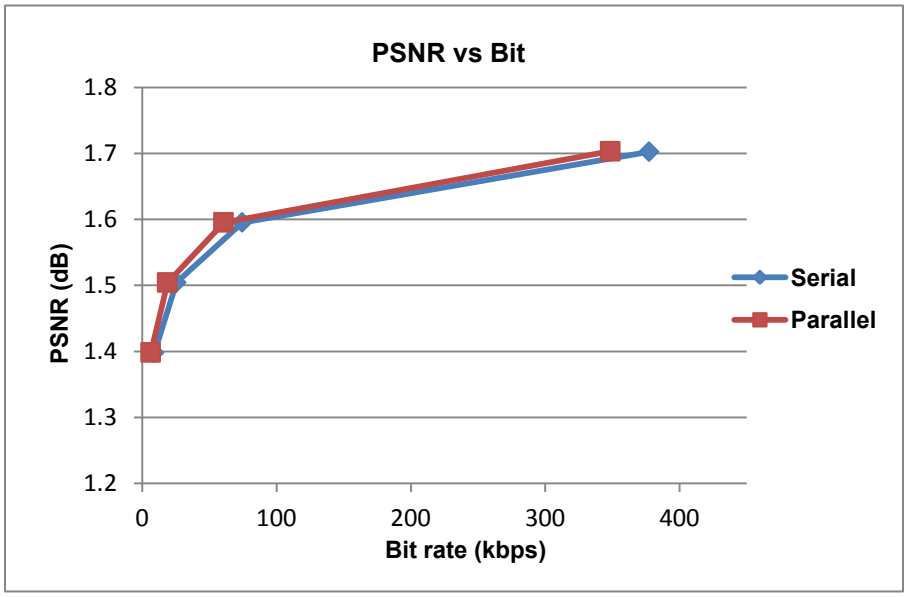

Figure 5.15 PSNR graph for news_qcif.yuv

5.7.3.7 Suzie_qcif.yuv

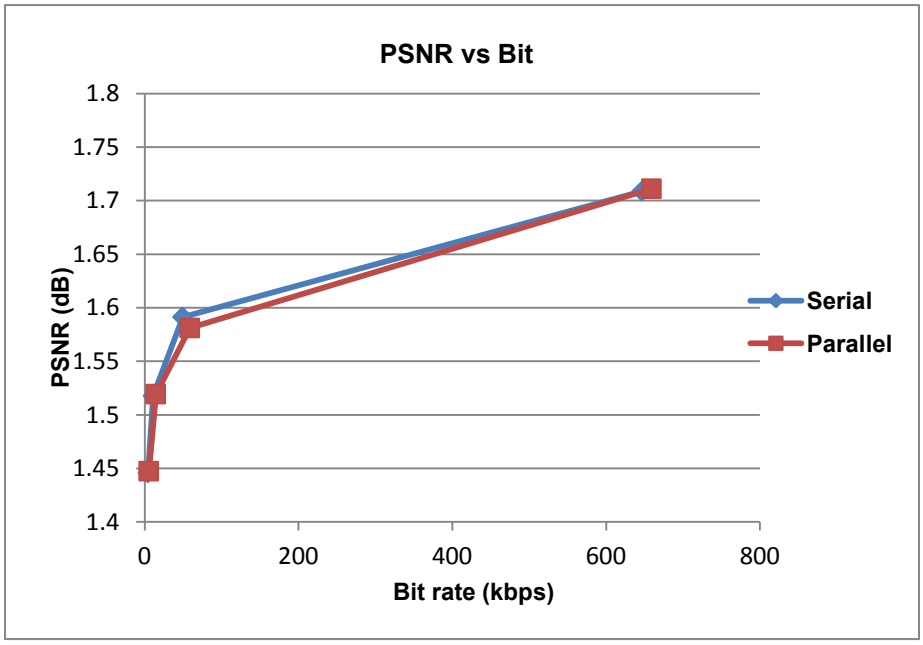

Figure 5.16 PSNR graph for suzie_qcif.yuv

5.7.3.8 Highway_qcif.yuv

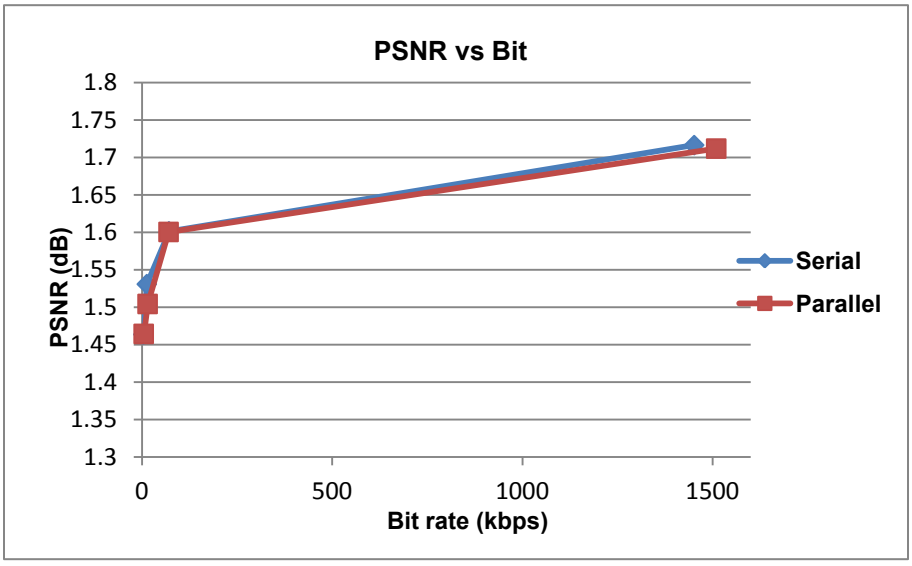

Figure 5.17 PSNR graph for highway_qcif.yuv

5.7.3.9 Mother-daughter_qcif.yuv

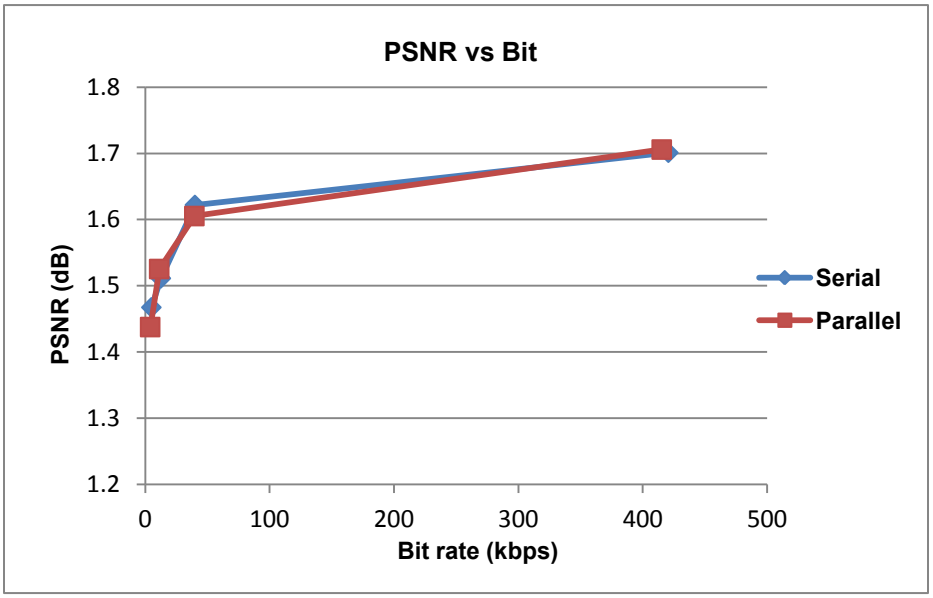

Figure 5.18 PSNR graph for mother-daughter_qcif.yuv

5.7.3.10 Salesman_qcif.yuv

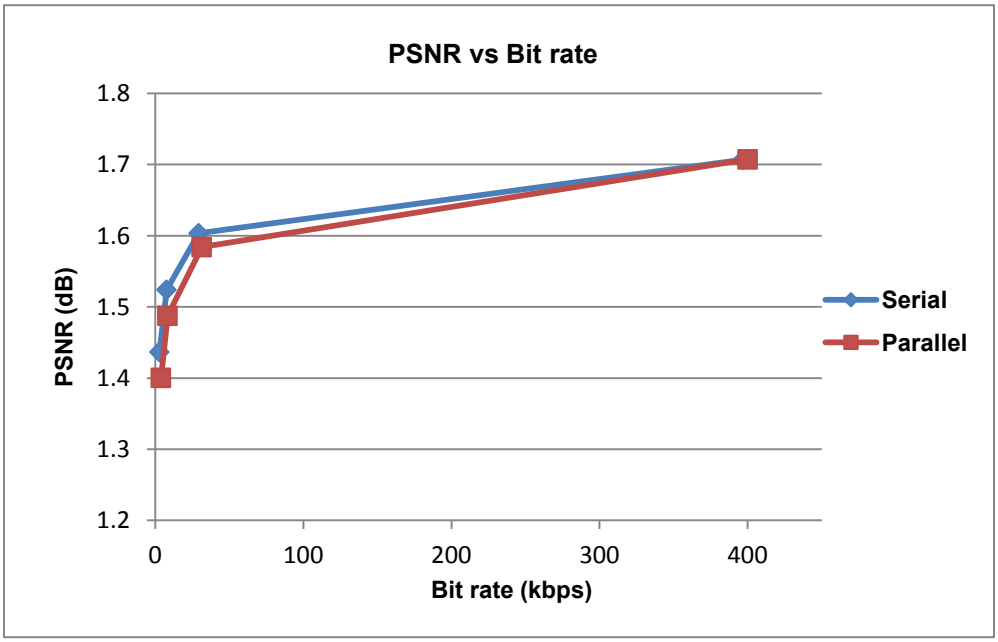

Figure 5.19 PSNR graph for salesman_qcif.yuv

5.7.3.11 Miss-america_qcif.yuv

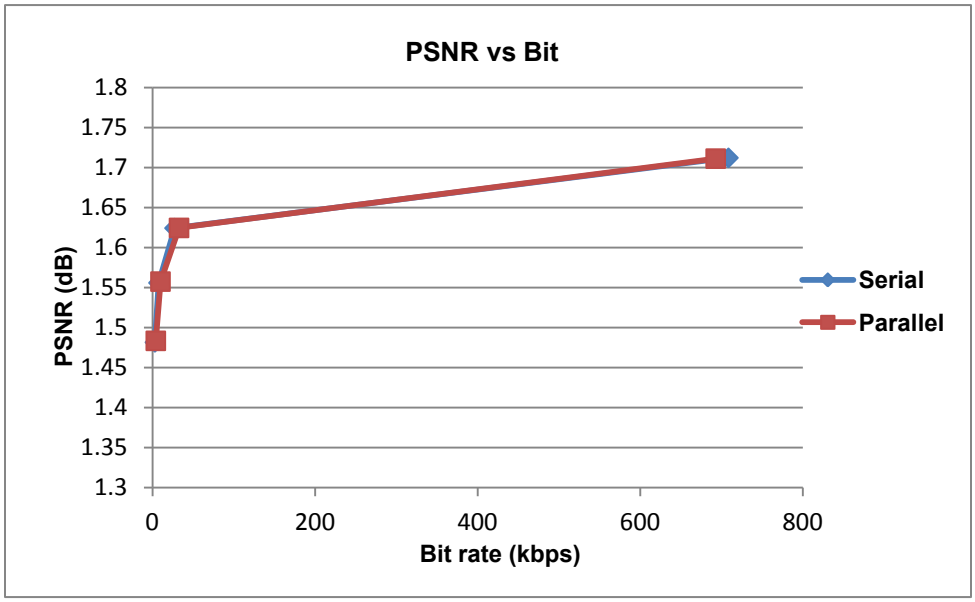

Figure 5.20 PSNR graph for miss-america_qcif.yuv

5.7.3.12 Container_qcif.yuv

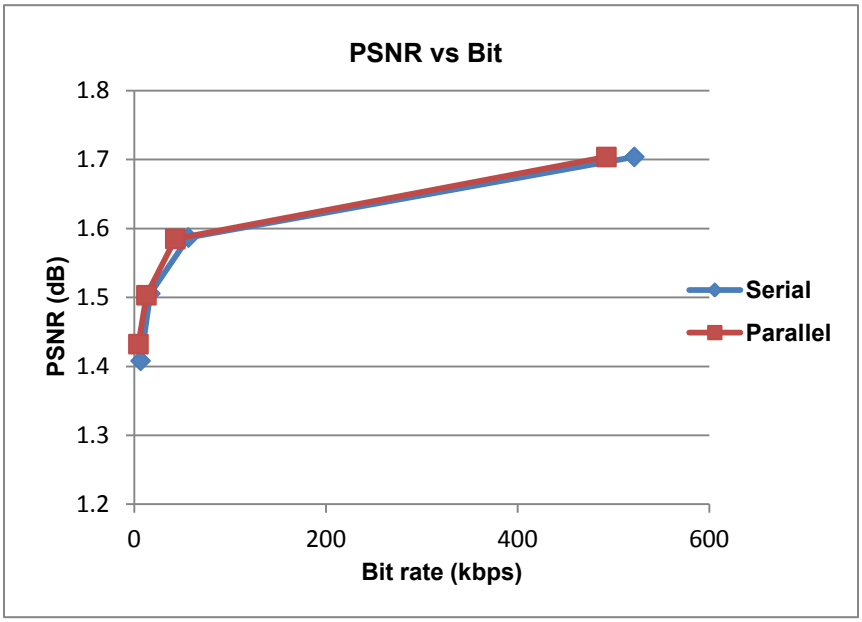

Figure 5.21 PSNR graph for container_qcif.yuv

5.7.4 SSIM graphs for QCIF sequences

5.7.4.1 Foreman_qcif.yuv

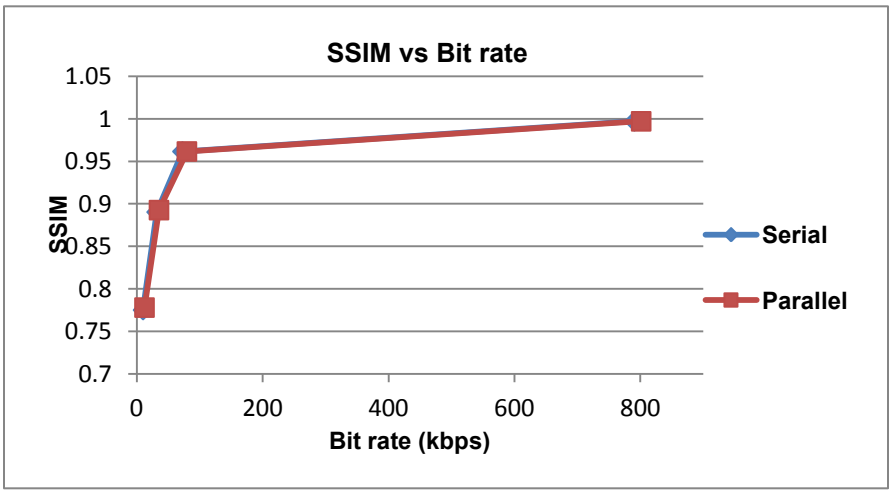

Figure 5.22 SSIM graphs for foreman_qcif.yuv

5.7.4.2 Bridge-close_qcif.yuv

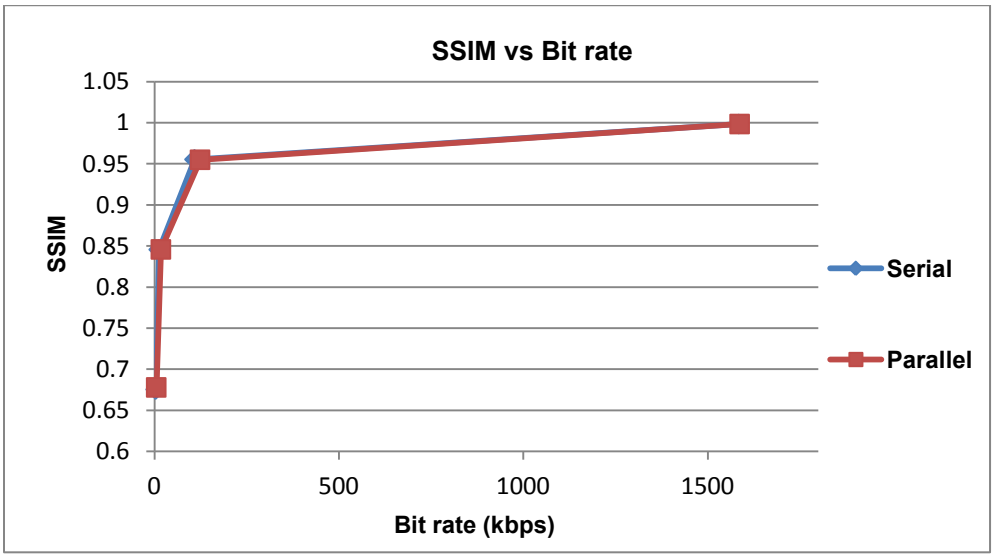

Figure 5.23 SSIM graphs for bridge-close_qcif.yuv

5.7.4.3 Hall_qcif.yuv

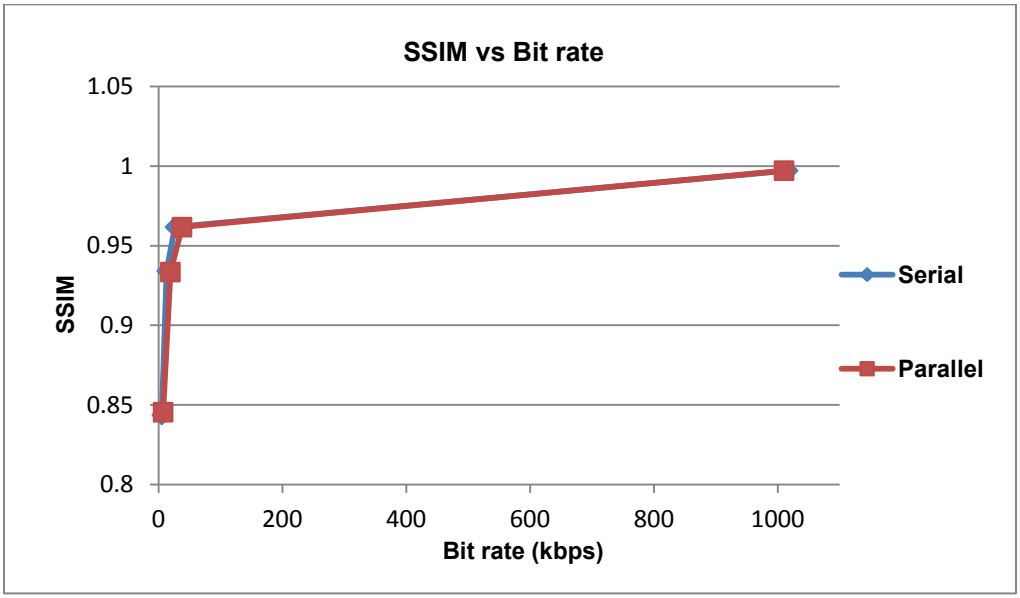

Figure 5.24 SSIM graphs for hall_qcif.yuv

5.7.4.4 Coastguard_qcif.yuv

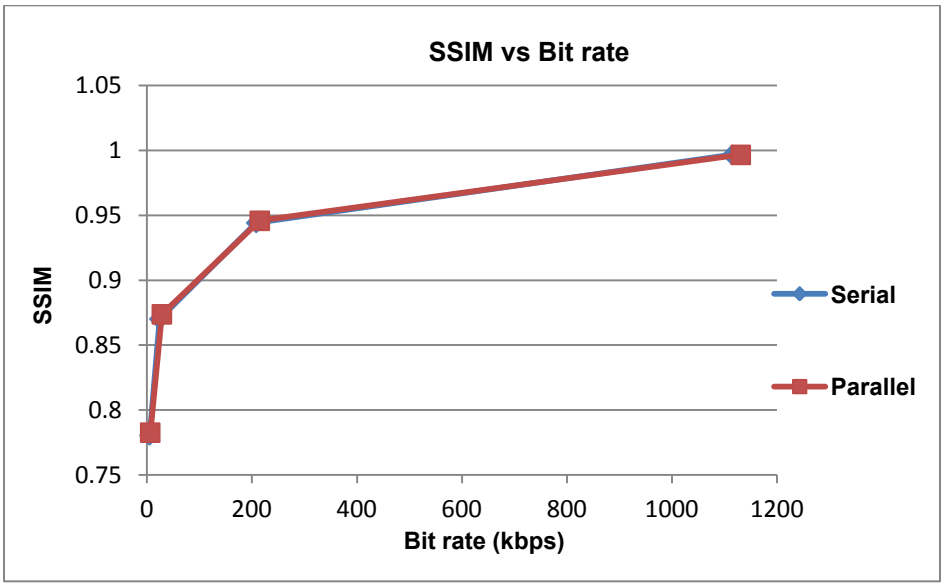

Figure 5.25 SSIM graphs for coastguard_qcif.yuv

5.7.4.5 Mobile_qcif.yuv

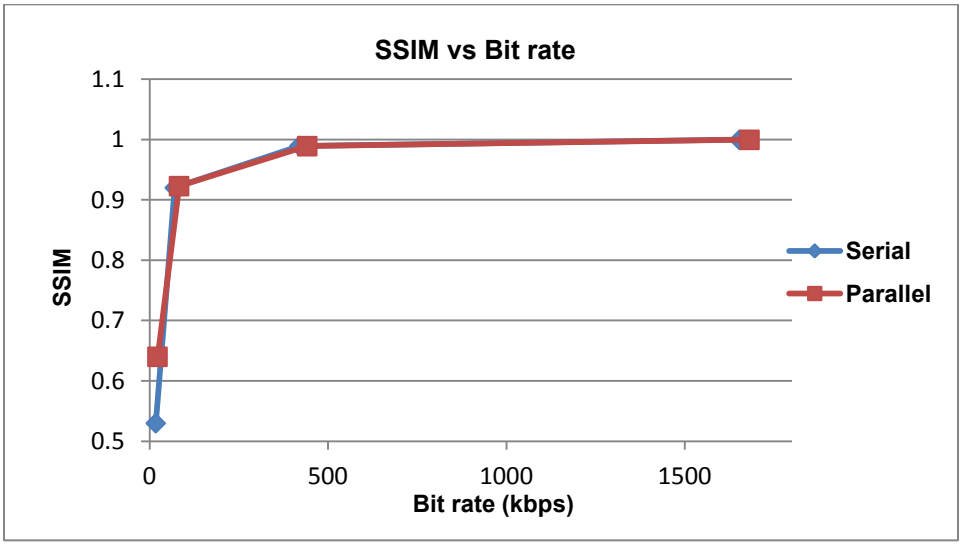

Figure 5.26 SSIM graphs for mobile_qcif.yuv

5.7.4.6 News_qcif.yuv

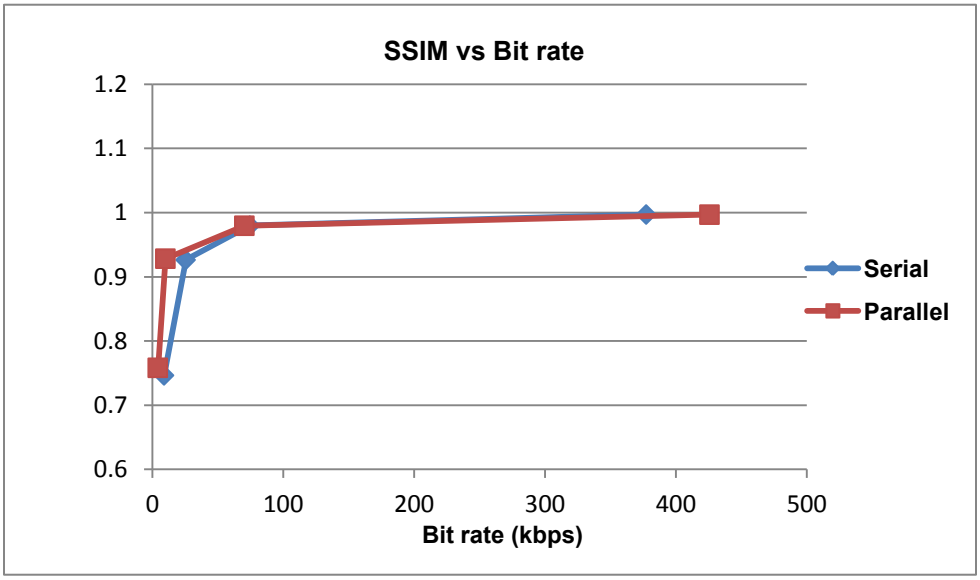

Figure 5.27 SSIM graphs for news_qcif.yuv

5.7.4.7 Suzie_qcif.yuv

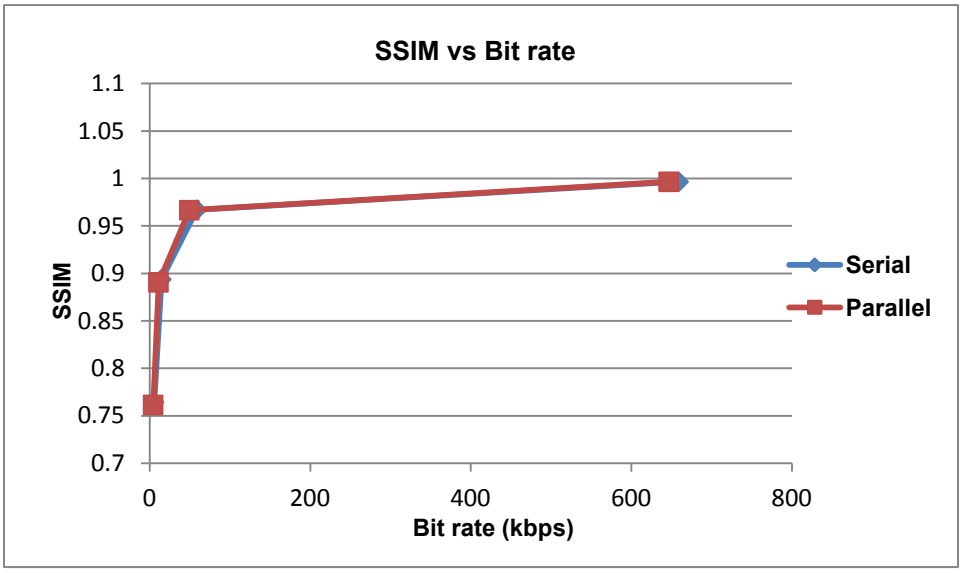

Figure 5.28 SSIM graphs for suzie_qcif.yuv

5.7.4.8 Highway_qcif.yuv

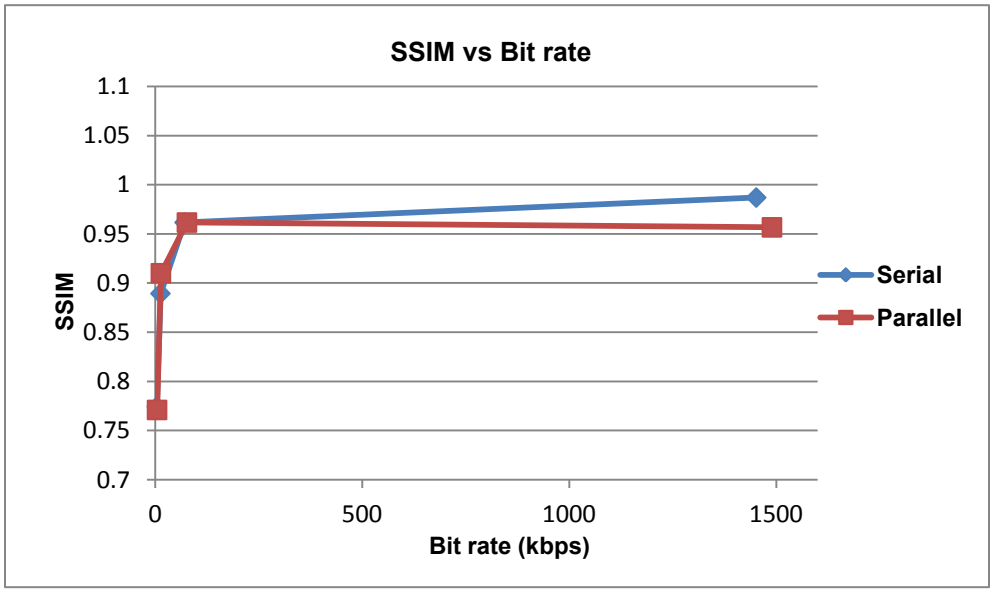

Figure 5.29 SSIM graphs for highway_qcif.yuv

5.7.4.9 Mother-daughter_qcif.yuv

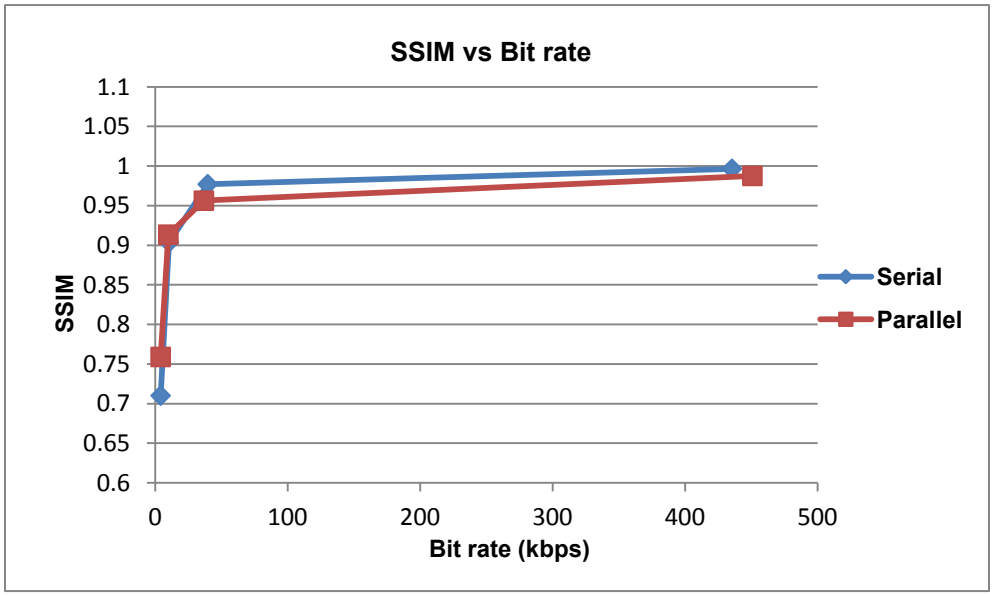

Figure 5.30 SSIM graphs for mother-daughter_qcif.yuv

5.7.4.10 Salesman_qcif.yuv

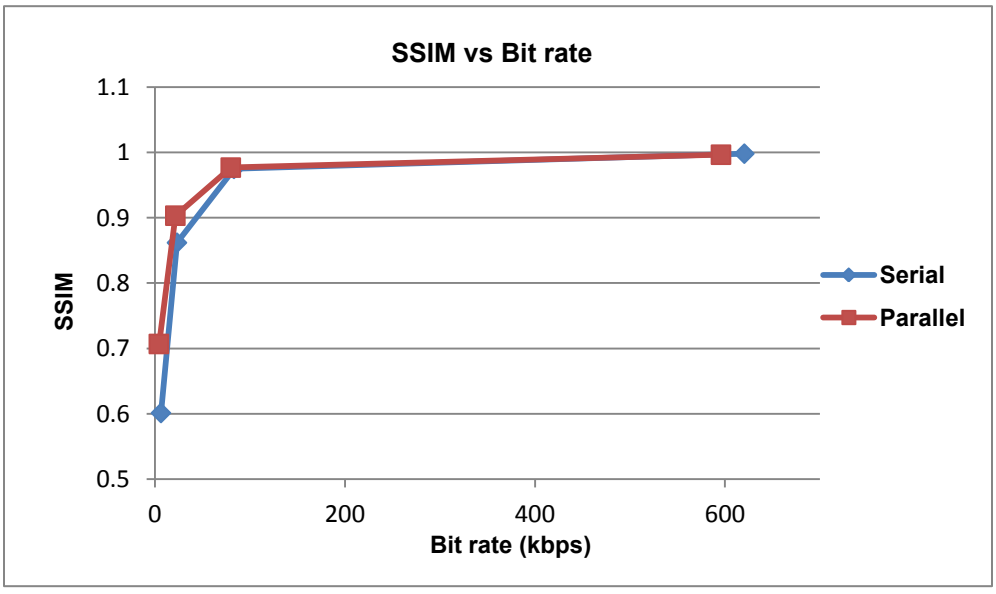

Figure 5.31 SSIM graphs for coastguard_qcif.yuv

5.7.4.11 Miss-america_qcif.yuv

Figure 5.32 SSIM graphs for miss-america_qcif.yuv

5.7.4.12 Container_qcif.yuv

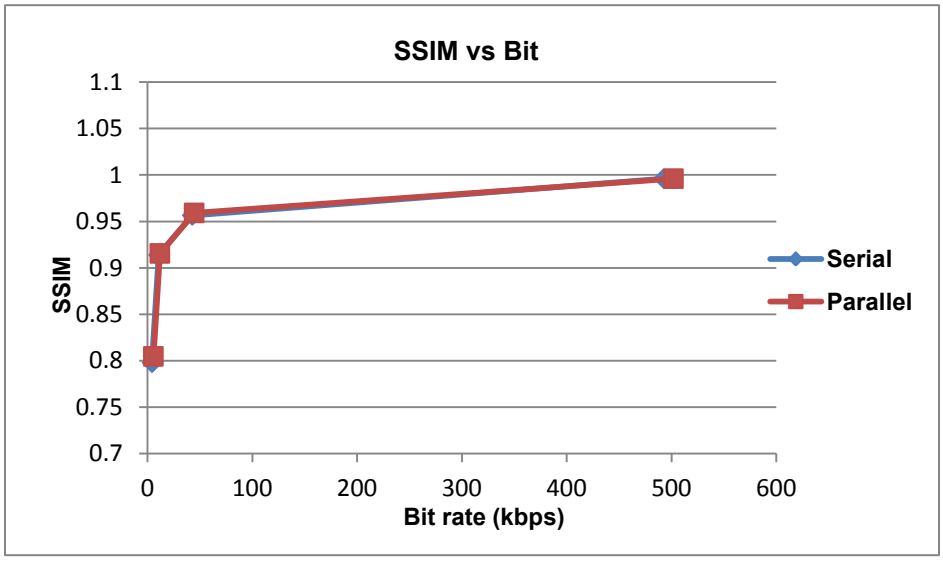

Figure 5.33 SSIM graphs for container_qcif.yuv

5.7.5 PSNR graphs for CIF sequence

5.7.5.1 Foreman_cif.yuv

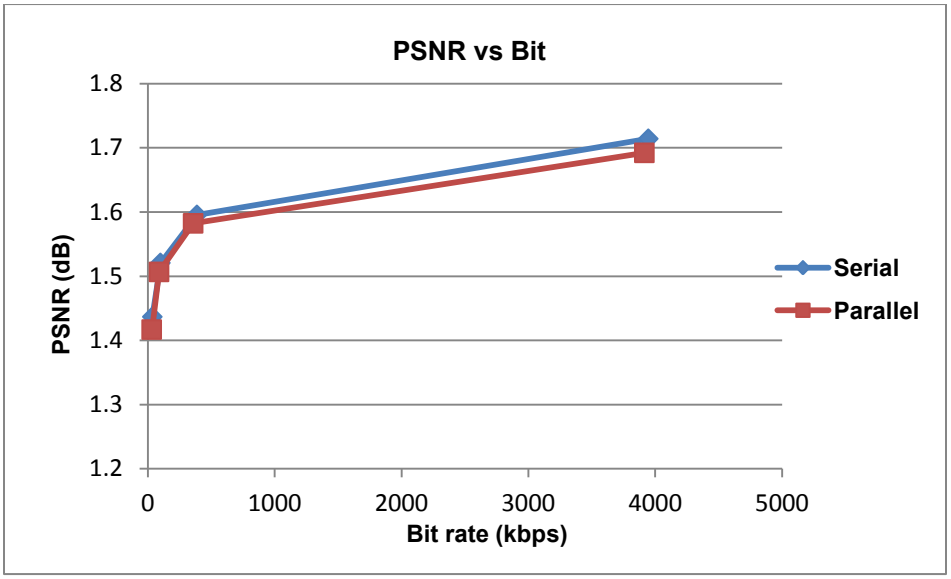

Figure 5.34 PSNR graph for foreman_cif.yuv

5.7.5.2 Coastguard_cif.yuv

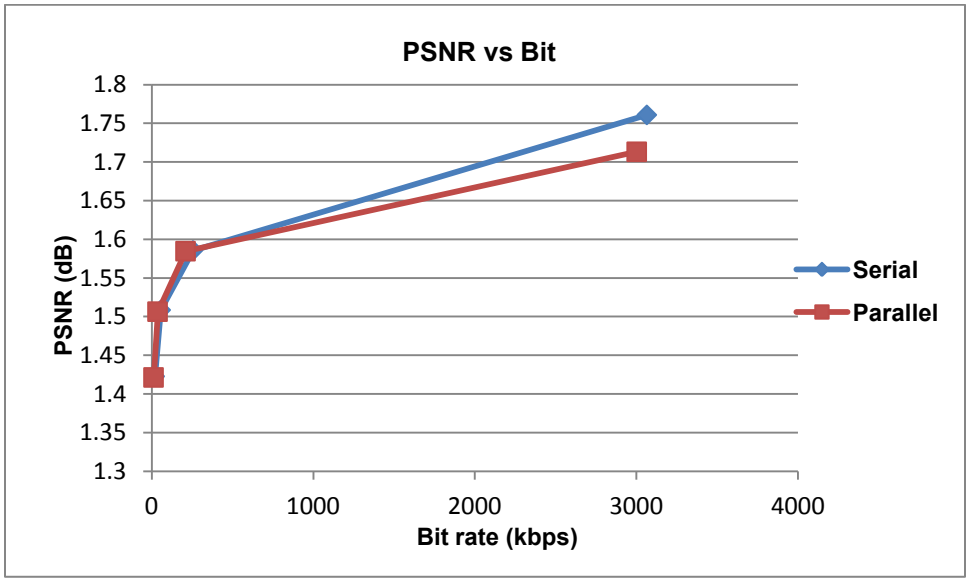

Figure 5.35 PSNR graph for coastguard_cif.yuv

5.7.5.3 Hall_cif.yuv

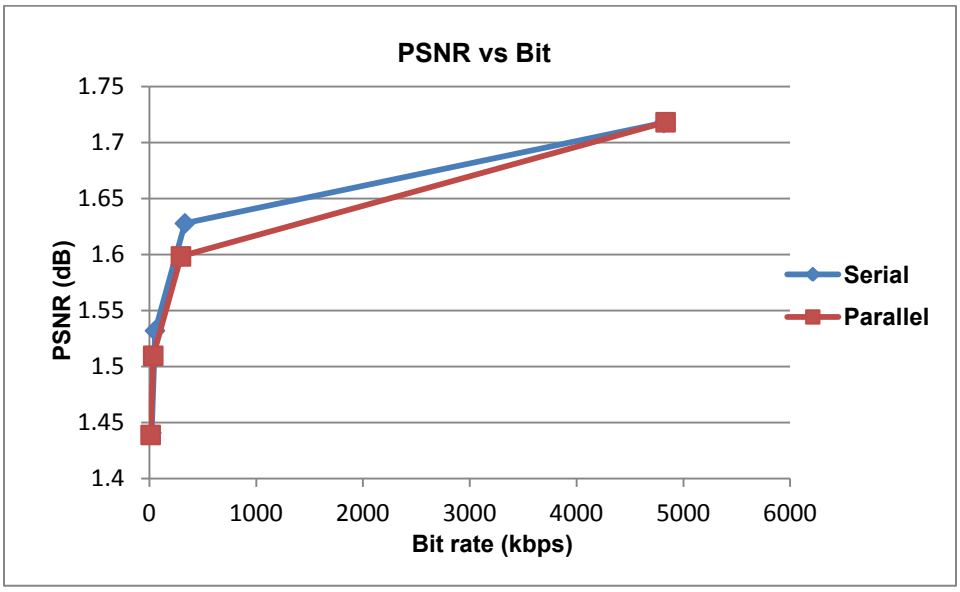

Figure 5.36 PSNR graph for hall_cif.yuv

5.7.5.4 Bridge-close_cif.yuv

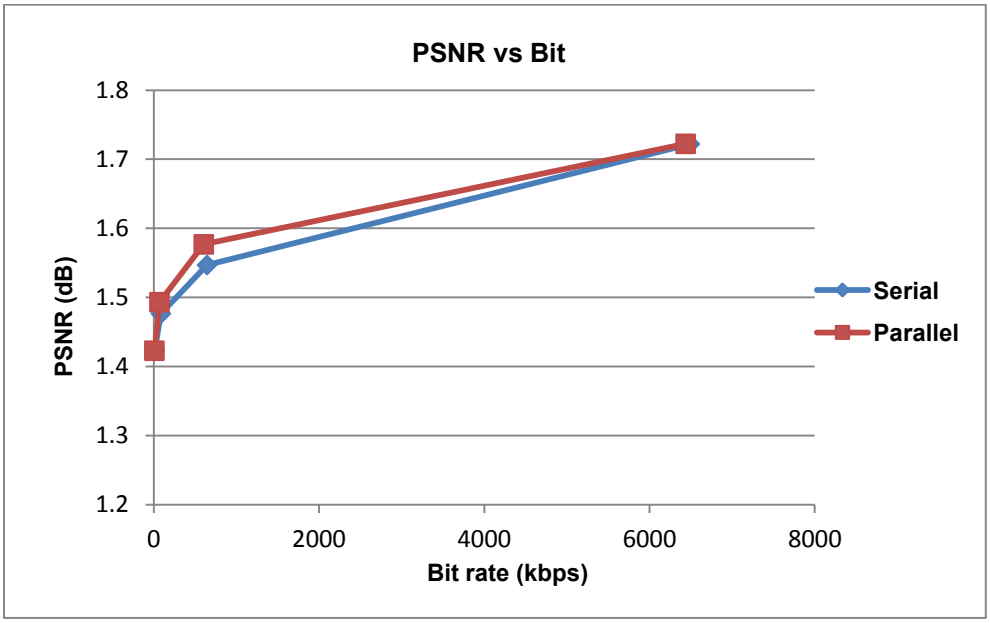

Figure 5.37 PSNR graph for bridge-close_cif.yuv

5.7.5.5 Mobile_cif.yuv

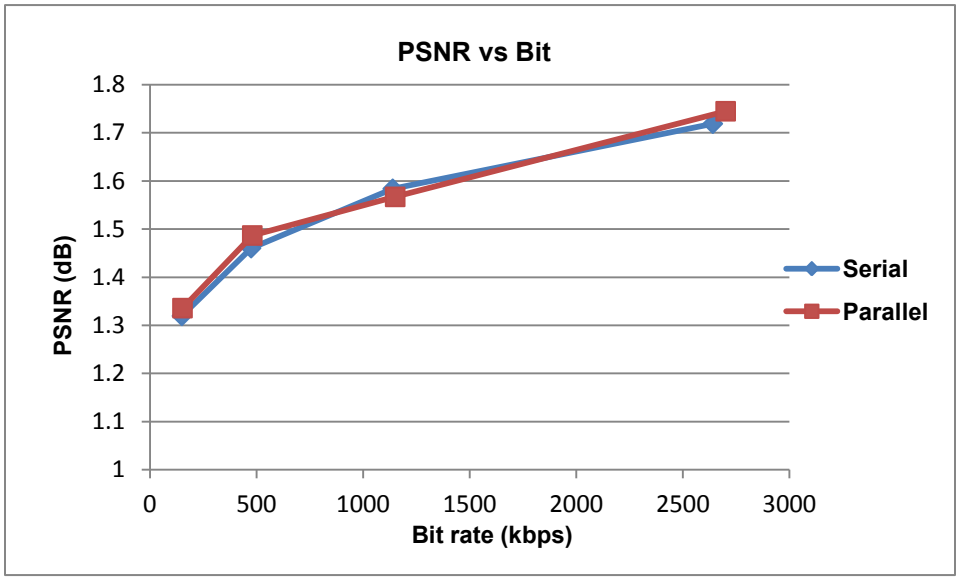

Figure 5.38 PSNR graph for mobile_cif.yuv

5.7.5.6 News_cif.yuv

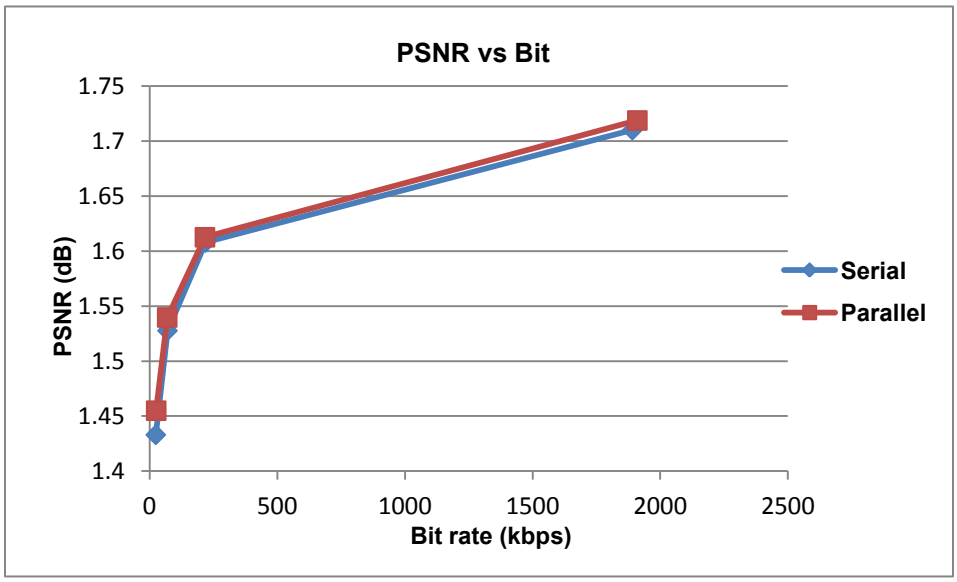

Figure 5.39 PSNR graph for news_cif.yuv

5.7.5.7 Bus_cif.yuv

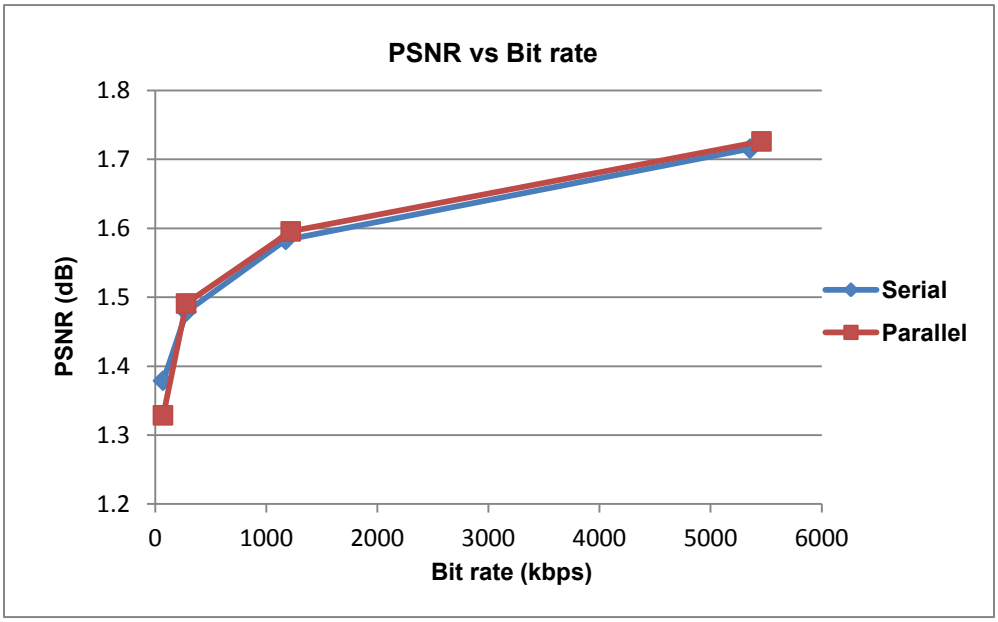

Figure 5.40 PSNR graph for bus_cif.yuv

5.7.5.8 Highway_cif.yuv

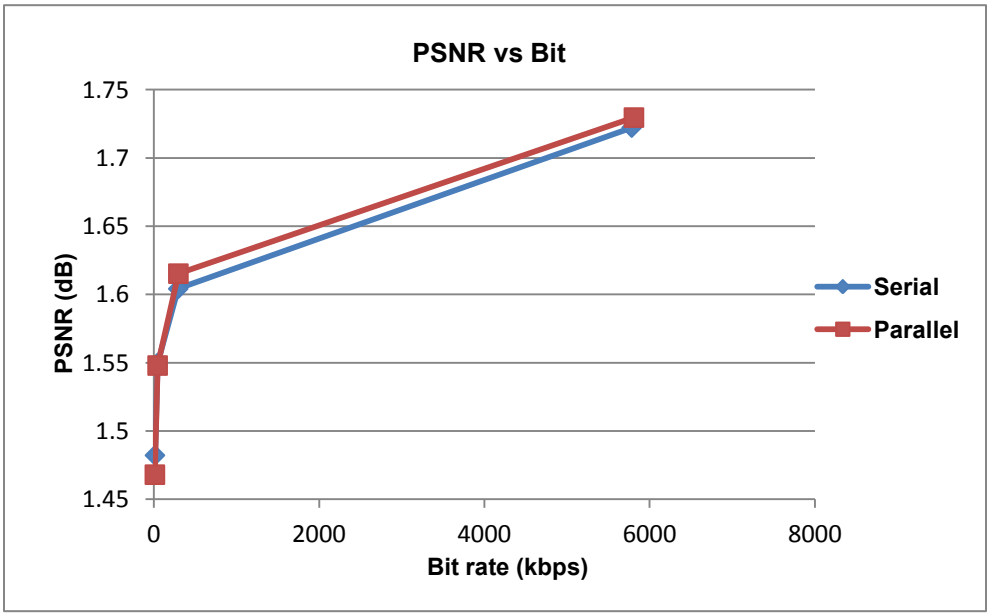

Figure 5.41 PSNR graph for highway_cif.yuv

5.7.5.9 Mother-daughter_cif.yuv

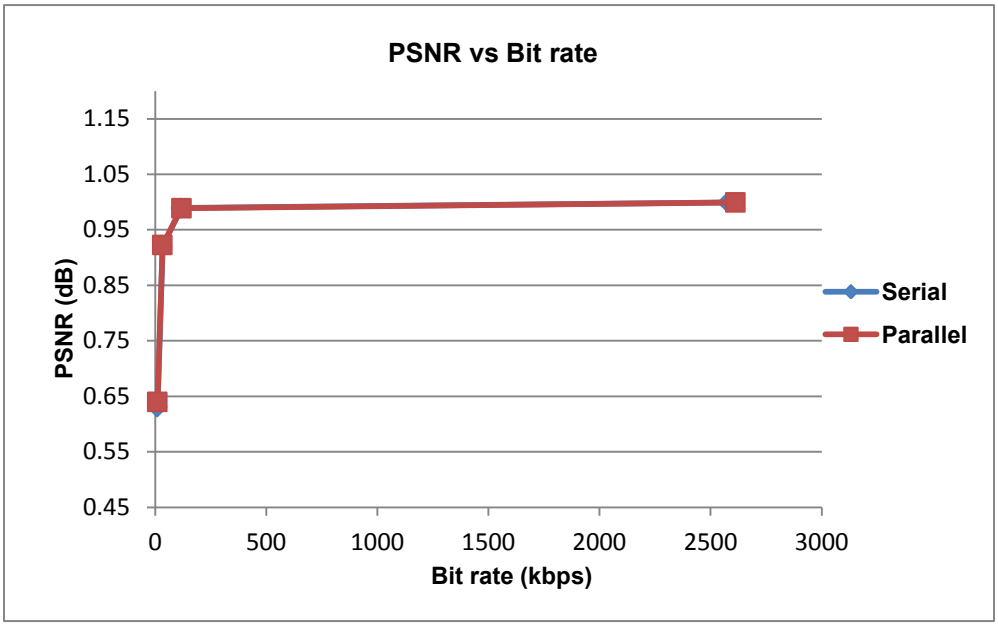

Figure 5.42 PSNR graph for mother-daughter_cif.yuv

5.7.5.10 Flower_cif.yuv

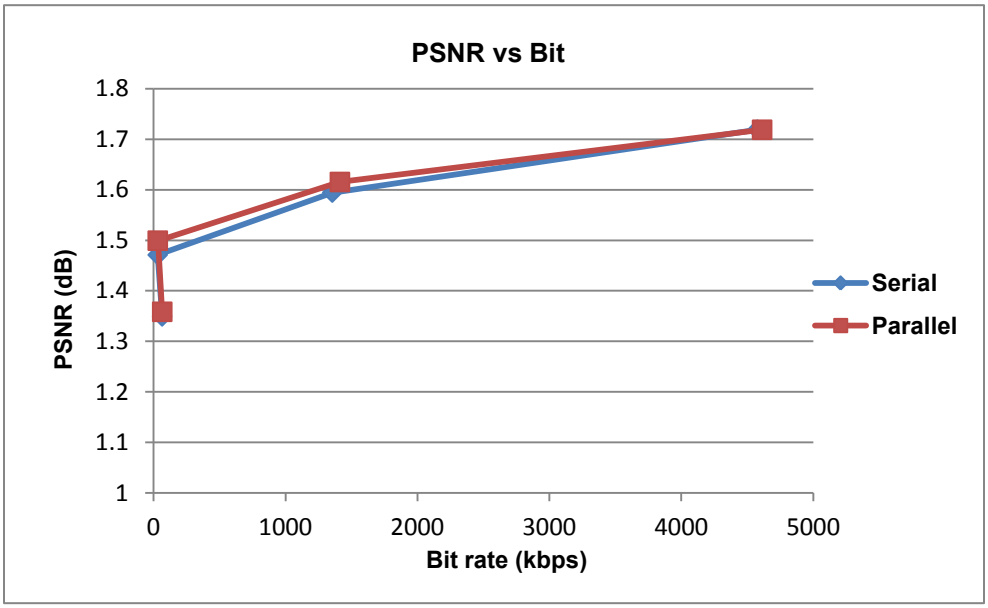

Figure 5.43 PSNR graph for flower_cif.yuv

5.7.5.11 Paris_cif.yuv

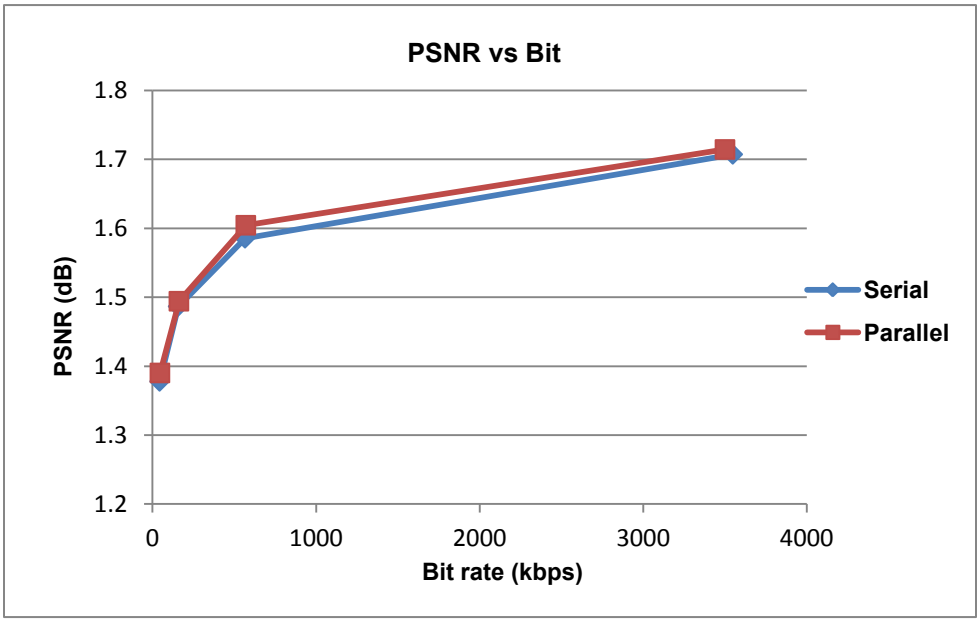

Figure 5.44 PSNR graph for paris_cif.yuv

5.7.5.12 Container_cif.yuv

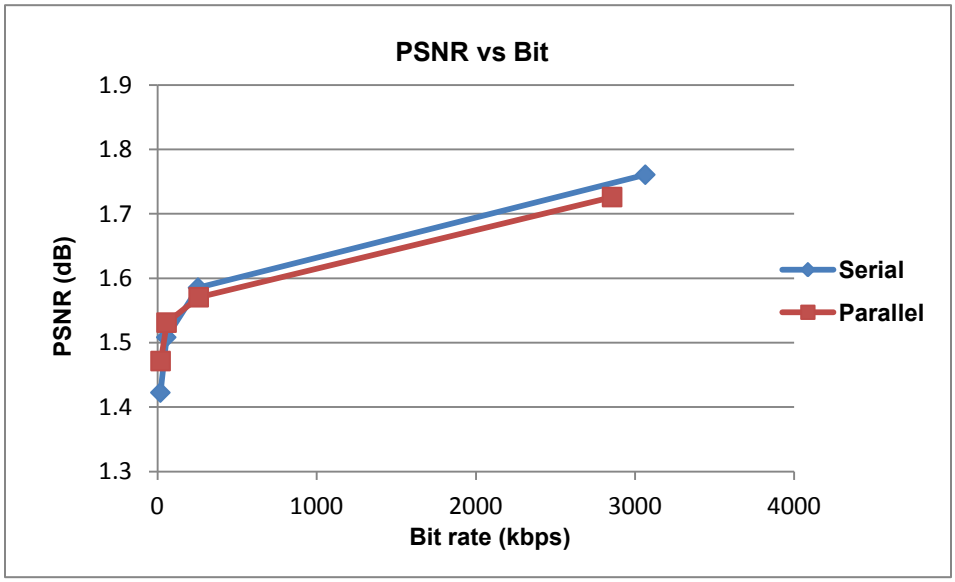

Figure 5.45 PSNR graph for container_cif.yuv

5.7.6 SSIM graphs for CIF sequences

5.7.6.1 Foreman_cif.yuv

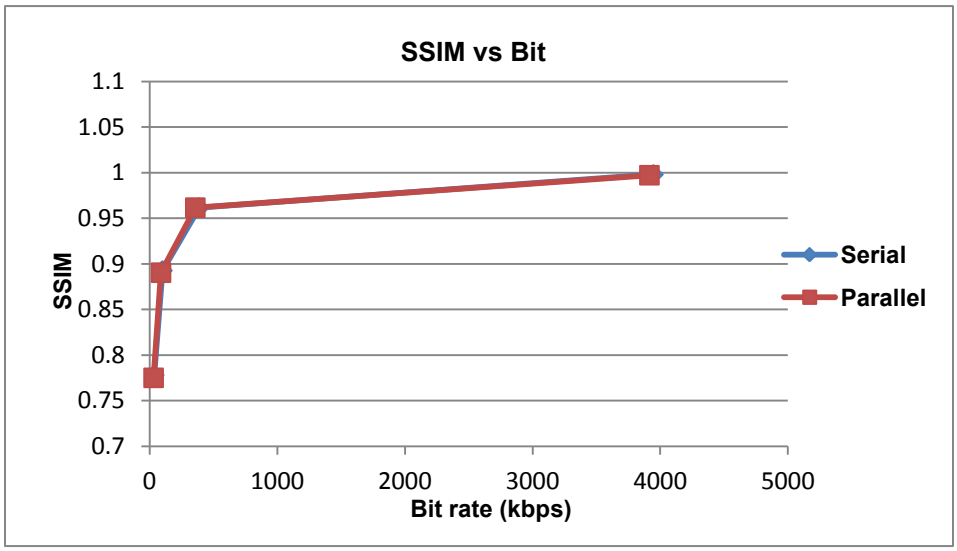

Figure 5.46 SSIM graph for foreman_cif.yuv

5.7.6.2 Bridge-close_cif.yuv

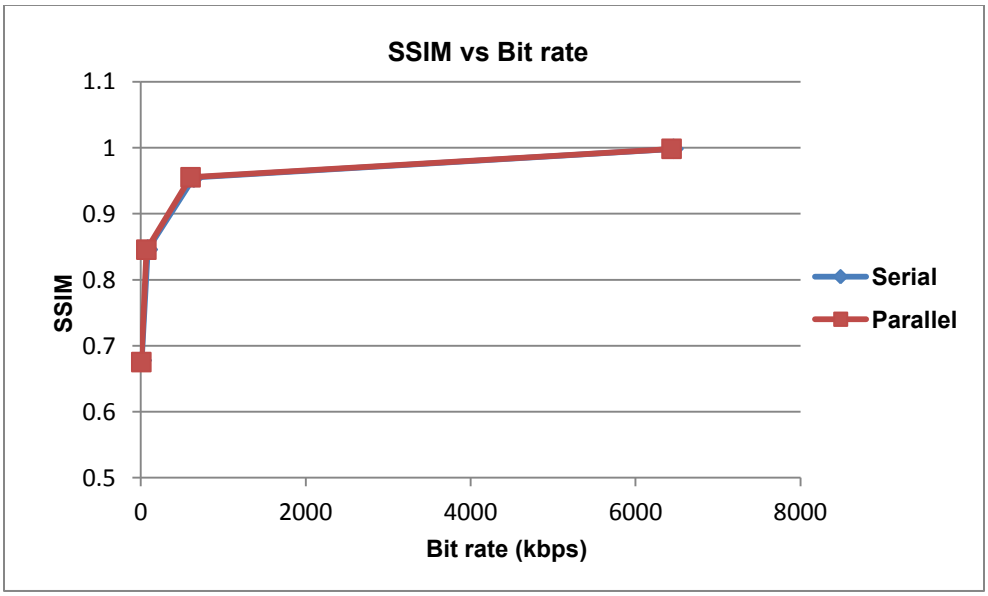

Figure 5.47 SSIM graph for bridge-close_cif.yuv

5.7.6.3 Hall_cif.yuv

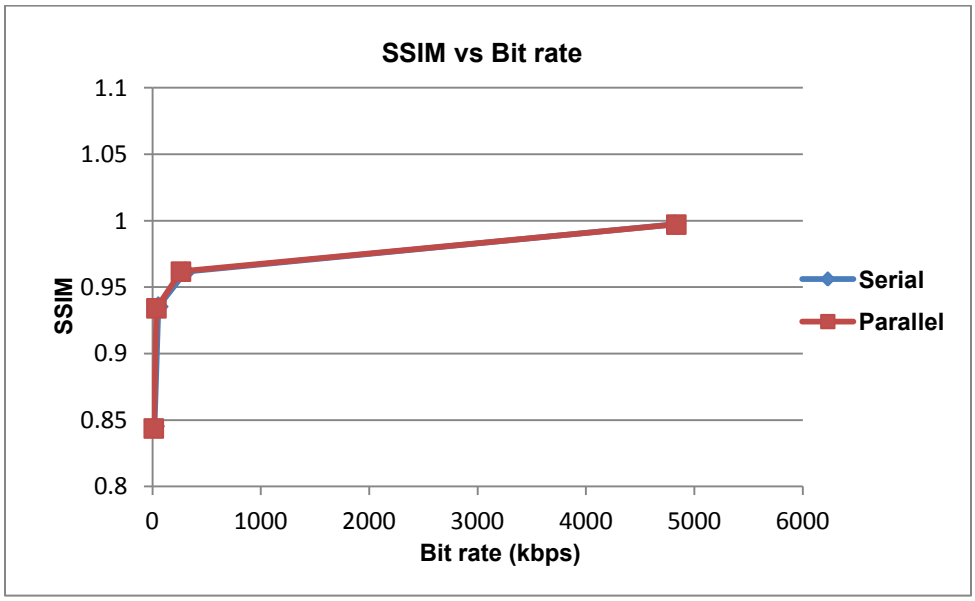

Figure 5.48 SSIM graph for hall_cif.yuv

5.7.6.4 Coastguard_cif.yuv

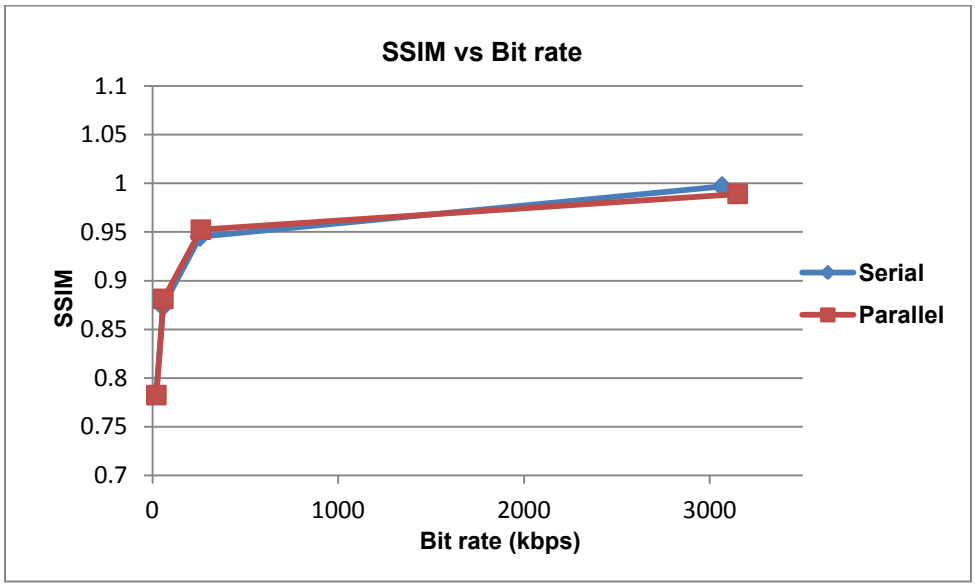

Figure 5.49 SSIM graph for coastguard_cif.yuv

5.7.6.5 Mobile_cif.yuv

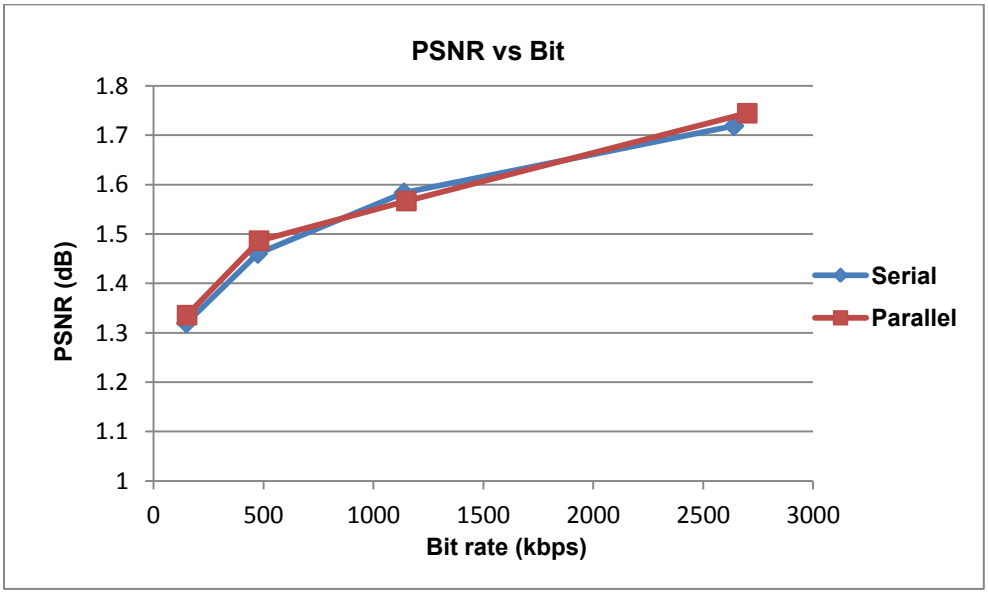

Figure 5.50 SSIM graph for mobile_cif.yuv

5.7.6.6 News_cif.yuv

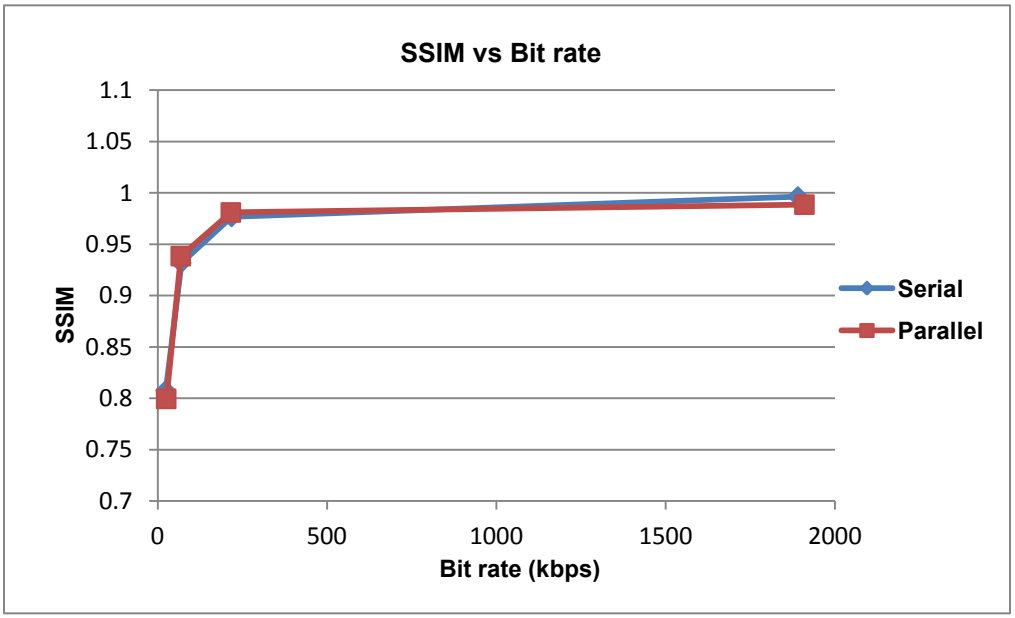

Figure 5.51 SSIM graph for news_cif.yuv

5.7.6.7 Bus_cif.yuv

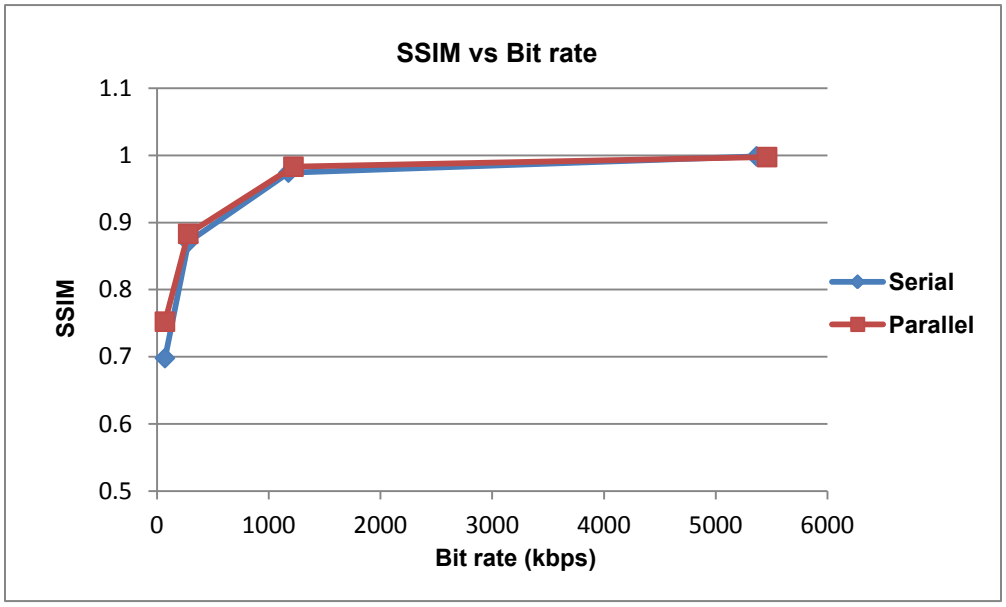

Figure 5.52 SSIM graph for bus_cif.yuv

5.7.6.8 Highway_cif.yuv

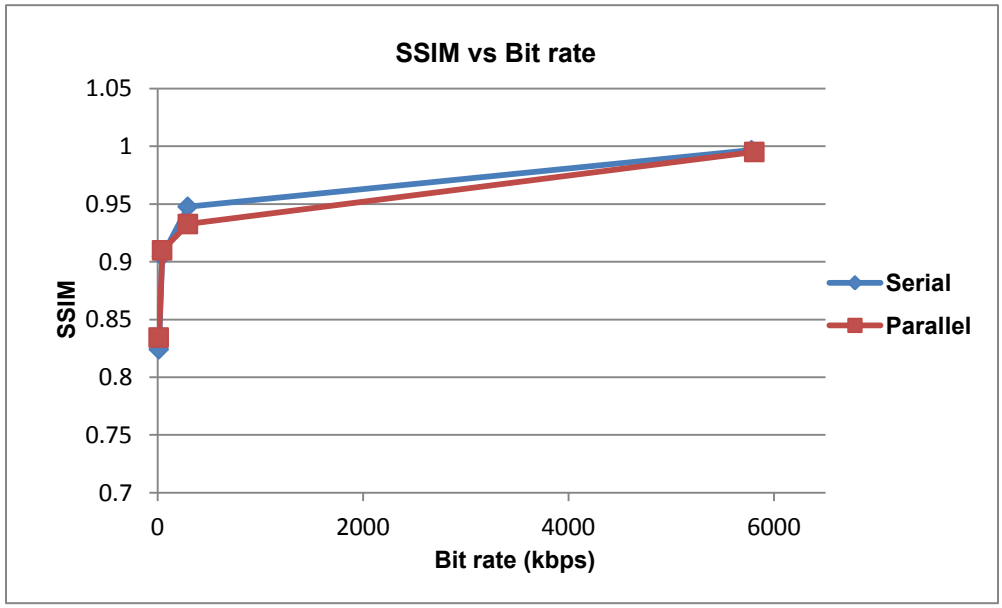

Figure 5.53 SSIM graph for highway_cif.yuv

5.7.6.9 Mother-daughter_cif.yuv

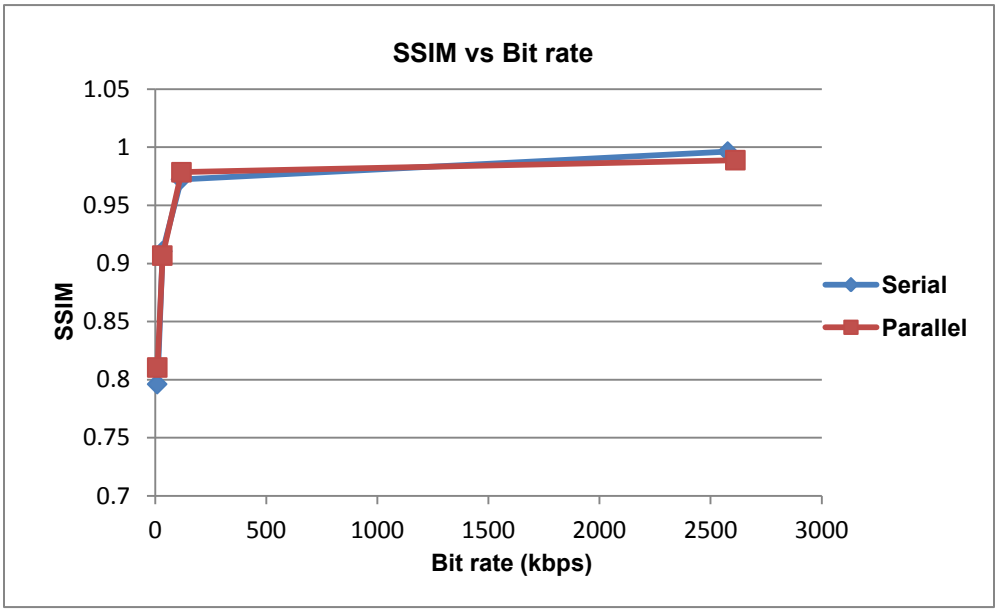

Figure 5.54 SSIM graph for mother-daughter_cif.yuv

5.7.6.10 Flower_cif.yuv

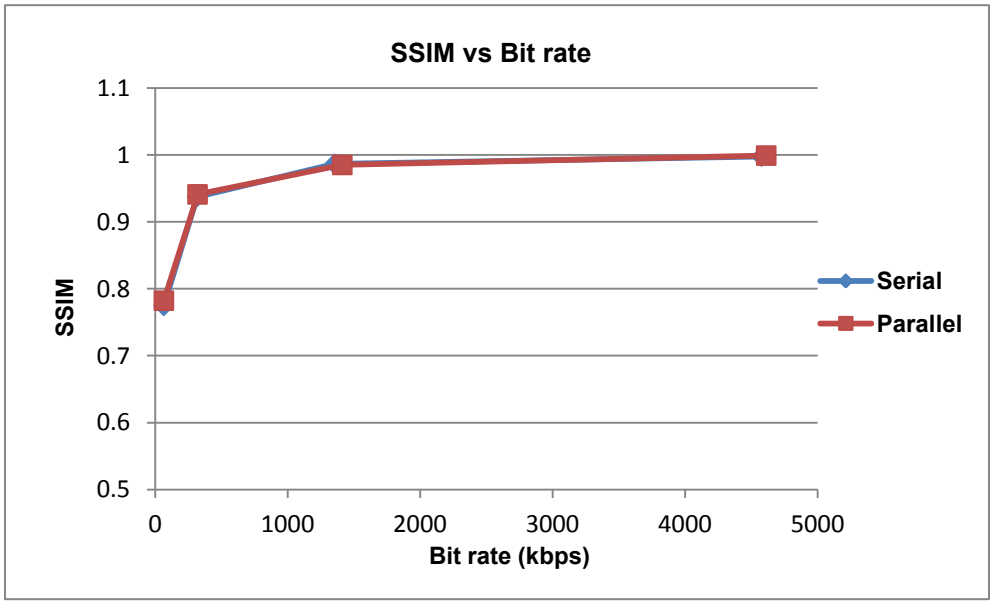

Figure 5.55 SSIM graph for Flower_cif.yuv

5.7.6.11 Paris_cif.yuv

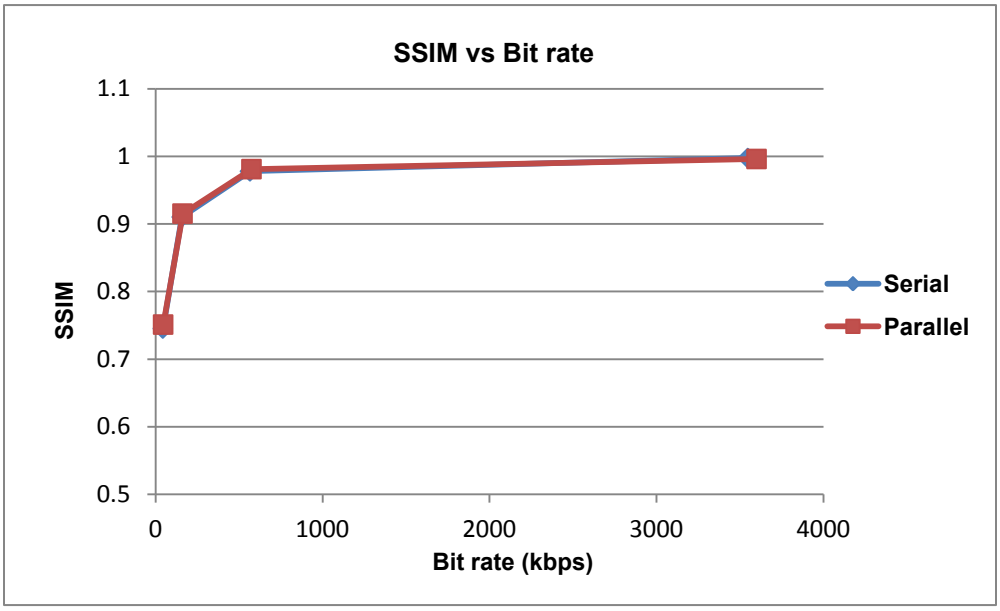

Figure 5.56 SSIM graph for paris_cif.yuv

5.7.6.12 Container_cif.yuv

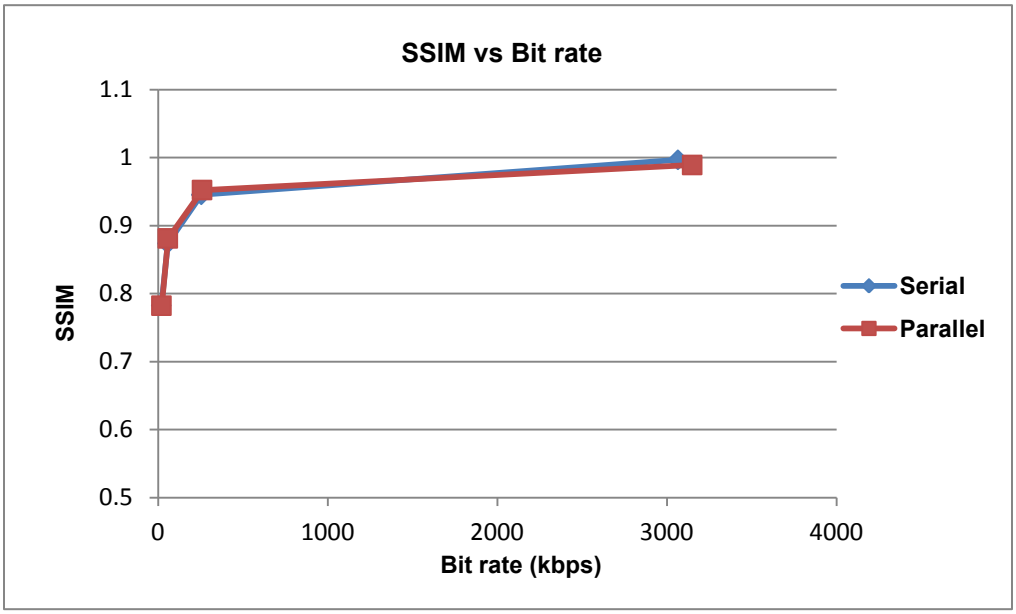

Figure 5.57 SSIM graph for container_cif.yuv

5.7.7 CPU power usage graphs for QCIF sequences

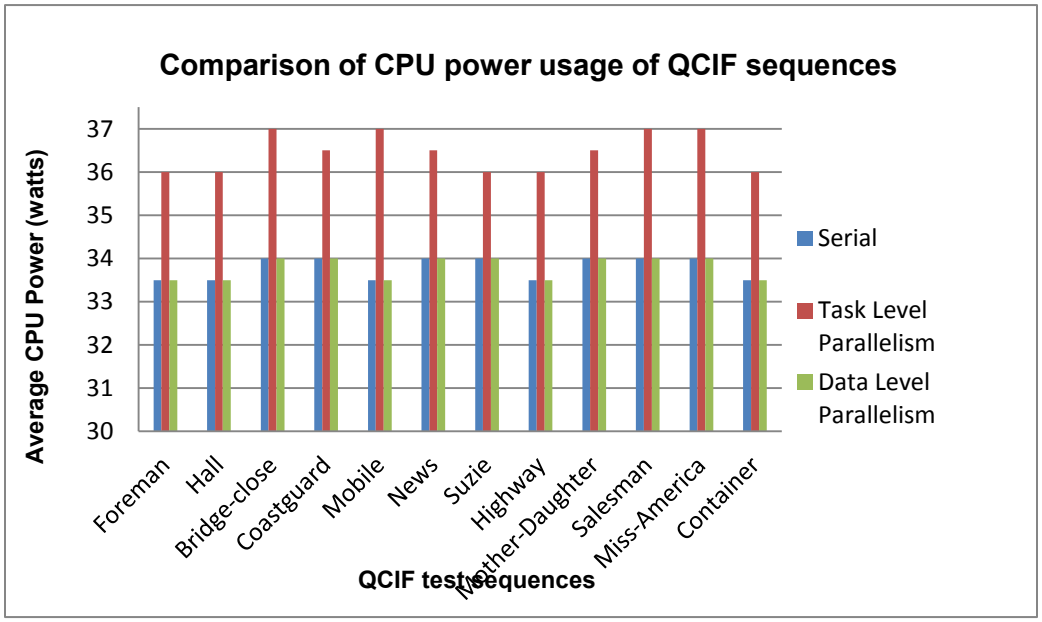

Figure 5.58 Comparison of average CPU power usage for all QCIF sequences

5.7.8 CPU power usage graphs for CIF sequences

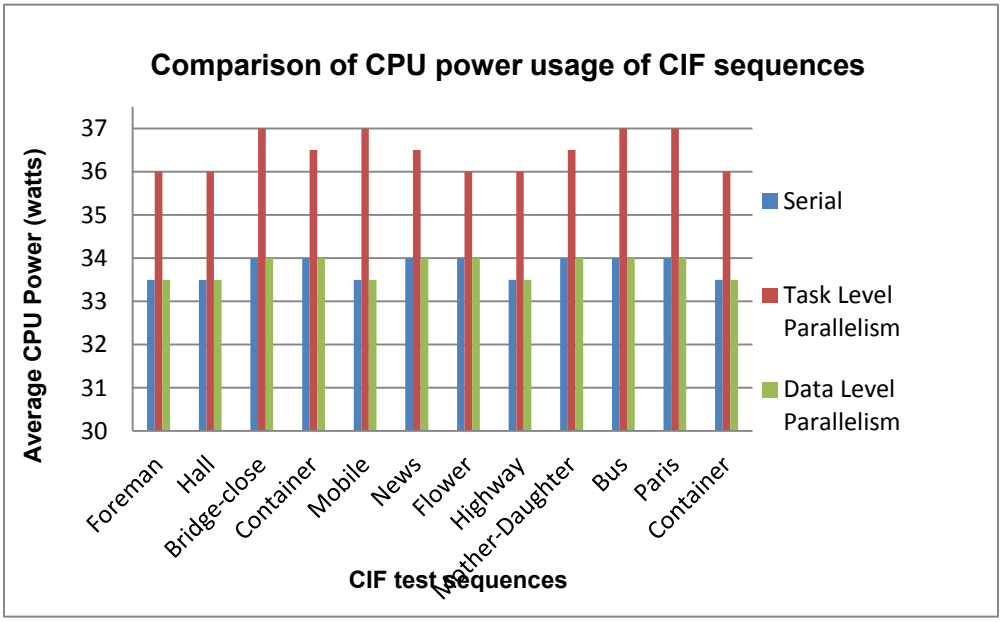

Figure 5.59 Comparison of average CPU power usage for all CIF sequences

5.7.9 Physical memory usage graphs for CIF sequences

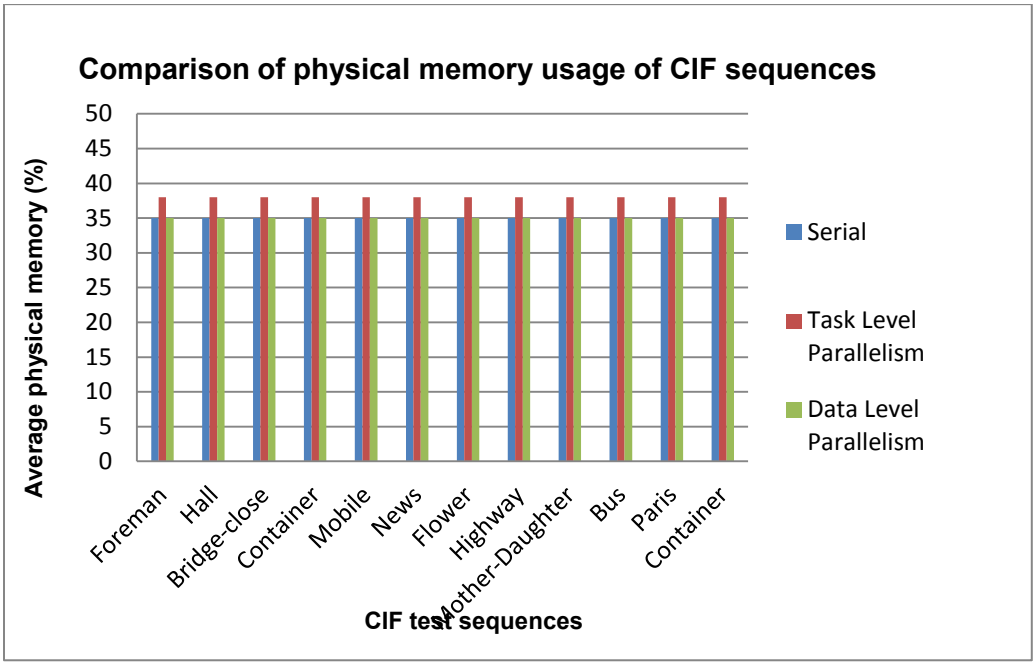

Figure 5.60 Comparison of physical memory usage for all CIF sequences

5.7.10 Physical memory usage graphs for QCIF sequences

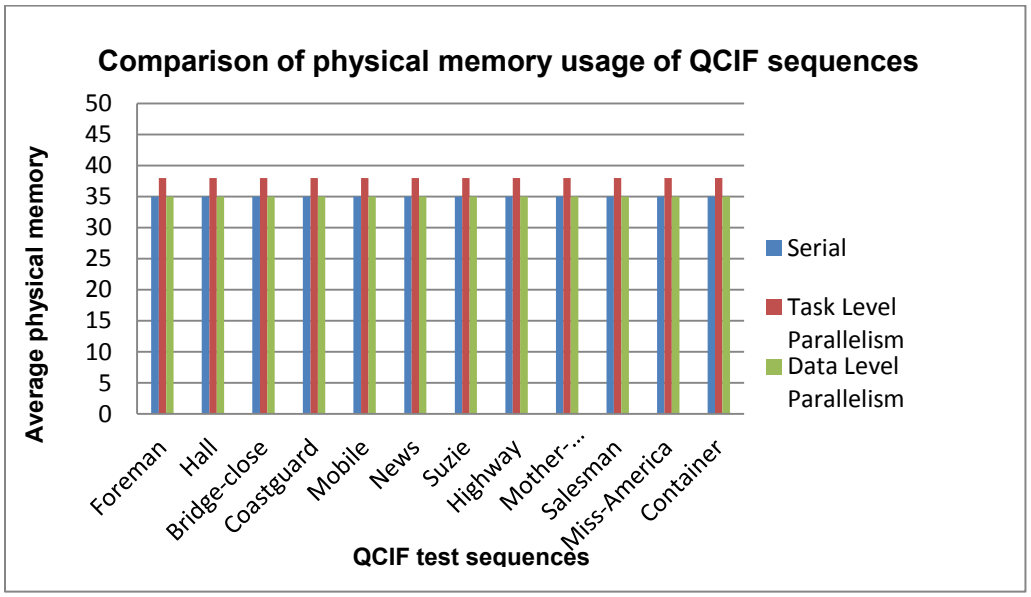

Figure 5.61 Comparison of physical memory usage for all QCIF sequences

5.8 Analysis of task based and data based parallelism

As per the design strategy adopted in this thesis to achieve task based parallelism, a GOP to be encoded is always divided equally into two sub GOP's and each sub GOP is encoded in a single dedicated thread. Hence a maximum of two threads are required to encode an entire GOP as the GOP is always divided into two equal sub GOP's for any video sequence. The number of frames in each sub GOP has to be preset before starting the encoding procedure by setting the 'intra frame period' parameter in the configuration file 'encodebaseline.cfg'. The 'intra frame period' parameter denotes the number of P frames between two successive I frames. Hence to encode a video sequence with a GOP of 30 frames, the 'intra frame period' parameter would be set to 15 with frame 1 and frame 16 as I frame for thread 1 and thread 2 respectively. The successive frames i.e. frame 2 to frame 15 would be encoded as P frames in thread 1 and frame 17 to frame 30 as P frames in thread 2 respectively. By adopting this approach the two threads become independent of each other and hence no data dependency. Also the overhead for thread creation is minimal as only a maximum of two threads are required to encode any video sequence for any number of GOP's. This design strategy for task based parallelism implemented in the JM 18.0 [1] reference software using OpenMP software for thread creation reduces the encoding time for a GOP approximately by 50%. This can be concluded from the readings shown in table 5.1 and 5.4. The other advantage of task based parallelism is the elimination of data dependency. It is eliminated by introducing an additional I frame for each sub GOP; hence the two threads can run in parallel without any race conditions. This reduces the software complexity. The most evident disadvantage of this design is that the encoding of the video sequence can only start after the raw data bytes for the middle frame in a GOP is made available. This can hamper the real time response for a video sequence. Also as an additional I frame is included for encoding a GOP, the bit rate increases marginally as compared to the original reference JM 18.0 [1] software.

The basic idea behind data based parallelism is to identify the hot spots in the software and parallelize them by creating new threads for each hot spot. This drastically increases the overhead for thread creation as the total number of threads created for encoding a GOP is dependent on the number of frames to be encoded and the number of hot spots encountered during encoding of a single frame. For example to encode a video sequence having a GOP of 30 frames with each frame having 5 hot spots to encode it completely and each hot spot being divided equally between a maximum of two parallel threads, the number of threads created would be :

$$\text{No of threads created} = 2 * 5 * 30$$

$$= 300 \text{ threads}$$

Hence compared to task based parallelism an additional overhead of 298 thread creation is required to encode the same video sequence using data based parallelism as per the design strategy explained above. Also the software complexity to eliminate race condition between two threads in data based parallelism is more as compared to task based parallelism. Hence from figures 5.8 and 5.9 it can be shown that the data based parallelism approach carried out on a CPU having a maximum of 2 cores brings about no reduction in the encoding time for most of the video test sequences [60] but increases the encoding time by few seconds for quite a few video test sequences [60]. .

5.9 Summary

This chapter explains the advantages and disadvantages of using task based parallelism approach to encode the video frames over data based parallelism approach. The results were populated on a general purpose CPU having a maximum of 2 cores and running a maximum of 2 threads i.e. 1 dedicated thread on each core to achieve parallelism. OpenMP software is used to create the threads and make it run in parallel on the underlying hardware. The results were populated on video test sequences [60] shown in figure 5.6. These test

sequences covered large variations in motion i.e. from fast moving objects to constant background. Based on the results shown in table 5.1 and 5.4, approximately 50% reduction in the encoding time was achieved by adopting task based parallelism approach. Data based parallelism showed no reduction in the encoding time as compared to the original reference JM 18.0 [1] software. .

The next chapter derives a conclusion based on the analysis of the results shown in chapter 5 and also gives brief introduction as to what can be further implemented using parallel programming methodologies.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

From the simulation results shown in Chapter 5, it is shown that the proposed task based parallel algorithm on a general purpose CPU having 2 cores with a maximum of 2 threads running in parallel created by OpenMP software brings about approximately 50% reduction in the encoding time with no degradation in the video quality as compared to the reference JM 18.0 [1] software. These results are shown in table 5.1 to 5.4 and figure 5.8 to 5.60. PSNR and SSNR are used as video quality metric.

Besides this data based parallelism causes thread creation overhead and there is no reduction in the encoding time as compared to the reference JM 18.0 [1] software. This result is shown in figure 5.8 and figure 5.9. But past research as shown in [65] proves that data based parallelism brings about 20% more reduction as compared to the task based parallelism in this thesis. Hence while deciding the design strategy to adopt to reduce the time complexity; one needs to consider the number of available cores. In case of a 2 core general purpose CPU, task based parallelism is a better approach than data based parallelism as thread creation over head is less in task based parallelism than data based parallelism. Also since data race condition between threads is eliminated by introducing an additional I frame, software for a task based approach is computationally less complex as compared to a data based approach.

6.2 Future work

In this thesis stress is given on adopting a task based parallelism approach to encode the video frames rather than adopting data based parallelism as it raises two major concerns –

1] Overhead in creating thread.

2] Data dependencies involved in intra prediction, inter prediction, loop filtering and CAVLC.

By having a parallel computing model like Compute Unified Device Architecture (CUDA) [64] invented by NVIDIA, the thread creation overhead can be reduced as CUDA threads are extremely lightweight, with very low creation overheads and switching time. The problem of data dependency can also be solved by dividing the frame into many macro-blocks (MB) and then mapping each MB's on different processor to execute in parallel.

REFERENCES

[1] "http://www.iphome.hhi.de/suehring/tml/", JM reference software, Fraunhofer Institute for Telecommunications Heinrich Hertz Institute.

[2] S. Kwon, A. Tamhankar, and K. Rao, "Overview of H. 264/MPEG-4 part 10", Journal of Visual Communication and Image Representation, vol.17, no.2, pp.186 – 216, April 2006.

[3] I. E. Richardson, "The H.264 advanced video compression standard", 2nd Edition, Wiley, 2010.

[4] H.264 / MPEG-4 Part 10 White Paper, I. E. G. Richardson. www.vcodex.com.

[5] I. E.G. Richardson, "H.264 and MPEG-4 video compression: video coding for next-generation multimedia", Wiley, 2003.

[6] "http://www.venlogic.com/vl2/tools/index.php?url=tools_Detail_h264" for H.264 codec reference

[7] "http://www.axis.com/products/video/about_networkvideo/compression.htm" reference for inter-frame coding pictures, Axis Communications.

[8] S.K.Muniyappa, "Implementation of complexity reduction algorithm for intra mode selection in H.264/AVC", M.S. Thesis, EE Dept., University of Texas at Arlington, Dec 2011.

[9] M. Jafari and S. Kasaei, "Fast intra- and inter-prediction mode decision in H.264 advanced video coding", International Journal of Computer Science and Network Security, vol.8, no.5, pp. 1-6, May 2008.

[10] F. Pan et al, "Fast intra mode decision algorithm for H.264/AVC video coding", in Proc.IEEE Int. Conf. Image Process., pp. 781–784, Singapore, Oct. 2004.

[11] F. Fu et al, "Fast intra prediction algorithm in H.264/AVC", in Proc. 7th Int. Conf. Signal Process., pp. 1191–1194, Beijing, China, Sep. 2004

[12] Y. Zhang et al, “Fast 4x4 intra-prediction mode selection for H.264”, in Proc. IEEE Int. Conf. Multimedia Expo, pp. 1151–1154, Taipei, Taiwan, Jun. 2004.

[13] S.Swaroop, “Low complexity H.264 encoder using machine learning for streaming applications”, M.S. Thesis, EE Dept., University of Texas at Arlington, May 2011.

[14] “https://computing.llnl.gov/tutorials/openMP/”, reference for OpenMP.

[15] T.Wiegand and G.J.Sullivan, “The picture is here. Really”, IEEE Spectrum, vol.48, pp. 51-54, Sept. 2011.

[16] “http://en.wikipedia.org/wiki/IPTV”, reference for bandwidth requirement

[17] P. R. Ramolia and K.R. Rao, “Low complexity AVS-M using machine learning algorithm C4.5”, TELSIKS 2011, Nis, Serbia, 5-8 Oct. 2011.

[18] D. Han et al, “Low complexity H.264 encoder using machine learning”, IEEE SPA 2010, pp. 40-43, Poznan, Poland, Sept. 2010.

[19] ITU-T H.264 “Advanced video coding for generic audiovisual services”.

[20] “http://www2.cs.uidaho.edu/~krings/CS240/Notes.F10/240-10-10.pdf”, Flynn’s Taxonomy.

[21] Q.Peng and J.Jing, “H.264 codec system-on-chip and design and verification”, ASIC 2003, 5th International Conference, vol.2, no.7, pp. 922-925, Mar. 2004.

[22] T.Bhatia, “Optimization for H.264 high profile decoder for Pentium 4 processors”, M.S. Thesis, EE Dept., University of Texas at Arlington, Dec. 2005.

[23] G.Sullivan, P.Topiwala and A.Luthra, “The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions,” *SPIE conference on applications of digital image processing XXVII*, Vol. 5558, pp. 53 – 74, Aug. 2004.

[24] D.Marpe, T.Wiegand and S.Gordon, “H.264/ MPEG-4 AVC fidelity range extensions: tools, profiles, performance and application areas”, *Proc. ICIP 2005*, Genova, Italy, Sept. 11-14, 2005.

[25] D.C.Garcia and R.L.De Queiroz, “Least squares directional intra prediction in H.264/AVC”, IEEE signal processing letters, vol.17, no.10, pp. 831-834, Oct. 2010.

[26] A. Al, B. Rao, S. Kudva, S. Babu, D. Sumam, and A. Rao, “Quality and complexity comparison of H.264 intra mode with JPEG2000 and JPEG,” in *Proc. ICIP 2004*, Singapore, Oct. 2004.

[27] S.B.Wang and X.L.Zhang, “H.264 Intra prediction architecture optimization”, Dept. of electronics engineering, Beijing university, Beijing, China.

[28] B.Luo and L.Zhang, “Fast intra prediction mode selection method for H.264 video coding”, ISDEA 2010 international conference, pp.721-724, Oct. 2010.

[29] B.H.Jie, “The new generation of video coding compression standard–H.264/AVC”, Beijing: posts & telecom press, pp.92-96, 2005.

[30] B.G.Kim, “Fast selective intra-mode search algorithm based on adaptive thresholding scheme for H.264/AVC encoding”, IEEE Trans. on circuits and systems for video technology, vol.18, pp.127-133, Jan 2008.

[31] C.C. Cheng, “Fast three step intra prediction algorithm for 4×4 blocks in H.264”, IEEE, ISCAS, vol.2, pp.1509-1512, May 2005.

[32] M.Nadeem, S.Wong and G.Kuzmanov, “An efficient realization of forward integer transform in H.264/AVC intra-frame encoder”, SAMOS, 2010 international conference, pp. 71-78, Nov 2010.

[33] “www.vcodex.com”, H.264/MPEG-4 Part 10: Transform & Quantization.

[34] JVT, Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T rec. H.264 ISO/IEC 14496-10 AVC), May 2003.

[35] H.Li, C.Zhang, J.Ren, L.Li, and D.Liu, “Stream algorithm for 4*4 integer transform in H.264”, Multimedia and ubiquitous engineering, National university of defense technology, China, May 2007.

[36] K.R. Rao and P. Yip, “Discrete cosine transform”, Academic Press,1990.

[37] T.Liu, E.Yang, R.Cheng, Y.Fu, “CUDA-based H.264/AVC deblocking filtering”, School of communication and information engineering, Shanghai University, Shanghai, China, Nov. 2010.

[38] M. Wien, “Variable block-size transforms for H.264/AVC”, IEEE Transactions on Circuits and Systems for video technology, vol.13, no.7, pp. 564-567, July 2003.

[39] A.G.Tescher, “The H.264 / MPEG-4 AVC video coding standard and its development status”, Beijing, China, pp. 719, July 2005.

[40] M.Karczewicz and R.Kurceren, “The SP and SI frames design for H.264/AVC”, IEEE Transactions on circuits and systems for video technology, vol.13, no.7, pp.637-644, July 2003.

[41] ISO/IEC 14496-2, “Information technology – coding of audio-visual objects – Part 2”, Dec. 2001.

[42] ITU-T Recommendation H.263, “Video coding for low bit rate communication”, Feb. 1998.

[43] “http://en.wikipedia.org/wiki/OpenMP”, reference for OpenMP.

[44] “http://openmp.org/wp/openmp-compilers/”, OpenMP Compilers.

[45] “http://openmp.org/wp/2008/10/openmp-tutorial-at-supercomputing-2008/ “, OpenMP Tutorial at Supercomputing 2008.

[46] “http://openmp.org/wp/2009/04/download-book-examples-and-discuss/”, Using OpenMP – Portable Shared Memory Parallel Programming – Download Book Examples and Discuss.

[47] “http://openmp.org/wp/about-openmp/”, About the OpenMP ARB and OpenMP.org.

[48] “http://openmp.org/wp/2008/11/openmp-30-status/”, OpenMP 3.0 Status.

[49] “https://computing.llnl.gov/tutorials/openMP/#Introduction”, OpenMP goals.

[50] T.Sathe, “Complexity reduction of H.264/AVC motion estimation using OpenMP”, M.S. Thesis, EE Dept., University of Texas at Arlington, May 2012.

[51] “http://ranger.uta.edu/~walker/CSE%205343_4342_SPR11/Web/Lectures/Lecture-4-Threading%20Overview-Ch2.pdf”, System Overview of Threading.

[52] “http://www.openmp.org”, OpenMP manual.

[53] H. Schwarz, D. Marpe and T. Wiegand, ‘Analysis of Hierarchical B Pictures and MCTF’, IEEE International Conference on Multimedia and Expo (2006), pp. 1929–1932.

[54] Z. Wang, et al, “Image quality assessment: From error visibility to structural similarity,” IEEE Trans. Image Processing, vol. 13, pp. 600–612, Apr. 2004.

[55] T.Wiegand et al, “Rate-constrained coder control and comparison of video coding standards,” IEEE Trans. Circuits Systems Video Technology, vol. 13, no.7, pp.688-703, July 2003.

[56] Z. Wang, et al, “Image quality assessment: From error visibility to structural similarity,” IEEE Trans. Image Processing, vol. 13, pp. 600–612, Apr. 2004.

[57] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.

[58] “http://en.wikipedia.org/wiki/Structural_similarity”, Introduction to SSIM.

[59] “http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio”, Introduction to PSNR.

[60] “http://trace.eas.asu.edu/yuv/”, YUV test sequences.

[61] “http://softpixel.com/~cwright/programming/simd/”, SIMD instruction sets.

[62] “http://www.ghacks.net/2010/05/31/track-pc-power-consumption-with-microsoft-joulemeter/”, Joule-meter introduction.

[63] “http://www.filecluster.com/downloads/Joulemeter.html”, Software setup for joule-meter.

[64]“http://www.nvidia.com/content/cudazone/download/Getting_Started_w_CUDA_Training_NVISION08.pdf”, Getting Started with CUDA.

[65] Shuwei Sun, et al, “A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition”, Springer-Verlag Berlin Heidelberg, pp. 577–585, 2007.

BIOGRAPHICAL INFORMATION

Tushar Ashok Saxena was born in Mumbai, India in 1986. He received the Bachelor's degree in Electronics and Communication Engineering from Mumbai University, India in 2008. He worked as an Embedded Software Engineer in Larsen & Toubro, Mumbai, India from September 2008 to December 2010.

He decided to pursue the Master's degree from The University of Texas at Arlington in spring 2011. He worked as a Graduate Research Assistant under Dr. Rao in the Multimedia Processing Lab from summer 2011 to fall 2011. He got an opportunity to work as Radio Software Protocol Stack Test engineer, intern at Research in Motion in Sunrise, Florida from spring 2012 to summer 2012. His interests lie in the field of video coding and embedded systems.