OPTIMIZATION OF H.264 HIGH PROFILE DECODER FOR PENTIUM 4 PROCESSOR

by

TARUN BHATIA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2005

ACKNOWLEDGEMENTS

I am grateful to Dr. K.R.Rao for his constant encouragement and support which made this thesis possible. The author would also like to thank Dr.Manry and Dr. Wang for being on the committee. A special note of thanks for Dr. Pankaj Topiwala, CEO, FastVDO LLC, and my colleagues at FastVDO and Multimedia Processing Lab for helping me understand and appreciate the field of video compression.

Finally, I am indebted to my family for their love, encouragement, moral and financial support. This thesis is dedicated to them.

December 14, 2005

ABSTRACT

OPTIMIZATION OF H.264 HIGH PROFILE DECODER

FOR PENTIUM 4 PROCESSOR

Publication No. _____

TARUN BHATIA, MS

The University of Texas at Arlington, 2005

Supervising Professor: K.R. Rao

H.264 is an emerging video coding standard, which aims at compressing high-quality video content at low-bit rates. While the basic idea behind the encoding and decoding process still remains the same as in the previous standards, many new and innovative tools have been added to provide higher compression capabilities with high perceptual quality. These capabilities have also contributed to the increase in complexity of implementation of H.264 significantly compared to the previous generation of video coding standards.

In this thesis, the performance analysis of the H.264 decoder for decoding of high profile high definition data is done for the Pentium 4 processor and accordingly the

optimization techniques based on single instruction multiple data and Windows based multithreading have been implemented. The comparison of optimized version of the H.264 decoder with the non-optimized version shows significant speed improvements after optimization thus making a contribution to the implementation of real-time decoding of high definition data encoded using H.264 high profile encoder.

TABLE OF CONTENTS

AKNOWLEDGEMENTS..... ii

ABSTRACT.....iii

LIST OF ILLUSTRATIONS.....ix

LIST OF TABLES.....xi

LIST OF ACRONYMS.....xii

Chapter

1.INTRODUCTION.....1

1.1 Thesis Outline.....2

2.OVERVIEW OF MPEG-4 AVC/H.264 DECODER.....4

2.1 Introduction..... 4

2.2 Profiles..... 5

2.2.1.Baseline Profile.....5

2.2.2 Extended Profile..... 6

2.2.3 Main Profile..... 6

2.2.4 High Profiles.....6

2.3 Video coding algorithm..... 8

2.4 Prediction Process..... 9

2.4.1 Inter Prediction..... 9

2.4.2 Intra Prediction..... 10

2.5 Transform and Quantization 11

2.6 Deblocking..... 12

2.7 Entropy Coding..... 13

2.8 Summary 14

3.PENTIUM 4 OPTIMIZATION ARCHITECTURE OVERVIEW……15

3.1 Main Features of Intel® NetBurst® microarchitecture 16

3.2 Pipeline 17

3.2.1 Front End 17

3.2.2 Execution Unit 18

3.2.3 Retirement Unit..... 19

3.3 Branch Prediction..... 19

3.4 Caches 20

3.4.1 Prefetch 21

3.5 Hyper-Threading Technology..... 22

3.5.1 Resource Sharing 23

3.5.2 Hyperthreading Pipeline 24

3.6 Summary 25

4.SINGLE INSTRUCTION MULTIPLE DATA (SIMD)
BACKGROUND AND APPLICATION TO H.264 DECODER……..26

4.1 SIMD Architecture..... 26

4.1.1 Registers..... 26

4.2 MMX Technology 28

4.3 SSE Technology..... 30

4.4 Application of SIMD 31

4.5 SIMD Optimization of High Profile H.264 HD Decoder..... 31

4.5.1 Performance Analysis of H.264 Decoder 32

4.5.2 Results of Performance analysis of H.264 Decoder 36

4.5.3 Application of SIMD Optimization to IDCT 4x4 transform in H.264 38

4.5.4 Application of SIMD Optimization to Motion Compensation 40

4.6 Summary..... 44

5.MULTITHREADING…….46

5.1 Basics of Multithreading..... 46

5.2 Multithreading Strategies..... 48

5.2.1 Data Level Parallelism..... 48

5.2.2 Task Level Parallelism..... 48

5.3 Thread Synchronization..... 49

5.4 Producer-Consumer Problem..... 50

5.5 Summary..... 51

6.MULTITHREADING GOP LEVEL H.264 DECODER……...52

6.1 H.264 Decoder Design-Strategies..... 52

6.1.1 Task-level..... 52

6.1.2 Data-level..... 53

6.2 H.264 GOP level Decoding 54

6.3 H.264 Decoder –Threading Architecture 57

6.3.1 Main Thread 57

6.3.2 Get IDR Position Thread 58

6.3.3 Decoder Threads 58

6.4 Multithreaded GOP Level Decoding - Results 61

6.4 Multithreaded GOP Level Decoding - Observations 65

6.5 Summary 67

7.CONCLUSIONS AND FURTHER RESEARCH……..…….69

Appendix

A. H.264 ENCODER SETTINGS FOR GENERATION OF TEST STREAMS USED…..... 71

B. AMDAHL'S LAW……74

REFERENCES……..76

BIOGRAPHICAL INFORMATION…….81

LIST OF ILLUSTRATIONS

Figure Page

2.1 Macroblock and Sub-macroblock partitions 5

2.2 Coding profiles of H.264 7

2.3 Block diagram of H.264 encoder 8

2.4 Block diagram of H.264 decoder 9

2.5 Intra prediction modes 11

2.6 Block Edges in a macroblock 13

3.1 Intel NetBurst Microarchitecture 15

3.2 Intel NetBurst Microarchitecture Pipeline 17

3.3 Resources Sharing in Hyperthreading Architecture 23

4.1 Architectural support for SIMD architecture 27

4.2 MMX Data Types 28

4.3 SIMD operations 30

4.4 Results of VTune Analysis of H.264 Decoder -System Level 34

4.5 Streams Used for Testing……35

4.6 Results of time based sampling of H.264 Decoder……..37

4.7 % Time consumed by 2-D IDCT 4x4 40

4.8 Segregation of % time consumption by motion compensation 42

4.9 % Time consumed by interpolation 44

5.1 Producer-Consumer Thread Synchronization..... 50

6.1 Multithreaded Architecture for H.264 GOP Level Decoding……..59

6.2 Flow Control in Get IDR Position Thread..... 60

6.3 Speed up in decoding time for Girl.264..... 62

6.4 Speed up in decoding time for Golf.264..... 62

6.5 Speed up in decoding time for Karate.264..... 63

6.6 Speed up in decoding time for Plane.264 63

6.7 Speed up in decoding time for Shore.264..... 64

6.8 Overhead Time..... 64

6.9 Processor Utilization for Karate.264 using 2 decoder threads..... 67

LIST OF TABLES

Table Page

4.1 Test Setup for performance analysis of H.264 Decoder……35

4.2 Results of performance profiling of the H.264 high profile decoder.....36

4.3 Results of SIMD optimization of the IDCT 4x4 block.....39

4.4 Results of comparison of SIMD optimized Interpolation with non-SIMD Interpolation……..…….....43

LIST OF ACRONYMS

AVC: Advanced Video Coding

DVD: Digital Video Disc

GOP: Group of Pictures

HD: High Definition

IDR: Intra Decoder Refresh

IEC: International Engineering Consortium

ISO: International Standards Organization

ITU: International Telecommunication Union

JVT: Joint Video Team

MPEG: Motion Picture Experts Group

NALU: Network Abstraction Layer Unit

PPS: Picture Parameter Set

SEI: Supplemental Enhancement Information

SI: Switching Intra

SIMD: Single Instruction Multiple Data

SP: Switching Prediction

SPS: Sequence Parameter Set

CHAPTER 1

INTRODUCTION

MPEG-4 Part 10/H.264 [4] is an emerging video coding standard, which aims at providing high quality video content at very low bitrates. It offers tools for higher compression and better error robustness than previous generation of video coding standards [9] [10] [19] [38] [39]. While the basic video coding algorithm for H.264 is still based on hybrid of motion compensated prediction and transform coding, new and improved algorithms used for them add significant complexity and require higher computational capabilities for implementation [1].

For real-time implementation of H.264 encoding and decoding algorithms on commonly used general purpose single processor hardware platforms such as Intel's Pentium 4, it needs significant speed optimizations in terms of computational implementation and memory bandwidth requirements. In this thesis, the SIMD (Single Instruction Multiple Data) [18] level speed optimization techniques at processor level and multithreading at the Operating System level are presented for the H.264 high profile decoder. The high profile of the H.264 standard defines a set of coding tools and options with the intention of making it a preferable choice for high-quality broadcast and entertainment applications. This makes the usage of H.264 high profile for high definition (HD) video compression in real-time on general purpose processor like Intel's Pentium 4

one of the most computationally intensive applications at the single processor level .The optimizations performed and the results of those optimizations in this thesis have been tested specifically for the decoding of H.264 high profile high definition (1280x720 progressive content) sequences. The real-time encoding of high definition sequences using the H.264 high profile is too complex to be implemented on a general purpose single processor hardware platform [2] [3].

1.1 Thesis Outline

Chapter 2 presents a basic introduction to the H.264 video compression standard. The video coding tools and arrangement of the tools in terms of profiles provided in the standard are discussed briefly.

Chapter 3 presents a basic introduction to the target hardware platform of the Pentium 4 processor. It presents a brief discussion of the optimization architecture i.e. Intel NetBurst Microarchitecture [18] along with the hyperthreading technology which offers an opportunity of achieving significant optimization of applications running on the Pentium 4 processor.

Chapter 4 presents the basics of implementation of the SIMD (Single Instruction Multiple Data) technology on the Intel's Pentium 4 processor i.e. MMX (Multimedia Extension) and SSE (Streaming SIMD) technology. Further, it presents the performance analysis of H.264 high profile decoder using the Intel's VTune performance analyzer [20]. Finally, application of SIMD optimization to the most time consuming blocks in the

H.264 decoding process and the performance improvement obtained as a result are discussed.

Chapter 5 presents the basics of multithreading based on Windows operating system. It discusses thread creation and synchronization using semaphores. An application of thread synchronization to the producer-consumer problem is also mentioned.

Chapter 6 presents a method of application of multithreading for the speed optimization of the H.264 high profile decoder. A specific case of optimization of decoding of the group of pictures (GOP) level encoded high profile HD sequence is discussed and a multithreaded decoding architecture with resulting speedup is presented for the same.

Chapter 7 outlines the conclusions and further research. The configuration files for the JM 9.6 H.264 encoder used for generation of H.264 high profile bitstreams are listed in Appendix A.

CHAPTER 2

OVERVIEW OF MPEG-4 AVC/H.264 DECODER

The new video coding standard MPEG-4 part 10 / H.264 [4] aims at having significant improvements in coding efficiency, and error robustness in comparison with the previous video compression standards such as MPEG-2 [19], H.263 [39], and MPEG-4 part 2[38]. It allows coding of non-interlaced and interlaced video and provides better visual quality even at higher bitrates. This chapter presents an overview of the decoding process and the coding tools provided in the standard which give the H.264 coding algorithm its compression capabilities.

2.1 Introduction

The design of the H.264/ MPEG-4 part10 [4] video codec is based on the traditional hybrid concept of block based motion-compensated prediction and transform coding [4] [5] [6] [7] [8]. The coding of the video is done on picture by picture basis which is further represented as one or more slices. A slice is the smallest independent coding element in the H.264 coding structure. The slice consists of a sequence of macroblocks with each macroblock consisting of 16x16 luminance and two chrominance components (Cb and Cr).

Each macroblock's 16x16 luminance component is partitioned into 16x16, 16x8, 8x16 and 8x8, and further, each 8x8 luminance can be sub-partitioned into 8x8, 8x4, 4x8

and 4x4. The 4x4 sub-macroblock partition is called a block. The hierarchy of data is as follows:

Picture [slices [macroblocks [sub-macroblocks [blocks [pixels]]]]]

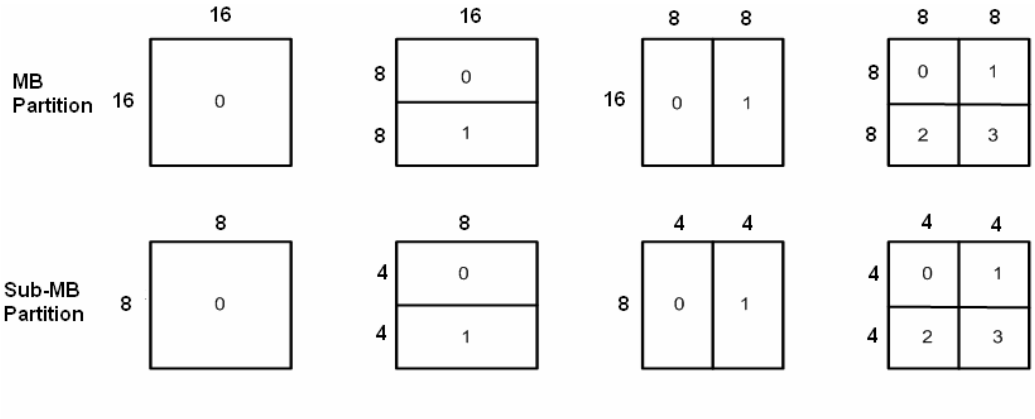


Figure 2.1 Macroblock and Sub-macroblock partitions [5]

2.2 Profiles

The various coding tools and capabilities of the H.264 video coding algorithm are arranged under various profiles (Fig. 2.2) which have been defined with a focus on specific target applications. The profiles and the specific tools offered in those profiles can be listed as [5]:-

2.2.1 Baseline Profile (mainly for video streaming applications)

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

- Flexible macroblock order (FMO): macroblocks may not necessarily be in the raster scan order. The map assigns macroblocks to a slice group.
- Arbitrary slice order (ASO): the ordering of the slices in the bitstream may not be in raster scan order.

2.2.2 Extended Profile

- All the features of baseline profile
- SP, SI slices
- B slices
- Weighted prediction

2.2.3 Main Profile

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

2.2.4 High Profiles [9] [10]

- Includes high profile (HP) supporting 8-bit video with 4:2:0 sampling, addressing high end consumer use and other applications using high resolution video
- High 10 profile (Hi10) supporting 10-bit video with 4:2:0 sampling

- High 4:2:2 profile (H422P), supporting upto 4:2:2 sampling and upto 10-bits per sample
- High 4:4:4 profile (H444P) , supporting up to 4:4:4 sampling, up to 12 bits per sample, lossless region coding and integer residual color transform for coding RGB video
- All the features of main profile
- Adaptive block size transform (introduction of 8x8 integer DCT)
- Perceptual quantization matrices
- 8x8 Intra prediction

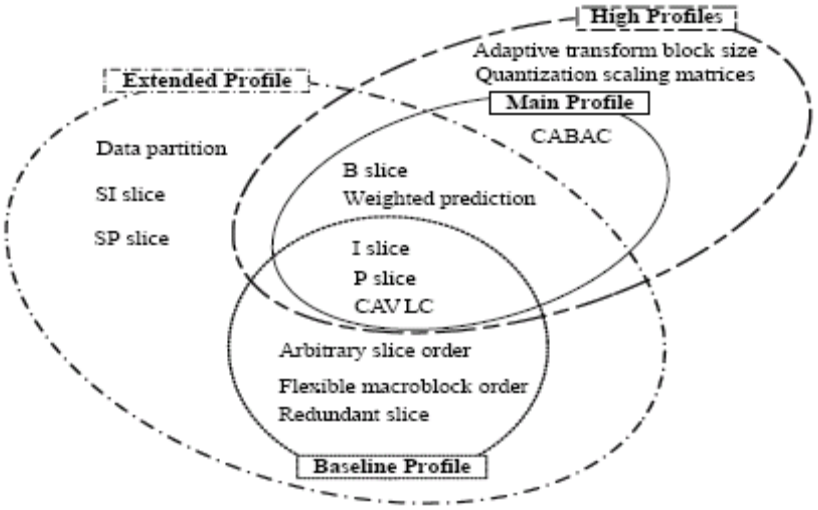


Figure 2.2 Coding profiles of H.264 [5]

2.3 Video coding algorithm

The block diagrams for the H.264 encoder and decoder are shown in Figs. 2.3 and 2.4 respectively. Intra-coding uses various spatial modes to reduce spatial redundancy in source data for a picture. Inter-coding uses block-based inter-prediction to reduce temporal redundancy between various pictures. The deblocking filter helps in reducing blocking artifacts at the block boundaries. Application of transform to the prediction residual leads to further compression and helps in removal of spatial redundancy through quantization. The quantized transform coefficients along with the motion vectors for prediction are then entropy coded to create the H.264 encoded bitstream.

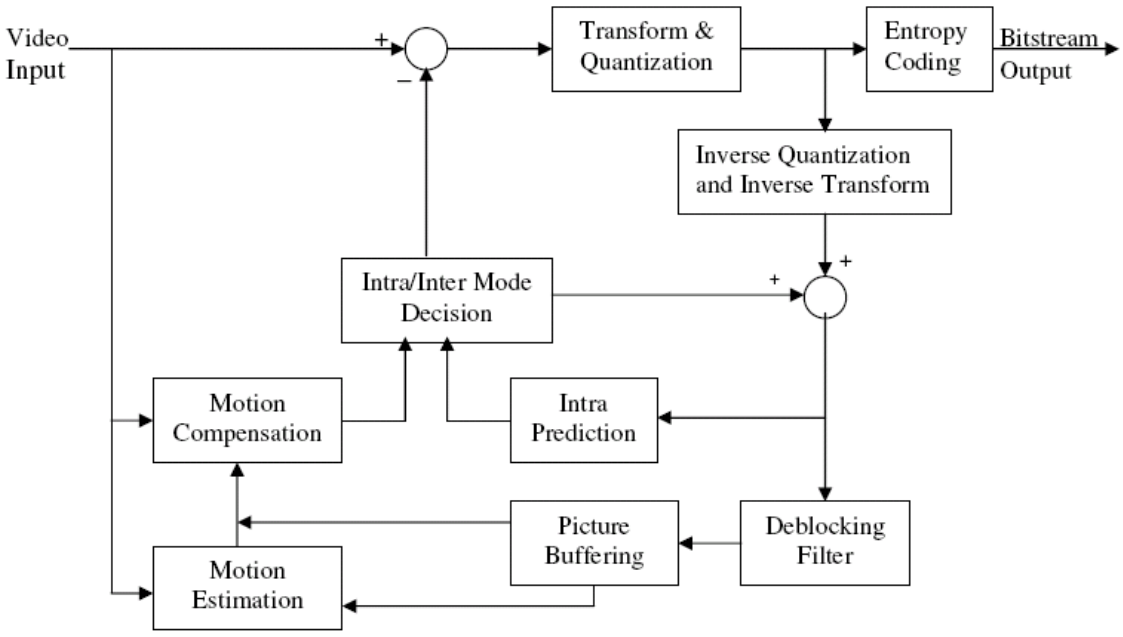


Figure 2.3 Block diagram of H.264 encoder

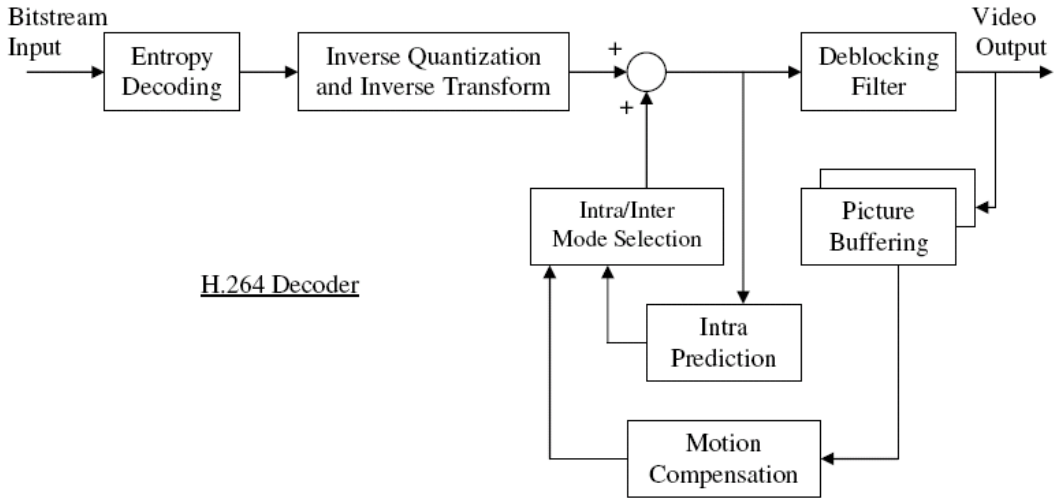


Figure 2.4 Block diagram of H.264 decoder

2.4 Prediction Process

All types of prediction used in the earlier standards are supported namely I (Intra), P (Predictive) and B (Bi-predictive) with addition of number of modes mainly for intra-prediction.

2.4.1 Inter Prediction

Unlike previous standards, the P slices can be predicted using reference from past or the future in the decoding order. It can also contain intra macroblocks (used in case of gradual decoder refresh (GDR) [26]). For B-slices the prediction uses two references but unlike the previous standards can be from the past or the future in the decoding order along with the regular method of one reference from past and one from future in the decoding order. The standard also defines two new types of slices; SI (Switching Intra) and SP (Switching Predictive) which as the name suggests are used for switching

between streams. The motion estimation also includes options of skipped macroblock, direct prediction and weighted prediction of a macroblock. The skipped macroblock uses a predicted motion vector to copy a motion compensated block instead of copying a co-located macroblock as no motion vector is transmitted on the encoder side in the absence of residual data. The option of direct prediction allows two modes for motion vector prediction: new spatial direct motion vector prediction (which uses motion vectors of co-located macroblocks to predict motion vectors of current macroblock) and the refined temporal direct prediction (which uses temporally scaled motion vectors of macroblocks in the reference frames to predict motion vectors of the current macroblock). The option of weighted prediction allows scaling of reference slice data in order to facilitate prediction for occurrences like scene fading.

The macroblocks in a picture can be partitioned from 16x16 up to 4x4. Larger partitions are for appropriate for homogeneous areas of the picture and smaller partitions are for detailed areas. The H.264 standard specifies usage of sub-pel motion estimation upto ¼ -pel accuracy. The reference pictures for inter-prediction are chosen from the decoded picture buffer which maintains the reference frames in the form of lists.

2.4.2 Intra Prediction

The intra coded macroblock is coded itself by without using temporal prediction. In the H.264 standard, the process of intra prediction involves predicting the current block from the decoded frame (not filtered) and then calculating the residual between the source block and predicted block. The residual data is then encoded.

The various modes of prediction offered are:-

For Luma: 9 modes for 4x4 blocks, 9 modes for 8x8 blocks, 4 modes for 16x16 blocks

For Chroma: 4 modes for each 4x4 chroma block

The directions of prediction for the all 9 possible modes are shown in the Fig 2.5 [6]

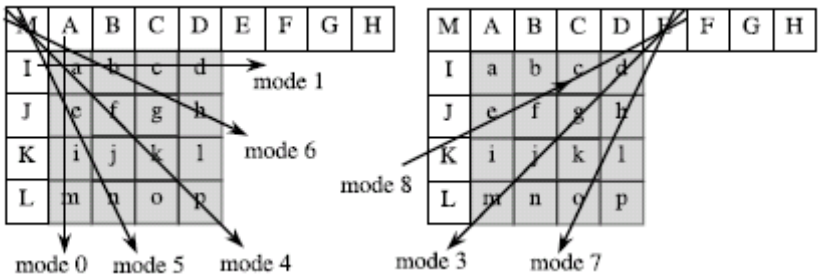


Figure 2.5 Intra prediction modes

2.5 Transform and Quantization

H.264 is based on the use of a block-based transform for spatial redundancy removal. H.264 uses an adaptive transform block size, 4x4 and 8x8 (high profiles only), whereas previous video coding standards used the 8x8 DCT. The smaller block size leads to a significant reduction in ringing artifacts. Also, the 4x4 transform has the additional benefit of removing the need for multiplications. For improved compression efficiency, H.264 also employs a hierarchical transform structure, in which the DC coefficients of neighboring 4x4 transforms for the luma signals are grouped into 4x4 blocks and transformed again by the Hadamard transform. For blocks with mostly flat pel values, there is significant correlation among DC coefficients of neighboring blocks. Therefore,

the standard specifies the 4x4 Hadamard transform for luma DC coefficients for 16x16 intra-mode only, and 2x2 Hadamard transform for chroma DC coefficients.

H.264 assumes a scalar quantizer. The basic forward quantizer operation is:

$$Zi\,j = round(Yi\,j\,/Qstep) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where *Yi j* is a coefficient of the transform described above, Qstep is a quantizer step size and *Zi j* is a quantized coefficient. The rounding operation here (and throughout this section) need not round to the nearest integer; for example, biasing the 'round' operation towards smaller integers can give perceptual quality improvements. A total of 52 values of Qstep are supported by the standard, indexed by a quantization parameter, QP. Qstep doubles in size for every increment of 6 in QP. The wide range of quantizer step sizes makes it possible for an encoder to control the tradeoff accurately and flexibly between bit rate and quality. The values of QP can be different for luma and chroma. Both parameters are in the range 0–51 and the default is that the chroma parameter.

2.6 Deblocking

A filter is applied to each decoded macroblock to reduce blocking distortion. The deblocking filter is applied after the inverse transform in the encoder (before reconstructing and storing the macroblock for future predictions) and in the decoder (before reconstructing and displaying the macroblock). The filter smoothes block edges, improving the appearance of decoded frames. The filtered image is used for motion-

compensated prediction of future frames and this can improve compression performance because the filtered image is often a more faithful reproduction of the original frame than a blocky, unfiltered image.

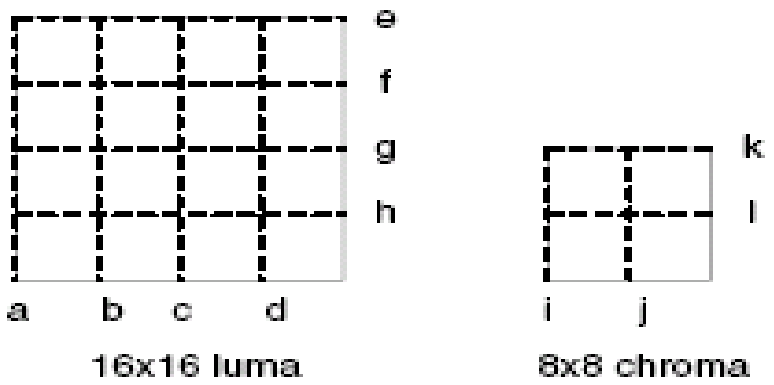


Figure 2.6 Block Edges in a macroblock

Filtering is applied to vertical or horizontal edges of 4×4 blocks in a macroblock (except for edges on slice boundaries), in the following order.

1. Filter 4 vertical boundaries of the luma component (in order a, b, c, d in Figure 4.4).
2. Filter 4 horizontal boundaries of the luma component (in order e, f, g, h, Figure 4.4).
3. Filter 2 vertical boundaries of each chroma component (i, j).
4. Filter 2 horizontal boundaries of each chroma component (k, l).

2.7 Entropy Coding

The entropy coder converts a series of symbols representing elements of a video sequence into a compressed bitstream suitable for transmission or storage. The elements

at the slice layer are coded using either context adaptive variable length coding (CAVLC) or context adaptive binary arithmetic coding (CABAC). Elements like header information and parameter sets (Sequence Parameter Set and Picture Parameter Set) are coded using Exponential-Golomb codes [6].

2.8 Summary

This chapter has presented a basic introduction to the MPEG-4 part 10 / H.264 video coding standard. H.264 performs better than the previous video coding standards by introducing new innovative algorithms and improving some previously used algorithms to provide superior visual quality at lower bitrates with better error resilience.

CHAPTER 3

PENTIUM 4 OPTIMIZATION ARCHITECTURE OVERVIEW

The first step for optimization of an application for a hardware platform is to analyze the abilities of the hardware platform. The important features which we need to look into before we start optimizing are the processor architecture, cache and processor bus. The Intel® Pentium® 4 processor is based on Intel® NetBurst® microarchitecture [11] [12] [17] replacing the P6 Architecture [11] used until the Pentium® III Processor. The architecture as shown in Fig. 3.1 [11] [17] is designed to provide the end user with new levels of performance, enabling computationally intensive tasks to be performed at the desktop level.

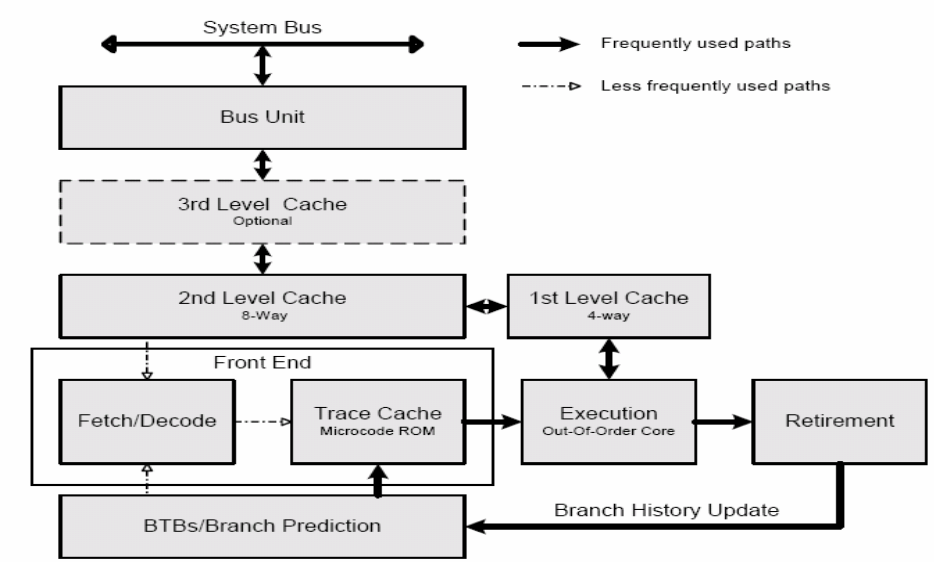


Figure 3.1 Intel NetBurst Microarchitecture

3.1 Main Features of Intel® NetBurst® microarchitecture

- hyper-pipelined technology that enables higher clock rates and higher frequency headroom (upto 10 GHz)
- high-performance, quad-pumped bus interface to the Intel® NetBurst® microarchitecture system bus
- rapid execution engine to reduce the latency of basic integer instructions
- out-of-order speculative execution to enable parallelism
- superscalar issue to enable parallelism
- hardware register renaming to avoid register name space limit
- cache line sizes of 64 bytes (up from 32 bytes in Pentium III)
- hardware prefetch
- a pipeline that optimizes for the common case of frequently executed instructions; the most frequently-executed instructions in common circumstances (such as a cache hit) are decoded efficiently and executed with short latencies
- Employments of techniques to hide stall penalties; among these are parallel execution, buffering, and speculation. The microarchitecture executes instructions dynamically and out-of-order, so the time it takes to execute each individual instruction is not always deterministic.

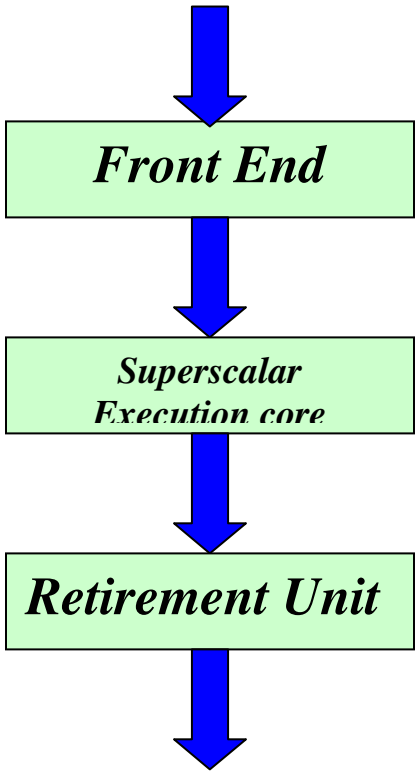


Figure 3.2 Intel NetBurst Microarchitecture Pipeline

3.2 Pipeline

As shown in Fig. 3.2 the three stages of the Intel NetBurst Microarchitecture pipeline are:-

3.2.1 Front End

The in-order front end of the Intel NetBurst microarchitecture consists of two parts:

- fetch/decode unit
- execution trace cache

It performs the following functions:

- prefetches IA-32 instructions that are likely to be executed
- fetches required instructions that have not been prefetched and decodes instructions into μops (micro-operations). The execution trace cache stores decoded instructions in the form of μops rather than in the form of raw bytes unlike the conventional processor caches. It allows for complex instruction decode logic to be removed from the execution unit. Also, it allows code from the target of a branch to be included in the same trace cache line as the branch itself reducing cache delays.
- generates microcode for complex instructions and special-purpose code
- delivers decoded instructions from the execution trace cache
- predicts branches using advanced algorithms
- the execution trace cache stores μops in the path of program execution flow, where results of branches in the in the code are integrated.

3.2.2 Execution Unit

The out-of-order execution unit reorders μops so that the μops whose inputs are ready (and have execution resources available) can execute as soon as possible. The core's ability to execute instructions out-of-order is the key factor in enabling parallelism. It considers a large number of μops (126 of which 48 can be load and 32 can be store operations) [17] from the program to find independent μops. This task is done by the μop schedulers who track the input operands to determine if a μop is ready to execute. The μops are then scheduled to execute in the data dependent order i.e. when the resources needed by them are available. The

Rapid Execution Engine is able to execute up to six μops per main clock cycle. The double-speed integer Arithmetic Logic Units (ALUs) are able to execute at a rate of two μops per clock cycle, providing higher ALU throughput compared to running the ALU at the main clock speed.

3.2.3 Retirement Unit

The retirement section ensures that the results of execution are processed according to original program order and that the proper architectural states (flags etc.) are updated. It also keeps track of branches and sends updated information to the branch target buffer (BTB).

3.3 Branch Prediction

Branch prediction is important to the performance of a deeply pipelined processor. It enables the processor to execute conditional/unconditional branch instructions long before their outcome is certain. Branch delay is the penalty incurred in case the branch is not predicted correctly. As the pipelines grow deeper and deeper, branch delay increases. On the other hand, branch delay for correctly predicted can be as low as zero cycles.

The branch prediction algorithm in Intel NetBurst microarchitecture predicts all near branches (conditional calls, unconditional calls, returns and indirect branches). It does not predict far transfers (far calls, irets and software interrupts).

The main features of the branch prediction in this architecture are:-

- The ability to dynamically predict the direction and target of the branches based on an instruction's linear address, using the branch target buffer.
- Static prediction in case of failure/non-availability of dynamic prediction in case of which backward branches are taken and forward branches are not. This works well in case of loop ending branches. The difference between a loop ending branch and a normal backwards branch is determined by comparing the distance between the branch and its target to a threshold. If the distance is greater than the threshold then the branch is predicted as taken.
- The ability to predict return addresses using 16-entry return address stack (as the return may be invoked from several call sites).
- The ability to build trace of instructions across predicted taken branches to avoid branch penalties.

3.4 Caches

The Intel NetBurst microarchitecture supports up to three levels of on-chip cache. At least two levels of on-chip cache are implemented in processors based on Intel NetBurst microarchitecture. The Intel Xeon MP [13] and certain Intel Pentium and Intel Xeon Processors [13] may also contain a third level cache.

The first level cache (L1 cache, nearest to the execution core) contains separate caches for instructions and data. These include first-level data cache and instruction cache. Other caches are shared between instructions and data. All caches use least recently used replacement algorithm. On processors without a third level cache, the

second-level cache miss initiates a transaction across the system bus interface to memory sub-system. On processors with a third-level cache, a third-level cache miss initiates a transaction across the system bus. A bus write transaction writes 64 bytes to cacheable memory, or separate 8 byte chunks in the destination are not cacheable. A bus read transaction from cacheable memory fetches two cache lines of data.

3.4.1 Prefetch

3.4.1.1 Software Prefetch

Software Prefetch is enabled using SSE instructions. It is not to be used for instruction prefetch as it can cause severe problems on a multiprocessor system. Using software prefetch requires comprehensive analysis of application by the programmer so that factors like type of prefetch to be used and how much in advance do we need to get the data can be decided for optimal cache usage. Depending on what type of prefetch we use the data is prefetched into the one of the caches, for e.g. the cache nearest to the processor which will minimize disturbance of other caches and avoids the need to access of-chip cache.

But using non-optimal software prefetch can cause reduction in performance because:-

- too many prefetches reduce efficiency
- bus bandwidth demand increase
- prefetching data too early in the application will result in data being removed from cache before it is actually used and the whole process of prefetch will just add to execution latency

3.4.1.2 Hardware Prefetch

Hardware Prefetch brings cache lines into the unified second level cache based on prior reference patterns. It is triggered automatically but requires regular access patterns. It does not cause any bandwidth overhead. But the process has a start-up penalty before it triggers and starts fetching data.

3.5 Hyper-Threading Technology

Intel Hyper-Threading Technology (HT) [11] [12] is available in Pentium 4 and Xeon processor families. The basic idea behind HT technology is to provide two logical processors on a single physical processor. It enables the software to take advantage of task-level and/or thread-level parallelism. Each logical processor has a set of architectural registers while sharing physical execution resources of the single physical processor. Doubling the architectural state (eight general purpose registers, control registers, machine state registers etc.) seem like two physical processors to the software application (including the Operating System) running on it and provide a performance boost over a traditional multiprocessor system.

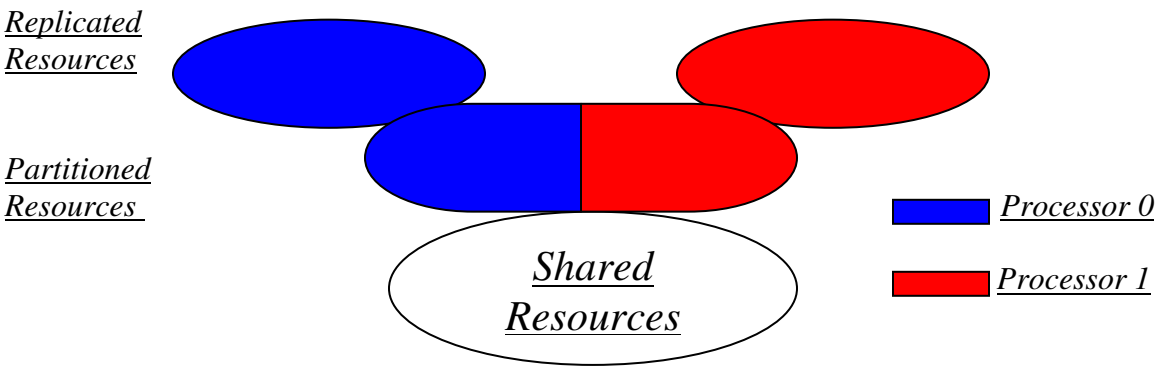


Figure 3.3 Resources Sharing in Hyperthreading Architecture

3.5.1 Resource Sharing

As shown in Fig. 3.3 there is a three level resource management in a hyperthreaded machine. They are:-

3.5.1.1 Replicated Resources

The architectural state is replicated for each logical processor. The architecture state consists of registers that are used by the operating system and application code to control program behavior and store data for computations. This state includes the eight general-purpose registers, the control registers, machine state registers, debug registers, and others.

3.5.1.2 Partitioned Resources

Several buffers are partitioned for usage by both logical processors by limiting entries to half for each of them. This is done to allow one processor to bypass operations of the other processor which may be stalled by an event like branch misprediction, cache miss or certain instruction dependencies. These buffers include μop queues after the

execution trace cache, the queues after the register rename stage, the reorder buffer which stages instructions for retirement, and the load and store buffers. In the case of load and store buffers, partitioning also provides an easier implementation to maintain memory ordering for each logical processor and detects memory ordering violations.

3.5.1.3 Shared Resources

Most other resources in the physical processor like the caches and the execution unit are completely shared to improve dynamic utilization of the resource. Some shared resources include a processor id to indicate to which logical processor the entry belongs.

3.5.2 Hyperthreading Pipeline

Instructions form the two logical processors are executed simultaneously in the execution unit not necessarily in program order. There are selection points in the pipeline where instructions from one or the second logical processor can be chosen. In case one logical processor cannot use the pipeline, then the other logical processor has complete control over the pipeline.

3.5.2.1 Front End

The execution trace cache is shared between the two logical processors. It is used alternately by the two logical processors every clock cycle unless one of them is stalled. After fetching the instructions and building traces of μops, the μops are placed in a queue. If both logical processors are active then the execution trace cache is partitioned for each of them to proceed independently.

3.5.2.2 Execution Unit

The execution unit does not differentiate between instructions of the two logical processors once they are placed in the queue. After execution, they are placed in re-order buffer. The re-order buffer decouples execution stage from the retirement stage and is partitioned equally between the two logical processors.

3.5.2.3 Retirement Unit

It tracks when the instructions form the two logical processors are ready to be retired. The instructions are retired for each logical processor in program order by alternating between two of them.

3.6 Summary

This chapter briefly discussed Intel NetBurst microarchitecture and optimization capabilities of the Pentium 4 processor. The features of the optimization architecture and Hyper threading technology which are useful in case of multithreading of applications were also mentioned.

CHAPTER 4

SINGLE INSTRUCTION MULTIPLE DATA (SIMD) BACKGROUND AND APPLICATION TO H.264 DECODER

Intel had included SIMD technology as an extension to the basic Intel architecture (IA-32) designed to improve performance of multimedia and communication algorithms. The technology includes new registers at the hardware level and new data types and instructions at the software level. This chapter explores the level of performance boost that can be obtained for H.264 high profile decoder.

4.1 SIMD Architecture

Figure 4.1 [11] [14] below shows block diagram of the hardware registers that can be used for implementation of code exploiting SIMD technology. By exploiting SIMD instructions on the floating point registers of x86 architecture, backward compatibility with all application and operating system code is maintained.

4.1.1 Registers in SIMD Architecture

- 8 XMM registers (128-bit) added Pentium III onwards to be used by SSE, SSE2, SSE3 instruction set
- 8 MMX registers (64-bit) introduced Pentium MMX onwards for to be used by the MMX instruction set.

- 8 general purpose registers (32-bit) from x86 architecture
- MXCSR register a special purpose control register that contains IEEE-754 flags and mask bits.
- EFLAGS register containing the all the major flags used by the processor to indicate the results of the operation performed for e.g. carry, sign, zero etc.

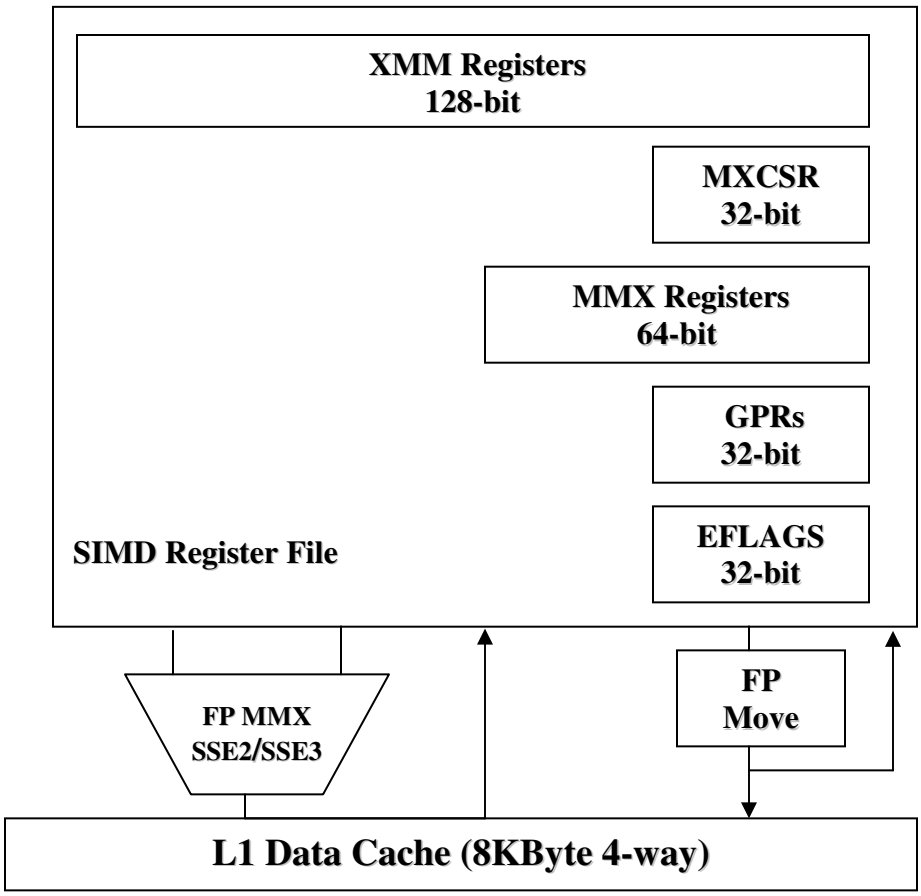


Figure 4.1 Architectural support for SIMD architecture [11] [17]

4.2 MMX Technology

Intel MMX™ (multimedia extension) technology [11] [15] [16] [18] is an extension to the basic Intel Architecture and includes new registers and instructions. It exploits the inherent parallelism in multimedia and communication algorithms which help to boost performance of these algorithms on Intel processors. Many multimedia applications require simple operations to be performed on large quantities of data items in parallel. Also, the size of data is typically 8-bit or 16-bit (e.g. pixel data in an image or video).The MMX technology includes 57 new instructions, and four new data types [18]. Each of the four data types represents 64-bit data interpreted as packed byte, packed word, packed double word and quadword. The representation of data types is shown in Fig. 4.2.

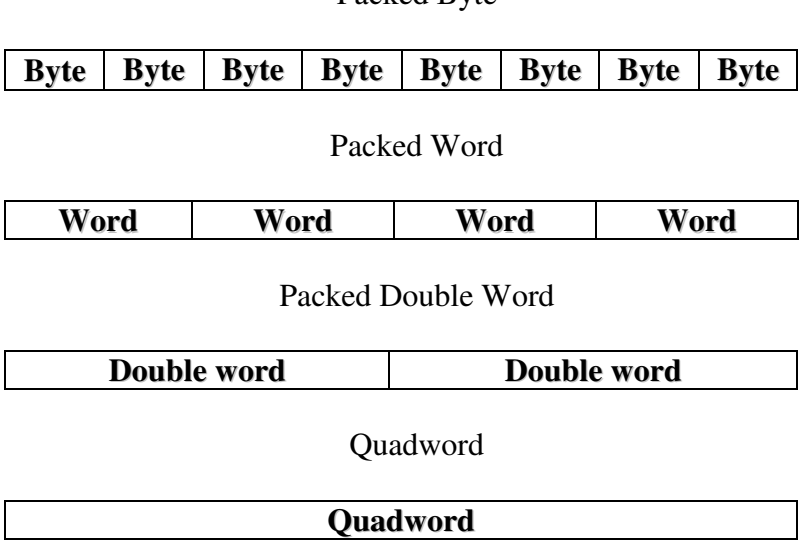


Figure 4.2 MMX Data Types [11]

4.2.1 Features of MMX Technology

- 57 new instructions
- 4 new data types
- 8 64-bit registers derived from 8 80-bit floating point registers in the Pentium architecture [11] [15] [16]. Therefore, regular floating operations are suspended when using MMX registers. At the end of the MMX operations, *emms* instruction is used to indicate to the CPU that floating point operations can be resumed.
- "saturation arithmetic": the results of operations can be set to zero or maximum range of the data type instead of a normal wraparound. This feature is particularly helpful in video coding algorithms like H.264 where truncation after an arithmetic operation [4] is used to limit the results to a certain range.
- the instruction types include arithmetic operations (add, sub etc.), comparison operations (<, > etc.), logical operations (AND , OR etc.), conversion operations (convert byte data into word data) , shift operations (right shift , left shift etc.) and data transfer operations. These instructions operate on packed data type. Figure 4.3 illustrates this property.
- support for integer, floating point, signed and unsigned arithmetic.

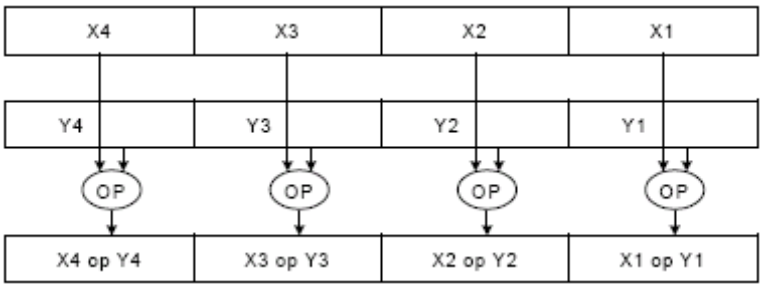


Figure 4.3 SIMD operations [11]

4.3 SSE Technology

Intel SSE (streaming SIMD extension) further extends the power of MMX technology by introducing more instructions (SSE, SSE2 and SSE3) and specifically providing 8 128-bit XMM registers on chip [11] [18]. The introduction of these 128-bit registers Pentium III onwards meant doubling the data processing power of MMX registers and execution of regular floating point instruction without interference when using SIMD instructions. Other than the extending the above mentioned instructions to 128 bits, SSE technology introduces instructions for operations like sum of absolute difference (PSADBW), masking (PMOVMSKB), average (PAVGB) ,finding minimum and maximum value (PMINUB, PMAXUB) and shuffle (PSHUFW). By providing instructions, which execute in a single clock cycle for these extensively used operations, Intel offers an opportunity of achieving significant optimization of computationally intensive applications like multimedia running on its processors.

4.4 Application of SIMD

SIMD technology operates on data arranged in a particular format. Therefore, usage of SIMD for optimization of an application like H.264 decoder needs the data in the application to be arranged in that format [12]. Basically, the data arrangement to efficiently use SIMD style processing should be converted to structure of arrays (SoA) format. An example of the concept stated is if the following operation needs to be implemented on 8 bytes of data $x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4$ in the memory

$$\text{Sum} = x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$$

Then before SIMD instructions can be used for this operation, the data needs to be rearranged in the memory as $x_1\ x_2\ x_3\ x_4\ y_1\ y_2\ y_3\ y_4$. Such a rearrangement of data will allow the first 4 bytes of data (all x) to be read in one SIMD register and next 4 bytes (all y) to be read into another SIMD register. Then 4 multiplication operations can be performed in parallel. But then a weakness of SIMD technology is encountered as no operations can be performed within the same register. Therefore, results of the 4 multiplications can be added only 1 addition operation at a time.

4.5 SIMD Optimization of High Profile H.264 HD Decoder

H.264 is an emerging video compression standard, which shows more compression capabilities than achieved by previous standards [5] [7] [8] [9]. H.264 provides higher coding efficiency through added features and functionality which also entail additional complexity in encoding and decoding processes [1]. The features are classified into profiles with each profile defined with a specific target application. In July

2004, addition of High Profile through an amendment called fidelity range extension (FRExt) [9] [10] further improves the performance of H.264 compared to previous and contemporary video compression standards but further adding to already computationally intensive encoding and decoding process. Moreover, the application of the codec for compression of HD (high definition – 1280x720p) data presents a significant challenge at the single processor desktop level [1] [4] [7]. The sheer volume of data to be processed puts a severe strain on the computational capabilities of even the fastest processors available at the desktop level. On the other hand, for H.264 to be popular among the consumers as the next generation of video compression algorithm and a replacement of the widely used MPEG-2 [18], real-time performance from at least the decoder is expected at the desktop level [3]. SIMD optimization is one of the methods which can contribute significantly towards achieving that goal.

4.5.1 Performance Analysis of H.264 Decoder

Application of SIMD requires a careful performance analysis of the target application which can reveal the "hotspots" in the application code. The "hotspots" in the application code are sections of code which utilize maximum of processor time out of the total time taken by the application to complete one execution. Application profilers are used for the task of performance analysis. For the performance analysis of FastVDO LLC's [37] H.264 High profile decoder, the profiler used is Intel's VTune performance analyzer [20]. The Intel VTune Performance analyzer is a powerful software-profiling tool for Windows and Linux. The VTune analyzer helps in understanding the

performance characteristics of a software application at all levels: system, application and microarchitecture. The VTune analyzer gives results in three formats:-

- Sampling : It can be time based and event based
- Call Graph: Flow of control in the application
- Counter Monitor : Monitoring of system based counters

For the purpose of H.264 decoder, time based sampling gives the hotspots in the decoding process. Accordingly, the targets for SIMD optimization can be identified. Figure 4.4 shows a sample graph generated at the system level by using Intel VTune performance analyzer while running the H.264 decoder. It shows that the complete decoding process is taking 37% (% Clockticks) of the processor time for the decoding test.

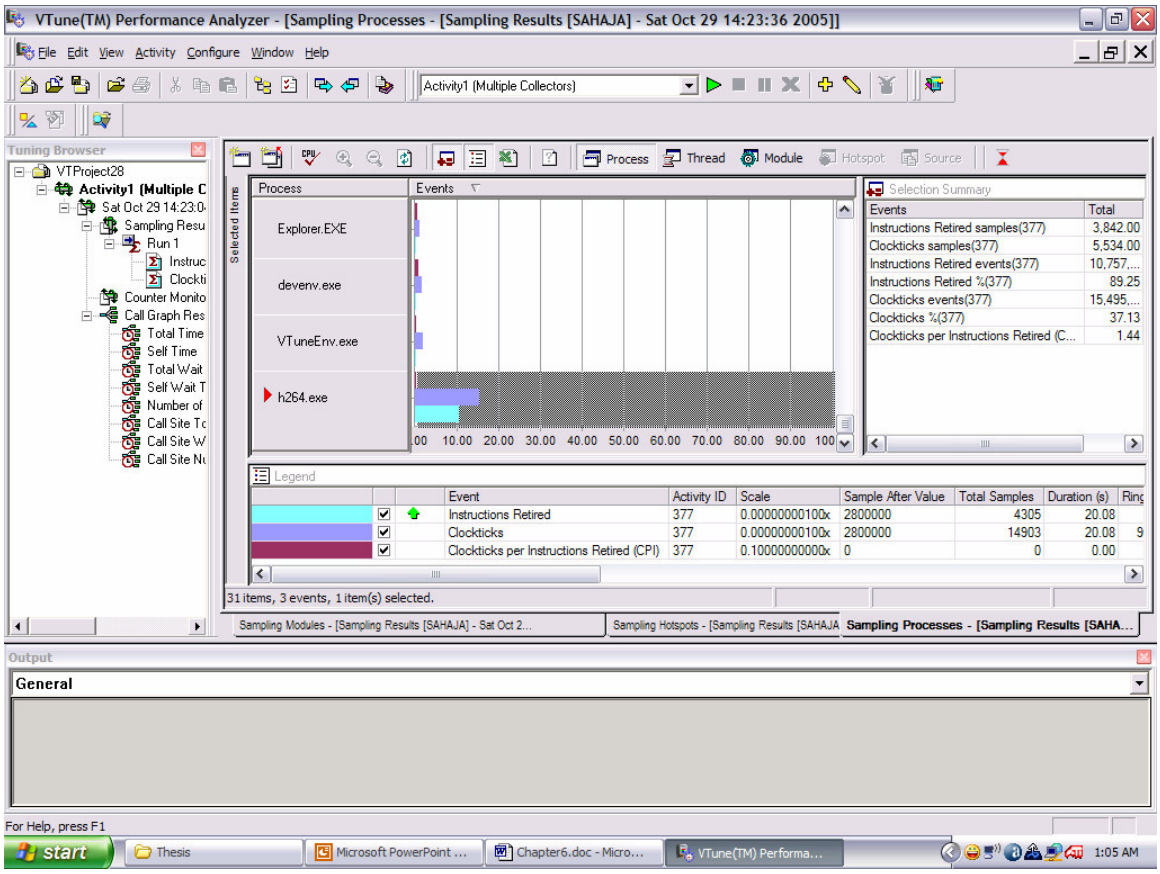
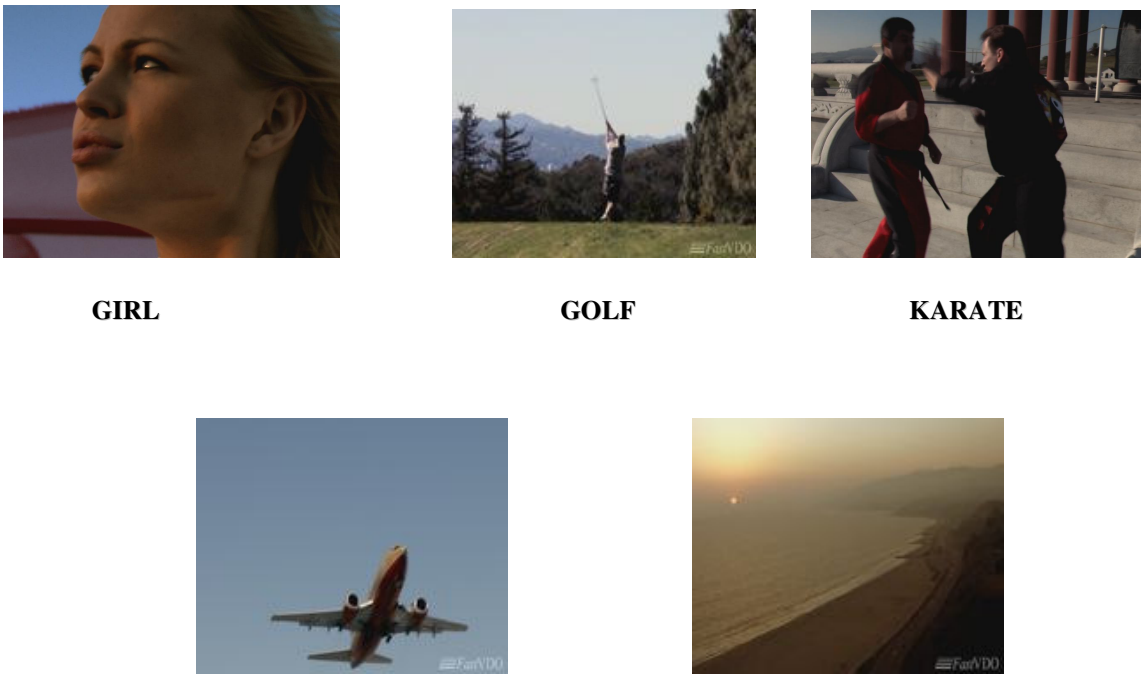


Figure 4.4 Results of VTune Analysis of H.264 Decoder -System Level

Table 4-1 lists the test conditions for performance analysis of H.264 decoder at the desktop level. Figure 4.5 shows the snapshots of the video sequences [ ] used in performance analysis of the H.264 decoder and to further test the optimization achieved by application of SIMD techniques for optimization.

Table 4.1 Test Setup for performance analysis of H.264 decoder

Platform	Pentium 4 Mobile 2.8 GHz, 702MB RAM, Windows XP
H.264 Decoder	FastVDO LLC High Profile H.264 Decoder© [37]
Streams	Girl, Golf, Karate, Plane, Shore (shown in Fig. 4.5) [37]
Encoder	JM 9.6 [36]
Encoding Parameters	See Appendix A
Profiler	Intel VTune Performance Analyzer



GIRL **GOLF** **KARATE**

PLANE **SHORE**

Figure 4.5 Streams used for performance analysis

4.5.2 Results of Performance analysis of H.264 Decoder

Results of time-based sampling of H.264 decoder using Intel VTune performance analyzer are listed in Table 4-2. It lists the segregation of total decoding time in terms of time spent in individual modules as % of the total decoding time. Figure 4.6 shows the graphical representation of the data listed in Table 4-2. The results indicate that the most time consuming computation modules in the H.264 are IDCT 4x4 (inverse discrete cosine transform) and motion compensation. Therefore, these modules are selected as the target of application SIMD optimization.

Table 4.2 Results of performance profiling of the H.264 high profile decoder

%	Girl.264	Golf.264	Karate.264	Plane.264	Shore.264
IDCT 8x8	4.957879	0.956737	3.2735859	1.894126	1.63884
CABAC	11.026452	5.293592	10.335945	10.01807	6.407274
Memcpy + Memset	13.33369	16.86905	11.849307	11.01987	14.59611
IDCT 4x4	17.02527	20.39636	15.568315	12.89757	17.79446
MC	29.53265	38.00137	40.045078	50.16149	41.66314
Others	24.12405	18.48289	18.927766	14.00887	17.90019

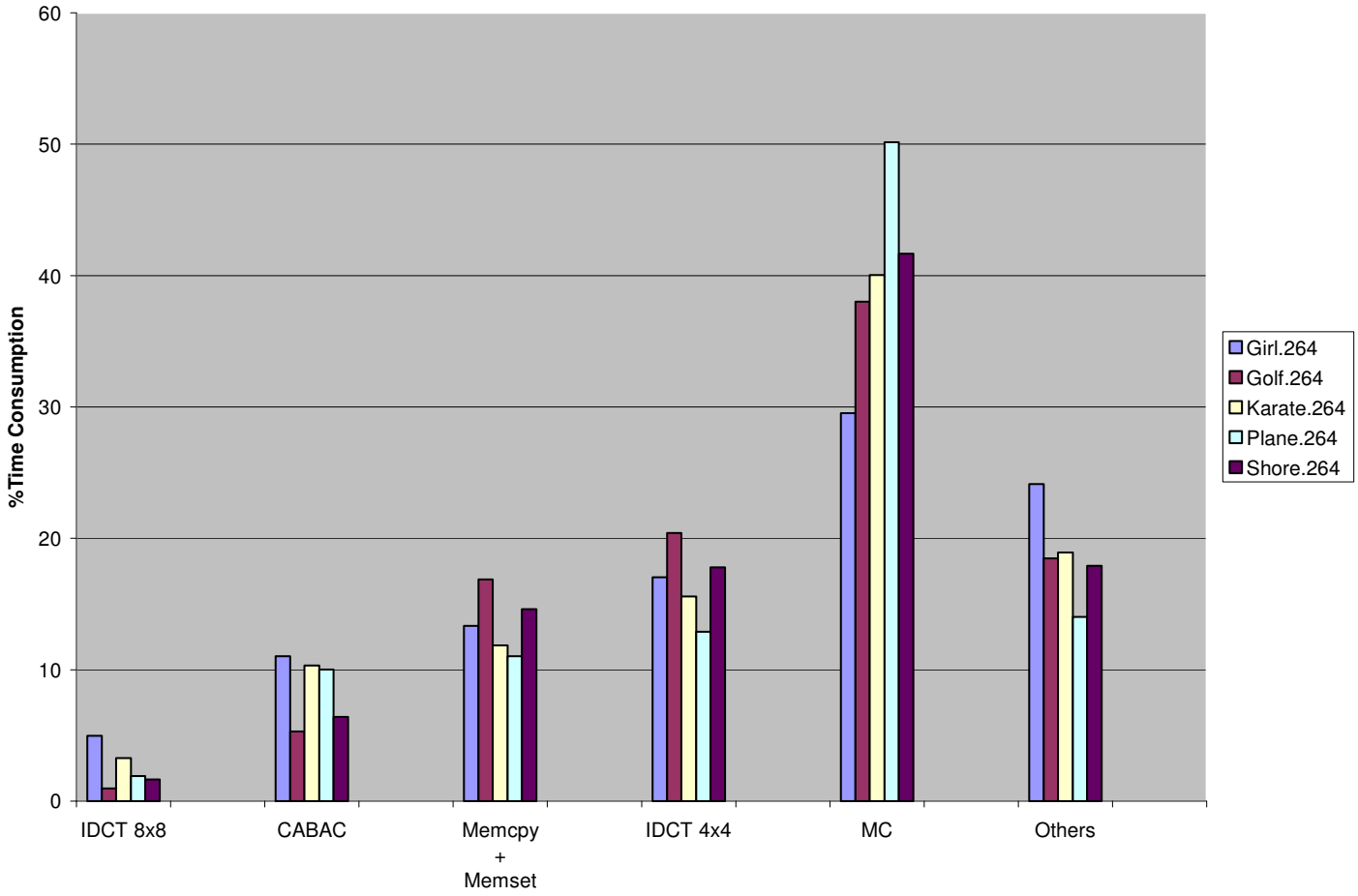


Figure 4.6 Results of time based sampling of H.264 Decoder

4.5.3 Application of SIMD Optimization to IDCT 4x4 transform in H.264

The H.264 standard uses an integer approximation of the discrete cosine transform which operates on 4x4 block of data after motion-compensated prediction or Intra prediction. The steps of implementation [4] [21] of the 1-D inverse transform are listed as follows: -

ei0 = di0 + di2, with i = 0..3

ei1 = di0 – di2, with i = 0..3

ei2 = (di1 >> 1) – di3, with i = 0..3

ei3 = di1 + (di3 >> 1), with i = 0..3

fi0 = ei0 + ei3, with i = 0..3

fi1 = ei1 + ei2, with i = 0..3

fi2 = ei1 – ei2, with i = 0..3

fi3 = ei0 – ei3, with i = 0..3

where, eii is the input to the transformation process. The resulting 4x4 matrix obtained after 1-D IDCT is then transposed and then 1-D IDCT is applied to its results to get the results of the 2-D IDCT 4x4 module. The final coefficients are obtained by adding 32 and dividing (right shifting) by 64.

In the 2-D IDCT process, since the operation remains the same for all 4 pixels, 4 resulting values of each of the operations mentioned above can be calculated in parallel (for e.g. using the instruction PADDB, PSUBB). This will lead to saving of crucial decoding time. Transpose of 4x4 matrix can be implemented using the PUNPCKLBW/ PUNPCKHBW.

Table 4-3 shows the comparison of the performance of implementations of 2-D IDCT 4x4 with and without SIMD. Figure 4.7 shows the speed improvement in the 2-D IDCT 4x4 module achieved by SIMD implementation of 2-D IDCT 4x4 over the non-SIMD implementation. Depending upon the % time consumed by the 2-D IDCT 4x4 module of the total time taken by the complete decoding process; speedup in the overall decoding process can be calculated using the Amdahl's Law (Appendix B). The speed improvement in the overall decoding process is approximately 6-10 % depending upon the type of motion in the sequence. If the motion in the sequence is high, then the time taken by 2-D IDCT 4x4 module is lower compared to when motion is low. Therefore, overall speedup by optimizing 2-D IDCT 4x4 module using SIMD is more in case of low motion sequences.

Table 4.3 Results of SIMD optimization of the IDCT 4x4 block

%	Girl.264	Golf.264	Karate.264	Plane.264	Shore.264
No SIMD (%)	17.02527	20.39636	15.56831	12.89757	17.79446
SIMD (%)	8.811405	11.61393	9.16206	7.050434	9.89151
Speedup Factor	1.932186	1.756198	1.69921	1.82993	1.79896
Overall (%) Speedup	8.9489	9.628	6.8447	6.21	8.581

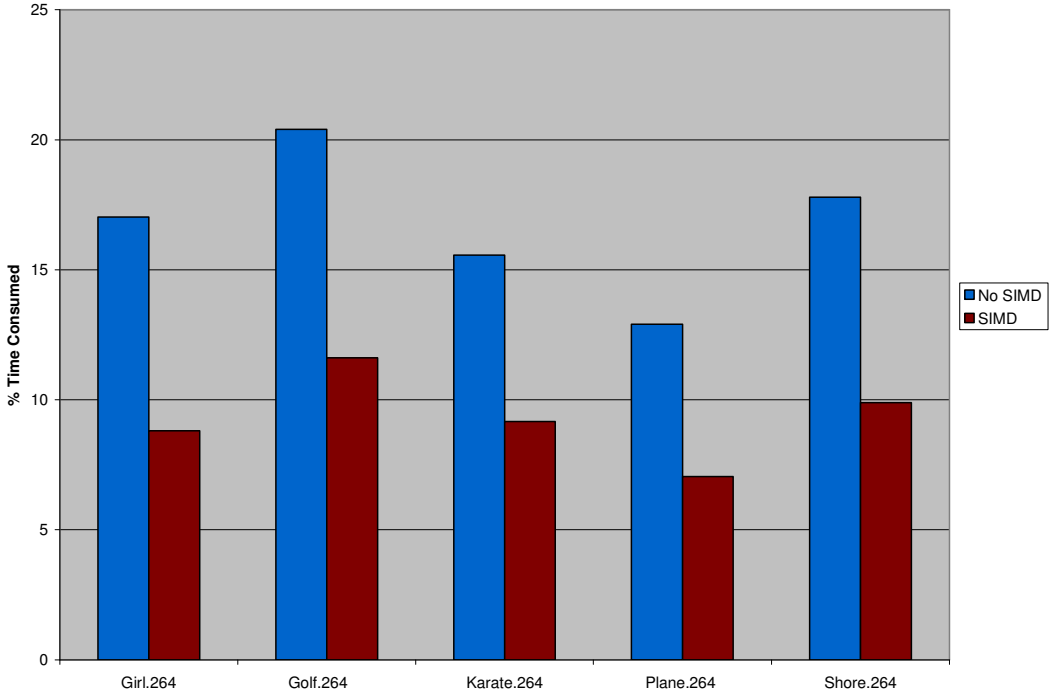


Figure 4.7 % Time consumed by 2-D IDCT 4x4

4.5.4 Application of SIMD Optimization to Motion Compensation

The motion compensation module [4] [5] [6] [7] in the H.264 standard can be classified into two sub-modules:-

4.5.4.1 Data Manipulation

The data manipulation module in motion compensation process is involved in the process of communication with the inverse transform and quantization module and decoded picture buffer (DPB). The task of the data manipulation module is to get the motion vectors for the target block from the bitstream, check for type of prediction used (skip, intra, direct etc. [4] [5] [7]) and search for the reference frame from the decoded

picture buffer if needed. Then the block from reference frame is copied into the interpolation sub-module. Since there are no computations involved, therefore application of SIMD optimization to this sub-module would not lead any performance improvement.

4.5.4.2 Interpolation

The interpolation sub-module in the motion compensation module performs all the arithmetic calculations of sub-pixel interpolation and linear interpolation required for generation of reference block in the reference frame. The H.264 standard allows the motion estimation process at the ½-pixel and the ¼ - pixel levels. Once the reference block is obtained from the reference frame, it can be converted to ½-pixel resolution applying the 6-tap filter [1 -5 20 20 -5 1]/32 and rounding the result. The resulting block can be further interpolated to ¼ - pixel resolution by linearly interpolating the horizontally or vertically adjacent ½-pixel or full resolution pixels. For the B-frames the final block used for compensation is derived by applying linear interpolation to the two prediction blocks representing forward and backward prediction. The chroma sample interpolation for the 4:2:0 format needs $^1/_8$ – pixel resolution if the luma component of the target block uses ¼ - pixel resolution. The $^1/_8$ – pixel resolution interpolation is generated as a linear combination of the neighboring integer sample position.

Figure 4.8 shows the segregation of % decoding time consumption by the motion compensation module into % time taken by the sub-modules of data manipulation and interpolation. It is evident from the plot that as motion increases in a video sequence, so does the decoding time spent in sub-module of interpolation.

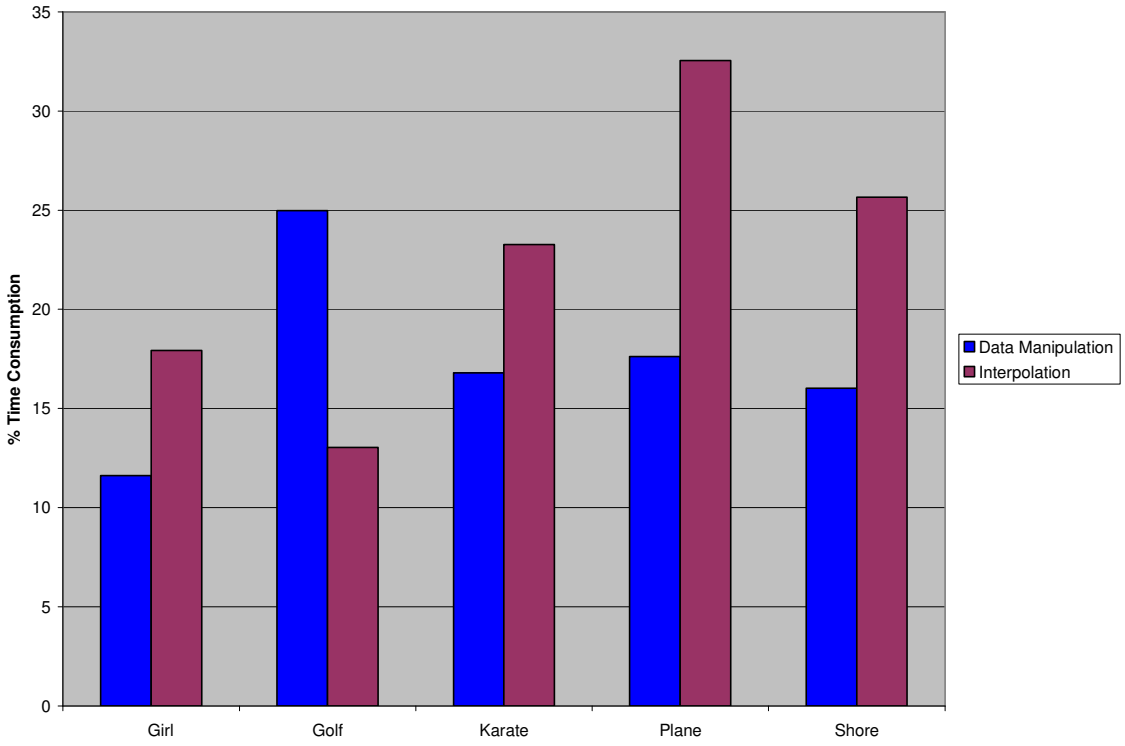


Figure 4.8 Segregation of % time consumption by motion compensation

4.5.4.3 SIMD Implementation of Interpolation

The linear interpolation section is similar to 2-D IDCT 4x4 block in terms of operations involved (shift, addition). Therefore, using packed arithmetic instructions helps in optimizing the module of linear interpolation since same operation is performed on all the pixels of the block. The main section of the interpolation is the ½-pixel resolution applying the 6-tap filter [1 -5 20 20 -5 1]/32. The filtering can be done by initially performing the multiplication (by 20 for 3, 4 positions and by -5 for 2, 5 positions) using PMULLW for a set of pixels and adding the results to get intermediate

pixel values which are then rounded to get the final set of pixels with ½-pixel resolution . Then linear interpolation of adjacent full and ½-pixel resolution values results in ¼ - pixel resolution values. Figure 4.9 shows the comparison of % time consumed of the overall decoding time the interpolation sub-module of the motion compensation process with and without SIMD. It shows that using SIMD for interpolation leads to speedup of the module by a factor of approximately 1.7.

Table 4.4 Results of comparison of SIMD optimized Interpolation with non-SIMD Interpolation

%	Girl	Golf	Karate	Plane	Shore
NO SIMD (%)	15.96824	13.02634	23.25319	32.54503	19.7399
SIMD (%)	9.51832	7.53874	14.32317	19.7414	11.608
Speedup Factor	1.68	1.73	1.62	1.65	1.7
Overall(%) Speedup	6.89	5.8	9.8	14.68	8.85

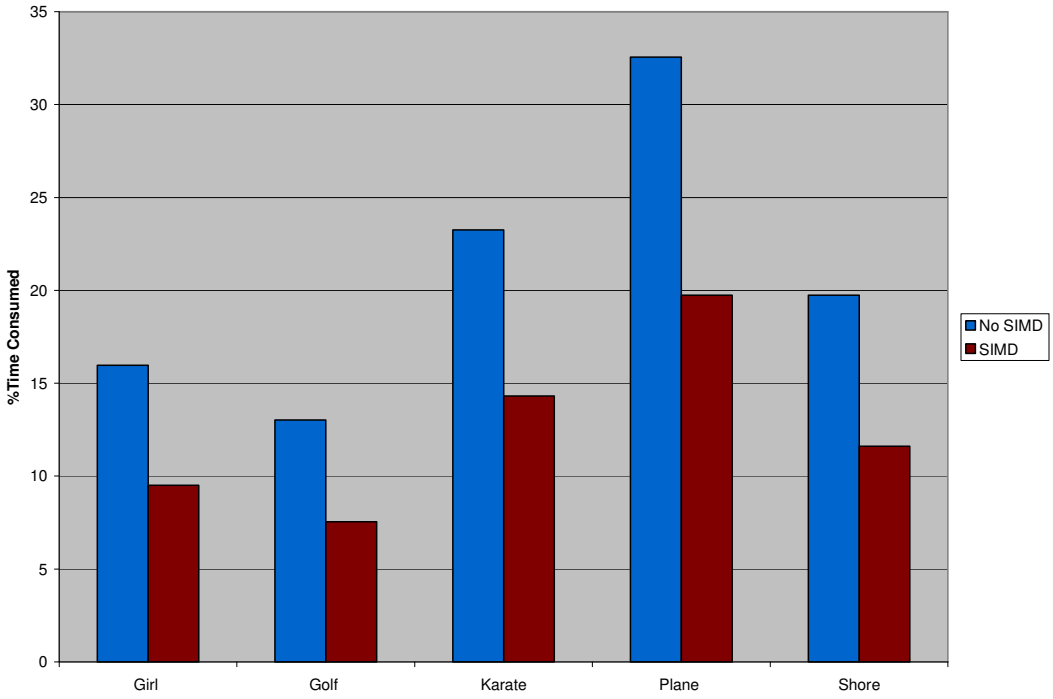


Figure 4.9 % Time consumed by interpolation

4.6 Summary

This chapter has presented the basics of implementation of single instruction multiple data technology in the Pentium 4 processor and its application for optimization of the H.264 decoder for HD video sequences. The performance of the H.264 decoder was analyzed using the Intel VTune performance analyzer to identify the hotspots in the application code of the decoder. Accordingly, the SIMD optimization was applied to two of the most time consuming modules of the H.264 decoder namely, 2-D IDCT 4x4 and the interpolation process in motion compensation. The results show significant reduction

in % of decoding time utilized by SIMD implementation of these modules as compared to pure high-level language (C, C++) implementation.

CHAPTER 5

MULTITHREADING

Multiprocessing is one of the key techniques for getting more performance from a system by performing more than one action at a time. The most common application of multiprocessing is the operating system e.g. *Windows XP* [22] where it is used to hide the latency associated with access to system resources. For software optimization, it is used to do more work in less time. Instruction-level parallelism gives the processor the ability to execute more than one instruction at the same time. The application performance can be improved even further by using multiple threads and increasing the level of parallelism.

5.1 Basics of Multithreading

On an operating system like Windows every application consists of one or more processes. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a base priority, minimum and maximum working set sizes, and at least one thread of execution. Each process starts with a single thread, often called the primary thread, but can create additional threads from any of its threads.

A thread is the entity within a process that can be scheduled for execution. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. [23] [24] [25]

When an operating system supports multitasking on a single processor system, it divides the processor time into *time slices* and allocates them to each of the running threads. After the time slice elapses, the execution of the thread using that time slice is suspended. Since the time slices allocated are very small, it gives an impression of the threads running simultaneously. On a multiprocessor system, the execution load is distributed among multiple processors and actual multiprocessing takes place with background process offloaded to the second processor. But running too many threads on the system can degrade performance, as more the number of threads are the longer each thread has to wait for its time slice of execution.

There are two ways of implementing multitasking:-

- Dividing the application into more than one process or *multiprocessing*, each executing independently consisting of one or more threads.
- Dividing the single process application into multiple threads or *multithreading*.

It is more efficient to implement *multithreading* rather than *multiprocessing* for the following reasons [24]:-

- The context switching process is easier between threads than between processes as process context is larger than thread context
- The threads share the address space and the global variables which simplify inter-thread communication and synchronization.
- All threads of a process can share open handles to resources.

5.2 Multithreading Strategies

To achieve, performance improvement by multithreading the application can be divided into tasks and data. Analysis of application in terms of tasks and data will allow the programmer to explore the possibilities of implementing one of these two types of parallelisms [12] :-

5.2.1 Data Level Parallelism

Applications that operate on large data sets can divide the calculations among multiple threads based upon the different sections of data. For example different filters can be applied to different sections of decoded frame data in a video coding application. Since the parallelism is driven by data, it is called data parallelism.

5.2.2 Task Level Parallelism

- *Coarse grained*: Multiple tasks performed at the same time
- *Fine grained*: Each task divided into independent sub-tasks.

Therefore, if the threads of an application are organized by tasks, they implement task level parallelism and if they are organized by data, they implement data level parallelism. In both cases, it is important to achieve maximum processor utilization by load balancing and minimizing overhead incurred by creating, managing and synchronizing the threads. By maximizing the work each thread performs individually, multithreading overhead can be minimized.

5.3 Thread Synchronization

As mentioned earlier, multiple threads in an application process share the system resources allocated to it. Therefore, for a programmer it raises the question of synchronization of thread execution so that conflicts between the threads do not occur and degrade performance. The most common types of thread conflicts [R] are:-

- "*Race condition*": It occurs when an assumption that one thread will complete execution before its results are needed by another thread fails.
- "*Deadlocks*": It occurs when two threads block have blocked each other's execution and can only unblock that execution by proceeding.

The following methods can be used to prevent thread conflicts:-

- "*Semaphores*": They block execution of the thread at a certain point in the code until another thread signals it can resume.
- "*Critical Sections*": They mark a section of the code which cannot be interrupted by the operating system so that race conditions can be avoided.

5.4 Producer-Consumer Problem

The producer-consumer problem illustrates one of the methods of implementation and importance of thread synchronization. The producer thread puts data in the buffer and the consumer thread performs operations on that data and then updates the buffer.

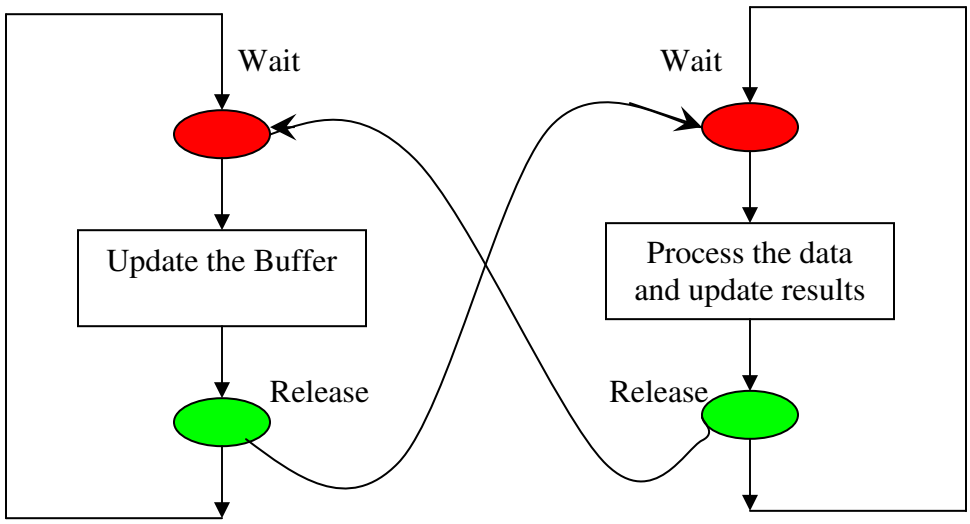


Figure 5.1 Producer-Consumer Thread Synchronization

The process of thread synchronization can be described as:-

- The primary thread creates the producer thread, consumer thread and the semaphores needed for synchronization.
- The control is then passed to the producer thread.
- The producer thread waits for the consumer thread to release control of the buffer. For the first time, this condition is true.
- The producer thread then proceeds to load the buffer with data.

- The producer thread then releases control of the buffer to the consumer thread.
- The consumer thread waits for the producer to release control and then takes the buffer data, processes it and updates the buffer.
- The consumer thread then releases control of the buffer to the producer thread.
- When the complete data set has been processed, a flag can be set to indicate to the threads to quit execution or the threads can be *killed* from the main thread.

5.5 Summary

This chapter briefly discussed the basic concepts of threads and multithreading. It listed some of the pitfalls that can occur due to multithreading and methods of thread synchronization. An example of usage of thread synchronization in the form of producer-consumer problem has been presented.

CHAPTER 6

MULTITHREADING GOP LEVEL H.264 DECODER

As mentioned in the previous chapter, multithreading is one of the methods of achieving performance improvement in an application. This chapter explores the application of multithreading to the H.264 decoder and presents a design for the same. Finally, the performance comparison between single threaded and multithreaded H.264 decoding is presented.

6.1 H.264 Decoder Design-Strategies

To design a multithreaded H.264 decoder, two possible approaches can be used:-

6.1.1 Task-level

- As shown in Fig. H.264 decoder pipeline consists of various stages through which the encoded bitstream passes through to generate the decoded data. Each stage for e.g. IDCT, Deblocking etc. can be seen as a task each of which can be performed as a separate thread. But implementing multithreading at the task level in the H.264 decoder can lead to the following problems:-
- The tasks share buffers among them and independent execution of each task will involve complicated synchronization of buffer access and lead to significant threading overhead.

- The decoding algorithm is sequential, which will lead to threads further in the pipeline to wait for the results from the threads earlier in the pipeline. This will lead to performance degradation compared to a single threaded decoder implementation.

6.1.2 Data-level

The second form of implementation of a multithreaded H.264 decoder can be done using data-level parallelism, which will use multiple decoder instances to decode the bitstream. This type of design would require for the bitstream to have sections which can be decoded independently. To efficiently implement a multithreaded H.264 Decoder that will decode different sections of the bitstream simultaneously, there should be minimal or no interaction between the different decoder threads.

This requirement is important as neglecting it can lead to the following threading pitfalls [24] [25]: -

- Increase in threading overhead
- To prevent “deadlocks” when two threads try to access the same variables and buffers shared between them. e.g. reference frame buffers.
- To prevent “race conditions” when one decoder thread tries to access data from the other decoder thread buffer, which hasn’t finished updating the buffer.

6.2 H.264 GOP level Decoding

The key to implementing a multithreaded H.264 decoder is the concept of using Group of Pictures (GOPs) in the encoding process of the bitstream. The H.264 like other video compression algorithms derives its compression from exploiting temporal correlation between frames or fields (or slices in case of H.264). The compression is obtained by encoding the difference between the actual frame and the corresponding predicted frame derived using previously encoded frames which occur in the past and the future in the decoding order (in case of P and B frames). In H.264, the previously encoded frames are maintained in the buffer known as a list [4] [6]. The reference frames or pictures are of two types: -

1) Short term reference: pictures which are regularly replaced using the "sliding window" memory control as the new pictures are decoded and sent to the decoded picture buffer.
2) Long term reference : pictures which are older in the decoding sequence and are removed explicitly by commands

If a bitstream not using GOPs in the encoding process is decoded using a multithreaded decoder then depending upon the bitstream it can possibly lead to the threading pitfalls mentioned above. Basically, if the first frame of the GOP cannot be decoded without a reference to the decoded picture buffer, then the frame cannot be decoded until the reference frame required by the current frame is decoded and made available for reference. This will lead to the current thread being stalled and the

parallelism in the decoding is lost. This problem does not end at the first frame. If any frame of the GOP n needs a reference from a decoded frame of GOP m, it cannot proceed with the decoding process until that reference frame is made available. The solution to this problem is sending an IDR frame at the start of each frame. An IDR or Intra Decoder Refresh [4] [7] frame is an I-frame which can be decoded without a reference frame from decoded picture buffer. Also, on receiving an IDR coded picture, the decoder marks all pictures in the reference buffer as "unused for reference" or flushes the decoded picture buffer.

The implications of using an IDR picture at the start of each sequence are listed as follows: -

- Insertion of IDR pictures at regular intervals gives the options of going forward and reverse in the video sequence to the media player. The process of going forward and reverse in a video or an audio sequence is known as "seeking" [40]. The process of seeking to a particular position in the bitstream means that the video decoder should have the capability of starting the decoding process from that position. For example, the design of the multithreaded decoder presented later in the chapter uses streams which are encoded using an IDR frame every 30 frames to form a GOP of 30 frames . Assuming a decoding rate of 30 frames per second, having an IDR frame every 30 frames would give the media player using the H.264 decoder seeking precision of 1 second. The capability of "seeking" is a must for a media player in playback of video material like movies stored on data

disks. An example of the application of "seeking" is playback of chapters in a DVD [40].

- Inserting an IDR frame every 30 frames also provides robustness and error correction capabilities to the decoder also with the precision of 1 second i.e. if an entire GOP or part of is lost in transmission over a noisy channel or contains encoding errors, the decoder will be able to decode from the next GOP (assuming that GOP is received error frame) and error is thus localized to that particular GOP. Also insertion of I-frames every few frames help in reducing drift due to predictive coding [24].
- Encoding Time Reduction: Insertion of I-frames also reduces the encoding time of the sequence as encoding using Intra- prediction for a frame is much faster than using Inter-prediction as Motion Estimation is one of the most time consuming block of the whole encoding process. In the H.264 standard, number of intra prediction modes [4] [5] [6] has increased thus making the process of choosing one of those modes for encoding a block of data more time consuming. But the process of inter-prediction in the H.264 standard [4] [5] [6] has also become more advanced resulting in complexity and time consumption by using reference lists even for P-frames, allowing any frame in the list to be chosen as reference frame for prediction.
- Increase in Bitrate: Insertion of and I-frame after a B or P frame causes a sudden surge in bitrate. This surge in bitrate will be detrimental in applications using a channel with bandwidth constraints for transmission of data between the encoder

and decoder (live capture and transmissions) as it will overload the channel and slow the decoding and display process at the decoder side making it non real-time. The bandwidth constraint is not a factor in case of playback of previously encoded bitstream as in the case of DVD. Also the surge in the bitrate can be curtailed by using a good rate control algorithm which can be set to choose higher quantization for an I-frame.

6.3 H.264 Decoder –Threading Architecture

The architecture proposed for multithreaded GOP level decoding of H.264 is shown in Fig. 6.1. The architecture consists of following threads:-

6.3.1 Main Thread

The main thread performs the following functions:-

- Initialize the input interface.
- Initialize the storage buffers required for the decoded data.
- Create N number of H.264 decoder objects where "N" is specified by the user.
- Read the input bitstream and parse the bitstream to get SPS and PPS NALUs
- Initialize all N decoders using the SPS and PPS NALUs.
- Create 1 thread to manage the search of position of next N IDR NALUs in the H.264 bitstream (which is basically the starting of GOPs)
- Create N decoder threads
- Create semaphores to manage synchronization between the threads created.

6.3.2 Get IDR Position Thread

This thread manages the control of the decoding process once the main thread passes control to it. The Get IDR position threads searches for starting position of N GOPs in the bitstream, where N is the number of decoder threads running on the system. The N starting positions are then passed on to N decoders and thus decoding of N GOPs starts simultaneously. Figure 6.2 shows the operations and flow of control in the Get IDR position thread.

6.3.2.1 Get IDR Position Function

The Get IDR position thread used Get IDR Position function to search for NALUs containing IDR frames in the bitstream. This function searches for the start code in the bitstream and then compares the next byte immediately following the start code to determine if the data in the NALU belongs to an IDR frame or not. The type of NALU can be determined from the NALU syntax and Table 7-1 in the [4].

6.3.3 Decoder Threads

The decoder threads use the H.264 decoder objects to decode the frames in the GOP. The input to the decoder threads is a pointer to starting position of the GOP, which is the start of the NALU containing the IDR frame. The decoded output can be stored to a buffer from which a video renderer can take frame data and display it on the screen.

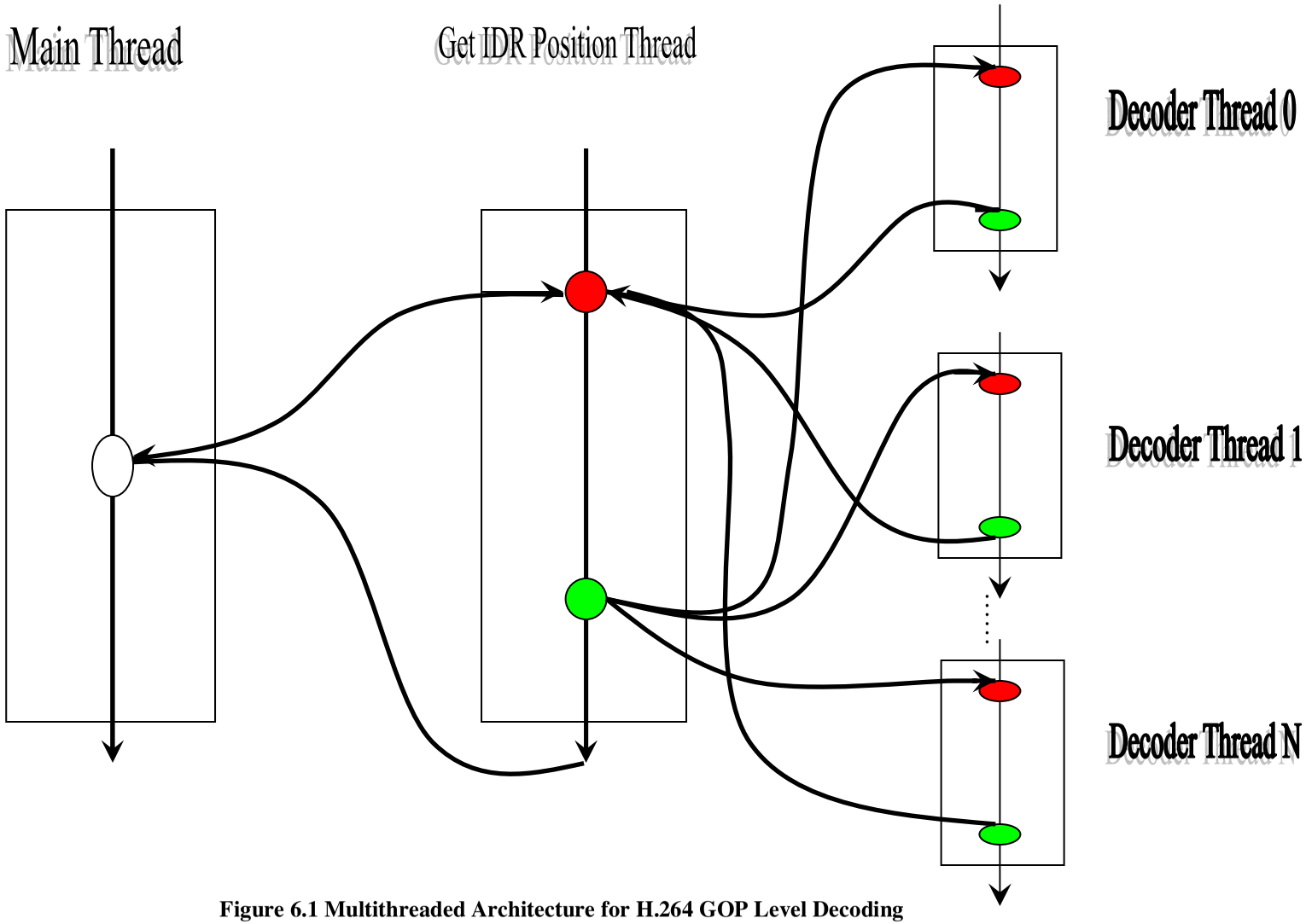


Figure 6.1 Multithreaded Architecture for H.264 GOP Level Decoding

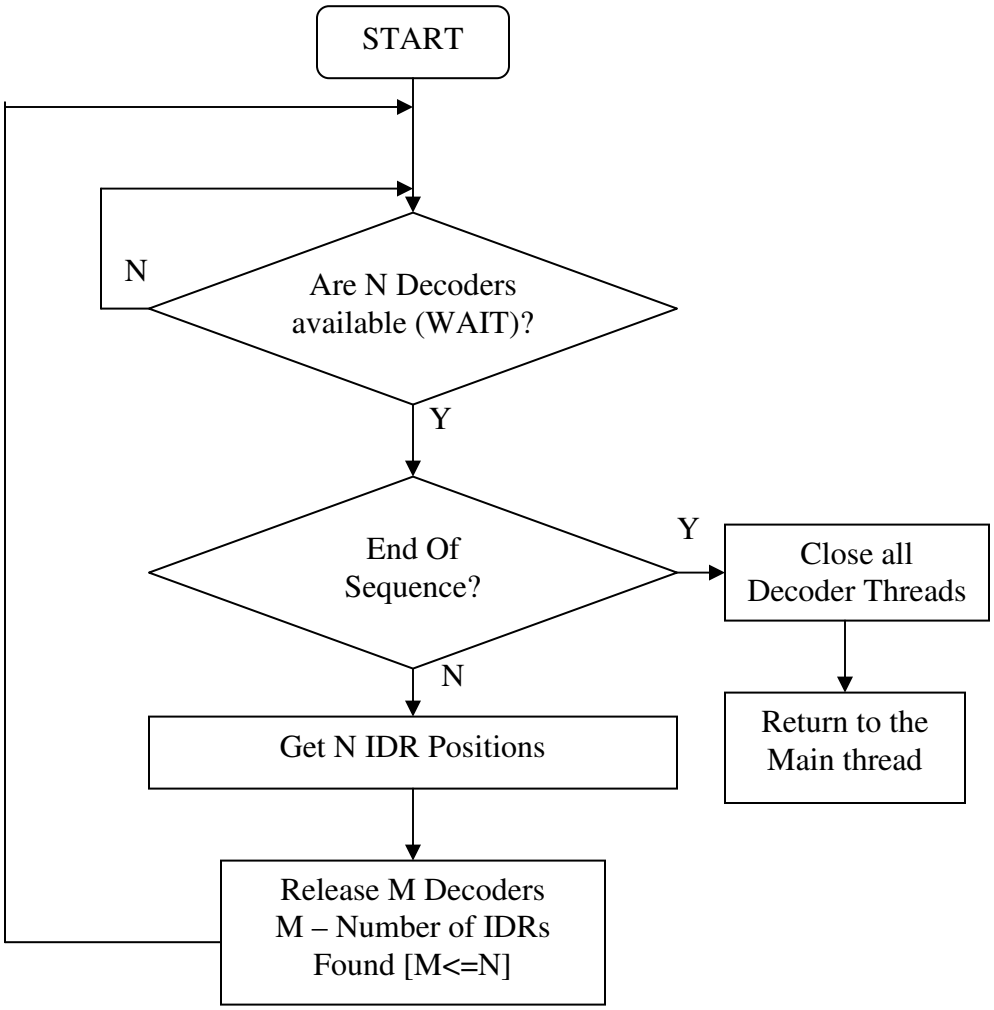


Figure 6.2 Flow Control in Get IDR Position Thread

6.4 Multithreaded GOP Level Decoding - Results

To compare the multi-threaded decoding to the single-threaded decoding of High Profile H.264 HD bitstreams following test setup was used:-

Platform : Pentium 4 HT Mobile, 2.8 GHz, 704MB of RAM

H.264 Decoder used : FastVDO LLC© H.264 High Profile Decoder

Performance Metrics : Decoding time in seconds

Resolution : 1280x720p

Encoder Used : JM 9.6 [36]

Encoding Parameters: Appendix A

Number of Decoder Threads used : 1,2,3,4

Algorithm Overhead: Time taken to search for starting position of NALUs containing IDR frames.

Figures 6.3-6.7 shows the % improvement (decrease) in the decoding time for 5 streams required using multiple threads as compared to the decoding time required using a single-thread.

Figure 6.8 shows % Time Overhead incurred by using the current architecture, which is fraction of the decoding time spent in searching for NALUs containing the IDR frames.

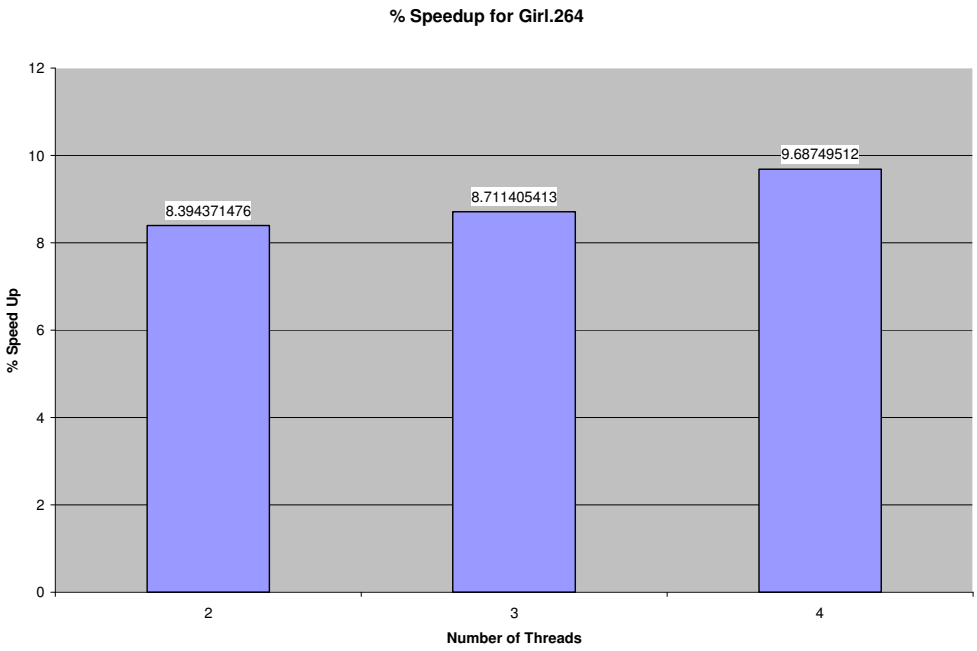


Figure 6.3 Speed up in decoding time for Girl.264

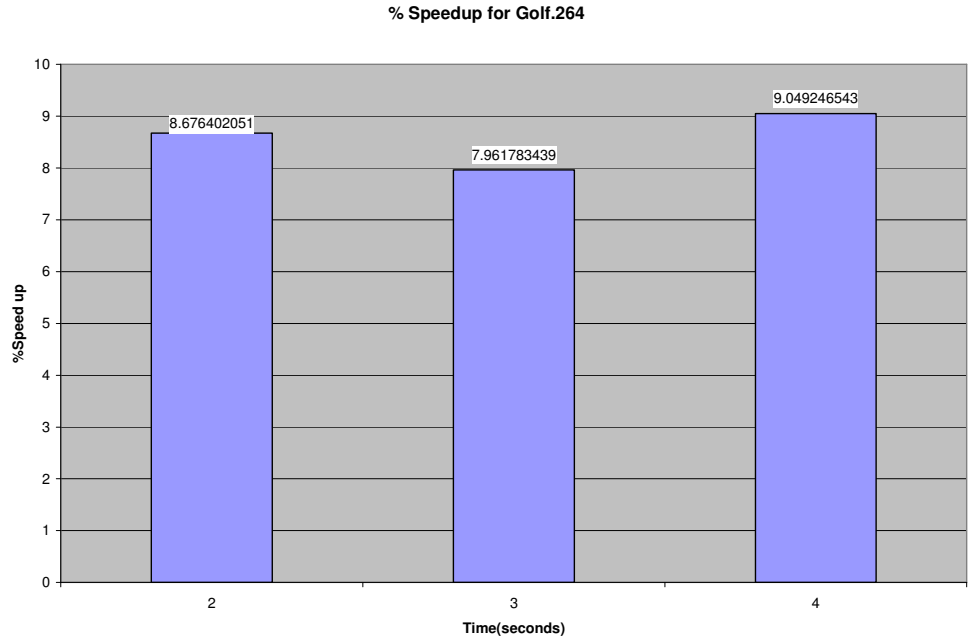


Figure 6.4 Speed up in decoding time for Golf.264

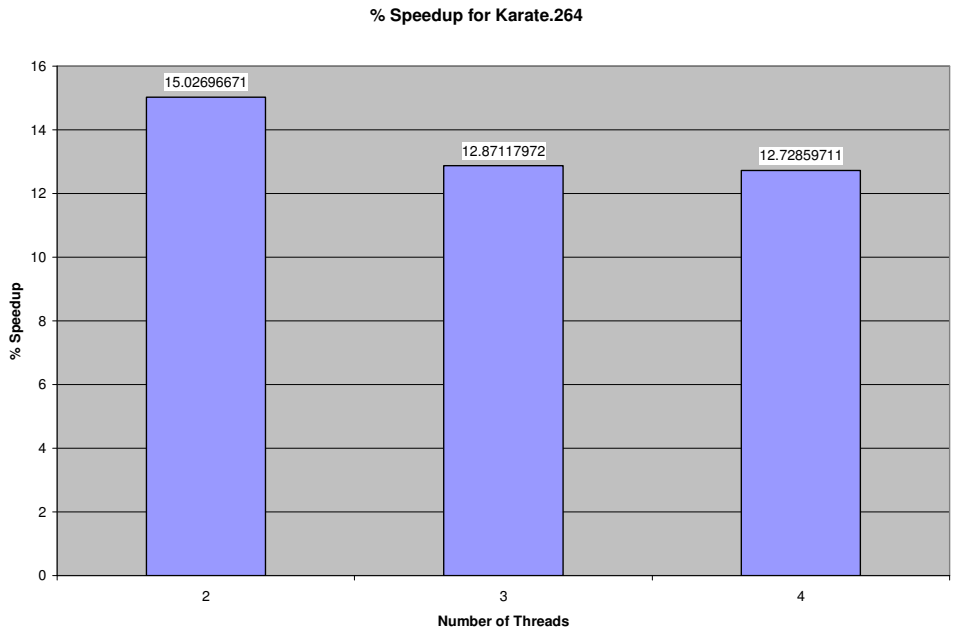


Figure 6.5 Speed up in decoding time for Karate.264

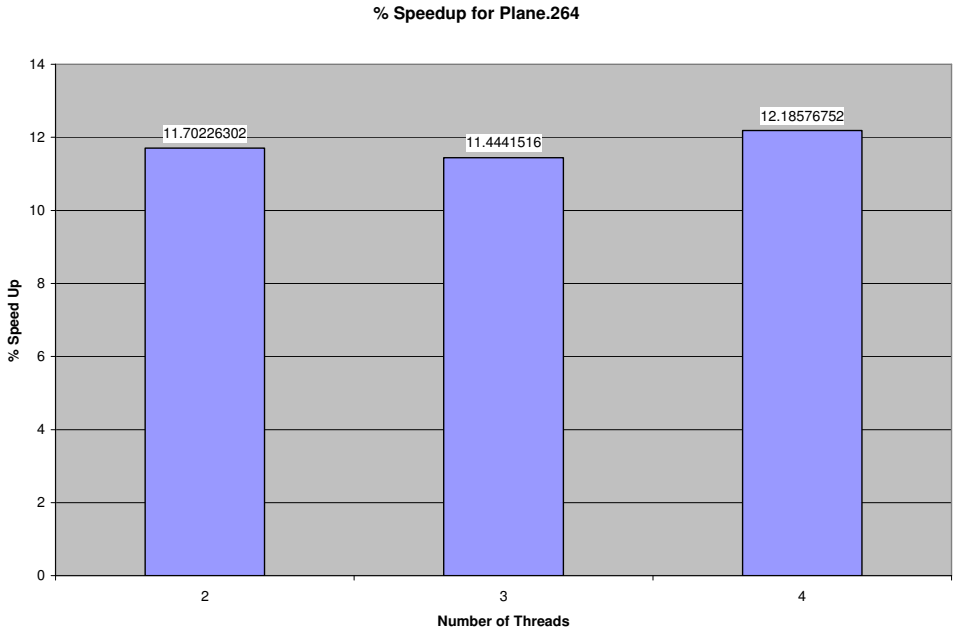


Figure 6.6 Speed up in decoding time for Plane.264

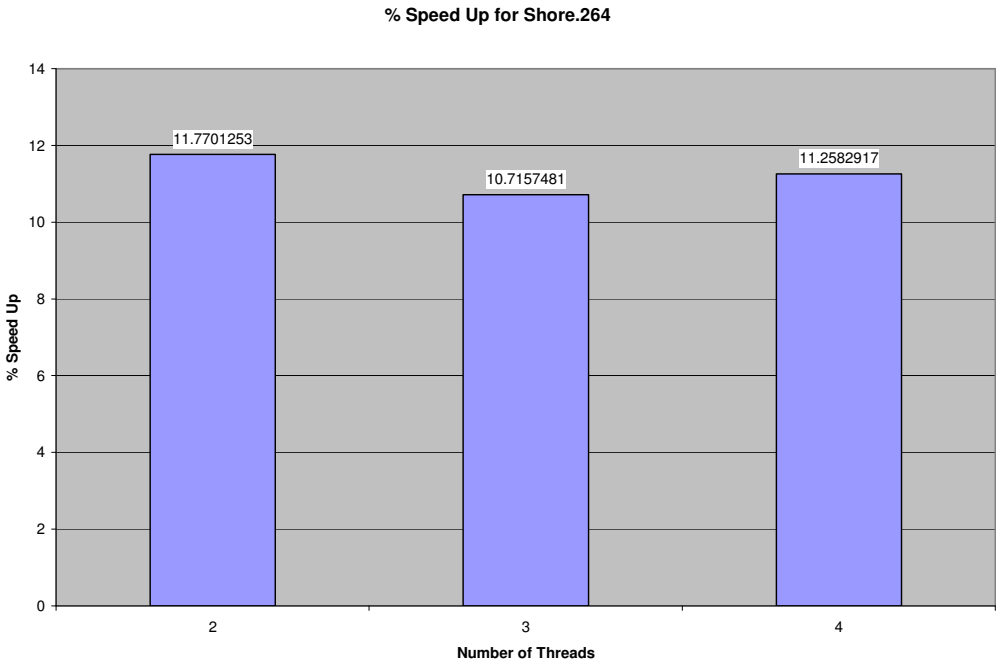


Figure 6.7 Speed up in decoding time for Shore.264

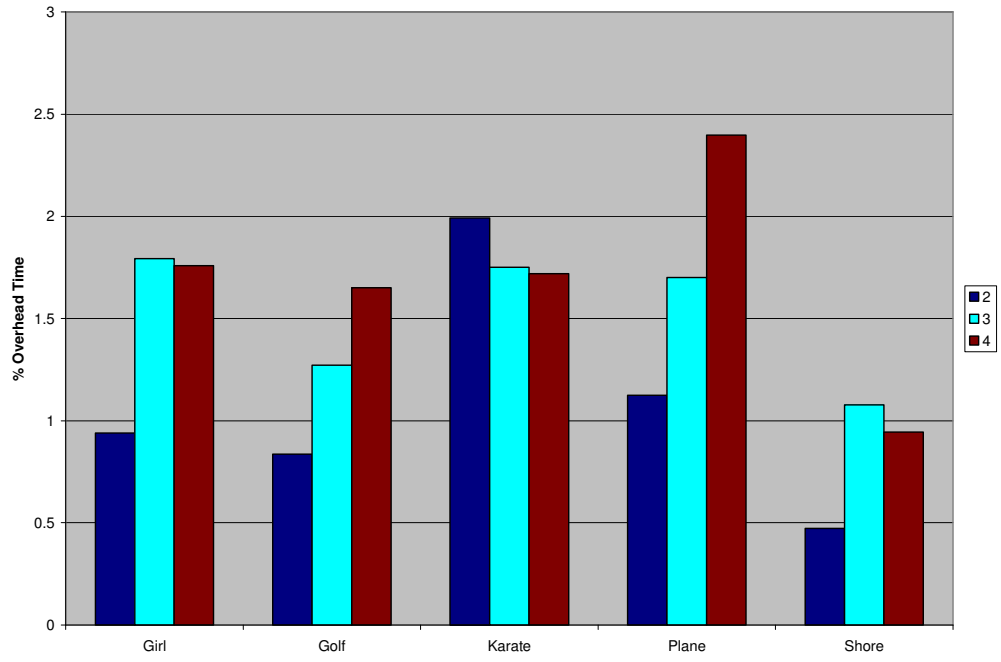


Figure 6.8 Overhead Time

6.4 Multithreaded GOP Level Decoding - Observations

As shown it can seen in Figs. 6.3 – 6.7, the decoding time for the streams using multiple decoder threads show a decrease of at least 8%. The best results are obtained by using 2 threads which is the actual capability of a Pentium 4 processor with HT technology [11]. If we increase the number of decoder threads, it leads to overloading of the processor and thus could lead to slowing down the application instead of boosting its performance. This fact is clearly visible in case of decoding of Karate.264 which shows maximum decrease in decoding time when using two decoder threads and percentage decrease in decoding time reduces as the number of decoder threads increase. The key factor determining whether increase in number of threads beyond two will improve or degrade performance is level of processor utilization achieved using a two decoder threads. If using two decoder threads have low processor utilization, then it indicates that the processor has some spare time slices available which are not being used by any process. Therefore, decoder threads more than two can be used can be used in the application to utilize those spare time slices and further boost the performance. Figure 6.9 shows the processor utilization while decoding Karate.264 using 2 decoder threads.

The second performance metric used in the performance analysis is the time spent in searching for NALUs containing the IDR frames or the start of GOPs. This metric is expressed as the time spent in the search as a % of the total decoding time. This time is classified as the algorithm overhead because the search of NALUs containing IDR frames in the bitstream is not required when a single decoder thread is used. However, this

overhead can be avoided if position of NALUs containing IDR frames is known beforehand. At the encoder side, the offset in terms of number of bytes from the start of the bitstream can be placed and sent to the decoder along with the bitstream. The H.264 standard provides a method of sending extra information in the bitstream which is not necessary for decoding purpose and a compliant decoder doesn't need to implement compulsorily. The method is to send additional information in the form of SEI messages [4]. NALUs containing SEI messages can be sent to the decoder at any position in the bitstream according to the requirement of the information. The decoder can choose to decode and use the information or simply discard the NALU. Another example of usage of SEI messages in the bitstream is logo insertion [27].

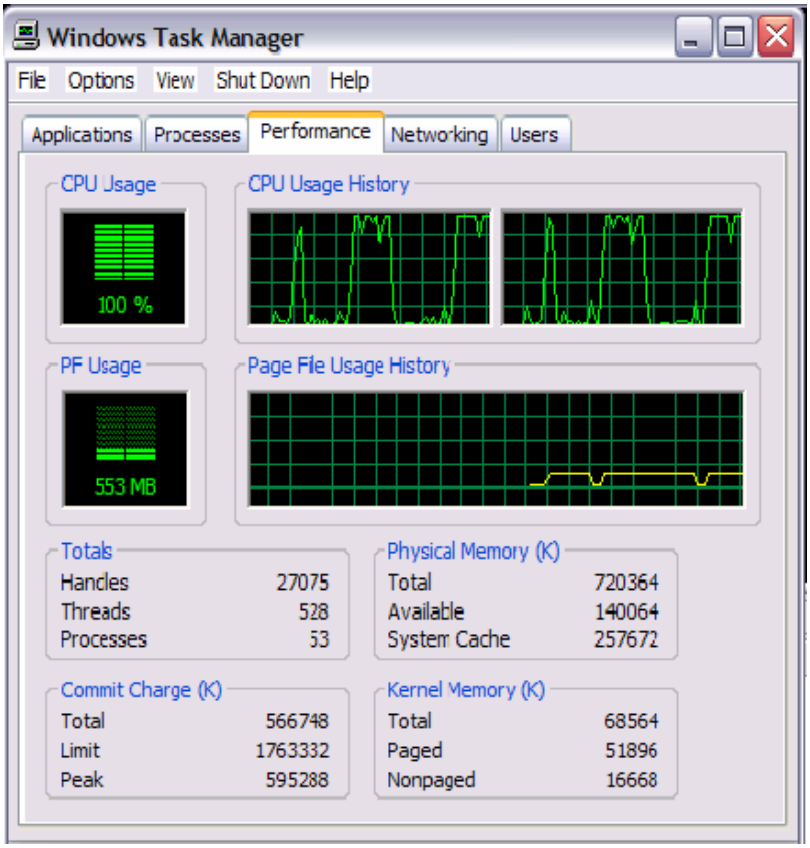


Figure 6.9 Processor Utilization for Karate.264 using 2 decoder threads

6.5 Summary

This chapter has presented multithreading as a method of boosting performance of decoding of an H.264 High Profile bitstream of HD (1280 x 720 p) resolution. Since the tests have been performed on a Pentium 4 processor with HT technology, the best performance can be achieved by using two decoder threads. With the advent of multicore processors like Intel's Pentium 4 Extreme edition [28] and AMD's Opteron [29] number of decoder threads more than two can be used to further improve performance of the

H.264 decoding process for the GOP based bitstream. The GOP based bitstream are sure to find an application at the HD level in the DVD standard for HD content using next generation DVD formats namely HD-DVD [30] and the Blu-ray disc [31].

CHAPTER 7

CONCLUSIONS AND FURTHER RESEARCH

In this thesis, speed optimization techniques for the H.264 High Profile Decoder at the processor level in terms of SIMD (Single Instruction Multiple Data) and multithreading at the operating system level have been presented and the speedup obtained as result of a application of those techniques have been presented. The specific case of decoding of High Definition sequences has been considered. After the performance analysis of the H.264 decoder using Intel VTune performance analyzer the IDCT 4x4 and Motion Compensation modules were targeted for application of SIMD optimization. Significant reduction in execution time of these modules was achieved as a result of the same. Application of multithreading to decoding of GOP level encoded HD (1280x720 progressive) bitstreams was further studied and a proposed multithreading architecture for the H.264 decoder was proposed. Speed comparisons of the multithreaded decoder with the single threaded decoding of H.264 bitstreams show significant reduction in decoding time making it a target of further study and improvement. Other multithreading architectures can be tested for further reduction in decoding time. Moreover, the performance of the current decoding architecture on the latest multicore processors (Intel's Pentium 4 Extreme Edition with Hyperthreading capabilities in each core [28]) can be tested to determine the limits of H.264 HD decoding performance at a single processor (general purpose) level.

The H.264 decoding process is a part of the H.264 encoding process (Fig. 2.1). Therefore, any improvement in the decoding performance in terms of speed contributes to reduction in encoding time. Also, techniques similar to ones employed in this thesis can be applied for improvement of speed of the encoding of HD sequences resulting in the real-time performance and minimizing hardware requirements (High Definition real-time encoding is impossible to achieve on any single processor currently available in the market [3]).

The rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability [32], have made graphics hardware a compelling platform for computationally demanding tasks in a wide variety of application domains. Kelly et al. in their paper on Fast image Interpolation for Motion Estimation using Graphics Hardware [33] explore the usage of graphic cards for video coding. Application of modules of H.264 on the graphic cards and analysis of performance of those modules can lead to a vital addition of processing power to a general purpose single processor machine [34] [35].

APPENDIX A

H.264 ENCODER SETTINGS FOR GENERATION OF TEST STREAMS USED

A.1 Configuration file settings for H.264 encoder JM 9.6 [36] for encoding of streams used for testing speed up of the decoding process using SIMD optimization

Input	.yuv (4:2:0) 8-bit
Output	.264
Number of frames	239
Frame Height	720
Frame Width	1280
Intra Period	0 (1st frame only)
QP I-slice	26
QP P-slice	26
QP B-slice	28
Bitstream Structure	I - B- P-B-P
Rate Control	Off
Deblocking filter	Off
Transform	4x4 and 8x8
Profile	High (100)
Number of Reference Frames	5
Entropy Coding	CABAC

A.2 Configuration file settings for H.264 encoder JM 9.6 [36] for encoding of streams used for testing speed up of the decoding process using Multithreaded GOP Level Decoding

Input	.yuv (4:2:0) 8-bit
Output	.264
Number of frames	239
Frame Height	720
Frame Width	1280
Intra Period	30 (1 every 30 frames)
QP I-slice	26
Frame Rate	30 frames per second
QP P-slice	26
Bitstream Structure	I – P –P ……..I-P-P
Rate Control	Off
Deblocking filter	Off
Transform	4x4 and 8x8
Profile	High (100)
Number of Reference Frames	5
Entropy Coding	CABAC

APPENDIX B

AMDAHL'S LAW

Amdahl's Law [11]

The Overall Speedup (O.S.) obtained by optimizing a part of code that takes a portion p of the total execution time by a factor s is

$$O.S. = \left(\frac{1}{\frac{}{1 - p + (p/s)}} - 1 \right) \times 100\ \%$$

p → fraction of the code being optimized

s → speedup factor for that fraction of

Therefore, overall speedup obtained by optimizing a portion of the code that takes 20% of the total execution time by a factor of 2 can be calculated as : -

$$O.S. = \left(\frac{1}{1 - 0.2 + (0.2/2)} - 1 \right) \times 100\ \% = 11.11\ \%$$

Amdahl's law also states the maximum theoretical limit on overall speedup that can be obtained by optimizing a particular module which takes portion p of the total execution time:-

$$O.S.\ (max) = \left(\frac{1}{1 - p} - 1 \right) \times 100\ \%$$

REFERENCES

[1] Horowitz, A. Joch, F. Kossentini, and A. Hallapuro,"H.264/AVC Baseline Profile Decoder Complexity Analysis" *IEEE Transactions for Circuits and Systems for Video Technology*, vol.13, no. 7, pp. 704-716, July 2003.

[2] Y.K.Chen, X.Tian, S.Ge and M.Girkar, "Towards Efficient Multi-Level Threading of H.264 Encoder on Intel Hyper-Threading Architectures" *Proceeding of the 18th International Parallel and Distributed Processing Symposium*, pp. 63-72, April 2004.

[3] Y.K.Chen, E.Q.Li, X.Zhou and S.Ge, "Implementation of H.264 encoder and decoder on personal computers" *In Press, Journal of Visual Communications and Image Representations*, 2005.

[4] H.264: International Telecommunication Union, "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," ITU-T, 2005.

[5] Soon-kak Kwon, A.Tamhankar and K.R.Rao, "Overview of MPEG-4 Part 10" *In Press, Journal of Visual Communications and Image Representations*, 2005.

[6] I.E.G.Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*, John Wiley & Sons, 2003.

[7] A.Puri, X.Chen and A.Luthra, " Video Coding using the H.264/MPEG-4 AVC Compression Standard", *Signal Processing: Image Communication*, Vol. 19, Issue 9,pp. 787-937, October 2004.

[8] Special Issue on H.264 / MPEG-4 AVC

IEEE Transactions for Circuits and Systems for Video Technology, Vol. 13, No. 7, July 2003.

[9] G.Sullivan, P.Topiwala and A.Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," *SPIE Conference on Applications of Digital Image Processing XXVII*, Vol. 5558, page 53-74, Aug 2004.

[10] D.Marpe, T.Wiegand and S.Gordon, "H.264/ MPEG-4 AVC Fidelity Range Extensions: Tools, Profiles, Performance and Application Areas", *Proc. ICIP 2005*, Genova, Italy, September 11-14, 2005

[11] *IA-32 Intel® Architecture Optimization Reference Manual*, http://developer.intel.com/design/pentium4/manuals

[12] *The Software Optimization Cookbook*, Intel Press, 2002.

[13] http://www.intel.com/products/processor/xeon/

[14] J.Lee, S.Moon and W.Sun, "H.264 Decoder Optimization Exploiting SIMD Instructions", Seoul National University. http://sips03.snu.ac.kr/pub/conf/c67.pdf Accepted at *IEEE Asia-Pacific Conference on Circuits and Systems, (APCCAS),* December 2004.

[15] *The Complete Guide to MMX™ Technology*, McGraw-Hill Inc, 1997.

[16] S.Krishnaprasad, "SIMD Programming Illustrated using Intel's MMX Instruction Set", *Journal of Computing Sciences in Colleges***,** Volume 19, Issue 3 , pp. 268 - 277 , January 2004

[17] The Microarchitecture of the Intel Pentium 4 processor, *Intel Technology Journal*, Volume 8, Issue 1, 2004.

[18] *IA-32 Intel® Architecture Software Developer's Manual: Volume 2* http://developer.intel.com/design/pentium4/manuals

[19] MPEG-2: ISO/IEC JTC1/SC29/WG11 and ITU-T, "ISO/IEC 13818-2: Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Video," *ISO/IEC and ITU-T*, 1994.

[20] http://www.intel.com/cd/software/products/asmo-na/eng/vtune/index.htm

[21] JVT B-038 *Low Complexity Transform and Quantization – Part 1: Basic Implementation* available from http://ftp3.itu.int/av-arch/jvt-site

[22] http://msdn.microsoft.com

[23] *Multithreading for Rookies* at http://msdn.microsoft.com

[24] S.Guptha, *Multithreaded Programming in a Microsoft Win32 Environment* at http://www.intel.com

[25] Petzold, *Programming Windows*, Fifth Edition, Microsoft Press, 1999.

[26] S.Kumar et al, "Overview of Error Resiliency Schemes in H.264 / AVC standard", *In Press, Journal of Visual Communications and Image Representations*, 2005.

[27] JVT- QO19 *SEI message for logo insertion* available from http://ftp3.itu.int/av-arch/jvt-site

[28] http://www.intel.com/personal/desktopcomputer/dual_core/

[29] http://www.amd.com/us-en/Processors/ProductInformation

[30] http://www.hddvd.org/hddvd/

[31] http://www.blu-ray.com/

[32] "A Survey of General Purpose Computation on Graphics Hardware", *Eurographics 2005,* pp.21-51, August 2005.

[33] F.Kelly and A.Kokaram, "Fast Image Interpolation for Motion Estimation using Graphics Hardware", *Proceedings of the SPIE*, Volume 5297, pp. 184-194, May 2004

[34] http://www.ati.com

[35] http://www.nvidia.com

[36] H.264 reference software: http://iphome.hhi.de/suehring/tml/download/

[37] http://www.fastvdo.com

[38] MPEG-4: ISO/IEC JTCI/SC29/WG11, "ISO/IEC 14 496:2000-2: Information on Technology-Coding of Audio-Visual Objects-Part 2: Visual," *ISO/IEC*, 2000

[39] H.263: International Telecommunication Union, "Recommendation ITU-T H.263: Video Coding for Low Bit Rate Communication," *ITU-T*, 1998.

[40] "*Writing a DVD Playback Application in DirectShow*" on http://msdn.microsoft.com

[41] http://www.gpgpu.com (Graphics Processing Unit for General Processing).

[42] http://www.iso.org

[43] http://www.itu.int

[44] N.C.Paver et al, "Accelerating Mobile Video: A 64-Bit SIMD Architecture for Handheld Applications", *Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, pp.21-34, Vol.41, August 2005.

[45] Shish-Hao Wang et al, "A Software-Hardware Co-Implementation of MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining", *Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, pp.93-110, Vol.41, August 2005.

[46] Tsu-Ming Liu et al, "An 865-μW H.264/AVC Video Decoder for Mobile Applications", IEEE Asian Solid-State Circuits Conference, Hsinchu, Taiwan, November 2005.

[47] http://www.mpeg.org

BIOGRAPHICAL INFORMATION

Tarun Bhatia received the Bachelors degree in Electronics Engineering from the University of Mumbai, Mumbai, India in 2001 and a M.S. degree in Electrical Engineering from University of Texas at Arlington in 2005.His research interests are video codec design and optimization and embedded systems.