

# DISTRIBUTED ENCODER AND DECODER FOR H.264

By

**DIPTI SONI**

**(08MCE025)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**AHMEDABAD-382481**

**MAY 2010**

# DISTRIBUTED ENCODER AND DECODER FOR H.264

## Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master Of Technology In Computer Science And Engineering

By

DIPTI SONI

(08MCE025)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2010

## Certificate

This is to certify that the Major Project entitled "DISTRIBUTED ENCODER AND DECODER FOR H.264" submitted by DIPTI SONI (08MCE025), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

**Prof. Priyanka Sharma**

Guide, Assistant Professor

Department Of Computer Engineering

Institute of Technology

Nirma University

Ahmedabad.

**Dr. S.N. Pradhan**

PG co-ordinator

Department Of Computer Engineering

Institute of Technology

Nirma University

Ahmedabad.

**Prof.D.J Patel**

Prof. and Head

Department Of Computer Engineering

Institute of Technology

Nirma University

Ahmedabad

**Dr.Ketan Kotecha**

Director

Institute of Technology

Nirma University

Ahmedabad

## **Abstract**

This project using H.264 for distributed encoder and decoder, that will be implemented using parallel computing. This is techniques for video compression starting from basic concepts. The rate-distortion performance of modern video compression schemes is the result of an interaction between motion representation techniques, intra-picture prediction techniques, waveform coding of differences, and waveform coding of various refreshed regions. The project starts with an explanation of the basic concepts of video codec design and then explains how these various features have been integrated into international standards, up to and including the most recent such standard, known as H.264.

This project is implemented by using the CABAC algorithm and for parallel computing using the beowulf cluster. First part of the project is implementation of H.264 systems on parallel machine. The same set up will be later tested with 8 core processor.

## Acknowledgements

With immense pleasure, I would like to present this report on the dissertation work related to DISTRIBUTED ENCODER AND DECODER FOR H.264. I am very thankful to all those who helped me for the dissertation work and for providing valuable guidance throughout the project work.

I would First of all like to thank Dr S N Pradhan, Professor In Charge MTech(CSE), Institute of Technology, Nirma University, Ahmedabad whose keen interest and excellent knowledge base helped me to finalize the topic of the dissertation work. His constant support and interest in the subject equipped me with a great understanding of different aspects of the required architecture for the project work.

My kind and sincere thanks and gratitude to Dr Ketan Kotecha, Director Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the Dissertation work. My sincere thanks and gratitude to Prof D J Patel, Professor and Head Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the Dissertation work.

I would also like to thank Prof Priyanka Sharma, Assistant Professor, CSE Department, Nirma Institute of Technology, Nirma University, whose knowledge and expertise in Computer architechure has been a great pillar of support for my work. She has shown keen interest in this dissertation work right from beginning and has offered a great motivating factor in outlining the flow of my work.

The blessings of God, faculty members and my family members makes the way for this major project. I am very much grateful to them.

**- DIPTI SONI**  
**08MCE025**

# Contents

Certificate	iii
Abstract	iv
Acknowledgements	v
List of Tables	ix
List of Figures	ix
Abbreviations	ix
<b>1 Introduction</b>	<b>2</b>
1.1 General Overview . . . . .	2
1.2 Motivation . . . . .	3
1.3 Objective . . . . .	4
1.4 Scope Of Work . . . . .	4
1.5 Organization Of Major Project . . . . .	5
<b>2 Literature Survey</b>	<b>6</b>
2.1 Concept Video Codec For H.264 . . . . .	6
2.1.1 H.264 Encoder . . . . .	6
2.1.2 H.264 Decoder . . . . .	7
2.2 Parallelization Of H.264 . . . . .	8
2.3 Overview Of Cluster Computing . . . . .	9
2.4 Conclusion . . . . .	9
<b>3 High Efficient Parallel Algorithm For H.264</b>	<b>10</b>
3.1 Parallel Algorithm For H.264 . . . . .	10
3.1.1 Data Dependencies In H.264 . . . . .	11
3.1.2 Data Dependencies Introduced By Inter-prediction . . . . .	11
3.2 CABAC Algorithm For H.264 . . . . .	12
3.2.1 Context Modeling In CABAC . . . . .	12
3.2.2 Proposed Grasp Algorithm . . . . .	13
3.2.3 Grasp For Group Of Picture . . . . .	16

<b>4</b>	<b>Related Work</b>	<b>17</b>
4.1	Example Of Distributed Encoding (Media Encoding Cluster ) . . . .	17
4.1.1	What Is Media Encoding Cluster . . . . .	17
4.1.2	Features Of Media Encoding Cluster . . . . .	17
4.1.3	Output Of Media Encoding Cluster . . . . .	19
4.2	Parallel Computing Using Erlang Distribution . . . . .	20
4.2.1	What Is Erlang . . . . .	20
4.2.2	Components . . . . .	21
4.3	Tools And Libraries . . . . .	22
4.3.1	Example Of Erlang Program . . . . .	24
4.4	Analysis Of Related Work . . . . .	24
<b>5</b>	<b>Implementation Details Of H.264</b>	<b>25</b>
5.1	Data Domain Decomposition Of H.264 . . . . .	25
5.2	Skeleton Of H.264 Reference Code . . . . .	26
5.2.1	Implementation Details Of H.264 . . . . .	27
5.2.2	NAL units, Slices,Fields And Frames . . . . .	27
5.2.3	Slice Types . . . . .	27
5.2.4	Motion Compensation . . . . .	28
5.2.5	Motion Vector Prediction . . . . .	29
5.2.6	Macroblock Layer . . . . .	29
5.2.7	Macroblock Modes For I Slice . . . . .	29
5.2.8	Macroblock Modes For P And B Slices . . . . .	30
5.2.9	Block Transformation And Encoding . . . . .	31
5.2.10	Entropy Coding Modes . . . . .	31
5.2.11	DC Transformation . . . . .	31
5.2.12	In Loop Deblocking Filter . . . . .	32
5.2.13	Advanced Prediction . . . . .	32
5.2.14	Arbitrary Slice Ordering . . . . .	32
5.2.15	Flexible MacroBlock Ordering . . . . .	33
5.3	CABAC Encoding In H.264 . . . . .	34
<b>6</b>	<b>Implementation Of Codec And Simulation Result</b>	<b>37</b>
6.1	Parallelization Of The H.264 Encoder Using C Code . . . . .	37
6.1.1	Function Documentation . . . . .	37
6.2	Parallelization Of The H.264 Decoder Using C Code . . . . .	40
6.2.1	Function Documentation . . . . .	40
6.3	Output Of Encoder And Simulation Result . . . . .	43
6.3.1	Simulation Result . . . . .	44
6.3.2	Performance And Analysis . . . . .	44
6.4	Conclusion . . . . .	46
<b>7</b>	<b>Parallel Computing Using The Beowulf Cluster</b>	<b>47</b>
7.1	Why Beowulf Cluster? . . . . .	47
7.2	Principal Specification . . . . .	47
7.3	Use Linux For Beowulf . . . . .	49

7.4	A Kernel And A Distribution . . . . .	50
7.5	Network Software . . . . .	50
7.5.1	Tcp/Ip . . . . .	50
7.5.2	Socket . . . . .	52
7.5.3	Remote Procedure Call . . . . .	53
7.5.4	Distributed File System . . . . .	54
7.5.5	Secure Shell . . . . .	54
7.5.6	Mpich . . . . .	54
8	Implementation Of Beowulf Cluster . . . . .	55
8.1	Introduction . . . . .	55
8.2	Logical View Of Beowulf Cluster . . . . .	55
8.3	Requirements . . . . .	56
8.4	Setting Up Cluster Installation And Configuration . . . . .	56
8.5	Testing For Cluster . . . . .	60
8.5.1	Xinetd Insatallment . . . . .	60
8.5.2	Rsh And Rlogin Configuration . . . . .	62
8.5.3	Installment Of Pmandel and Cpilog Files . . . . .	62
8.5.4	Output Of Cluster Testing . . . . .	63
8.5.5	Configuration Between Master And Slave Node . . . . .	63
9	Conclusion And Future Work . . . . .	66
9.1	Conclusion . . . . .	66
9.2	Future Work . . . . .	66
	References . . . . .	67
	Website References . . . . .	67
	Index . . . . .	68



## Abbreviations

AVC	Advanced Video Coding
CABAC	Context Analysis Based Adaptive Codec Algorithm
CAVLC	Context Adaptive Variable Length Coding
CPU	Central Processor Unit
DCT	Discrete Cosine Transform
FMO	Flexible Macroblock Ordering
GOP	Group Of Picture
GRASP	Growing, Reordering and Selection by Pruning (GRASP)
IDR	Instantaneous Decoding Refresh
JVT	Joint Video Team
MB	Macro Block
MC	Motion Compensation
MPEG	Motion Picture Expert Group
MPI	Message Passing Interface
PSNR	Peak Signal-To-Noise Ratio
QP	Quantization Parameter
RPC	Remote Procedure Call

# List of Figures

2.1	Encoder and Decoder for H.264[4]	7
2.2	Hierarchy of Data Domain Decomposition in H.264[5]	8
3.1	Data Dependencies Introduced By Inteprediction[5]	11
3.2	Relationship Between a Template of Context Parameters (left)and a Full Balanced Tree For Binary Data (right)[3]	13
3.3	Example Of Pruned Tree Over Reordered Tree Each Internal Final Tree Is Labeled By Its Assosiate Context Parameter Index (right)[3]	16
4.1	Media Encoding Processing system	18
4.2	Output on client side	19
4.3	Output on Server side	20
4.4	Output of Erlang Program	24
5.1	Data Domain Decomposition Of H.264[9]	26
5.2	Parameter Set Use With Reliable "Out Of Band" parameter set exchange[5]	28
5.3	Macroblock Partition For H.264[8]	30
5.4	Subdivision Of Picture Into Slices(When Not Using FMO)[7]	33
5.5	Subdivision Of A QCIF Frame Into Slices When Utilizing FMO.[7]	33
5.6	Skeleton Of H.264 Reference Code	35
5.7	FlowChart Of CABAC Encoding Process	36
6.1	Output OF Encoder For H.264	43
6.2	Performance According To Rate Distortion	45
8.1	Logical View Of Beowulf Cluster	56
8.2	Requirements for Beowulf Cluster	57
8.3	Output Of Xinetd Configuration	61
8.4	Output Of Rsh And Rlogin Configuration	62
8.5	Output Of Pmandel Files	63
8.6	Output Of Cpilog Files	64
8.7	Output Of Cluster Testing	65
8.8	Configuration Between Master And Slave Node	65

# Chapter 1

## Introduction

This chapter covers general overview of the Thesis work. It include the motivation, objectives, and the scope of the Thesis work. It also guide about the organization of the thesis report.

### 1.1 General Overview

We Know last generation video encoding standards increasing computing demand in order to reach the limits on compression efficiency. This is particularly the case of that is gaining interest in industry, we are interested in applying parallel processing and parallel computing to H.264 in order to fulfill the computation requirement imposed by stressing applications like video on demand , video conference live broadcast etc[1].

In this work we propose a hierarchical parallelization of H.264 encoder very well suited to low cost cluster our proposal uses message passing parallelization at two levels GOP and frame optimization at the lowest parallelization level.

The high computational demands of the H.264 decoding process pose serious challenges on current processor architectures. A natural way to tackle this problem is the use of multi-core systems. The contribution of this project lies in a systematic overview and performance evaluation of parallel video decoding approaches.

Video Compression is based on removing sensitive redundant information and in the high spatial and tempral correlation. Last generation video encoding technique particularly H.264 push the capabilities of these technique to their limits, the result is the reduction in bandwidth requirements the several order of magnitude.

A video sequence is a stream of frames generated at a certain frequency or a frame rate. H.264 specification allows definition of a number of consecutive frames as an independent unit(GOP) to be encoded. H.264 also allows defining slices inside a frame as a frame portion that can also be independently encoded.

Encoding efficiency has a price that is computation power H.264 encoders has a very high CPU demand, the most critical case is encoding with latency and real time response requirements. When this is combined with high quality video format the only adequate platform are those with super computing capabilities clusters, multiprocessor, and special purpose we are interested in cluster platform because they are becoming a commonly available resources in an increasing number of companies and institution that require a high performance system able to cope with large scale application. Parallel programming on cluster is also very flexible and it allows the design of parallel video encoders adapted to almost very requirements. Resources available on clusters vary from single to multiple CPU per node and every node we can have a multimedia extension in the CPU .

## 1.2 Motivation

We know today's work stations are about hundred times faster than those made just a decade ago but some computational scientists and engineers need even more speed. They make great simplifications to the problem they are solving must wait hours , days or even weeks for their programs to finish running

Faster computers tackle larger computations. Suppose we can afford to wait overnight for our program to produce a result. If our program suddenly runs 10 times faster, previously out of reach computations would be within our grasp. We could produce in 15 hour an answer that previously required nearly week to generate.

We could simply wait for CPU to get faster . In about 5 years single CPU will be 10 times faster than today(A consequence of moore's law).

On the other hand , parallel computing is proven way to get higher performance now.

### 1.3 Objective

Implementation of Distributed encoder and decoder for H.264 using the parallel machines. The same set up will be tested with 8 core processor. After that we will compare the for parallel computing and parallel processing.

### 1.4 Scope Of Work

- The role of parallelism in accelerating computing speeds has been recognized for several decades.
- Parallel platforms provide increased bandwidth to the memory system.
- Parallel platforms also provide higher aggregate caches.
- Principles of locality of data reference and bulk access, which guide parallel algorithm design also apply to memory optimization.
- Applications such as information retrieval and search are typically powered by large clusters.

## 1.5 Organization Of Major Project

**Chapter2, *Literature Survey***, This Chapter covers the basic Concept of the Video Codec and high efficient parallel Algorithm for H.264 . It give description of how to parallelize H.264

**Chapter3, *High Efficient Parallel Algorithm For H.264***, This Chapter covers the description of how to parallelize H.264 using the CABAC algorithm.

**Chapter4, *Related Work***, This Chapter covers the Example of distributed encoding that is Media Encoding Cluster and problem of parallel computing using the Erlang Distribution .

**Chapter5, *Implementation Details Of H.264*** This Chapter covers the parallelization of H.264 Encoder and Decoder

**Chapter6, *Implementation And Simulation Result*** This Chapter covers the parallelization of H.264 Encoder and Decoder Using C language

**Chapter7, *Parallel Computing Using Beowulf Cluster***, This Chapter covers the Solution of Erlang Distribution and how to do parallel computing using the Beowulf Cluster.

**Chapter8, *Implementation Of Beowulf Cluster***, This Chapter covers the how to implement the beowulf cluster using the fedoa8

**Chapter9, *Conclusion And Future Work*** This Chapter covers conclusion and future work of this project.

## Chapter 2

# Literature Survey

This chapter covers the brief explanation of the H.264 Encoder relevant to the thesis work. It covers the parallelization of H.264 and high efficient parallel algorithm for H.264.

### 2.1 Concept Video Codec For H.264

The standard defines the syntax of an encoded video bitstream together with the method of decoding this bitstream. The basic functional elements prediction, transform, quantization, entropy encoding are little different from previous standards MPEG1, MPEG2, MPEG4, H.261, H.263 the important changes in H.264 occur in the details of each functional element.[10]

#### 2.1.1 H.264 Encoder

An input frame is presented for encoding. The frame is processed in units of a macro block corresponding to  $16 \times 16$  pixels in the original image. Each macro block is encoded in intra or inter mode. In either case, a prediction macro block is formed based on a reconstructed frame. In Intra mode, prediction is formed from samples in the current frame that have previously encoded, decoded and reconstructed. In Inter mode, prediction is formed by motion-compensated prediction from one or more reference frame.

However, The for each macro block may be formed from one or two past or future frames in time order that have already been encoded and reconstructed. The prediction is subtracted from the current macro block to produce a residual or difference macro block. This

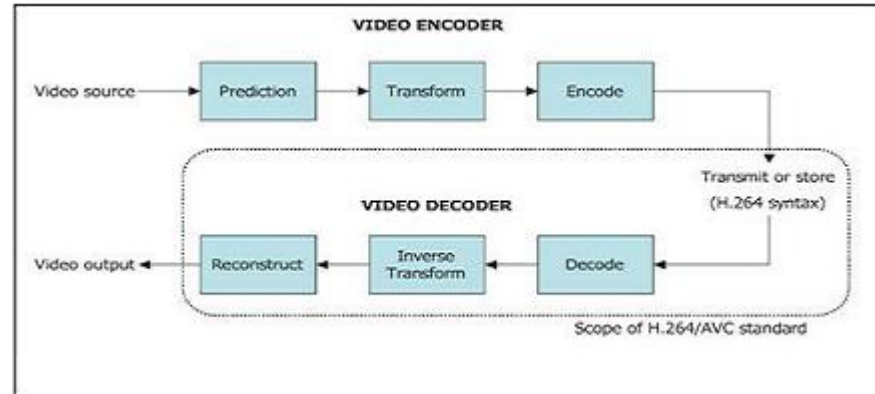


Figure 2.1: Encoder and Decoder for H.264[4]

is transformed using a block transform and quantized to give , a set of quantized transform coefficients. These coefficients are re-ordered and entropy encoded.

The entropy encoded coefficients, together with side information required to decode the macro block , quantizer step size, motion vector information describing how macro block was[5] motion-compensated, etc form the compressed bitstream. This is passed to a Network Abstraction Layer NAL for transmission or storage.[3]

### 2.1.2 H.264 Decoder

The decoder receives a compressed bitstream from the NAL. The data elements are entropy decoded and reordered to produce a set of quantized coefficients . These are rescaled and inverse transformed to give . Using the header information decoded from the bitstream, the decoder creates a prediction macro block , identical to the original prediction formed in the encoder. Prediction is added to inverse transform to produce reconstructed macro block which this is filtered to create the decoded.

The purpose of the reconstruction path in the encoder is to ensure that both and use identical reference frames to create the prediction . If this is not the case, then the predictions in encoder and decoder will not be identical, leading to an increasing error or drift between the encoder and decoder.



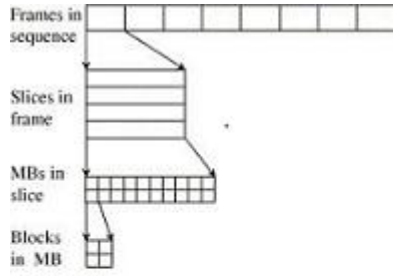


Figure 2.2: Hierarchy of Data Domain Decomposition in H.264[5]

## 2.2 Parallelization Of H.264

In this section we answer these questions for video coding/decoding workloads by analyzing their parallel scalability. Multimedia applications remain important workloads in the future and video codecs are expected to be important benchmarks for all kind of systems.[11] In a data-level decomposition the data is divided into smaller parts[8] and each assigned to a different processor. Each processor runs the[3] same program but on different multiple data elements . In H.264 data decomposition can be applied at different levels of the data structure: Group-of-Pictures level, frame-level, slice-level, macro block-level, and block-level.

## 2.3 Overview Of Cluster Computing

A computer cluster is a group of linked computers, working together closely so that in many respects they form a single computer. The components of a cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve performance and availability over that of a single computer. Computing with a Beowulf cluster engages four distinct but interrelated areas of consideration.

- Hardware system structure.
- A Resource administration and management environment.
- Distributed programming libraries and tools.
- Parallel algorithms.

## 2.4 Conclusion

According to literature survey we got that we can parallelize the H.264 using the CABAC algorithm . Parallel computing easily implemented by using the Beowulf cluster.

## Chapter 3

# High Efficient Parallel Algorithm For H.264

### 3.1 Parallel Algorithm For H.264

This chapter proposes a highly efficient parallel algorithm for H.264 encoder, which is based on the analysis of data dependencies in H.264 encoder. In the algorithm, the video frames are partitioned into several MB regions, each of which consists of several adjoining columns of macro-blocks (MB), which could be encoded by one processor of a multi-processor system. While starting up the encoding process, the wave-front technique is adopted, and the processors begin encoding process orderly.[7] In the MBRP parallel algorithm, the quantity of data that needs to be exchanged between processors is small, and the loads in different processors are balanced. The algorithm could efficiently encode the video sequence without any influence on the compression ratio.

Compared with previous standards, H.264 developed by the Joint Video Team formed by ISO MPEG and ITU-T VCEG achieves up to 50 percent improvement in bit rate efficiency and more than 4 times of the computational complexity, due to many new features including quarter-pixel motion estimation (ME) with variable block sizes and multiple reference frames up to 16, intra-prediction, integer transformation based on discrete cosine transform (DCT), alternative entropy coding mode Context-based Adaptive variable Length Coding Context-Based Adaptive Binary Arithmetic Coding, in-loop de-blocking filter and so

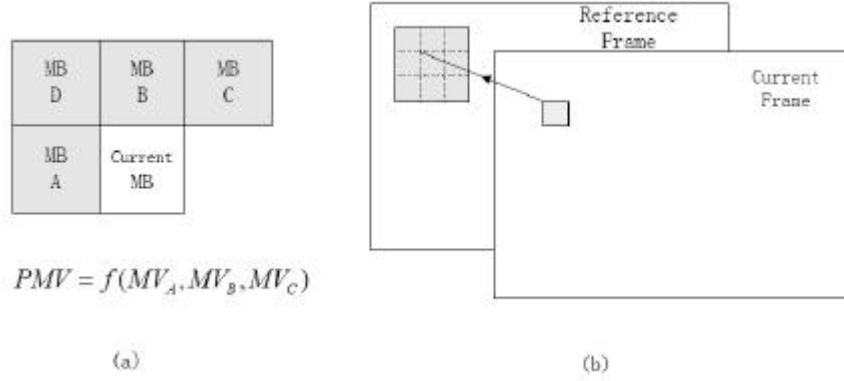


Figure 3.1: Data Dependencies Introduced By Inteprediction[5]

on . Therefore, the parallel structure and parallel algorithm are an alternative ways for real-time H.264 video application.

### 3.1.1 Data Dependencies In H.264

In the H.264 encoder, a MB is composed of (16\*16) luma pixels and 882 chroma pixels. Which results in several types of data dependencies that should be avoided in parallel algorithm.

### 3.1.2 Data Dependencies Introduced By Inter-prediction

In inter-prediction, the PMV defines the search center of ME, which comes from the motion vectors (MV) of the neighboring sub-blocks, A MV, B MV, C MV, and the corresponding reference indexes, as shown in Fig. Only the difference between the final optimal MV and the PMV will be encoded. Accordingly, the ME processes of the left, top, and top-right neighboring MBs should be finished before encoding the current MB.

## 3.2 CABAC Algorithm For H.264

A new algorithm for context modeling of binary sources with application to video[8] compression is presented. Our proposed method is based on a tree rearrangement and tree selection process for an optimized modeling of binary context trees. We demonstrate its use for adaptive context-based coding of selected syntax elements in a video coder. For that purpose we apply our proposed technique to the H.264/AVC standard and evaluate its performance for different sources and different quantization parameters[7].

CABAC method use some a priori gathered knowledge about the typical properties of the underlying source for a proper (fixed) selection and ordering of context parameters. However, in cases where this prior knowledge is not available or a higher degree of agreement between input source and context model is desired, the proposed GRASP method can be applied.

### 3.2.1 Context Modeling In CABAC

Context modeling in CABAC involves pre-defined sets  $T$  of past symbols, so-called context templates. For each symbol  $x$  to encode, in a first step the conditional probability  $p(x|T)$  is estimated by switching between different probability models according to the already coded neighboring symbols in  $T$ . Then, the estimated probability distribution  $p(x|T)$  is used to drive an adaptive binary arithmetic coding engine for the actual encoding of the symbol  $x$ . After encoding, the probability model is updated with the value of the encoded symbol  $x$ . Thus,  $p(x|T)$  is estimated on the fly by tracking the actual source statistics. However, estimating  $p(x|T)$  using past sample statistics may cause the problem of context dilution, if there are no appropriate limits on the symbol alphabet size or on the size of the context templates. In CABAC, this problem is avoided by using very simple context templates consisting of at most two neighbors (i.e.  $T = (y_0, y_1)$  in Fig. (left)) and by restricting the symbol alphabet to a binary alphabet. For non-binary valued symbols, CABAC provides appropriately defined binarization schemes.

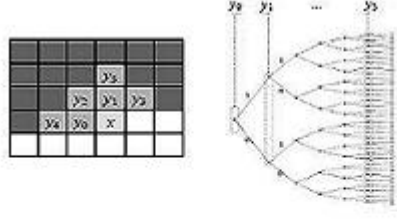


Figure 3.2: Relationship Between a Template of Context Parameters (left) and a Full Balanced Tree For Binary Data (right)[3]

### 3.2.2 Proposed Grasp Algorithm

#### Tree Growing By Reordering

In this section, we describe a method for exploiting given inter- symbol redundancy to a greater extent than it is usually achievable with an ad-hoc design of context models like e.g. that used in CABAC. Our proposed method is based on context trees. The basic motivation behind this approach is given by the observation that the finite memory model for a binary alphabet can be represented as a binary tree structure .

In this section, we describe a method for exploiting given intersymbol redundancy to a greater extent than it is usually achievable with an ad-hoc design of context models like e.g. that used in CABAC. let  $y_j^d$  denote the assignment of the context parameter  $y_j$  with  $(j \in 0, \dots, N-1)$  to a node of the tree at depth  $d$ . Let the set  $(y_j^0, y_j^1, \dots, y_j^d - 1, y_j^d - 1)$  denote the sequence of assignments of context parameters to the sequence of nodes over which the node at depth  $d$  is reached in the tree.

In CABAC, this sequence is fixed to  $(y_0^0, y_1^1)$  where the canonical order of context parameters in is used. Tree growing starts in Step 0 with gathering the population of each basic context, which correspond to the leaves of a regular full balanced tree. Any tree reordered or with regular order of context parameters can be reconstructed from these basic contexts. For  $N$  context parameters there are  $(2^N)$  basic contexts. The populated initial full balanced

tree is then grown using the following Steps 1 to 3.

**step0** Based on an initial order for the context parameters of a given context template ( $T = y_0, \dots, y_N - 1$ ), construct the basic contexts. Populate the corresponding nodes by gathering the given source statistics such that each basic context contains the occurrence counts of 0s and 1s.

**step 1** Start constructing a new, reordered tree by associating the empty subsequence with the root node at depth 0.

**step 2** Determine the assignment of a context parameter( $y_j$ ) to the current node at depth  $d \leq N$ . Let the subsequence  $y_j^0, y_j^1, \dots, y_j^{d-1}$  be the sequence of assignments to reach the current node at depth  $d$  from the root node the assignment ( $y_j^d$ ) has to be determined.

$$j_d = \operatorname{argmin}_j (L_j) j \in (0, \dots, N-1) (j_0, \dots, j_{d-1}) \quad (3.1)$$

with  $(L_j)$  denoting the adaptive code length to encode symbols  $x$  from nodes at level  $d+1$  given as

$$L_j = \sum_{i=0}^1 (c_0^{z_{ij}}, c_1^{z_{ij}}, k_0^{z_{ij}}, k_1^{z_{ij}}) \text{ where} \quad (3.2)$$

$$l(c_0, c_1, k_0, k_1) = -\log_2 \left( \frac{(c_0-1)!(c_1-1)!(k_0+k_1-1)!}{(c_0+c_1-1)!(k_0-1)!(k_1-1)!} \right) \quad (3.3)$$

$z_{ij} = y_j^0, y_j^1, \dots, y_j^{d-1}$  is the subsequence of assignment to reach the node at depth  $d+1$  and initial counts of 0's and 1's at that node, respectively

**step 3** Repeat Step 2 recursively until the maximum depth  $N$  is reached.

### Tree Selection By Pruning

The structure of the chosen tree model has to be transmitted and the gain in information per context parameter in one branch of a context tree cannot always compensate the increased side information when the number of nodes grows, an appropriate method for selecting the best subset of nodes is used. Thus, in order to reduce the dimensionality of the context tree, a tree selection by pruning is carried out to choose the best performing subtree as follows:

**step0** To each node of the full balanced tree, as constructed in the tree growing by reordering stage, assign the code length.

$$l_s = l(c_0^s, c_1^s, k_o^s, k_1^s) \quad (3.4)$$

where  $(c_0^s, c_1^s, k_o^s, k_1^s)$  are the occurrence and initial counts at that node. Set the cost functional  $J$  of each terminal node to its code length  $j_{(s)} = l(s)$  Note that evaluation of nodes starts with the terminal nodes of the reordered full balanced tree, i.e., at depth  $N - 1$ .

**step1** Compare the code length of the current node  $s$  with the sum of the cost functional  $J$  evaluated at the two child nodes  $sch0$  and  $sch1$ . If the sum of the costs of the children is greater than or equal to the adaptive code length at the parent node, prune the branch below the parent node otherwise, assign the sum of the childrens cost plus a model cost term  $mc$  to the parent node.

**step2** Repeat Step 1 recursively until the root is reached.



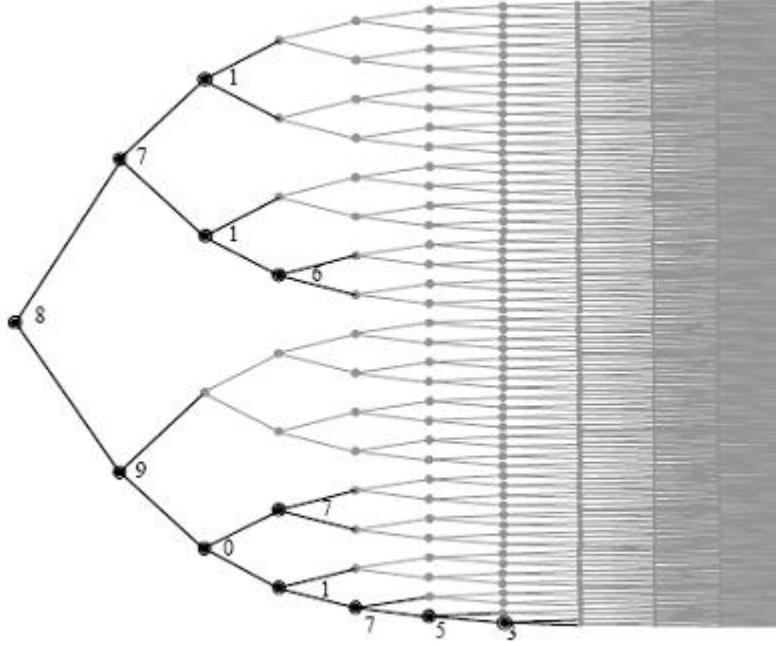


Figure 3.3: Example Of Pruned Tree Over Reordered Tree Each Internal Final Tree Is Labeled By Its Associate Context Parameter Index (right)[3]

### 3.2.3 Grasp For Group Of Picture

Grasp Algorithm results in the design of locally optimal trees for a single coding unit. Usually, frames of the same type (P or B) within one GOP have similar statistical properties which may result in similar context tree models and further in bit-rate reduction. Thus, we extend the GRASP algorithm to design a single optimal tree for a given syntax element of each slice type (I, P or B) within a GOP. For that, the basic GRASP algorithm is slightly changed.

First, the population of each basic context prior to the tree growing process is gathered by using all frames of a given type within a GOP. Then the tree is grown by choosing the best context parameter for splitting each node together with selecting the best initial state for that node. For the transmission of the corresponding initial count, a fixed length indicator of 6 bits is used to indicate the choice of the best fit to our pre-defined state table of 64 model probability states.

## Chapter 4

# Related Work

### 4.1 Example Of Distributed Encoding (Media Encoding Cluster )

#### 4.1.1 What Is Media Encoding Cluster

Media Encoding Cluster is the first Open Source Cluster Encoding Solution that is written in C/C++ for distributed Media(Video and Audio) Encoding. It distributes Video Chunks over Network to Client Nodes and parallelize the Encoding Task for one File over even more than one Computer to reduce the Encoding Time per File. What it does?

- Collect and Ingest our Multimedia Content.
- Management of our Content.
- Encode and Transport Our Content.
- Distribute our Content

#### 4.1.2 Features Of Media Encoding Cluster

**Scalability** Enable our network to grow with our business Media Encoding Cluster installations scale easily as clusters that can be managed from a single interface. Batch encode video sources, and automate encoding processes to achieve optimal efficiency.

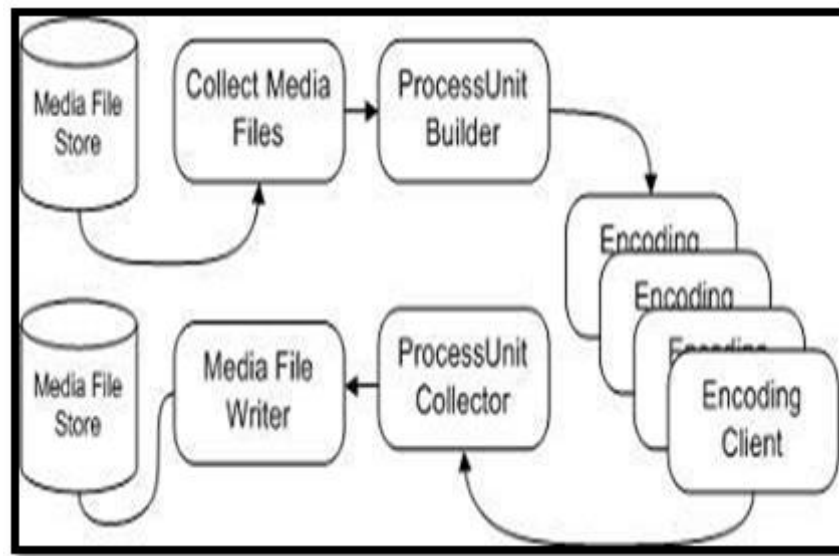


Figure 4.1: Media Encoding Processing system

**Broad Format Support** Broad format support ingest content in over 40 major formats giving a wide range of video sources to choose from when encoding content or processing user submissions.

**Video and audio filtering** Modify and enhance video and audio parameters as encode, normalize audio, adjust volume, adjust color balance, sharpen video, and apply many more filtering options.

**Efficient encoding** Convert more source files faster with this high-performance encoder.

**Flexible encoding and editing** Modify source file parameters on the fly as we encode, editing our content and making it suitable for delivery to multiple devices. Manipulate frame size and aspect ratio, crop for delivery to mobile devices, insert logos or watermarks, extract key frames, and much more.[5]

### 4.1.3 Output Of Media Encoding Cluster

This is consist of the output of Media Encoding Cluster for both client and server side.

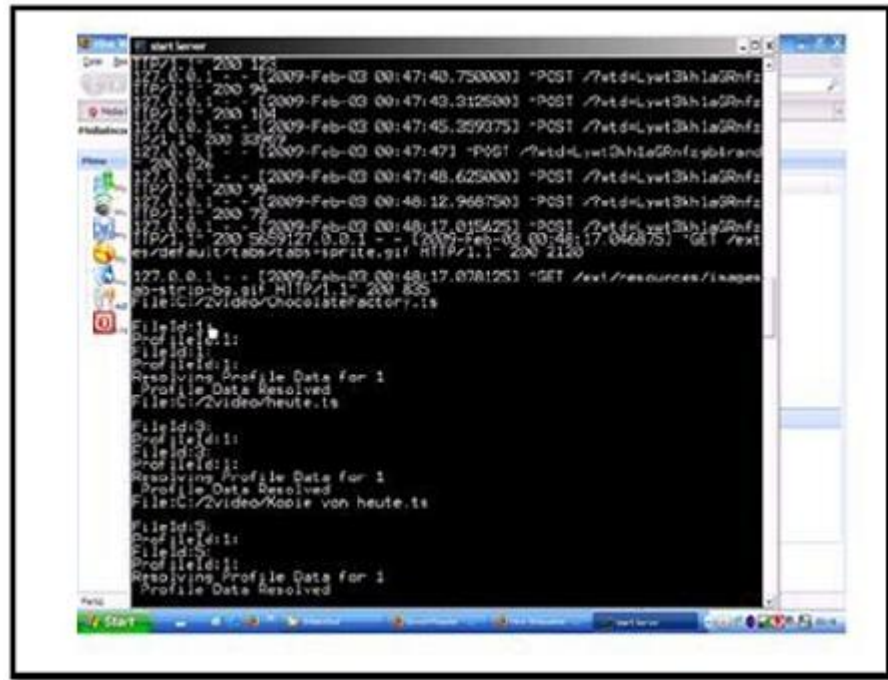


Figure 4.2: Output on client side

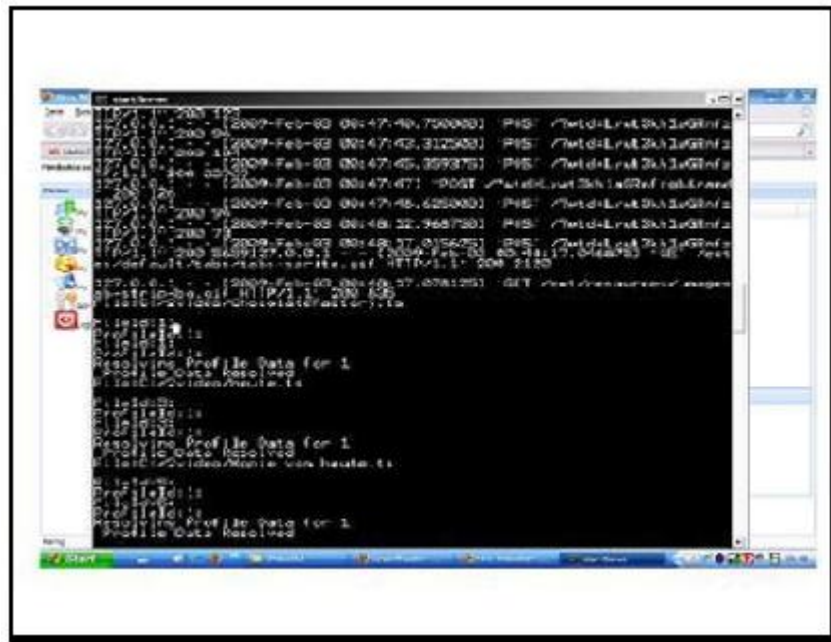


Figure 4.3: Output on Server side

## 4.2 Parallel Computing Using Erlang Distribution

### 4.2.1 What Is Erlang

Erlang is a programming language which has many features more commonly associated with an operating system than with a programming language: concurrent processes, scheduling, memory management, distribution, networking, etc. The initial open-source Erlang release contains the implementation of Erlang, as well as a large part of Ericsson's middleware for building distributed high-availability systems.

**Concurrency** Erlang has extremely lightweight processes whose memory requirements can vary dynamically. Processes have no shared memory and communicate by asynchronous message passing.

**Distribution** Erlang is designed to be run in a distributed environment. An Erlang virtual machine is called an Erlang node. A distributed Erlang system is a network of Erlang nodes (typically one per processor). An Erlang node can create parallel processes running on other nodes which perhaps use other operating systems. Processes residing

on different nodes communicate in exactly the same way as processes residing on the same node.

**Robustness** Erlang has various error detection primitives which can be used to structure fault-tolerant systems. For example, processes can monitor the status and activities of other processes, even if these processes are executing on other nodes. Processes in a distributed system can be configured to fail-over to other nodes in case of failures and automatically migrate back to recovered nodes.

**Soft real-time** Erlang supports programming "soft" real-time systems, which require response times in the order of milliseconds. Long garbage collection delays in such systems are unacceptable, so Erlang uses incremental garbage collection techniques.

**Hot code upgrade** Many systems cannot be stopped for software maintenance. Erlang allows program code to be changed in a running system. Old code can be phased out and replaced by new code. During the transition, both old code and new code can coexist. It is thus possible to install bug fixes and upgrades in a running system without disturbing its operation.

**Incremental code loading** Users can control in detail how code is loaded. In embedded systems, all code is usually loaded at boot time. In development systems, code is loaded when it is needed even when the system is running. If testing uncovers bugs, only the buggy code needs to be replaced.

**External interfaces** Erlang processes communicate with the outside world using the same message passing mechanism as used between Erlang processes. This mechanism is used for communication with the host operating system and for interaction with programs written in other languages. If required for reasons of efficiency, a special version of this concept allows e.g. C programs to be directly linked into the Erlang runtime system.

#### 4.2.2 Components

**Inets** HTTP 1.0 server and FTP client.

**Mnesia** Distributed real-time database for Erlang Supports RAM-replication as well as disk storage allows dynamic schema changes allows arbitrarily complex data structures to be stored. Mnesia is very fast since it runs in the same address space as the applications that use it.

**Orber** CORBA (v2.0) Object Request Broker (ORB).

**SNMP** Extensible SNMP v1/v2 agent and MIB compiler.

### 4.3 Tools And Libraries

**Appmon** Graphical monitoring of process groups locally and on remote nodes.

**ASN.1** Compile-time and runtime package which supports the ASN.1 . Basic Notation and the encoding rules BER, DER and PER .

**Compiler** Erlang Compiler.

**Debugger** Graphical Erlang debugger.

**ERTS** Erlang runtime system, including the virtual machine, the garbage collector, and the port mapper daemon.

**GS** Library for writing graphical user interfaces.

**IC** Compiler from OMG's Interface Definition Language (IDL) to Erlang and C and Java.

**Kernel** C code necessary to run the Erlang system. Erlang built-in functions (BIFs) code, boot and name servers networking and distribution support. Loaders, linkers and loggers, OS and file system interfaces.

**Mnemosyne** Optional query language for Mnesia.

**Mnesia Session** Foreign languages interface to Mnesia defined in IDL providing Mnesia access via the IOP and erlang interface protocols.

**OS monitor** Monitoring of CPU, disk and memory utilization including SNMP (v1/v2) MIBs. Interfaces to Solaris syslogd and Windows NT event log.

**Parse tools** LALR 1 parser generator for Erlang (yecc) similar to yacc. Yecc takes a BNF grammar definition as input and produces Erlang code for a parser as output. Yecc is used to generate the Erlang parser .

**PMan** Tool for tracing and viewing the state of Erlang processes ,locally or on remote nodes.

**SASL** Progress,error,crash report handling, report browsing release handling, overload regulation.

**Table visualizer** Tool for viewing ETS and Mnesia tables.

**Tool Bar** Simplifies access to the Erlang Tools.

**Tools** Coverage analyser profiler, text-based tracer Emacs mode, Emacs TAGS file generator, make utility, call graph utility.



### 4.3.1 Example Of Erlang Program

ERLANG programming language is use just run code from the shell. So here is a small Erlang program. Enter it into a file called tut.erl

```
-module(tut). -export([double/1]). double(X)-2 * X.s
```



Figure 4.4: Output of Erlang Program

## 4.4 Analysis Of Related Work

Firstly, We worked parallel computing using erlang distribution that was using ASN1 , IDL Compiler with Erllide for graphical environment . But in result we got complexity .

We worked in media encoding cluster for C and Erlang language after that we got , parallel computing easily implemented by the C language and complexity is reduced by using the BEOWULF cluster.

## Chapter 5

# Implementation Details Of H.264

### 5.1 Data Domain Decomposition Of H.264

We Know Video Sequence is divided into many Independent units of Group Of picture(GOP) and then they are processed by different nodes synchronously. At last connect the processed bit -stream correctly accordingly to the original bit stream structure. From the result, We can find that it is exactly the same that the data of Video quality and encoding rate[8]. Which are obtained from the serial algorithm and this parallel it is possible to the encoding process consider by on the system architechure of cluster.

A Video Consist Of each group of picture include each frame divided into, which is the self content encoding unit and independent of other slices in the same frame, which are the unit of motion and entropy coding.

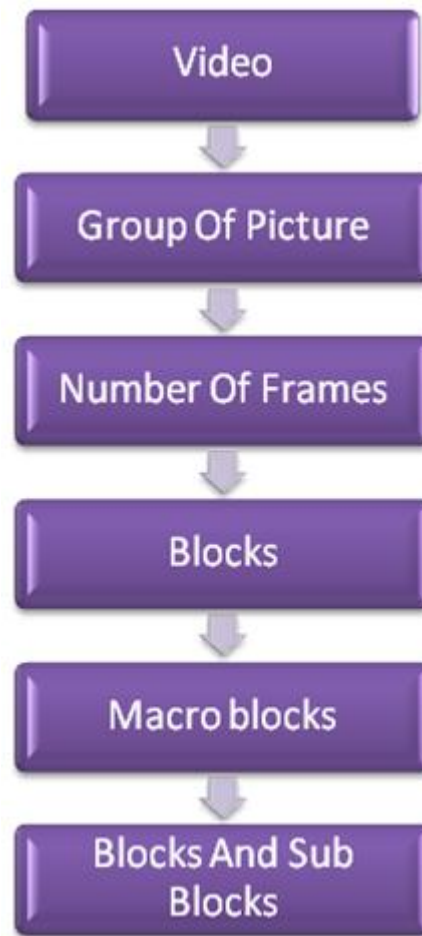


Figure 5.1: Data Domain Decomposition Of H.264[9]

## 5.2 Skeleton Of H.264 Reference Code

The Implementation presented here only handles a minimal feature set. This means that the following features are implemented[6]

- CABAC.
- Interlacing.
- Any kind of data partitioning.
- Any kind of arbitrary slice ordering (ASO) or flexible macroblock ordering (FMO).

- B and switching slices.
- Any kind of long-term prediction (i.e. only the last frame may be used as reference frame).
- In-Loop deblocking filtering.

### 5.2.1 Implementation Details Of H.264

H.264 is a block-based, motion-compensated video compression method. It is designed to be scalable, that is, its efficiency is roughly equally high for all purposes from low-bandwidth streaming up to high definition broadcast and storage.

### 5.2.2 NAL units, Slices, Fields And Frames

A H.264 video stream is organized in discrete packets, called NAL units (Network Abstraction Layer units). Each of these packets can contain a part of a slice, that is, there may be one or more NAL units per slice. But not all NAL units contain slice data, there are also NAL unit types for other purposes, such as signalling, headers and additional data. The slices, in turn, contain a part of a video frame. In normal bitstreams, each frame consists of a single slice whose data is stored in a single NAL unit. Nevertheless, the possibility to spread frames over an almost arbitrary number of NAL units can be useful if the stream is transmitted over an error-prone medium. The encoder may re synchronize after each NAL unit instead of skipping a whole frame if a single error occurs. H.264 also supports optional interlaced encoding. In this encoding mode, a frame is split into two fields. Fields may be encoded using spacial or temporal interleaving. To encode color images, H.264 uses the YCbCr color space like its predecessors, separating the image into luminance (or luma, brightness) and chrominance (or chroma, color) planes. It is, however, fixed at 4:2:0 subsampling, i.e. the chroma channels each have half the resolution of the luma channel.

### 5.2.3 Slice Types

H.264 defines five different slice types: I, P, B, SI and SP.

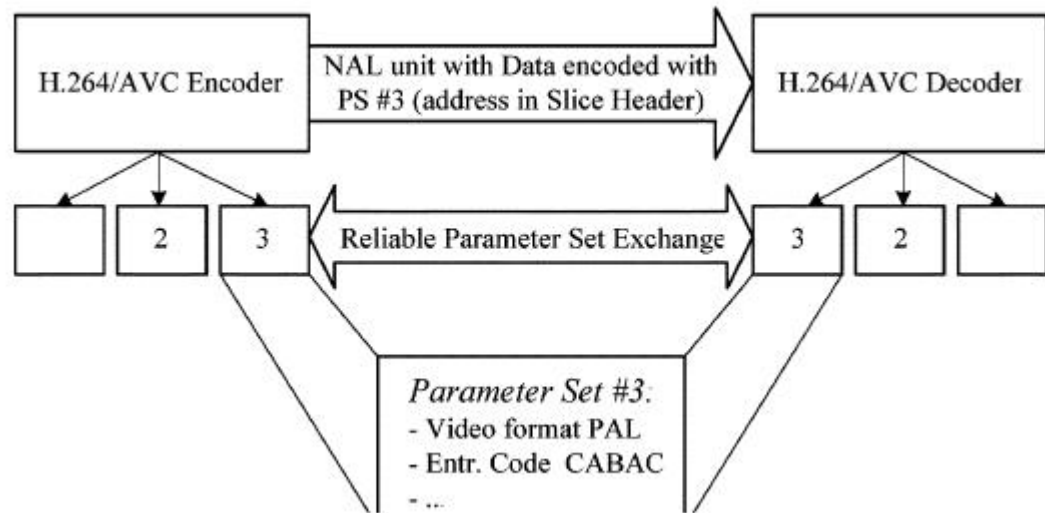


Figure 5.2: Parameter Set Use With Reliable "Out Of Band" parameter set exchange[5]

- I slices or Intra slices describe a full still image, containing only references to itself. A video stream may consist only of I slices, but this is typically not used. However, the first frame of a sequence always needs to be built out of I slices.
- P slices or Predicted slices use one or more recently encoded slices as a reference (or prediction) for picture construction. The prediction is usually not exactly the same as the actual picture content, so a residual may be added.
- B slices or Bi-Directional Predicted slices work like P slices with the exception that former and future I or P slices (in playback order) may be used as reference pictures. For this to work, B slices must be encoded after the following I or P slice.
- SI and SP slices or Switching slices may be used for transitions between two different H.264 video streams. This is a very uncommon feature.

#### 5.2.4 Motion Compensation

Since MPEG-1, motion compensation is a standard coding tool for video compression. Using motion compensation, motion between frames can be encoded in a very efficient manner. A typical P-type block copies an area of the last decoded frame into the current frame

buffer to serve as a prediction. If this block is assigned a nonzero motion vector, the source area for this copy process will not be the same as the destination area. It will be moved by some pixels, allowing to accomodate for the motion of the object that occupies that block. Motion vectors need not be integer values: In H.264, motion vector precision is one-quarter pixel (one eighth pixel in chroma). Interpolation is used to determine the intensity values at non-integer pixel positions. Additionally, motion vectors may point to regions outside of the image. In this case, edge pixels are repeated.

### 5.2.5 Motion Vector Prediction

Because adjacent blocks tend to move in the same directions, the motion vectors are also encoded using prediction. When a blocks motion vector is encoded, the surrounding blocks motion vectors are used to estimate the current motion vector. Then, only the difference between this prediction and the actual vector is stored.[4]

### 5.2.6 Macroblock Layer

Each slice consists of macroblocks (or, when using interlaced encoding, macroblock pairs) of 16x16 pixels. The encoder may choose between a multitude of encoding modes for each macroblock.

### 5.2.7 Macroblock Modes For I Slice

In H.264, I slices also use a prediction/residual scheme: Already decoded macroblocks of the same frame may be used as references for this so-called intra prediction process. The macroblock mode indicates which of two possible prediction types is used.

- Intra 16x16 uses one intra prediction scheme for the whole macroblock. Pixels may be filled from surrounding macroblocks at the left and the upper edge using one of four possible prediction modes. Intra prediction is also performed for the chroma planes using the same range of prediction modes. However, different modes may be selected for luma and chroma.
- Intra 4x4 subdivides the macroblock into 16 subblocks and assigns one of nine prediction modes to each of these 4x4 blocks. The prediction modes offered in Intra4x4

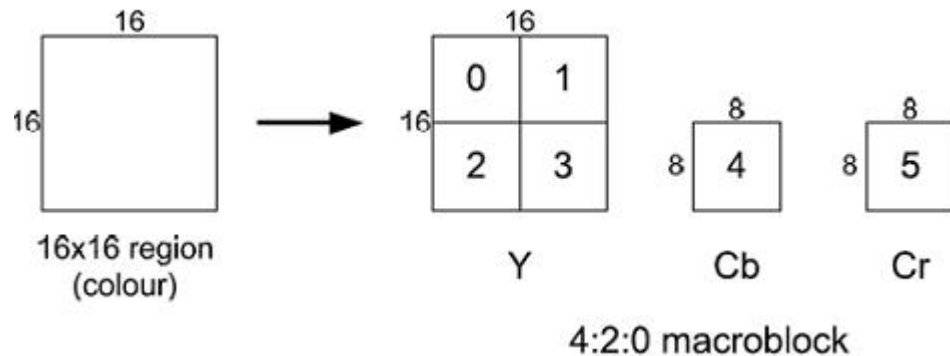


Figure 5.3: Macroblock Partition For H.264[8]

blocks support gradients or other smooth structures that run in one of eight distinct directions. One additional mode fills a whole subblock with a single value and is used if no other mode fits the actual pixel data inside a block.

### 5.2.8 Macroblock Modes For P And B Slices

P and B slices use another range of macroblock modes, but the encoder may use intra coded macroblocks in P or B slices as well. In regions of uniform motion without texture changes, macroblocks may be skipped. In this case, no further data is stored for the macroblock, but it is motion-compensated using the predicted motion vector. Otherwise, the macroblock of 16x16 pixels is divided into macroblock partitions.

- It may be stored as a single partition of 16x16 pixels.
- It may be split horizontally into two partitions of 16x8 pixels each.
- It may be split vertically into two partitions of 8x16 pixels each.
- It may be split in both directions, resulting in four sub-macroblock partitions of 8x8 pixels. Each of these may be split similarly into partitions of 8x8, 8x4, 4x8 or 4x4 pixels. Not all sub macroblock partitions need to be split in the same manner, thus allowing for any number of partitions from 1 to 16.

### 5.2.9 Block Transformation And Encoding

The basic image encoding algorithm of H.264 uses a separable transformation. The mode of operation is similar to that of JPEG and MPEG, but the transformation used is not an 8x8 DCT, but an 4x4 integer transformation derived from the DCT. This transformation is very simple and fast. It can be computed using only additions/subtractions and binary shifts. It decomposes the image into its spacial frequency components like the DCT, but due to its smaller size, it is not as prone to high frequency mosquito artifacts as its predecessors.

An Image block B is transformed to B0 using the following formula. The necessary post-scaling step is integrated into quantization (see below) and therefore omitted.

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} B' = MBM^T \quad (5.1)$$

### 5.2.10 Entropy Coding Modes

H.264 supports two different methods for the final entropy encoding step: CAVLC, or Context-Adaptive Variable Length Coding, is the standard method using simple variable length huffmann-like codes and codebooks. CABAC, or Context-Adaptive Binary Arithmetic Coding, on the other side, is an optional, highly efficient binary encoding scheme.

### 5.2.11 DC Transformation

The upper left transform coefficient (i.e. the first coefficient in scan order) is treated separately for Intra 16x16 Macroblocks and chroma residuals. Because this coefficient indicates the average intensity value of a block, correlations between the DC values of adjacent equally predicted blocks can be exploited this way. For Intra 16x16 Macroblocks, the DC coefficients



are transformed using a separable 4x4 Hadamard transform with the following matrix.

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \quad (5.2)$$

Chroma residuals are always transformed in one group per Macroblock. Thus, there are 4 chroma blocks per Macroblock and channel. A separable 2x2 transform is used.

$$M = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (5.3)$$

#### 5.2.12 In Loop Deblocking Filter

After each individual frame has been decoded, a de blocking filter that reduces the most visible blocking artifacts is applied. This has been available since MPEG-4 Simple Profile as an optional post-processing operation, but in H.264, it is closely integrated into the encoding and decoding process. The already deblocked frames are used as reference frames by the following P or B slices. This technique circumvents noticeable blocking artifacts as far as possible.

#### 5.2.13 Advanced Prediction

H.264 may not only use interframe references to the last decoded frame, but to a arbitrary number of frames. This greatly improves the encoding efficiency of periodic movements prediction. Moreover, multiple predictions may be mixed by arbitrary ratios.

#### 5.2.14 Arbitrary Slice Ordering

Since the slices of a picture can be decoded independently, slices need not be decoded in the correct order to render an image in acceptable quality. This is useful e.g. for UDP streaming where packets may be delivered out-of-order.

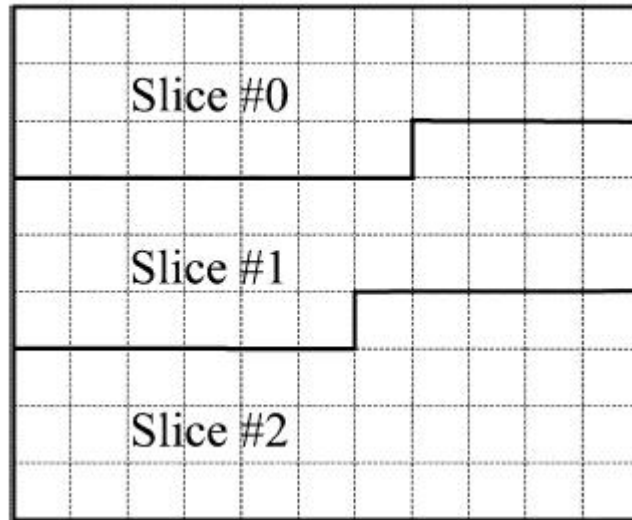


Figure 5.4: Subdivision Of Picture Into Slices(When Not Using FMO)[7]

### 5.2.15 Flexible MacroBlock Ordering

The Macroblocks inside a slice may be encoded in any order. This can be used to increase robustness against transmission errors, for example. It is also reminiscent of MPEG-4s video object planes system which could be used to encode each object of a scene individually. In normal video streams, however, this feature is not used, and macroblocks are sent in the normal scanline order.

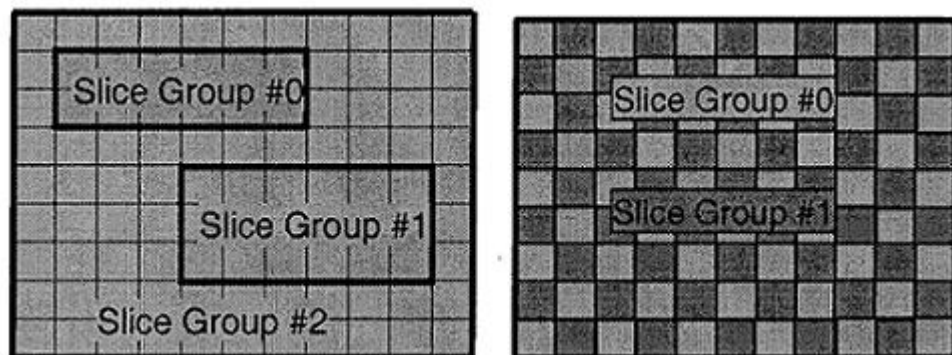


Figure 5.5: Subdivision Of A QCIF Frame Into Slices When Utilizing FMO.[7]

### 5.3 CABAC Encoding In H.264

The CABAC Encoder reads in bit stream and outputs meaningful information. The H.264/AVC standard defines this meaningful information as syntax element (SE). It also defines three variables: `codlOffset`, `codlRange` and MPS (the most probable symbol). The CABAC Encoder determines the value of output bit according to the values of `codlOffset` and `codlRange`. For example, when `codlRange` is larger than `codlOffset`, the value of output bit will be equivalent to that of MPS.

The H.264/AVC standard defines for the CABAC Encoder five tables: context table, initial table, `qCodIRangeIdx` table, `transIdxLPS` table, and `transIdxMPS` table. The context table is constructed from the initial table and is indexed by the variable context. Each entry of the context table contains two variables: `pStateIdx` and MPS. The bigger the value of `pStateIdx` is, the more probable the output bit equals to MPS. During the encoding process, we update `codlRange` by looking up the `qCodIRangeIdx` table and update the corresponding entry in the context table by looking up the `transIdxLPS` table and the `transIdxMPS` table. The flow chart of CABAC encoding is shown in Figure.[9] At the beginning of a new slice, we build the context table from the initial table. Then we initialize `codlOffset` and `codlRange` by using the first 2 bytes of the bit stream. After initializing the context table, `codlOffset` and `codlRange`, we go on to Encode one macroblock.

In the macroblock layer, the CABAC Encoder should first decide which syntax element (SE) to be Encoded Secondly the Encoder calculates context by referring to the syntax elements of the left, top, or current macroblock. The Encoding process is now divided into three stages: normal encoding process, bypass Encoding process and terminal Encoding process.



Figure 5.6: Skeleton Of H.264 Reference Code

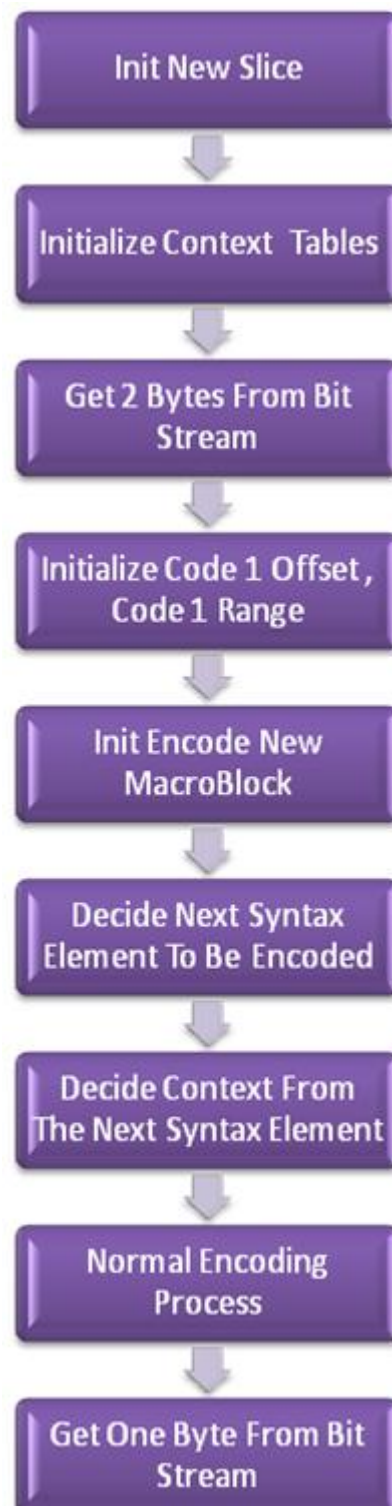


Figure 5.7: FlowChart Of CABAC Encoding Process

## Chapter 6

# Implementation Of Codec And Simulation Result

### 6.1 Parallelization Of The H.264 Encoder Using C Code

Here, the main() of H.264 Encoder is described by the following function

```
int main (int argc ,char ** argv)
```

#### 6.1.1 Function Documentation

- static void alloc encoder( EncoderParams \*\* p Enc )  
It allocate encoder structure.
- static void alloc img ( ImageParameters \*\* p Img ) [static]  
It allocate the Image structure
- static void alloc params ( InputParameters \*\* p Inp ) [static]  
It allocate the Input structure.
- static void chroma mc setup ( ImageParameters \* p Img ) [static]  
It setup Chroma MC Variables.
- static int determine coding level ( ImageParameters \* p Img, InputParameters \* p Inp, int curr frame)  
Determine coding level a frame belongs to.

- void encode one redundant frame ( ImageParameters \* p Img, InputParameters \* p Inp)  
It encode one redundant frame.
- static void encode sequence ( ImageParameters \* pImg, InputParameters \* p Inp)  
It Encode a sequence.
- static void free encoder ( EncoderParams \* p Enc ) [static]  
Free memory allocated for the encoder.
- void free encoder memory ( ImageParameters \* p Img, InputParameters \* p Inp)  
Free allocated memory of frame size related global buffers buffers are defined in global.h, allocated memory is allocated in int get mem4global buffers().
- static void free global buffers ( ImageParameters \* p p Img, InputParameters \* p Inp )  
Free allocated memory of frame size related global buffers buffers are defined in global.h, allocated memory is allocated in int get mem4global buffers().
- static void free img ( ImageParameters \* p Img, InputParameters \* p Inp )  
Free the Image structures.
- void free mem ACcoeff ( int \*\*\*\*\* cofAC )  
Free memory of AC coefficients.
- void free mem ACcoeff new ( int \*\*\*\*\* cofAC )  
Free memory of AC coefficients.
- void free mem DCcoeff ( int \*\*\* cofDC )  
Free memory of DC coefficients.
- void free orig planes ( ImageParameters \* p Img, InputParameters \* p Inp, ImageData \* imgData )  
Free allocated memory of original picture buffers.
- static void free params ( InputParameters \* p Inp )

- void free picture ( Picture \* pic )  
Free the Input structures.
- int get mem ACcoeff ( ImageParameters \* p Img, int \*\*\*\*\* cofAC )  
Allocate memory for AC coefficients.
- int get mem ACcoeff new ( int \*\*\*\*\* cofAC, int chroma )  
Allocate memory for AC coefficients.
- static void init encoder ( ImageParameters \* p Img, InputParameters \* p Inp )  
It Initialize encoder.
- static int init global buffers ( ImageParameters \* p Img, InputParameters \* p Inp )  
Dynamic memory allocation of frame size related global buffers buffers are defined in global.h, allocated memory must be freed in void free global buffers().
- static void init img ( ImageParameters \* p Img, InputParameters \* p Inp )  
Initializes the Image structure with appropriate parameters.
- int init orig buffers ( ImageParameters \* p Img, InputParameters \* p Inp, ImageData \* imgData )  
Memory allocation for original picture buffers
- static void init poc ( ImageParameters \* p Img, InputParameters \* p Inp )
- void Init redundant frame ( ImageParameters \* p Img, InputParameters \* p Inp )  
Initializes the POC structure with appropriate parameters.
- int main ( int argc, char \*\* argv )
- Picture\* malloc picture ( )  
It initialize key frames and corresponding redundant frames
- static void prepare first layer ( ImageParameters \* p Img, InputParameters \* p Inp, int curr frame to code)  
Prepare first coding layer



- static void prepare second layer ( ImageParameters \* p Img, InputParameters \* p Inp, int enh frame to code)  
Prepare second coding layer.
- void Set redundant frame ( ImageParameters \* p Img, InputParameters \* p Inp )  
It allocate redundant frames in a primary GOP.
- static void SetImgType ( ImageParameters \* p Img, InputParameters \* p Inp, int gop frame num )  
Set the image type for I,P and SP pictures (not B!).
- static void SetLevelIndices ( ImageParameters \* p Img )  
Sets indices to appropriate level constraints, depending on current level idc.

## 6.2 Parallelization Of The H.264 Decoder Using C Code

Here, the main() of H.264 Decoder is described by the following function

```
int main (int argc, char ** argv )
```

### 6.2.1 Function Documentation

- static void alloc decoder ( DecoderParams \*\* p Dec ) [static]  
Allocate the Decoder Structure
- static void alloc img ( ImageParameters \*\* p Img ) [static]  
Allocate the Image structure.
- static void alloc params ( InputParameters \*\* p Inp ) [static]  
Allocate the Input structure.
- DataPartition\* AllocPartition ( int n )  
Allocates a stand-alone partition structure. Structure should be freed by FreePartition(); data structures.
- static void conf read check ( int val, int expected )  
exit with error message if reading from config file failed.

- `void Configure ( ImageParameters * pImg, InputParameters * p Inp, int ac, char * av )`  
set default bitstream name, set default output file name, set default reference file name.
- `void error ( char * text, int code)`  
Error handling procedure. Print error message to stderr and exit with supplied code.
- `void free global buffers ( ImageParameters * pImg )`  
Free allocated memory of frame size related global buffers buffers are defined in global.h, allocated memory is allocated in `int init global buffers()`.
- `static void free img ( ImageParameters * p Img ) [static]`  
Free the Image structure.
- `static void free slice ( Slice * currSlice ) [static]`  
Memory frees of the Slice structure and of its dependent data structures.
- `void FreePartition ( DataPartition * dp, int n )`  
Frees a partition structure (array).
- `static void init ( ImageParameters * p Img ) [static]`  
Initilize some arrays.
- `static void init conf ( ImageParameters * p Img, InputParameters * p Inp, char * config filename )`  
Read parameters from configuration file.
- `void init frext ( ImageParameters * p Img )`  
Initialize FREXT variables.
- `int init global buffers ( ImageParameters * pImg )`  
Dynamic memory allocation of frame size related global buffers buffers are defined in global.h, allocated memory must be freed in `void free global buffers()`.
- `void JMDecHelpExit( void )`
- `int main ( int argc, char ** argv )`  
main function for decoder.

- static void malloc slice ( InputParameters \* p Inp, ImageParameters \* p Img ) [static]  
Allocates the slice structure along with its dependent data structures.
- static void Report ( ImageParameters \* p Img ) [static]  
Reports the gathered information to appropriate outputs.

Input YUV file	:	foreman_part_qcif.yuv				
Output H.264 bitstream	:	test.264				
Output YUV file	:	test_rec.yuv				
YUV Format	:	YUV 4:2:0				
Frames to be encoded I-P/B	:	2/1				
PicInterlace / MbInterlace	:	0/0				
Transform8x8Mode	:	1				
<hr/>						
Frame	Bit/pic	QP	SnrY	SnrU	SnrV	Time(ns)
<hr/>						
1000(NUE)	176					
1000(IDF)	20544	28	37.442	41.308	43.113	672
SAD(standard filter) = 1367414						
SAD(adaptive filter) = 1362353						

Figure 6.1: Output OF Encoder For H.264

### 6.3 Output Of Encoder And Simulation Result

Firstly We Parallelize the Code For H.264 Encoder Using The CABAC Algorithm. Then We Got The Output Of Encoder.

### 6.3.1 Simulation Result

The simulator of our MBRP parallel algorithm for H.264 encoder is developed using C language and implemented on a PC with a P4 1.7GHz processor and a 512MB memory. The simulation results are compared with , which is a sequential encoding structure. In our software simulation of H.264 encoder, processors are simulated. The main encoder parameters are shown in Table . The simulator collects the maximal encoding time among concurrently processed MB regions and the corresponding time spent on data exchanging. Some of the simulation results are presented in Table. we used "Foreman as video source, and frames were encoded.

Table I: Simulation Result Simulation Result Of Encoder	
QP	28
Snr Y	37.442
Snr U	41.308
Snr V	43.113
Time(ms)	672

### 6.3.2 Performance And Analysis

We Can measure the performance of the H.264 Encoder And Decoder By using the following points.

**Quantization Parameter** A quantization parameter (QP) is used for determining the quantization of transform coefficients in H.264/AVC. It can take on 52 values. The quantization step size is controlled logarithmically by QP rather than linearly as in previous standards, in a manner designed to reduce decoding complexity and enhance bit rate control capability. Each increase of six in QP causes a doubling of the quantization step size, so each increase of one in QP increases the step size by approximately 12 percent.

**Distortion Measures** Rate Distortion Optimization Requires an ability to measure the distortion. However The Perceived Distortion in visual content is very difficult quantity to measure as the characteristics of the human visual system. This problem is

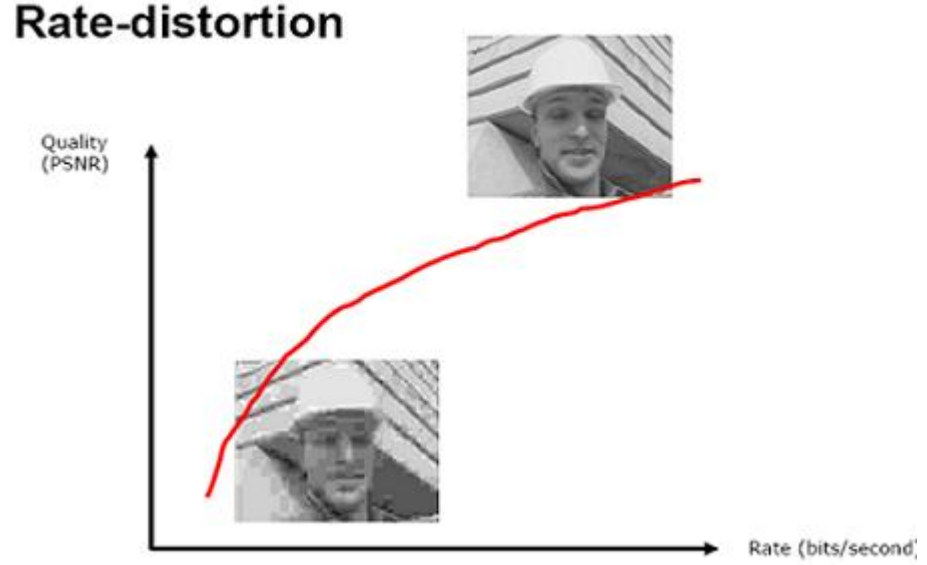


Figure 6.2: Performance According To Rate Distortion

aggravated in video coding because the addition of the temporal domain relative to still picture coding further complicates.

In practice highly imperfect distortion models such as the sum of the squared differences(SSD) or its equivalent, known as mean squared error(MSE) , or Peak signal to noise ratio (PSNR)are used in motion actual comparison, they are defined as.

$$SSD_A(F, G) = |F(S) - G(S)|^2 \quad (6.1)$$

$$MSE_A(F, G) = 1/|A| \quad SSD_A(F, G) \quad (6.2)$$

$$PSNR_A(F, G) = 10 \log_{10} (255)^2 / MSE_A(F, G) \quad (6.3)$$

## 6.4 Conclusion

H.264/AVC represents a number of advances in standard video coding technology, in terms of coding efficiency improvement, error/loss robustness enhancement, and flexibility for effective use over a broad variety . Its VCL design is based on conventional block-based motion-compensated hybrid video coding concepts, but with some important differences relative to prior standards, which include:

- Enhanced Motion Prediction Capability.
- Use of a small block-size exact-match transform.
- Adaptive in-loop deblocking filter.
- Enhanced entropy coding methods.

## Chapter 7

# Parallel Computing Using The Beowulf Cluster

### 7.1 Why Beowulf Cluster?

The node is responsible for all activities and capabilities associated with executing an application program and supporting a sophisticated software environment. These fall into four general categories[2]

- Instruction execution.
- High-speed temporary information storage.
- High-capacity persistent information storage.
- Communication with the external environment, including other nodes.

### 7.2 Principal Specification

In selecting the proper node configuration for a new Beowulf. Fortunately, only a small number of critical parameters largely characterize a particular Beowulf node. These parameters usually relate to a few peak capabilities or capacities and are only roughly predictive of the performance of any given application.



**Processor Clock Rate** The frequency (MHz or GHz) of the primary signal within the processor that determines the rate at which instructions are issued.

**Peak floating-point performance** The combination of the clock rate and the number of floating-point operations that can be issued and retired per instruction (Mflops)

**Cache size** The storage capacity (KBytes) of the high-speed buffer memory between the main memory and the processor.

**Main memory capacity** The storage capacity (MBytes) of the primary system node memory in which resides the global dataset of the applications as well as myriad other supporting program, buffering, and argument data.

**Disk capacity** The storage capacity (GBytes) of the permanent secondary storage internal to the processing node.

**SAN network port peak bandwidth** The bandwidth (Mbps) of the network control card and system area network communication channel medium.

The node is responsible for performing a set of designated instruction specified by the application program code or system software. The lowest -level binary encoding of the instructions and the actions they perform are dictated by the microprocessor instruction set architecture (ISA).

### 7.3 Use Linux For Beowulf

The most important reason for using Linux to build a Beowulf is its flexibility. Because Linux is open source, it can easily be modified, rearranged, and tweaked for whatever the task. Linux is actually very friendly. Because of the distributed development environment that has helped it become so successful, it is also easily modified and tweaked. Companies such as IBM, Fujitsu, NEC, Compaq, and Dell have all incorporated Linux into their business model, creating a marketplace around a distribution of kernel source code that is free. Other companies are simply using Linux because it makes practical business sense. Another reason to choose Linux is its support for many types of processors. Alpha, PowerPC, IA32, IA64, and many others are all supported in Linux. We can choose to build our Beowulf from the fastest Apple Macintosh servers or IBM pSeries servers. As an example of the flexibility and speed .

The first reason that smaller is better comes from decades of experience with source code development and stability. Whenever a line of code is added to a source tree, the probability increases that a hidden bug has been introduced. For a kernel that controls the memory system and precious data on disk, robustness is vital. Having fewer functions running in privileged mode makes for a more stable environment. A small kernel is a kernel that is more likely to be stable. Linux was chosen for its stability, robustness, and the ease with which it could be modified for the task.

The second reason for a small kernel is that the most stable code path is the most used code path. It tends to congregate in out-of-the-way locations, away from the well-worn code paths. The smaller the kernel, the fewer the hidden and rarely tested code paths. Finally, smaller is better when it comes to kernel memory and CPU cycles on a Beowulf. Because kernel operations such as task switching are run extremely often, even a small amount of additional kernel overhead can noticeably impact application performance. For Beowulf, a small kernel is advantageous.

## 7.4 A Kernel And A Distribution

The term "Linux" is most correctly applied to the name for the Unix-like kernel, the heart of an operating system that directly controls the hardware and provides true multitasking, virtual memory, shared libraries, demand loading, shared copy-on-write executables, TCP/IP networking, and file systems. The term "Linux" has also been applied in a very general way to mean the entire system, the Linux kernel combined with all of the other programs that make the system easy to use, such as the graphic interface, the compiler tools, the e-mail programs, and the utilities for copying and naming files.

A Linux distribution packages up all the common programs and interfaces that most users think of when they imagine Linux, such as the desktop icons or the Apache Web server or, more important, for scientific users, compilers, performance monitoring tools. Many Linux distribution companies exist. In fact, companies Red Hat, Turbo linux, Caldera, SuSE, and a host of smaller companies, have the freedom to customize, optimize, support, and extend their Linux distribution to satisfy the needs of their users. How the Linux kernel and Linux distributions are developed and maintained is key to understanding how to get support and how to get a Beowulf cluster up and running on the network as quickly as possible.

## 7.5 Network Software

In this part we turn to the networking software options available to the Beowulf programmer, administrator, and user. Networking software is usually described as a stack, made up of different protocol layers that inter operate with one another. We survey a few of the layers in the networking stack, focusing on those services and tools that are used extensively on Beowulf system.

### 7.5.1 Tcp/Ip

Parallel computers have traditionally used special high-performance inter processor communication networks that use custom protocols to transmit data between processing elements. In contrast, Beowulf clusters rely on commodity networks whose original design goals did not include serving as the interconnect for a commodity supercomputer. The use

of commodity networks implies the use of commodity protocols when costs must be kept down. Growth of the Internet during the last decade of century, has become the de facto standard network communication protocol. The IP protocol is conceptually divided into different logical layers that combine to form a protocol stack. The IP layer is a routable datagram layer. Data to be transferred is fragmented into datagrams individual packets of data. Packet length is limited by the physical transport layer, and the IP layer contains the logic to fragment requests that are too large into multiple IP packets that are reassembled at the destination. Each datagram is individually routable and contains a four-byte IP address that specifies the destination host. This version of IP is called IPv4. A new version, called IPv6, will increase the address space available to IP applications. The four-byte addresses used in IPv4 are too small for the total number of computers currently connected to the world's networks. This address depletion will be remedied by IPv6, which uses 16 bytes to represent host addresses. Currently, however, IPv4 remains dominant, particularly in the United States. The IP stack commonly supports two services: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP, the most common IP service, provides a reliable, sequenced byte stream service.

While the underlying physical transport layer usually provides error checking, TCP provides its own final data integrity checking. Most multiple-hop physical transports provide only a best-effort delivery promise. TCP incorporates a positive acknowledgment sliding-window retransmission mechanism that recovers from packet loss. It also tolerates latency while maintaining high performance in the normal case .

### IP Addresses

The destination of an Internet Protocol packet is specified by a 32-bit (or 128 bits) for IPv6 that uniquely identifies the destination host. The portion of the address that remains fixed within a network is called the network address, and the remainder is the host address. The division between these two parts is specified by the netmask. A typical netmask is 255.255.255.0, which specifies 24 bits of network address and 8 bits of host addresses. Three IP address ranges have been reserved for private networks:

- 10.0.0.0 10.255.255.255

- 172.16.0.0 172.31.255.255
- 192.168.0.0 192.168.255.255

### Zero Copy Protocol

One way to improve network performance, especially for high-performance networks, is to eliminate unnecessary copying of data between buffers in the kernel or between the kernel and user space. So-called zero-copy protocols give applications nearly direct access to the network hardware, which copies data directly to and from buffers in the application program.

### 7.5.2 Socket

are the low-level interface between user-level programs and the operating system and hardware layers of a communication network. They provide a reasonably portable mechanism for two (or more) applications to communicate, and they support a variety of addressing formats, semantics, and underlying protocols.

On Linux, the socket API is supported directly by the operating system, but research projects have proposed lower-level zero-copy protocols that would allow applications more direct access to the kernel. The socket API is powerful but not particularly elegant. Many programmers avoid working with sockets directly, opting instead to hide the socket interface behind one or more additional layers of abstraction (e.g., remote procedure calls or a library like MPI).

### Client Tasks

The client has four basic tasks:

- Create a local socket with an otherwise unused address.
- Determine the address of the server.
- Establish a connection (TCP only).
- Send and receive data.

### Server Tasks

Servers are more complicated than clients. There are a number of different design choices for servers, with various tradeoffs between response time, scalability ,how many clients can be supported, resource consumption, programming style, and robustness. Popular choices include a multithreaded server, a server that forks a new process for every connection, or a server that is invoked by the Internet daemon `inetd`. A few tasks are common to all these design choices.[2]

- Create a local socket.
- Select a port number.
- Bind the port number to the socket.
- Make the port number known to clients.
- Listen for connections (TCP only).
- Accept connections (TCP only).
- Send And Receive data.

### 7.5.3 Remote Procedure Call

Programming with sockets is part of the client/server programming model, where all data exchange is explicitly performed with sends and receives. A (RPC) follows a different paradigm of distributed computation, removing the programmer from explicit message passing. The idea behind an RPC is to make distributed programs look like sequential programs. A procedure is called inside a program rather than executing on the local machine, however, the local program suspends while the procedure executes on a remote machine. When the procedure returns,the local program wakes up and receives any results that may have been produced by the procedure.

RPC was designed not so much for parallel programming as for distributed programming. Parallel programming is a more tightly coupled concept where a single program works on a problem, concurrently executing on multiple processors. Distributed programming is a looser concept where two or more programs may require services from one another

and therefore need to communicate, but they are not necessarily working on the same problem. RPC can be used effectively on Beowulf systems, especially for porting applications that are already designed .

Two different RPC implementations are commonly found on Unix systems. The first is ONC ,for Open Network Computing. This is the RPC standard used by Linux and Beowulf systems. Distributed Computing Environment. The two systems are incompatible and offer different features.

#### 7.5.4 Distributed File System

Every node in a Beowulf cluster equipped with a hard drive has a local file system that processes running on other nodes. The worldly node's file system that they may boot and execute programs. The need for internode file system access requires Beowulf clusters to adopt one or more distributed file systems.

#### 7.5.5 Secure Shell

or SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices. It used on Linux and Unix based systems to access shell accounts. It send information, notably passwords, in plaintext, leaving them open for interception. The encryption used by SSH provides confidentiality and integrity of data over an insecure network, such as the Internet .

#### 7.5.6 Mpich

Parallel computing system is a robust and flexible implementation of the MPI (Message Passing Interface). MPI is often used with parallel or distributed computing projects. MPICH is a multi-platform, configurable system (development, execution, libraries, etc) for MPI. It can achieve parallelism using networked machines or using multitasking on a single machines.

## Chapter 8

# Implementation Of Beowulf Cluster

### 8.1 Introduction

This chapter describes a step by step Instruction on building Beowulf cluster. We implement the Beowulf Cluster using the Fedora8 with two nodes that consist of Redhat Distribution.

### 8.2 Logical View Of Beowulf Cluster

Beowulf is a class of computer clusters similar to the original NASA system. Originally developed by Thomas Sterling and Donald Becker at NASA, Beowulf systems are now deployed worldwide, chiefly in support of scientific computing. It is a group of what are normally identical, commercially available computers, which are running a Free and Open Source Software (FOSS), Unix-like operating system, such as BSD, GNU/Linux, or Solaris. They are networked into a small TCP/IP and have libraries and programs installed which allow processing to be shared among them. There is no particular piece of software that defines a cluster as a Beowulf. Commonly used parallel processing libraries include Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). Both of these permit the programmer to divide a task among a group of networked computers, and collect the results of processing. Examples of MPI software include OpenMPI or MPICH .



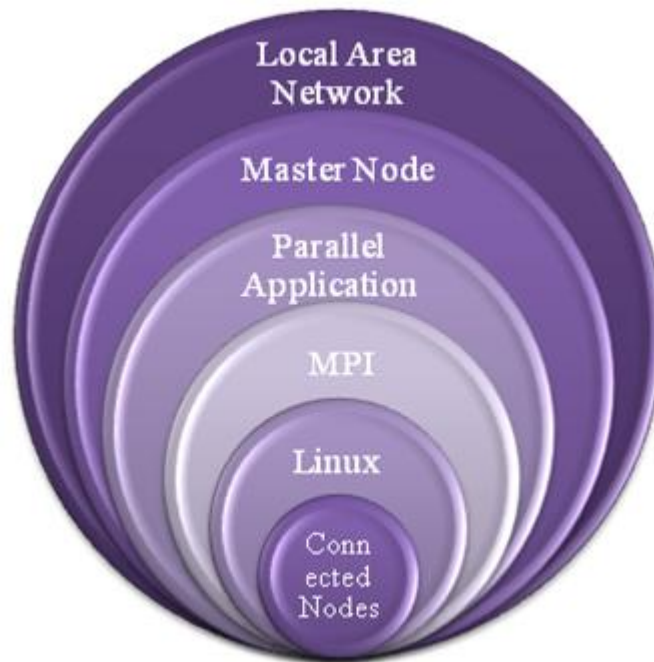


Figure 8.1: Logical View Of Beowulf Cluster

### 8.3 Requirements

The following figure specify the requirement of the Beowulf Cluster with setup the two nodes.

### 8.4 Setting Up Cluster Installation And Configuration

THE FOLLOWING STEPS FOR MASTER AND SLAVE BOTH ,WE ADD ANOTHER USER CALLED "ROOT"

- a. First step to get rsh and similar tools installed,for that we will do the following steps:
  - Make sure that xinetd is installed
  - If xinetd will not install in our pc ,it can be done by the following command.  
yum install xinetd

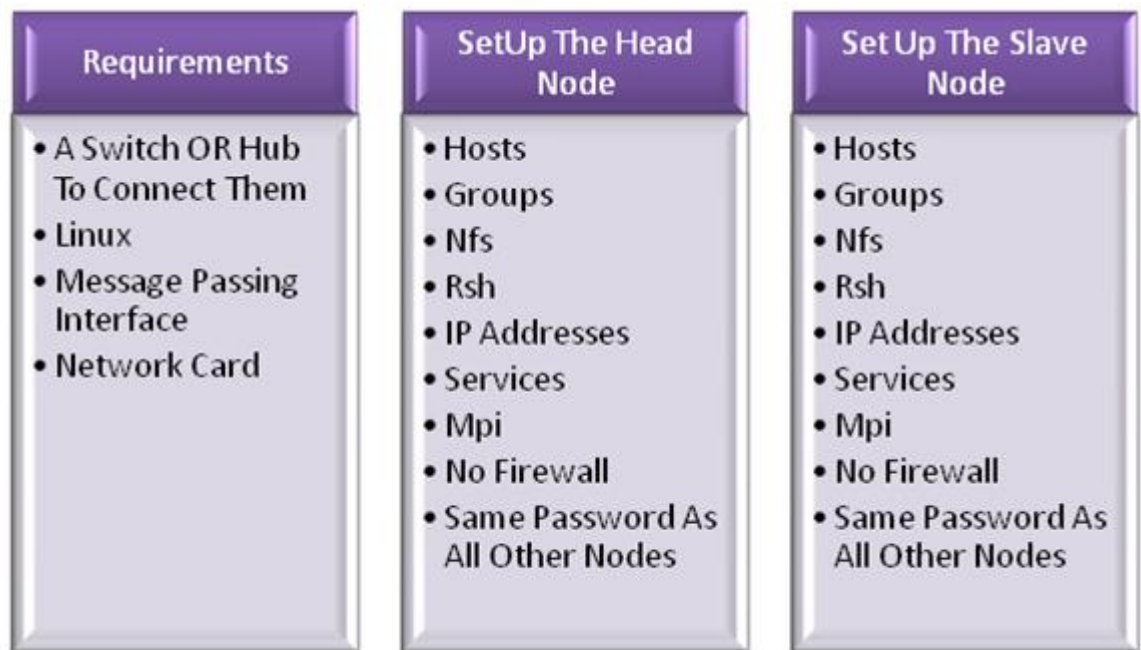


Figure 8.2: Requirements for Beowulf Cluster

- Then We will install The two necessary packages

```
yum install rsh-server
```

```
yum install rsh
```

- Go to the `/etc/securetty..` then add the following command line by line Rsh

```
rlogin
```

```
rexec
```

```
pts/0
```

```
pts/1
```

- Go to the `/etc/pam.d/rsh` when something in this file then replace it and add the following line

```
auth sufficient /lib/security/pam_nologin.so
```

```
auth optional /lib/security/pam_securetty.so
```

auth	sufficient	/lib/security/pam env.so
auth	sufficient	/lib/security/pam rhosts auth.so
account	sufficient	/lib/security/pam stack.so service=system-auth
session	sufficient	/lib/security/pam stack.so service=system-auth

- d. Go to the `/etc/xinetd.d` there `rlogin`, `rexec`, `rsh` and `telnet` are available, in their we will modify, disables yes to no

service shell

socket type=stream

wait =no

user=root

log on sucess +=USERID

log on failure +=USERID

Server=/in/usr/sbin.rshd

Disable=no

- e. Check command for remote service

- `chkconfig -list—grep rsh`

It checks that our remote service is active or not?

- `rpm -qa —grep rsh-server`

It specifies what rsh version is installed our computer

- `Netstat -tap—grep LISTEN`

It check the rsh service

- f. SERVICES make sure that services ,we want are up using following command

- `Chkconfig add sshd`

- Chkconfig add nfs
- Chkconfig add rexec
- Chkconfig add rlogin
- Chkconfig level 3 rsh on
- Chkconfig level 3 nfs on
- Chkconfig level 3 rexec on
- Chkconfig level 3 rlogin on
- Chkconfig del atd
- Chkconfig del rsh
- Chkconfig del sendmail

g. We will create a file

/root/.rhosts in user Root directory.

Node00 Root

Node01 Root

Here, Node00 is our Master Node and Other are slave node

h. Go to the /etc/hosts and add ip address ,host name of all nodes, and other are ip address node number like this

example: suppose 10.1.3.25 is IP address of my machine ,then 10.1.3.25 localhost.localdomain

Node00 10.1.3.4 Node01 THE FOLLOWING STEPS IS FOR MASTER NODE:

i. Configuration For MPICH-1.2.7p1

- mkdir /home/Root/mpich1
- ./configure --prefix=/home/Root/mpich1
- make

- make install

WHERE MPICH1 IS THE NEW DIRECTORY.

- j. Go to the /Root/.bash profile and add the following path, where mpich is installed, this will look like that:

```
/root/mpich1.2.7p1/bin:/Root/mpich-1.2.7p1/util
```

- k. Go to the /mpich-1.2.7p1/util/machines/vi machines.LINUX and add host name of all nodes.

```
Node00
```

```
Node01
```

## 8.5 Testing For Cluster

- a. GO to the mpich-1.2.7p1/examples/basic;make -cpilog files are comes.
- b. Go to the mpich-1.2.7p1/mpe/contribe.mandel;make—pamndel files are comes.
- c. Go to the mpich-1.2.7p1/examples/basic—./cpilog .

### 8.5.1 Xinetd Insatallment

Xinetd performs the same function as inetd. It starts programs that provide Internet services. Instead of having such servers started at system initialization time, and be dormant until a connection request arrives, xinetd is the only daemon process started and it listens on all service ports for the services listed in its configuration file. When a request comes in, xinetd starts the appropriate server. Because of the way it operates, xinetd (as well as inetd) is also referred to as a super-server. The services listed in xinetd's configuration file can be separated into two groups. Services in the first group are called multi-threaded and they require the forking of a new server process for each new connection request. The new server then handles that connection. For such services, xinetd keeps listening for new requests

so that it can spawn new servers. On the other hand, the second group includes services for which the service daemon is responsible for handling all new connection requests. Such services are called single-threaded and `xinetd`.

```

my beowulf (~/.beowulf) ~
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
my beowulf [x]
--> Finished
Dependencies Resolved

=====
Package      Arch      Version      Repository    Size
=====
Installing:
xinetd       i386      2:2.3.14-14.fc8  fedora        123 k
=====

Transaction Summary
=====
Install      1 Package(s)
Update       0 Package(s)
Remove       0 Package(s)

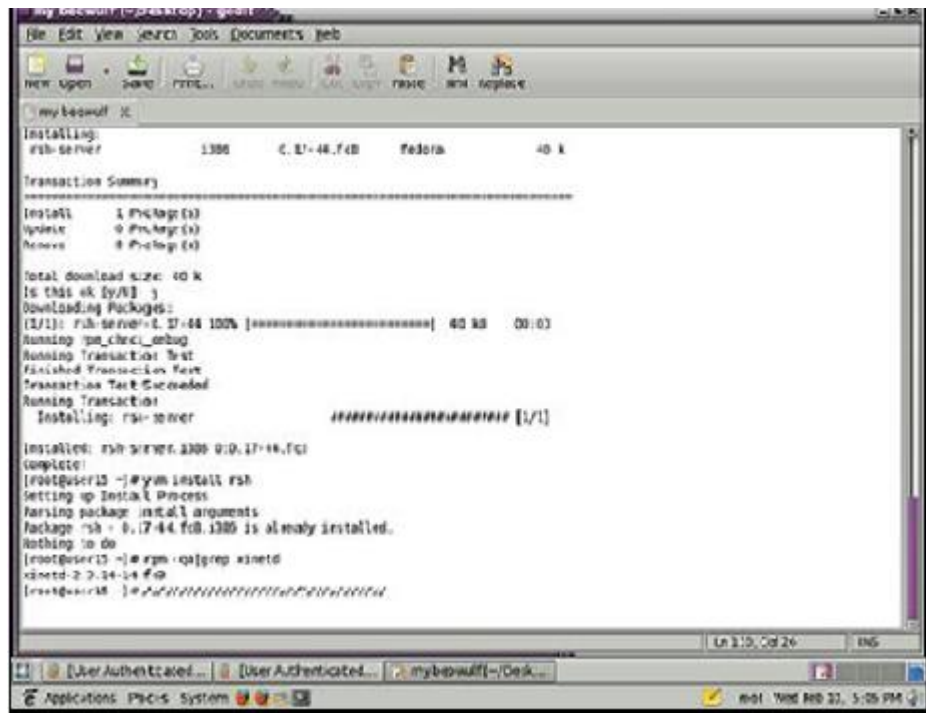
Total download size: 123 k
Is this ok [y/N]: y
Downloading Packages:
(1/1): xinetd-2.3.14-14.fc8.rpm [=====] 123 kB   00:16
warning: rpm2_hdr_from_fdso: Header V3 DSA signature: NOKEY, key ID 4f2a6fd2
Importing GPG key 0x4f2a6fd2 "Fedora Project <fedora@redhat.com>" from /etc/pki/rpm-gpg/RPM-GPG-KEY-fedora
Is this ok [y/N]: y
Importing GPG key 0x042480E "Red Hat, Inc <security@redhat.com>" from /etc/pki/rpm-gpg/RPM-GPG-KEY
Is this ok [y/N]: y
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
Installing: xinetd
===== [1/1]
Ln 1, Col 1
INS
[User Authenticated...] [User Authenticated...] my beowulf (~/.beowulf)
Applications Places System root Wed Feb 10, 5:06 PM

```

Figure 8.3: Output Of Xinetd Configuration

### 8.5.2 Rsh And Rlogin Configuration

Rsh stands for remote shell and allows us to execute non-interactive programs on another system. On some systems, this command is sometimes called remsh or rcmd. It executes the command on the other system and returns the program's standard output and standard error output. The other system must be running a remote shell daemon (rshd) to handle the incoming rsh command. Unix and Linux systems include a remote shell daemon, The rsh command does not require us to enter a password for the other system. Trust is established by defining host equivalency.



```

my beowulf
Installing:
rsh-server 1386 0.17-44.fc8 Fedora 40 k

Transaction Summary
-----
Install 1 Package(s)
Upgrade 0 Package(s)
Remove 0 Package(s)

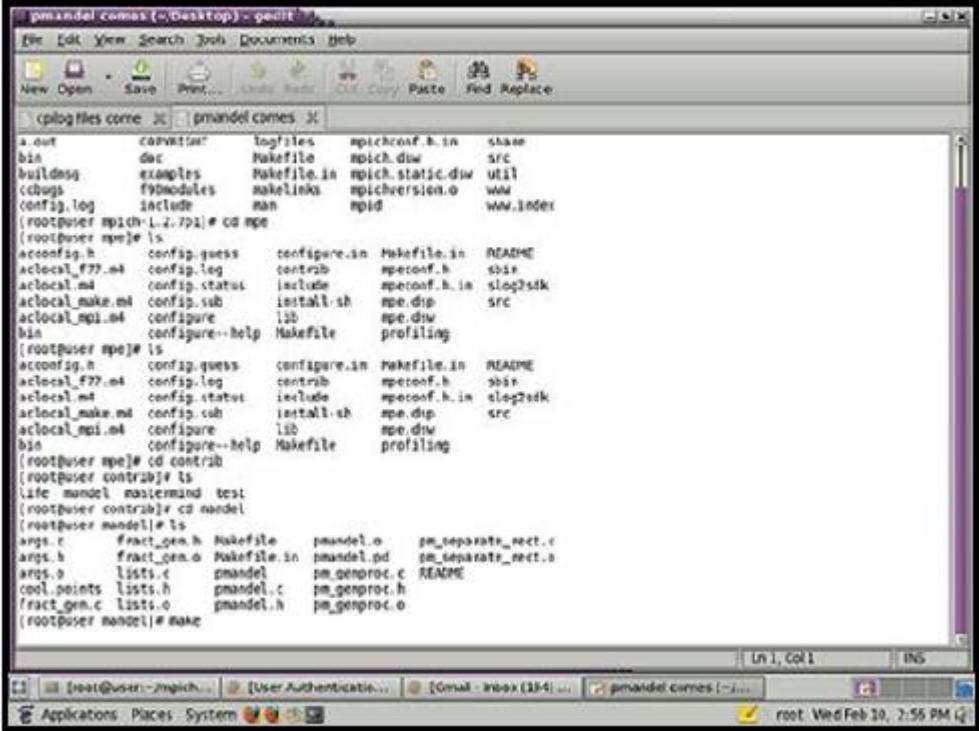
Total download size: 40 k
Is this ok [y/N] : y
Downloading Packages:
(1/1): rsh-server-0.17-44.fc8 100% [#####] 40 k 00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: rsh-server
Installed: rsh-server.x86_64 0:0.17-44.fc8
complete!
[root@beowulf ~]# yum install rsh
Setting up Install Process
Parsing package install arguments
Package rsh-0.17-44.fc8.x86_64 is already installed.
Nothing to do
[root@beowulf ~]# rpm -qal | grep rsh
rsh-0.17-44.fc8.x86_64

```

Figure 8.4: Output Of Rsh And Rlogin Configuration

### 8.5.3 Installment Of Pmandel and Cpilog Files

A Mandelbrot program that uses the MPE graphics pack age that comes with mpich It should work with any other MPI implementation as well but we have not tested it This is a good demo program if we have a fast server and too many processes.



```

pmandel comes (~/.Desktop) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace

cplog files corre 3C pmandel comes 3C

a.out      convstat  logfiles  mpichconf.h.in  share
bin         doc       Makefile  mpich.dsw       src
buildseq   examples  Makefile.in mpich.static.dsw util
cbugs      f90modules  makefile.in mpichversion.o  www
config.log  include   man       mpd              www.index

(root@user mpich-L.2.7p1) # cd mpe
(root@user mpe) # ls
acconfig.h  config.guess  configure.in  Makefile.in  README
aclocal_f77.m4  config.log  contrb       mpeconf.h  sbir
aclocal.m4    config.status  include      mpeconf.h.in  slog2stk
aclocal_make.m4  config.sub  install.sh   mpe.dip     src
aclocal_mpi.m4  configure    lib          mpe.diw     profiling
bin          configure--help  Makefile     profiling

(root@user mpe) # ls
acconfig.h  config.guess  configure.in  Makefile.in  README
aclocal_f77.m4  config.log  contrb       mpeconf.h  sbir
aclocal.m4    config.status  include      mpeconf.h.in  slog2stk
aclocal_make.m4  config.sub  install.sh   mpe.dip     src
aclocal_mpi.m4  configure    lib          mpe.diw     profiling
bin          configure--help  Makefile     profiling

(root@user mpe) # cd contrb
(root@user contrb) # ls
life mandel mastermind test
(root@user contrb) # cd mandel
(root@user mandel) # ls
args.c      fract_gen.h  Makefile  pmandel.o  pm_separate_rect.c
args.h      fract_gen.o  Makefile.in  pmandel.pd  pm_separate_rect.o
args.o      lists.c      pmandel    pm_genproc.c  README
cool_points lists.h      pmandel.c  pm_genproc.h
fract_gen.c lists.o     pmandel.h  pm_genproc.o

(root@user mandel) # make

```

Figure 8.5: Output Of Pmandel Files

### 8.5.4 Output Of Cluster Testing

It contains a few short programs in Fortran C and C++ for testing the simplest features of MPI. It contains multiple test directories for the various parts of MPI Enter make testing in this directory to run our suite of function Tests.

### 8.5.5 Configuration Between Master And Slave Node

For Doing Parallel Computing Between Master And Slave Node We do Configuration Between Master And Slave Node.







## Chapter 9

# Conclusion And Future Work

### 9.1 Conclusion

We propose the MBRP parallel algorithm for H.264 encoder in which frames are split into several MB regions each MB region includes several adjoining columns of MBs and is mapped onto a different processor to be encoded by the CABAC algorithm . We can do parallel computing using Beowulf cluster with reduction of the complexity. Commonly used parallel processing libraries include MPI (Message Passing Interface) and PVM (Parallel Virtual Machine). Both of these permit the programmer to divide a task among a group of networked computers, and collect the results of processing.

### 9.2 Future Work

C code for H.264 ENCODER and DECODER that will be implemented in Beowulf Cluster for parallel computing and The same set up will be tested with 8 core processor. Then we will compare the result of parallel computing and parallel processing .

# References

- [1] M.P Malumbres A Rodriguez, A. Gonzalez. Hybrid parallelization of an h.264/avc video encoder. *Albacete*, page 6, September, 2006.
- [2] Gorden Bell. *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.
- [3] Tian-Sheuan Chang Chao-Chung Cheng. An hardware efficient deblocking filter for h.264/avc. page 10, 2003.
- [4] T. Hidaka. Description of the proposing algorithm and its score for moving image. *ISO/IEC JTC 1/SC*, page 4, oct,1989.
- [5] youn-long lin jian-ven chain, cheng-vu cheng. A hardware accelerator for context based adaptive binary arithmetic decoding in h.264 /avc. *IEEE.*, page 4, 2005.
- [6] Gopal Raghavan Kermin Fleming, Chun-Chieh Lin. H.264 decoder: A case study in multiple design points. page 10.
- [7] Detlev Marpe Marta Mrak and Thomas Wiegand. A context modeling algorithm and its application in video compression. *Proc. ICIP.*, September 14-17, Barcelona, Spain.:4, 2003,.
- [8] S. Ramachandran 2 N. Keshaveni 1 and K.S. Gurumurthy. Implementation of context adaptive variable length coder for h.264 video encoder. *International Journal of Recent Trends in Engineering.*, vol.2:5, November 2009.
- [9] G. M. Schuster and A. K. Katsaggelos. a video compression scheme with optimal bit allocation among segmentation, motion and residual error. *IEEE Trans. Image Process.*, 6:5, 1997.
- [10] Gary J. Sullivan and Thomas Wiegand. Video compression from concepts to the h.264/avc standard. *Proceedings Of The IEEE*, vol.93:14, January 2005.
- [11] Liang P Zhao, Z. A highly efficient parallel algorithm for h.264 encoder based on macro-block region partition. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 31st:9, 2006.

# Website References

- [1] <http://google.com>
- [2] <http://www.erlang.org>
- [3] <http://www.codergrid.de>
- [4] <http://iphone.hhi.de/suehring/tml/>
- [5] <http://www.videolan.org/developers/x264.html>
- [6] <http://www.tandberg.net>
- [7] <http://www.scyld.com>
- [8] <http://www.lam-mpi.org>
- [9] <http://www.mcs.anl.gov/mpi/mpich>
- [10] <http://www.myricom.com>
- [11] <http://www.niu.edu/mpi>
- [12] <http://www.ens-lyon.fr/mercierg/mpi.html>
- [13] <http://www.disi.unige.it/project/gamma/mpigamma/>
- [14] <http://www.mpi-softtech.com>
- [15] <http://www.lhpca.univ-lyon1.fr/mpibip.html>
- [16] <http://www.lfbs.rwth-aachen.de/users/joachim/MP-MPICH/>
- [17] <http://www.nersc.gov/research/ftg/mvich/>
- [18] <http://www.stewe.org/itu-recs/h264.pdf/>
- [19] <http://www.bs.hhi.de/wiegand/ct2003.pdf/>
- [20] <http://www.nersc.gov/research/ftg/mvich/>
- [21] <http://www.ffmpeg.sourceforge.net/>

# Index

- Beowulf cluster, 55
- IP address, 51
- TCP/IP, 51
- Beowulf, 47
- CABAC, 10
- CAVLC, 10
- Decoder, 7
- Encoder, 7
- Erlang, 20
- LAN, 55
- MBRP, 10
- Media Encoding Cluster, 17
- MPI, 2
- MPICH, 54
- Predicton, 6
- Remote procedure call, 53
- Secure Shell, 54
- Sockets, 52