Real Time HD Visual Communication Using H.264 Codec for Windows Based System

Ramesh Naik M[1], Dr. Jayendra Kumar[2] and Vasanth Subramanyam[3]

Department of Electronics & Communication Engineering, Department of Automation Division[3]

NIT Jamshedpur, India[1, 2], Tata-Steel, Jamshedpur, India[3]

E-mail:- *rameshnaik466@gmail.com[1], jkumar.ece@nitjsr.ac.in[2], v.subramanyam@tatasteel.com[3]*

DOI:- https://doi.org/10.47531/MANTECH/ECC.2021.38

Abstract

H.264 standard is the most recent and widely used standard for developing a video codec. Enhanced compression and network friendliness made it very useful in teleconferencing applications. This paper presents F to develop a windows application that could perform video communication. And we are designing the codec part of it. For performance improvement, we have made use of two kinds of parallel processing. One is Data parallelism, and another is Thread-level parallelism. To implement the first technique, we used processor-specific SIMD instructions, and for the second technique we made use of the Windows thread libraries.

Keywords:- *Data and Thread level parallelism, JM reference software, SIMD instructions*

INTRODUCTION

Video A codec can compress/decompress the digital video given to it. A codec can be implemented either in software or hardware. In software, we implement the compression algorithms on a general-purpose processor, whereas a hardware codec can be an ASIC or FPGA. Both of the implementations have their own advantages and disadvantages. In this project, our focus is to implement a software H.264 codec for HD video. The main characteristics of a codec are the decoded video quality, amount of compressed data and processing time. These three characteristics are dependent on each other. If one wants to improve codec performance with respect to one characteristic, he has to compromise on the other. One needs to maintain the complex balance among these three characteristics to develop a good codec. There are several standard measures to estimate the performance of these characteristics. PSNR measures the quality, bit rate gives the amount of compressed data, and frame rate represents the encoding/decoding time.

REFERENCE IMPLEMENTATION (JM SOFTWARE)

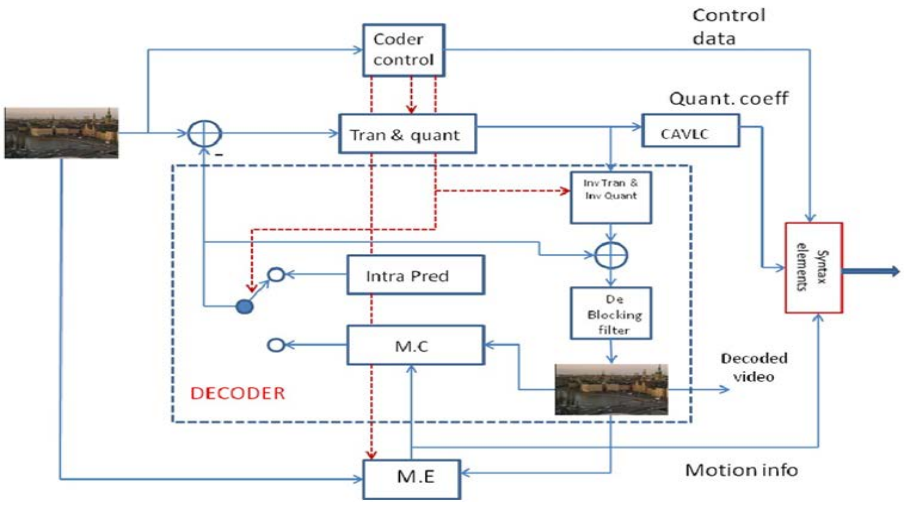

Figure 1 Video encoder block

The main source of information used in the implementation is the Joint Video Team's (JVT) working Document H 264 draft, which is freeware JM code [1] is software written in c, c++ without exploiting the parallelism possible in video processing techniques we try to analyze the JM source code and find the typical time required by the different modules of a video encoder; we found on average JM is encoding around 2 to 3 fps of frame size 720p (1280 x 720) For Interframe prediction & compensation 33 to 35% Transformation and quantization 30% CAVLC ending around 18% and remaining time are required for bitstream generation & NAL implementation.

Transformation is the most computationally intensive part of video codec design. So we concentrate on optimizing this module with the different architecture and algorithm.

Figure 1 H.264 [2] is a standard that describes the syntax and the semantics of a compressed bitstream. Any codec that could generate a bit stream following this standard is called as a H.264 encoder.

Optimization Strategy

The performance of the software is limited by the underlying performance processor and the algorithm used to implement it. This means the execution speed of the software is limited by the processor on which the code is written. It may also depend on the OS in some special cases (like system calls and i/o services).

These days processors are getting powerful by the developments in their architectures. We are now having multi-core processors where each core is superscalar in nature. So our aim is to develop our codec software by making the best use of available processor resources.

For that, we need to have a good idea of the processor architecture and instruction set. The reference codec software (JM) is not processor-specific that is, it can be run on any kind of processor. It is serial in nature with no parallel instructions.

In our case, we need to improve our codec performance; the processor supports data-level parallelism with its SIMD instructions and Thread level parallelism.

TRANSFORM AND QUANTISATION

A video encoder [2] consists of three main distinct functional units: a temporal model, a spatial model, and an entropy encoder. The input to the temporal model is an uncompressed video sequence. The temporal model tries to reduce the temporal redundancy by exploiting the similarities between the neighbouring video frames by constructing the prediction of the current video frame. In H.264 video codec standard, the prediction is formed from one or more previous or future frames and is improved by compensating for differences between the frames (motion compensated prediction).

In the below encoder block diagram, blocks T, Q represent transform, quantisation and T-1, Q-1 represent the reverse processes. The necessity of the transformation of data in an image or video codec is to convert image or motion-compensated residual data into another domain (transform domain).

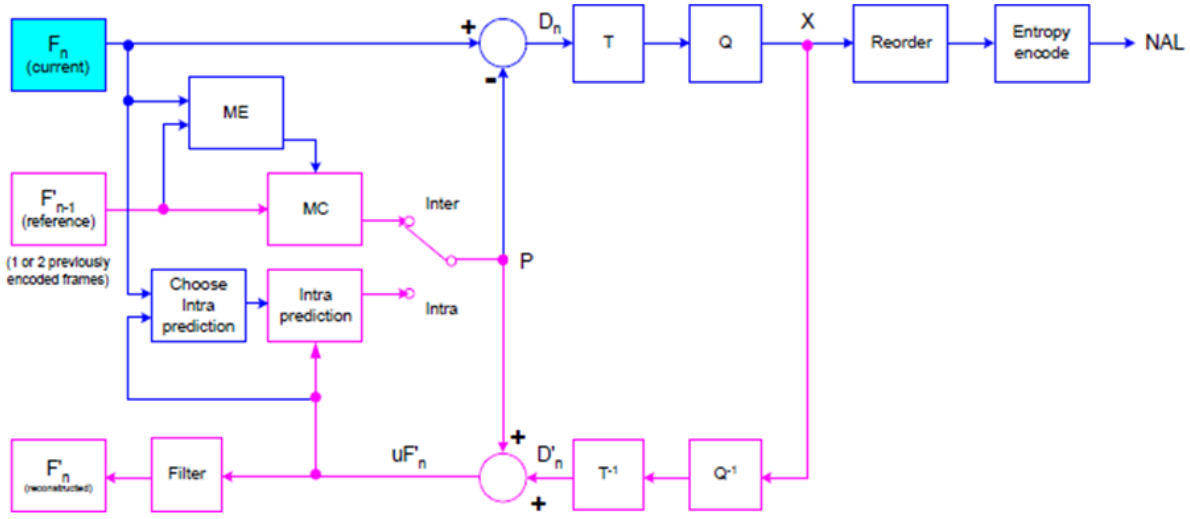

Figure 2 H.264 encoder block diagram

The choice of transform depends on some criteria like:

- Data in the transform domain should be separated into components with minimum Interdependence.
- Most of the energy in the transformed data should be concentrated into a small number of values so that complete data can be reconstructed at the other end by transmitting less number of values which helps to lower the bit rates that we have to transmit.
- The transform should be reversible, which means inverse transform should be able to reproduce the original data from the transformed data with minimum error possible.
- The transform should be implementable with low memory requirements, using limited precision arithmetic.
- It will be very easy to implement if it can use less number of arithmetic operations and if it doesn‘t involve complex operations like multiplications and divisions.

The entire process of transform and quantization can be carried out using 16-bit integer arithmetic and only a single multiply per coefficient, without any loss of accuracy.

SMID TECHNIQUE

We used a programming technique known as Single-Instruction, Multiple-Data, which is available in the latest INTEL general-purpose processors to implement some of the functional blocks in the video codec like integer transform, quantisation and inverse integer transform, inverse quantisation. This technique speeds up software performance by processing multiple data elements in parallel, using a single instruction.

Parallel and vector processor architectures support SIMD-style (Single-Instruction, Multiple- Data) programming where a common operation is performed simultaneously on multiple data elements to improve performance. Intel Architecture's MMX technology, designed and optimized for multimedia applications, supports SIMD-style programming. The intention was to provide a set of optimized instructions for multimedia applications. These instructions support SIMD-style (Single-Instruction Multiple-Data) machine-level instructions, which perform a single operation on multiple data elements in parallel.

The transformation equation Y=AIAT involves matrix multiplication. So as the multiplication is more time taking than addition and subtraction and as the transformation matrix A contains only ones and twos, the transformation is realised in assembly code using additions, subtractions, shifts only with the help of MMX instructions and streaming SIMD extension(SSE) instructions. During quantization some single-precision floating-point instructions are used for multiplication. When the multiplying transformation matrix columns‘contain only ones, then simply adding the elements in the rows of matrix to be transformed gives the required result. Here the matrix to be transformed is a 4x4 residual image block after intra prediction or motion-compensated prediction. If the multiplying transformation matrix columns‘contain some twos, then it needs a shift operation. Using an XMM register, we can do arithmetic operations on two eight words data with a single instruction. Each word is residual pixel data. PMADDWD can do eight-word- integer multiplications at a time. As it uses multiplications, there is not much improvement in timing when compared to the multiplication free method. The authors have done transformation without using that instruction. In this method of implementation, operations are quite easy, but the data transfer and adjustment is a significant problem. That too transformation is performed on 4x4 blocks; every time I have to load a 4x4 block, transform it store it and then go for another block. And for quantization use of multiplication instruction is compulsory. Inverse transform and inverse quantization are also realised on the same lines.

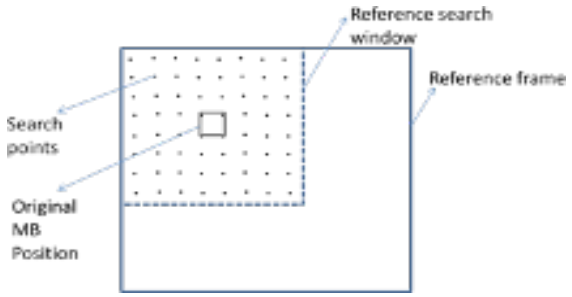

Fig. 3: Showing reference window and search points

TEMPORAL AND SPATIAL REDUNDANCY CODING

Initially, we have implemented a windowed based search of windows size w, which requires a total (2w +1)2∗256 computation is required, and there is about 3600 Macroblock per frame and 108000 macroblock computation is required per sec So motion estimation & Compensation is Highly computationally intensive algorithm almost around 33% of time consumed by this operation.

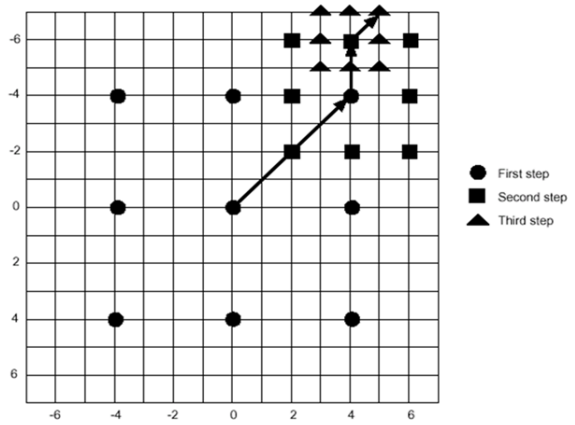

Fig. 4: The three-step search algorithm (3SS) [7]

In the first stage, the centre of the search position is (0,0) and 8 other point with window size w=8 as shown in the figure. First nine position is to be searched. After that, the minimum out of nine become center, and we reduce the search window by half. Repeat this procedure until the search size reduces to one. So it requires in first step 9 and two times 8 computation is required. So total just (1 + 8n) = 25 where n = 3 computation per block is required in three-step search algorithm.

If the macroblock is coded in 16x16 Intra mode, then the block labelled "-1' is transmitted first, containing the DC coefficient of each 4x4 luma block. Next, the luma residual blocks 0-15 are transmitted in the order shown with the DC coefficient of each 16x16 Intra macroblock) Blocks 16 and 17 contain a 2x2 array of DC cots from the Cb and Cr chroma components, respectively. Finally, chroma residual blocks 18-25 with zero DC coefficients are sent.

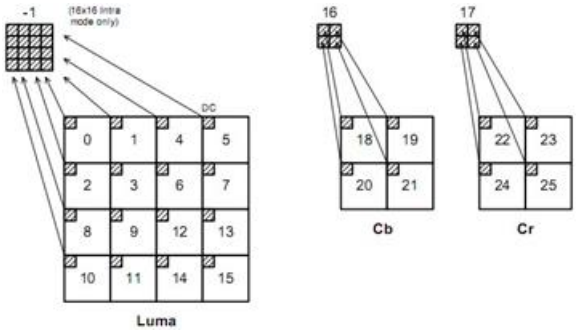

Fig. 5: Scanning order of residual blocks within a macroblock

For figure 5 The 4x4 DCT of an input array X is given by:

$$\mathbf{T} = \mathbf{AXA^T} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} [X] \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}^{\mathbf{T}}$$

$$a = \frac{1}{2}$$

$$b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$$

$$c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$$

The core part of the transform is multiply-free, i.e. it only requires additions and shifts.

$$\mathbf{Y} = \mathbf{CXC^T} \otimes \mathbf{E}$$

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} [X] \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

Where CXC^T is a "core' 2-D transform. E is a matrix of scaling factors and the symbol.

To ensure that the transform remains orthogonal, b also needs to be modified so that:

$$a = \frac{1}{2}$$

$$b = \sqrt{\frac{2}{5}}$$

$$c = \frac{1}{2}$$

2^{nd} and 4^{th} rows of matrix C and the 2^{nd} and 4^{th} columns of matrix CT are scaled by a factor of 2, and the post-scaling matrix E is scaled down to compensate. This avoids multiplications by 1/2 in the "core' transform CXCT, which would result in loss of accuracy using integer arithmetic). The final forward transform becomes:

IMPLEMENTATION DETAIL OF INTEGER TRANSFORM

- Integer Transform Require two 4X4 Matrix Multiplication So It Require total $2n^3 = 128$(n = 4) Multiplication.
- Here we used architecture [8]which efficiently use high-level parallelism of intel processor using intel MMX instruction.
- Initially, we first Transpose Matrix with Punpck instruction as shown below.
- This Punpck just Shuffles Lower/Higher-order word or double word with just 8 instruction; it gives Transpose Matrix.

In figure 6 after that. we add & subtract this register in a parallel fashion with PADDV & PSUBV instruction as shown in the figure. The PADDV instruction adds packed word integers. So it requires 8 add/sub & 2 shift instruction Total in 18 Instruction whole 2-dimensional IDCT is performed.

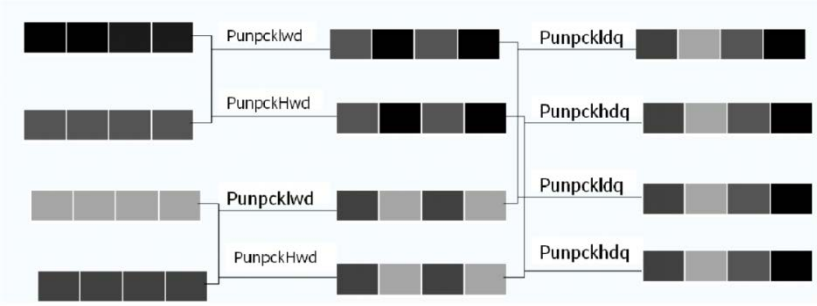

Figure 6: Parallel Implementation to find Transpose of Matrix

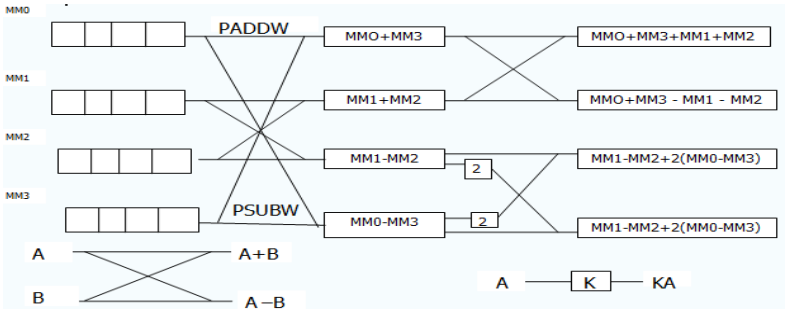

Figure 7: Parallel Implementation of DCT

RESULTS

Table 1: Comparison with JM

Video Sequence	PSNR (dB)		Bitrate (Mbps)		fps	
	JM	OURS	JM	OURS	JM	OURS
STOCKHOLM	34.72	36.83	9.6	13.6	0.82	5.71
MOBCAL	33.85	36.14	12.32	16.4	0.89	5.84
PARKRUN	32.94	34.08	30.4	32.4	0.85	5.76
SHIELDS	34.97	37.04	12.8	16.8	0.86	5.8
TCSEQUENCE	41.169	43.827	41.09	8.16	0.86	6.16

We put the same parameters to JM and our codec obtained various results.

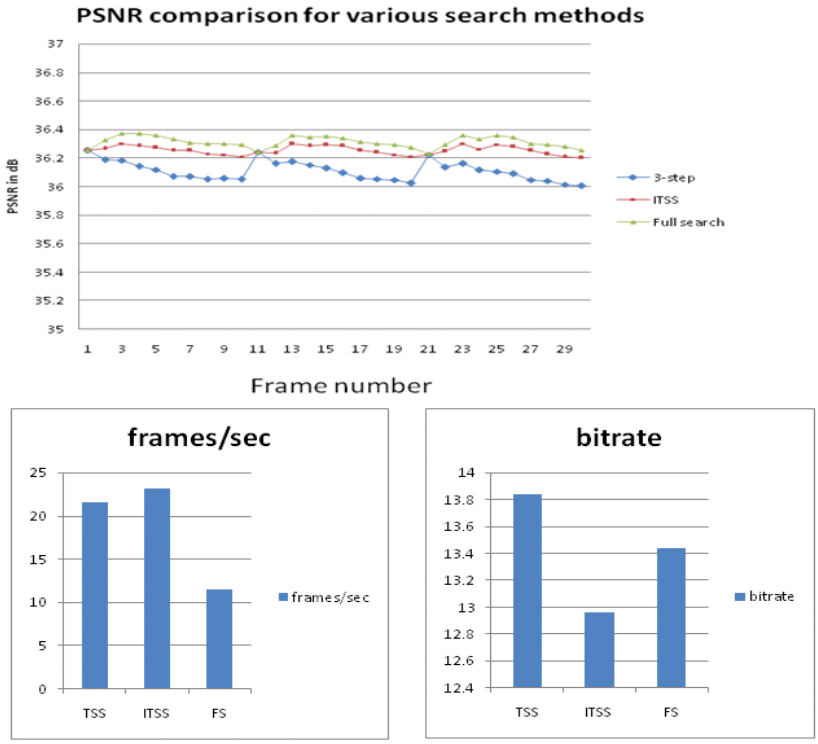

Figure 8: PSNR, frame rate and bitrate comparison for various search methods

Reasons for this ITSS to performs better might be,

- Least number of search points of all
- Searching is carried out mostly around the centre point.
- Small motion vectors, so better bitrates

CONCLUSION

Improvement in the PSNR is observed. This is due to the cost criterion taken by us in motion estimation. We took SAD as the cost criterion, while JM took a combination of SAD and Motion vector cost. The bit rate of our codec is poorer than the JM for the same reason. The performance of our codec is around 7 times better than JM. This is due to employing parallelization in the codec.

REFERENCES

1. IIntel® Integrated Performance Primitives for Intel® Architecture Reference Manual, Volume 2: Image and Video Processing
2. Iain E. G. Richardson, A text book on _H.264 and MPEG-4 VIDEO COMPRESSSION"
3. ISO IEC 14496-10 AND ITU-T REC.H.264, ADVANCED VIDEO CODING
4. H.264/AVC REFERENCE SOFTWARE Link : http://iphome.hhi.de/suehring/tml/
5. Source of Uncompressed Video Link : http://media.xiph.org/ldv/pub/test_sequences/720p/
6. Intel®64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture.
7. Donglai Xu; Bailey, C; Sotudeh, R.; An improved three-step search block-matching algorithm for low bit-rate video coding applications; Signals, Systems, and Electronics, 1998. ISSSE98. 1998 URSI International Symposium on Digital Object Identi_er: 10.1109/ISSSE.1998.738061 Publication Year: 1998 , page(s): 178 - 181
8. Xueming LI;Fang WE1;_An Improved Practical E_cient Implementation of ICT Used in H.264_;2004 IEEE International Conference on Multimedia and Expo (ICME).
9. http://software.intel.com/en-us/intel-ipp/
10. http://en.wikipedia.org/wiki/High-de_nition_television
11. http://www.tommesani.com/Docs.html
12. T. Wiegand, G. J. Sullivan, G. Bj.ntegaard, and A. Luthra, Overview of the H.264/AVC video coding standard, ‖ IEEE Trans. Circuits Syst. Video Technol., vol. 13, pp. 560–576, July 2003.