Efficient Constant-time Entropy Decoding for H.264

Nabeel Iqbal and Jörg Henkel
University of Karlsruhe, Chair for Embedded systems, Karlsruhe, Germany
{Iqbal, henkel} @ informatik.uni-karlsruhe.de

***Abstract*--Diverse approaches to parallel implementation of H.264 have been proposed; however, they all share a common problem. The entropy decoder in H.264 remains mapped on a single processing element (PE). Due to the inherently sequential and context-adaptive nature of the entropy decoder, it cannot be parallelized. This renders a bottleneck to the performance of the entire decoding process. Depending on the type of the processing core and the video bit-rate, the performance of the entire decoding process is subject to the process of entropy decoding. It is, therefore, needful to research and implement new algorithmic solutions to compensate for this bottleneck, and thereby make optimal use of parallel implementation of H.264 decoder on mainstream multi-core systems.**

This paper presents a new CAVLC decoding method which is de-rived by constructing custom CAVLC decoding tables using 'table grouping'. Compared to the conventional [5] 'sequential table look-up' method, which requires multiple memory accesses. Our proposed method accesses the custom tables only once for the decoding of any symbol. Moreover, in our proposed method, the symbol decoding time does not depend on the symbol length and it is constant for each symbol, resulting in a nearly linear increase in computational complexity with increase in video fidelity as compared to an non linear increase in earlier proposed methods. Experimental results show that our proposed algorithm features up to 7x higher performance and 83% less memory accesses compared to conventional methods. We compare to three commonly used, state-of-the-art CAVLC algorithms, such as table look-up by sequential search [5], table look-up by binary search [9], and "Moon's method" [16] .

I. INTRODUCTION

Advances in semiconductor technology allow billions of transistors to be integrated on a single chip. Intel has already built the first Billion transistor chip featuring quad cores and a large cache [19] and prototype eighty core chip [21]. Industry's prowess continues to drive Moore's Law, providing a double of transistor density every two years. The availability of terra-scale integration capacity and small feature size resulted in the dawn of multi-core systems [2].

According to Pollack's rule, the increase in area or energy consumption of a single core doesn't lead to a linear increase in performance. In general, 2x energy consumption results only in a 1.4x increase in performance. On the other side multi-core performance increases approximately linearly with energy consumption by integrating more cores on a single die, which makes it ideal for embedded mobile devices where performance-to-energy ratio is always a critical design goal. According to the 'kill rule' for multicore [1], the smaller the processing element (PE), the better the energy savings; however, more cores on a single die are only meaning full when application parallelism exists and communication cost between the cores and memory does not dominate the energy consumed in actual computations. Multicore systems are already available commercially and systems with more than 100 cores on a single die are envisioned in the near future. The promise of high computational performance with low energy consumption makes multicore processors ideal for multimedia enabled mobile devices. The promised computational performance will only be available to parallelized applications with suitably partitioned tasks. This parallelization requirement gives rise to an urgent need to study and devise new algorithms that are able to exploit the available computing power.

In multimedia, video encoding/decoding is the most computation-intensive process. Many video compression standards are in use like MPEG-2, H.263. MJPEG-4, H.264 etc. The latest video codec from Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T Video Coding Expert Group (VCEG) is H.264 or MPEG-4 AVC part 10, which offers better compression as compared to other standards under the envelop of the same video fidelity[3]. The primary goals of H.264 are better compression, suitability for network transmission and error resilience. To achieve these goals, a wide variety of tools are included in the standard, which makes it computationally complex and predominantly control-flow oriented [4]. Due to its versatility and coding efficiency, H.264 is widely accepted by industry and would likely be the leading video codec for mobile multimedia embedded systems for years to come.

The increased performance in bit rate owes to the inclusion of additional tools which lead to increased computational complexity in encoding as well as decoding. This increased complexity in decoding is a major problem when it comes to devise cost effective H.264/AVC-based video solutions [18]. To meet the computational requirement of the H.264 decoder, many parallel approaches have been proposed for Multi-Processor systems [7, 8, 10, 13, 14] utilizing from 2 PEs to 60 PEs. The common design metric in all these approaches is to map the entropy decoder on single PEs because entropy decoding is an inherently sequential process and cannot be parallelized. Hence entropy decoding in parallel designs becomes the bottle neck [7, 14] and the total throughput of the decoder depends on the performance of entropy decoder.

In H.264, there are two kinds of entropy encoding schemes: the conventional Variable Length Coding (VLC) and Context Adaptive Variable Length Coding (CAVLC). VLC is based on Exp-Golumb codes; its decoding process is simple and straightforward, and it is only being used to transmit the header data. On other hand, CAVLC is used to encode actual video data and is a combination of entropy and 'run length' coding with context-adaptive mechanism. The whole entropy encoding/decoding process is elaborated in [17]. The H.264 standard document defines the symbol tables for the CAVLC encoding process and is fixed for all profiles which include coeff_token, total_zero, and run_before syntax elements for "luma" and

978-3-9810801-5-5/DATE09 © 2009 EDAA

"chroma". At decoding end, in a typical implementation, look-up tables are used to decode the syntax elements in incoming bit-stream. The decoding based on look-up table requires multiple memory accesses until the desired codeword is found. In this paper we propose an efficient constant-time decoding algorithm for CAVLC. Our algorithm accesses custom table and coded bit-stream only once and decoding time is independent of symbol length. We built small decoding tables by exploiting the regularity and correlation in code words given in standard with the focus on simplest possible decoding algorithm.

The rest of the paper is organized as follows: Section II highlights the existing CAVLC decoding methods; Section III presents our methodology and in Section IV experimental setup and comparison with existing approaches are presented. Section V concludes the paper.

II. RELATED WORK

In this section, we briefly review existing CAVLC decoding methods. Most of the work done so far targets hardware implementation of CAVLC decoding such as special instructions or SoC design. Regardless of hardware or software solution, the primary goal of the work done is to reduce the memory used for lookup tables and minimize computational costs and memory accesses. The basic method is to parse encoded stream bit-by-bit to decode the symbol information, but a more sophisticated method may take a chunk of bits e.g. 8 bits at a time to speed-up the decoding process. The most fundamental CAVLC decoding method is Table Lookup by Sequential Search (TLSS) and has been implemented by H.264 reference software [5]. This method sequentially compares one bit at a time with decoding table until the exact code and its length is found. As TLSS-based decoding is done bit-by-bit, it is very slow and requires a lot of computational power with numerous memory accesses. To mitigate computational cost of TLSS, Table Lookup by Binary Search (TLBS) is proposed in [9]. In TLBS method, VLC codes are first arranged in descending order with respect to their numerical value, and then binary search algorithm is used to search the entry in the decoding table. It is reported that TLBS algorithm is faster than the TLSS algorithm because only six table look-up operations are required to search the code from a table with 64 entries. In TLBS method, table access is random and may show inefficiency in some systems due to random memory accesses which are contrary to sequential memory accesses in TLSS, and its speed-up gain may evaporate when the length of incoming symbol is large.

To reduce the memory access of CAVLC decoding process, software decoding for some components is proposed in [16] and it is known as Moon's method. The main idea is to use arithmetic operations for highly probable short VLC codes to reduce memory access, and use conventional TLSS method to decode symbols with low probability of occurrence. Moon's method proposes arithmetic equations for some entries of coeff_token table 1 and 2 and for all entries of run_before tables only. The proposed algorithm contains many conditional statements with large arithmetic operations which may not help in reducing CAVLC decoding time. Moreover, it assumes that 95% of the symbols will have length shorter than 9 bits which is not valid in case of high fidelity video. Therefore, the decoding time of this approach varies drastically with length of the symbol and is not a constant, and computational complexity increases drastically with increase in symbol length. Moreover, in arithmetic operations, integer division and modulo operations are used which are not helpful to reduce the computational cost effectively, but is also contrary to the standard that it does not involve any floating point operation.

In [11], for some components, CAVLC decoding solution without look-up tables is proposed which is not general and only valid under assumptions and constraints. A VLSI decoding solution is suggested in [6] and is composed of several modules with large area overhead and not suitable for software implementation. Memory efficient solution using look-up tables is proposed in [15]. In this solution, small decoding tables are designed only with the perspective of hardware based solution.

III. OUR METHODOLOGY

As mentioned in Section 2, the approaches use either tables or derive arithmetic equations (possible for some components only) to decode the CAVLC. We propose a new way to construct CAVLC tables by exploiting the regularity and correlation among codeword symbols in the decoding table.. This regularity and correlation within table is not evident at first; and to make use of it, one has to manually check each entry with all possible choices, which is a lengthy and tedious job. Our method gives many fold benefits first decoding tables are small secondly decoding algorithm is fairly simple and lastly the decoding time is independent of incoming symbol length. We arranged the decoding table entries according to occurrence of regular pattern and correlation among code words. we used self grouping of code words based on number of leading zeros present in the code word and classify the symbols in groups, this self grouping exists in H.264 tables with few exceptions in total_zero table. Then we sorted the resultant group entries in ascending order after the occurrence of the first '1'. By grouping and sorting the stored information required to identify the symbols can be reduced significantly, hence reducing memory accesses. Based on this grouping we derived your decoding tables and decoding algorithm. Moreover the way we grouped the symbols leads to a very small table requirement and fairly simple decoding algorithm. Our grouping method makes possible the classification of incoming symbol in a group using one single instruction, available in most modern processors, as compared to other works which use multiple conditional statements to narrow down the search. Beside simple decoding process the prime aspect of our method is that decoding time of coded symbol is independent of its length e.g. the decoding time of 16 bit coded symbol is same as 4 bit coded symbol. It is worthwhile to mention that our decoding process comprises of two steps and total decoding time required to decode the coded symbol is smaller than the methods with we compared in results Section IV. Which means even if there is high probability of occurrence of shorter length coded symbols even then our method will out perform other methods though the overall benefits will diminish as compared to longer coded symbols.

A. Table Grouping

We exploited the regularity and correlation found among decoding symbols of CAVLD tables and classify them into groups according to number of leading zeros. The process is shown in Figure 1 for example data of coeff_token table 1 in H.264 CAVLD. The first column shows the resultant groups based on leading zeros and second column shows the actual code words and information required to be stored. The whole process can be divided in three steps.

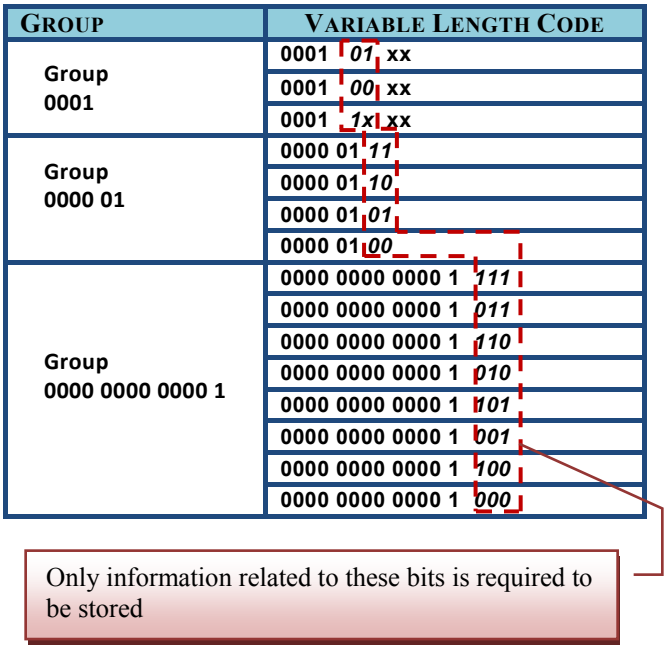


Figure 1: Table grouping process and information required to be stored.

- Arrange the decoding symbols in ascending order for leading zeros count.
- Arrange symbols which have same number of leading zeros in a same group, first column of Figure 1.
- After leading zero(s), the first occurrence of '1' is a signal, which acts as a classification of group. The occurrence of first 1 after leading zero also acts as sentinel and information bits follow it. Then sort the suffix bits and merge the information associated with the symbol to build the decoding tables with minimum memory requirements.

After classifying the group, the information that needs to be stored (for decoding the incoming symbol) comprises of a few bits and is shown in Figure 1 as a vertical group. Considering an example in Figure 1, the table size is totally 114 bits. After grouping the table entries, there is need to store the information related to 38 bits only. Information bits occurring after the first 1 are few as compared to the total length of code, which significantly reduces the permutations, and hence memory requirements of tables. Here afterwards we will call the leading zeros and first '1' as prefix and remaining information bits as suffix.

B. CAVLC Final Tables

The CAVLC tables are generated by grouping and sorting as described previously and the procedure is similar for each CAVLC table. In H.264, coeff_token syntax has four tables and selection of appropriate table depends on the context. Context table selection is based on average number of non-zero transform coefficient levels of top and left 4x4 luma blocks and it is -1 for chroma blocks. VLC table of the current block is adaptively selected according to neighboring blocks' non-zero transform coefficient levels.

To elaborate our method further, Table I and II shows the grouped table with associated information. As CAVLC tables are large and it is not possible to show complete tables here, only small portions are shown to elaborate our grouping and merging of tables. Table I shows the part of coeff_token table 0, TC and T1 represent the total number of coefficient and total number of ones respectively associated with the symbol. We partitioned the code in prefix and suffix. As explained earlier, group is made on the basis of prefix and we only stored the information related to suffix. We divided the whole table in 14 groups and similarly coeff_token table 2 is divided in 10 groups. In this way our method is seamlessly applicable to nearly all tables. Table II show the parts of total_zero table which is classified into 21 groups. The appropriate group selection is based on context, total coefficients decoded already, and leading zeros, and difference here is that already decoded information, total number of coefficient and symbol reveal the total zeros. There is exception in total_zero table that a few of the entries cannot be merged and grouped because of irregularities. We grouped all these entries in a separate group and used all of their permutations to keep the decoding algorithm simple which incurr a little memory overhead.

Similarly, all coeff_token, run_before, zero_left tables for luma and chroma tables are generated by the same processs. As mentioned earlier we only need to store the tables related to suffix entries as classification in a group can be done by finding the number of leading zeros and from now onwards we will use the term suffix table as decoding table.

Table I: The grouping for part of coeff_token table 0.

Code	Prefix	Suffix	[TC, T1]
0000 0100	0000 01	00	[6, 3]
0000 0101		01	[4, 2]
0000 0110		10	[3, 1]
0000 0111		11	[2, 0]
0000 0010 0	0000 001	00	[7, 3]
0000 0010 1		01	[5, 2]
0000 0011 0		10	[4, 1]
0000 0011 1		11	[3, 0]
0000 0000 0100 0	0000 0000 01	000	[8, 0]
0000 0000 0100 1		001	[9, 2]
0000 0000 0101 0		010	[8, 1]
0000 0000 0101 1		011	[7, 0]
0000 0000 0110 0		100	[10, 3]
0000 0000 0110 1		101	[8, 2]
0000 0000 0111 0		110	[7, 1]
0000 0000 0111 1		111	[6, 0]
0000 0000 0010 00	0000 0000 001	000	[12, 3]
0000 0000 0010 01		001	[11, 2]
0000 0000 0010 10		010	[10, 1]
0000 0000 0010 11		011	[10, 0]
0000 0000 0011 00		100	[11, 3]
0000 0000 0011 01		101	[10, 2]
0000 0000 0011 10		110	[9, 1]
0000 0000 0011 11		111	[9, 1]

Table II: The grouping for part of total_zero table.

Code	Prefix	Suffix	[TC, T0]
100	1	00	[3, 1]
101		01	[3, 2]
110		10	[3, 3]
111		11	[3, 6]
0000 10	0000 1	0	[1, 7]
0000 11		1	[1, 8]
0000 010	0000 01	0	[1, 9]
0000 011		1	[1, 10]

We need to store only the necessary information: in our case it is just the suffix bits occurring after first one and constructed the suffix tables. It is evident that there is variation in the length of suffix bits in a given table but is limited to one or two bits only. We decided to use maximum length of suffix bits occurring in a table across multiple groups to construct our decoding tables. Surely this will increase the number of permutations which results in increased size of suffix table. This increase in size is negligible and is very little compared to the total size of the tables. This will eliminate the exceptions in decoding algorithm which results in simple decoding algorithm without conditional statements. All CAVLC suffix tables take less than 8 KB in total which make them ideal to be placed in scratchpad memory, and if placed in main memory it is highly likely that when accessed tables will be cached automatically due to small size.

C. CAVLC Decoding Algorithm

We developed the fairly simple decoding algorithm in conjunction to the our table grouping method. The Algorithm 1 represent our decoding algorithm; though it shows the decoding of coeff_token syntax elements for luma, but can be adapted for remaining syntax elements just by changing the suffix table. Moreover it is the algorithm of full routine to decode coeff_token table and conditional statements in the algorithm are for context adaptation means selecting the different decoding table depending on the context. The decoding algorithm is three step process which include reading of bit-stream, classification of incoming symbol to its group and then decoding the suffix table using our suffix tables.

We first read the bits to decode from bit stream, and the number of bits we read is equal to the maximum length of the symbol in a given table. To decode coeff_token and run_before we read 16 bits and read 8 bits to decode total_zero, line 1 in algorithm. So we in our method bit-stream reading is done only once for decoding of syntax element in comparison to multiple readings in other works. After reading the sufficient bits, we classify the codeword into its group by counting the number of leading zeros. The counting of leading zeros is fairly fast and simple as most of the modern processors include assembly instruction for this purpose (commonly used for cryptography and graphics operations). Assembly instruction MSU and NSAU are examples in Intel x86 and Xtensa processors respectively. Even if the leading zero count instruction is not available in the processor, it is possible to compute it in constant time using bit-twiddling algorithms with little overhead of memory and cycles given in [20], line 2 in algorithm counts leading zeros. Till now we have classified the incoming symbol to its group after this the the extraction of information bits or suffix bits is done by shifting, line 4 and 5. We have obtained the suffix bits and use these bits to extract the decoding information from suffix table, line 7 to 11. The remaining part is repetition of last step in decoding which is for different suffix table selection based on context.

Our decoding method is straightforward and is independent of codeword length and without any conditional statement which results in constant decoding time for a incoming symbol. Due to constant decoding time, regardless of its codeword length, the performance of our algorithm is scalable for high fidelity videos in which probability of occurrence of shorter length code-words decreases significantly and adversely affects the performance of conventional algorithms.

Algorithm 1: Pseudo code of our coeff_token decoding routine

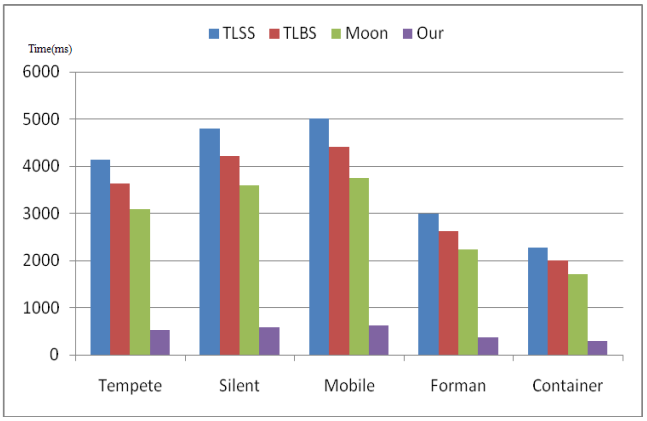
```
1:  code ← show_bits(16)  // read 16 bits from bitstream
2:   idx_one ← find_index_one(code) // using only
3:                                  // one instruction
4:  code ← code >> 16 – (idx_one + 3)
5:  suffix ← code && 0x0007 //extract last three bits
6:  if(vlcnum == 0) { // first vlc table
7:       info ← coeff_token_suff_0[idx_one>>3 + suffix]
8:              // access 32 bits info from table
9:      num_coeff ← info && 0xff
10:     num_trailing_one ← ( info >> 8) && 0xff
11:     code_length ← ( info >> 16) && 0xff
12
13: }else if(vlcnum == 2) { // second vlc table
14:       info ← coeff_token_suff_1[idx_one>>3 + suffix]
15:              // access 32 bits info from table
16:     num_coeff ← info && 0xff
17:     num_trailing_one ← ( info >> 8) && 0xff
18:     code_ length ← ( info >> 16) && 0xff
19:
20: }else { // Third vlc table
21:       info ← coeff_token_suff_2[idx_one>>3 + suffix]
22:              // access 32 bits info from table
23:     num_coeff ← info && 0xff
24:     num_trailing_one ← ( info >> 8) && 0xff
25:     code_ length ← ( info >> 16) && 0xff
26: }
27: frame_bitoffset ← frame_bitoffset + length
```

IV. EXPERIMENTAL RESULTS

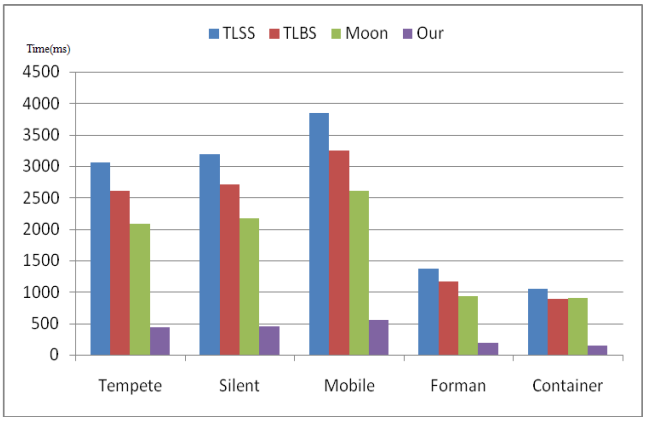
A. Experimental Setup

We performed multiple experiments using test sequences with diverse characteristics to avoid the influence of system and data properties. To encode the sequences, latest reference encoder JM 13 is used. Table III shows the encoding parameters. Each sequence is encoded with four different QPs to highlight the scalability of proposed method. Though baseline profile is used for encoding but proposed method is not limited to baseline and conformance testing for Main profile was done successfully. Table IV shows the test sequences used in experiments and their parameters. We integrated each CAVLC algorithm in our optimized decoder to get fair results. Application binaries were generated with instrumentation code which is required to get different performance metrics at runtime. The inclusion of instrumentation code will definitely increase the execution time, but it is not important because we are interested in relative performance of algorithms.

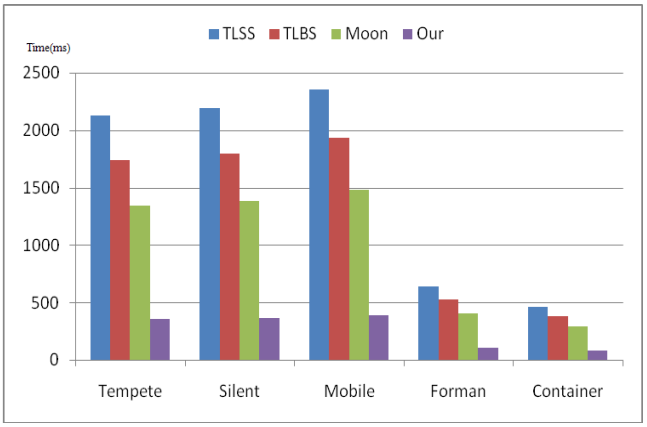
Experiments were performed on 1.6GHZ Intel Core Duo machine with 2 MB L2 cache and application thread was locked to one of two processors. We measured the performance metrics using Intel VTune performance analyzer which give system wide performance analysis. To get precise readings, data was gathered by Event-based sampling using processor counters.



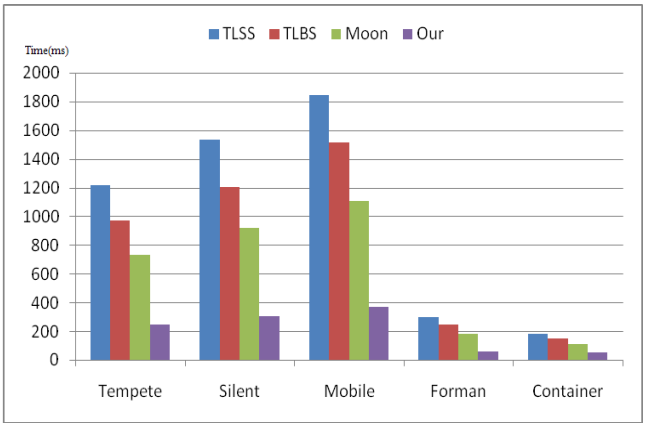
(a)



(b)



(c)



(d)

Figure 2: Execution time comparison for test sequences (a, b, c and d are for QP=16, 20, 24 and 28 respectively).

Table III: Encoding parameters of test sequences.

Parameter	Selected option
Profile	Baseline
RDO	On(fast algorithm)
MV search range	+/- 32 pixels
Reference frame	4
QP	16, 20, 24, 28
Motion search	Fast search
Intra interval	0
Encoder	JM 13

Video output file writing was disabled and input bit-stream was loaded first into memory to eliminate thread stalling due to IO operations. In each experiment, data was gathered using five applications runs including one calibration run. We gathered execution time as clock ticks and converted it to millisecond units. Memory access was counted as all data memory references made to the L1 data cache, including all loads from and to the memory within CAVLC routine.

B. Results

We compared our proposed method with TLSS, TLBS and Moon's algorithms in term of execution time and memory access count. Our method clearly outperforms others in execution time and performs better in memory access count. Figure 2 shows the execution time comparison for different values of QP and Figure 1

Table IV: Parameters of test sequences.

Sequence	Resolution	Frame rate	Frames
Tempete	CIF	30	60
Silent	CIF	25	60
Mobile	CIF	20	60
Foreman	CIF	35	60
Container	CIF	15	60

(a), (b), (c) and (d) corresponds to QP values 16, 20, 24 and 28 respectively. As mentioned earlier time is measured in milliseconds and binaries were compiled with instrumentation code so it has the meaning of relative time. The comparison shows that the proposed method is 7 times faster than TLSS and roughly 5 times faster than TLBS and Moon's method. Maximum relative speed is achieved when QP is equal to 16 in which longer length symbols are more probable. Our speed-up gain diminishes with increase in QP (decrease in video fidelity) and at QP = 28 it is just 5 times the TLSS. Which means compared to other methods computational requirement of our method increases less sharply with the decrease of QP. Contrary to TLSS, TLBS and Moons method, our method reads bit-stream once and there are no conditional statements or loops involved while decoding the incoming coded symbol. Moons method assumes that 85% of codes will have length shorter than 8 bits but we are not making any assumption on the probability of occurrence of code length. Due to these reasons, our decoding is independent of code length within table and its

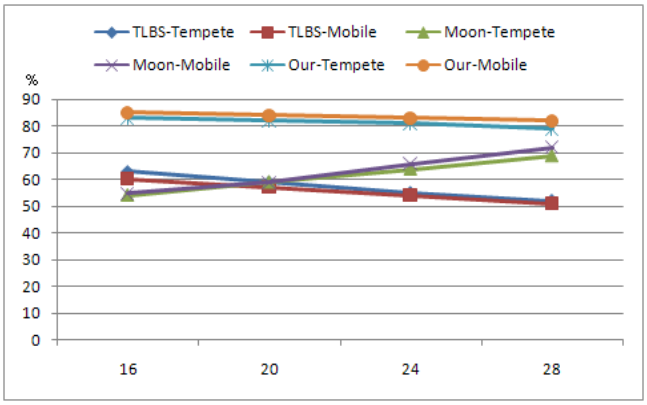


Figure 3: Memory access savings.

computation time only depends on bit-rate. Constant time decoding of code makes it scalable with increase in bit-rate and is ideal for multi-core implementations in which a single processor has to bear the work load of entropy decoding.

Our proposed algorithm performs well in the sense of memory access count compared to other algorithms. While in memory access count, we excluded memory accesses related to encoded bit-stream for all algorithms as it will be accessed sequentially and will always be in cache memory. Figure 3 shows the percentage savings in memory access count with TLBS, Moon and our method compare to TLSS. Our method uses only 13% memory accesses as compared to TLSS and savings are around 20% better than TLBS and Moon's method, when QP =16. The memory access savings gain decreases monotonically and reaches Moon's method gain with increase in QP and is in aligned with the execution time results. But our method uses small table (less than 8 KB), and it accesses table only once for the decoding of one code compared to others.

V. CONCLUSION

In this paper we proposed efficient constant time CAVLC entropy decoding method for H.264/MPEG-4 AVC decoder. The proposed algorithm uses table grouping method which leads to reduced table look-up requirement, where codes are grouped using the regularity and correlation within table. To classify the code into its group, we used a single instruction. We defined custom tables for all entries of coeff_token, run_before, total_zeros CAVLC tables. The proposed method and decoding algorithm can overcome conventional table look-up methods drawbacks, such as high memory acesses and slow decoding time due to bit-by-bit processing. The proposed method also saves memory by using minimal possible CAVLC tables. Experiment results show that reduction in memory access is upto 83% and speed-up of CAVLC decoding time is upto 7x. Due to constant code decoding time, the computational requirement of the proposed method increases steadily with increase in bit-rate which makes it scalable and ideal for multicore implementations, as only one processor has to bear the load of whole entropy decoding process.

VI. REFERENCES

[1] Anant Agarwal, Markus Levy, "The kill rule for multicore". Design Automation Conference (DAC), pp. 750-753, 2007.

[2] Shekhar Borkar, Norman P. Jouppi and Per Stenström, "Microprocessors in the Era of Terascale Integration", Design, Automation and Test in Europe Conference (DATE), pp. 237-242, 2007.

[3] JVT Draft recommendation and final draft international standard of joint video specification. ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC .

[4] Dragorad Milovanovic, Zoran Bojkovic and Andreja Samcovic, "Video Coding with H.264/AVC : Tools performance and complexity", WSEAS AMTA-EE-ICAI-MCBC-MCBE Conference, Venice, Italy, 2004.

[5] http://iphome.hhi.de

[6] W. Di, G. Wen, H. Mingzeng, and J. Zhenzhou, "A VLSI architecture design of CAVLC decoder," 5th IEEE International Conference on ASIC, vol. 2, pp. 962-965, Oct. 2003.

[7] C. Meenderinck, A. Azevedo, M. Alvarez, B. Juurlink, and A. Ramirez, " Parallel Scalability of H.264", In Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG), 2008.

[8] Van der Tol, E., Jaspers, E., Gelderblom, R, " Mapping of H.264 Decoding on a Multiprocessor Architecture", In: Proc. SPIE Conf. on Image and Video Communications and Processing, vol. 5022 pp. 707-718, 2003.

[9] Yong-Hwan Kim, Yoon-jong Yoo, Jeongho Shin, Byeongho Choi, and Joonki Paik, " Memory-Efficient H.264/AVC CAVLC for Fast Decoding", IEEE Trans. On Consumer Electronics, Vol.52, No3, August 2006

[10] S. Sudharsanan et al., "Image and video processing using MAJC 5200", Proceedings of the IEEE International Conference on Image Processing, Vancouver, Canada, 2000.

[11] Jeonhak Moon, Seongsoo Lee, "Design of H.264/AVC Entropy Decoder Without Internal ROM/RAM Memories", ISCCSP, March 2008.

[12] Y. K. Chen, E. Q. Li, X. S. Zhou and S. Ge, "Implementation of H.264 Encoder and Decoder on Personal Computers", Journal of Visual Communication and Image Representation, pp.509-532, 2006.

[13] Jike Chong, Nadathur Rajagopalan Satish, Bryan Catanzaro,Kaushik Ravindran, Kurt Keutzer. "Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling". Multimedia and Expo, IEEE ICMCS Conference, pp. 1874-1877, 2007.

[14] P. Li, B. Veeravalli and A. A. Kassim, " Design and implementation of parallel video encoding strategies using divisible load analysis", IEEE Trans. Circuits and Systems for Video Technology, pp. 1098-1112, 2005.

[15] Hsiu-Cheng Chang, Chien-Chang Lin, Jiun-In Guo "A Novel Low-Cost High-Performance VLSI Architecture For MPEG-4 AVC/H.264 CAVLC Decoding",IEEE ISCAS, pp. 6110-6113, 2005.

[16] Y. H. Moon, G. Y. Kim, and J. H. Kim, " An Efficient Decoding of CAVLC in H.264/AVC Video Coding Standard," IEEE Trans. On Consumer Electronics, Vol.51, No3, August 2005.

[17] Iain E.G. Richardson, "H.264 and MPEG-4 Video Compression – video coding for next generation multimedia", John Wiley & Sons, 2003.

[18] M. Horowitz, A. Joch, F. Kosssentini, and A.Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," IEEE Trans. Circuits and Syst. Video Technology., vol. 13, no.7, July 2003.

[19] http://www.intel.com/products/processor/core2quad/index.htm

[20] http://www-graphics.stanford.edu/~seander/bithacks.html

[21] http://www.intel.com/pressroom/kits/Teraflops/index.htm