H264/AVC Video Stream Parser For Baseline Profile

Vinita Malu
Dept.VLSI & EMBEDDED SYSTEM
Ganpat University
Mehsana, INDIA

***Abstract*—H264/AVC has been a popular technology for video compression due to its high coding efficiency. The H264 bit stream contains the encoded information such as motion vector, coded block pattern (CBP), residual data, macroblock types which can directly use in many algorithms like motion detection and object detection and tracking. This compressed domain algorithms use the encoded information which shows fast computation in real time.**

This paper presents entropy level decoding of H264 baseline profile video stream. Partial decoding is an efficient way to improve the performance of many video applications. In this paper the time taken by partial decoding is compared with whole decoding process. The time taken by partial decoding is very less compare the decoding process.

Keywords—H264/AVC, Video bit stream, Network abstraction layer (NAL) Sequence parameter set (SPS), Picture parameter set (PPS), Macroblock , CODEC, Context Adaptive variable length coding (CAVLC).

I. INTRODUCTION

The recent technology for video and computer vision has evolved more intelligence due to video analytics. Video analytics includes analyzing video streams and extracting information like background object motion, scene situation and so on. The object information illustrated above can be used in surveillance or interactive broadcasting services. There are two approaches to provide intelligence to video content: *pixel domain approach* and *the compressed domain approach*. Pixel domain approach requires additional computation to decode the video stream. As an effective alternative of this problem is compressed domain approach. Unlike the pixel domain approach, the compressed domain algorithms utilize the encoded information like motion vector, DCT coefficient and macroblock types which are include in encoded bit stream. The encoded information is beneficial to reduce the computational complexity because it can be directly exploited as effective clues for many compressed domain algorithms. The method presented in this paper provide this partially decoded data without full decoding which with compressed domain algorithm analysis video in real time.

A. H264 And H264 Profiles

H264/MPEG-4 Part 10 or AVC (Advance Coding Standard) is a video compression format which is currently most widely used for video recording, compression and distribution of video content. H264 is a method for video compression, the process of converting digital video into a format that takes up less capacity to store or transmit. This compression technique is used in multiple applications such as DVD- Video, mobile TV, video conferencing and internet video streaming.

In H264 each profiles define specific set of tools define in standards. A H264 bit stream that conforms to a particular profile that can be coded using some or all the tool with in profile. The profiles therefore act as constraints on the capabilities required by decoder.
Baseline profile features are following:
1) Bit depth per sample is 8.
2) Chroma format is 4:2:0.
3) Slices – I slice, P slice (Present) and B slice (Absent).

B. H264 codec work

Compression involves complimentary pair of system, a compressor (encoder) and decompressor (decoder). The encoder reduce the original bit stream before it store or transmit and the decoder convert the reduced bit stream into original bit stream before it plays. The Encoder/Decoder pair often describe as CODEC.

As per above definition of codec H264 codec have encoder and decoder with in it. An H264 encoder carries out prediction, transformation and encoding process and H264 decoder carries out complimentary process of decoding process, inverse transformation and reconstruction. The decoded video stream is not identical to original video steam; there were some bits loss because H264 is a lossy compression technique.

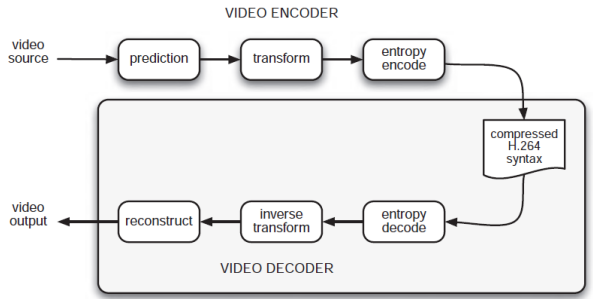

Fig. 1 the H264 encoding and decoding process [1]

C. *Decoding process*

Decoding process includes three processes which are following:
1) Bit stream decoding
2) Rescaling and inverse transform
3) Reconstruction

1) Bit stream decoding

A video decoder receive compressed H264 video bit stream, extract each of information such as quantized transform coefficients, prediction information, compressed data structure, compression technique and decodes each of syntax elements.

2) Rescaling and inverse transform

The quantized coefficients are rescaled. These coefficients are multiply by integer value to restore the original scale. Then the inverse DCT transform is applied on rescaled values.

3) Reconstruction

Decoder forms an identical prediction for each macroblock of slice. This prediction can be either from the same frame which known as *Intra prediction* or from the other frame know as *Inter prediction*. So these predictions are added with residual and reconstruct macroblock which can display as part of video frame.

This paper is organized as follows. In section 2, describe about the hierarchical structure of H264 syntax. Then in section 3 Algorithm for partial decoding is discussed. In section 4 results is shown. Finally the conclusion is drawn in section 5.

II. THE H264 SYNTAX

The H264 syntax is a hierarchical structure which consists of various elements. The syntax is hierarchical, from the highest level, the video sequence level, down through the separate frames or fields (access units), subset of access units (slices), to macroblocks. Control parameter and video information are stored in separate syntax such as Picture parameter set and Sequence parameter set
.

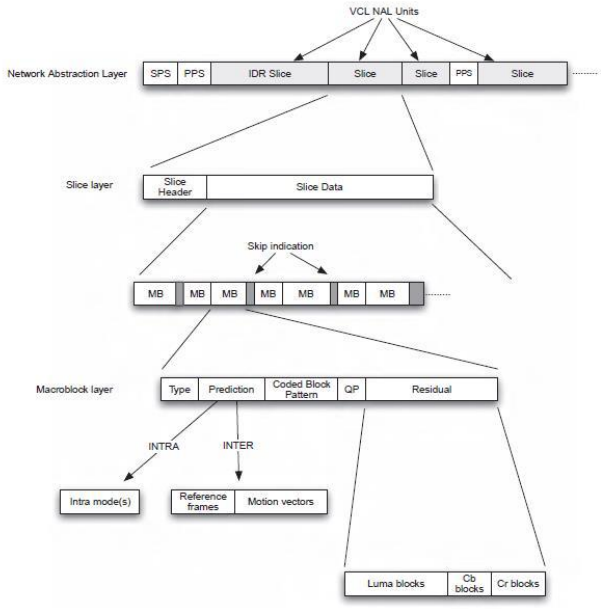

Fig. 2 the H264 syntax [1]

A. *NAL*

The Network abstraction layer consist series of NAL units. Each NALU contains one byte of header. This header contains:
1 bit of forbidden_zero_bit indicates error. If it is zero means NALU should not contain any bit stream error. If it is one means NALU may contain bit error.
2 bit of nal_ref_idc specifies that content of NAL unit may or may not be used to reconstruct reference pictures.
5 bit of nal_unit_type describes the type of NAL Unit. The types of NAL are shown in following figure:

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13..23	Reserved	
24..31	Unspecified	

Fig. 3 NAL unit [1]

Sequence Parameter set

A Sequence parameter set contains parameters which are common to entire video sequence such as profile id, level id, constrains flag, maximum number of reference frame, and macroblock size, height, width etc.

B. Picture Parameter Set

A Picture parameter set contains parameters which are common to a sequences or subset of coded frames such as number of active reference frames, entropy coding type and initialization parameters. It also contain its own identifier, picture parameter set id, points to SPS identifier.

C. Slice

Slice contains the slice header and slice data. The slice header is of variable bit length. The slice header is contain common information for all macroblock in that slice such as slice type which determine which type of macroblock types are allowed, the frame number, reference picture setting, and default quantization parameter.

The slice data contains series of macroblock that make up a slice. The coding technique can be either CAVLC or CABAC depend on entropy coding type flag to decode the macroblock. As B slice is absent in baseline profile therefore CAVLC coding technique is used in this parsing process.

D. Macroblock

Macrobock contains encoded information such as motion vectors, coded block pattern, DCT coefficient, macroblock type, macroblock address, prediction type and residue information.

In these five syntax macroblock have all the encoded information so to get all this information the parser should decode up to the macroblock. To reach macroblock first we decode the entire syntax one by one and at the end macroblock. Then we can directly use that encoded information in different algorithms.

III. Algorithms For Partial Decoding

Partial decoding includes two algorithms. First is Exp-Golomb Coding which is used to decode the starting NAL unit SPS, PPS and the second one is CAVLC which is used for further decoding in which it includes the slice data that are macroblock header and data.

A. Exp-Golomb Coding

The main algorithm in H264 is Exp-Golomb-coded integer. Exp-Golomb codes are special Huffman codes with regular construction that favours small numbers by assigning them short codes.

Exp-Golomb codes are variable length codes with the following properties:

1) Code length is increases with index code number.
2) Each code can be constructed logically and decoded arithmetically without the need of look up table.
The Exp-Golomb code word has following structure:

[Zero prefix][1][INFO] [1]

The code word consist of a prefix of M zeros, where M is zero or positive integer, a 1 and M bit of information, INFO. Each code word arithmetically generated in following manner

$$M = \text{floor}(\log 2\ [\text{code\_num} + 1])$$
$$\text{INFO} = \text{code\_num} + 1 - 2^{M}$$

Conversely code_num may be generated.

Step1: Read a series of consecutive zeroes until first 1 is detected. Count the number of zeroes.
Step2: The First detected 1 is ignore.

Step3: Read the next bits after 1 which should be equal to the total number of zeros which were in suffix.

Step4: code_num = 2^{M} + INFO – 1. [1]

code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

Fig. 4 Exp-Golomb Codewords [1]

B. Context Adpative Variable Length Coding

This method is used to decode the residual, zigzag ordered 4x4 or 2x2 block of transform coefficients. CAVLC process of decode the transform coefficient is as follows:

Step1: Decode the number of coefficients and trailing ones

Number of coefficient can be anything from 0 to 16 and trailing ones can be from 0 to 3.
For calculating coefficients and trailing ones there are four choices of look-up table which are define in standards to use for decoding coefficient token, the choice of table depends on the number of non-zero coefficients in the left and upper previously coded blocks, nA and nB respectively. Non-Zero coefficients nC is calculated as follows:

1) If upper and left blocks are both available, i.e. in the same coded slice, nC = (nA + nB + 1) >> 1, where >> indicates binary right shift.
2) If only the upper is available, nC = nB.
3) If only the left block is available, nC = nA.
4) If neither neighboring block is available, nC = 0.

Match the bit stream till the long prefix is match with each entry in column of table. The unique code is obtained from the bit stream which gives trailing one and number of coefficient.

Step2: Decode the sign of each trailing ones

For each trailing +/−1 T1 the sign is decoded with a single bit. Decode the next bit stream, substitute +1 for 0 and -1 for 1.

Step3: Decode the levels of the remaining non-zero coefficients.

If number of coefficient is greater than trailing ones, then the remaining non zero coefficient are called level, and those level are decode by Prefix – Suffix method.

Prefix – Suffix Method:

This method includes four steps:
1) Prefix Calculation: Prefix calculated by reading a series of consecutive zeroes until first 1 is detected. <Prefix> will have <zeroes 1>.

2) Suffix Calculation:

If it is first level:

```
If (number of coefficient  > 10 && trailing ones < 3)
        suffix length = 1
else
        suffix length = 0
```

Else

```
If  (suffix length = 0 && abs(level_code) > 3)
        suffix length = 2
else if  (suffix length = 0)
        suffix length = 1
else if (abs (level_code) > (3 << (suffix length -1)
      && suffix length < 6)
        suffix length = suffix length + 1
```

Here abs (level_code) = previous level code

3) Level Code Calculation:

If it is first level:

```
If (number of coefficient > 10 && trailing ones < 3)
   If (prefix < 14)
     If (suffix length)
        level_code = (prefix << 1) + read 1 bit
        else
        level_code = prefix
     else if (prefix == 14)
          If (suffix length)
             level_code = (prefix << 1) + read 1 bit
          else
             level_code = prefix + read 4 bit
     else
        level_code = 30
        if (prefix >= 16)
           level_code = level_code + (1 << (prefix-
                                    3)) -4096
                level_code = level_code + read (prefix –
                                    3) bit
Else
        If (prefix < 15)
           level_code = (prefix << suffix length) + read
                        suffix length bit
        else
           level_code = (15 << suffix length) + read
                        (prefix - 3) bit
        if (prefix >= 16)
           level_code = level_code + (1 << (prefix-3)) -4096
```

Step4: Decode the total number of zeroes before the last coefficient (Total zeroes)

Total Zeroes is the sum of all zeroes preceding the non-zero coefficient, read the next bit stream match with the each entry of table, the unique code obtained from bit stream which gives total number of zeroes.

Step5: Decode each run of zeros (the location of those embedded zeroes [run before]).
The number of zeroes preceding each non-zero coefficient (run before) is decoded in following steps:

```
If (number of coefficients = maximum number of coefficients)
        Total zeroes = 0
Else
```

Read the next bit stream, match with the each entry of table, the unique code obtained from bit stream which tells the position where the zeros will be embed.

IV. Result

I have used ubuntu 12.04 machines and ffmpeg library for the results. I have checked our results for two different frame resolutions that are 640 x 480 and 800 x 600. The result got from the machine; we can say the total time taken in parsing technique is reduced by 66% to the whole decoding process.

Conclusion

Our study showed that the parsing technique can be used effectively to get encoded data. The performance of parsing technique is sufficient to get not only the encoded data but also perform algorithms on it in real time. It shows better performance than whole decoding process. Therefore we conclude that using parsing technique the compress domain algorithm can implement.

Acknowledgement

I would like to express my deep gratitude to Mr. Sudhir Bhadauria, my research supervisor and Professor Vijay K. Patel at Ganpat University for their expert guidance, enthusiastic encouragement of this research work.

REFERENCES

[1] The H.264 Advanced Video Compression Standard Second Edition Iain E. Richardson Vcodex Limited, Uk.
[2] ITU-T Rec. H.264 (05/2003) – Series H: Audio-visual and Multimedia Systems.
[3] RFC3984 – H.264.
[4] H.264 / Mpeg-4 Part 10 White paper.
[5] CAVLC_Example.pdf.
[6] Link: Wikipedia.org.
[7] H.264 and MPEG-4 Video Compression – Video Coding For next-generation Multimedia, Iain E.G.Richardson, The Robert Gordan University, Aberdeen, UK.
[8] Michael Horowitz, Anthony Joch, Faouzi Kossentini "H.264/AVC Baseline Profile Decoder Complexity Analysis" in Ieee Transactions On Circuits And Systems For Video Technology, Vol. 13, No. 7, July 2003.
[9] Atul Puria,, Xuemin Chenb, Ajay Luthrac "Video Coding Using The H.264/MPEG-4 AVC Compression Standard" in Puri Signal Processing: Image Communication 19 (2004) 793–849.