A PATTERN-SEARCH METHOD FOR H.264/AVC CAVLC DECODING

Shau-Yin Tseng *Tien-Wei Hsieh*

SoC Technology Center
Industrial Technology Research Institute
Hsinchu, Taiwan 310, R.O.C.
{Tseng, TWHsieh }@itri.org.tw

ABSTRACT

In this paper, a new implementation method is proposed for Context-Adaptive Variable Length Coding (CAVLC) used in H.264 Baseline Profile. We analyze the correlation between bit patterns and 4x4 (or 2x2) blocks and have an idea of a pattern-search method before CAVLC decoding. If a pattern is matched in our look-up table, we can skip the standard CAVLD procedure and reconstruct a block directly. However, if there is not any pattern matched in the table, we have to reconstruct a block by CAVLD. Our look-up tables are built up according to our statistics and analysis. The experimental results show that the performance can be improved 10% compared with the standard CAVLD procedure.

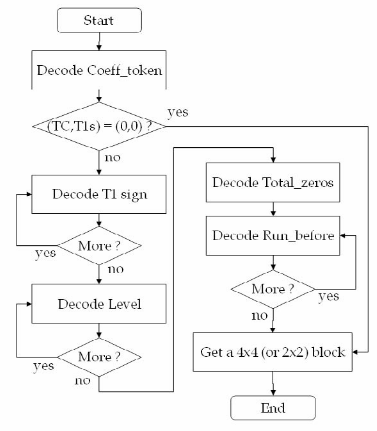

Fig. 1 CAVLC decoding flow

1. INTRODUCTION

H.264/AVC is a novel video coding standard, which is developed by Joint Video Team (JVT) of ISO/IEC Motion Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group [1] [2]. It has a number of features and functionalities which provide a considerable improvement over the previous coding standards.

In order to make the implementation flexible and cost effective over a variety of products and product generations, it is interesting in developing multimedia application software running on a programmable CPU or DSP. However, the complexity of H.264/AVC is more than previous standards. A software-based real-time decoder requires more powerful processors and faster algorithms [3] [4]. According to computational complexity, we know that motion compensation with pixel interpolation, entropy decoding with CAVLD and de-block filter consume more time.

The essential behavior of CAVLD is similar with VLD of previous standards. There are several works for VLD software implementation [5] [6]. Those fall into two classes: bit-serial methods and bit-parallel methods. Bit-serial methods are not very suitable for high-performance real-time application software because of the long time period needed for decoding a long codeword. On the contrary, bit-parallel methods can reduce memory access and increase performance. The easiest implementation is to use a table look-up; input bits address a table which contains the decoded symbol and the length of the code. The length of the code determines at which point of the bit-stream is advanced. The look-up table must be addressed by the maximum size of codes. However, it is wasteful because the shorter codes have many repeated entries in the table. One approach is to use multi-pass look-ups. First, a few bits of the bit-stream is looked up in a table. If there is not any code matched in the table, the second look-up into another table then is executed. Although this method saves memory, it consumes more processing time.

Besides, there are several works for CAVLC decoding [7] - [11]. Most of them are implemented in hardware and adopt table look-ups. Three interesting ideas can be observed among them. First, they use various methods to partition VLD tables to save memory space. Second, it is inefficient to look up Run_before tables, so arithmetic operation is adopted to replace them. Third, building multi-symbol VLD tables, CAVLD can decode consecutive

000100 111100 → 2,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0

Codewrod	Element	Value	Decoding output
000100	Coeff_token	TC=2, T1=1	Empty
1	Sign of T1	-	-
1	Level	+2	2,-1
110	Total_zeros	1	2,-1
0	Run_before	1	2,0,-1

Fig. 2 An example of the CAVLC decoding process

multiple symbols once.

According to the statistics of [12], it tells us that only 60% of 4x4 blocks are decoded within 15 bits. With this characteristic, we propose an efficient algorithm for CAVLD in this paper. Based on the statistics of frequencies of bit patterns, we propose a pattern-search method which reconstructs 4x4 (or 2x2) blocks directly if patterns are matched in look-up tables. In addition, this efficient pattern-search method also reduces memory access.

The rest of this paper is organized as follows. In Section 2, we analyze the correlation between bit patterns and 4x4 (or 2x2) blocks, and propose a pattern-search method before CAVLD. Experimental results are presented in Section 3. Finally, Section 4 gives a conclusion.

2. THE PROPOSED PATTERN-SEARCH METHOD

The essential behavior of CAVLD is similar with VLD of previous standards. However, it uses several extensive dedicated code tables, and those tables are chosen by the context of previous blocks or symbols. There are six decoding steps of CAVLD, and each step uses different tables. Fig. 1 shows the CAVLD flow.

1. Coeff_token: The total number of non-zero coefficients (TC) and the number of trailing ±1 values (T1s) are decoded. TC is ranged from 0 to 16, and T1s is ranged 0 to 3. The choice of look-up tables depends on nC, which is an average of numbers of non-zero coefficients in upper and left-hand decoded blocks.
2. Sign of T1: According to T1s, a number of single bits are decoded.
3. Level: According to TC, a number of non-zero coefficients are decoded. The choice of look-up tables depends on the previous decoded Level value.
4. Total_zeros: The total number of zeros preceding non-zero coefficients is decoded. The choice of look-up tables depends on TC.
5. Run_before: The number of zeros preceding each non-zero coefficient is decoded. The choice of look-up tables depends on the number of zeros left.

Table I
The most frequent patterns and
the corresponding zig-zag ordered coefficients

	Pattern	nC	Zig-zag ordered 16 coefficients
1	1	0,1	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2	11	2,3	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3	0111	0,1	-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4	01	-1	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5	0101	0,1	1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6	1111	4~7	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7	1011	2,3	-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8	1001	2,3	1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
9	011011	0,1	0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10	010011	0,1	0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0

6. Reconstruct a 4x4 (or 2x2) block according to a number of Signs, Levels and Run_befores.

For example, 16 zig-zag ordered coefficients (2, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) are reconstructed from the bit pattern (0001 0011 1100), as shown in Fig. 2. In other words, if the bit pattern (0001 0011 1100) is matched, then zig-zag ordered coefficients of a 4x4 block (2, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) is obtained, assumed that nC is 0. This is the basic concept of our proposed pattern-search method.

2.1. Analysis of correlation between patterns and blocks

Although [12] has gathered statistics of frequencies of 4x4 blocks, it did not distinguish those blocks with nC values. The same 4x4 blocks can be encoded as different bit patterns due to different nC values. For example, while nC is 0, the 4x4 block (2, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) is encoded as the bit pattern (0001 0011 1100). However, while nC is 2, the 4x4 block (2, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) is encoded as the bit pattern (0011 1111 100). Our patterns come from the combination of blocks and nCs. We analyze seven test sequences, Mobile, Foreman, Carphone, Silent, News, Highway and Claire. The format of each sequence is 4:2:0 QCIF and 100 frames (IPPPP). Top 10 frequent patterns and the corresponding zig-zag ordered coefficients are shown in Table I. Bit lengths of these patterns are within 6 bits.

We sample 4,000 frequent patterns and arrange them according to their frequencies. In our statistics, sum of frequencies of top 4,000 patterns occupies 67.63% of number of decoded block, as shown in Fig. 3. Besides, sum of frequencies of top 500 patterns occupies 64.85%. It is shown that patterns rarely appear except top 500 patterns. Among these 500 patterns, there are 38.11% of patterns belonging to nC = 0 or 1; the maximum size of these patterns is 16 bits, as shown in Fig. 4.

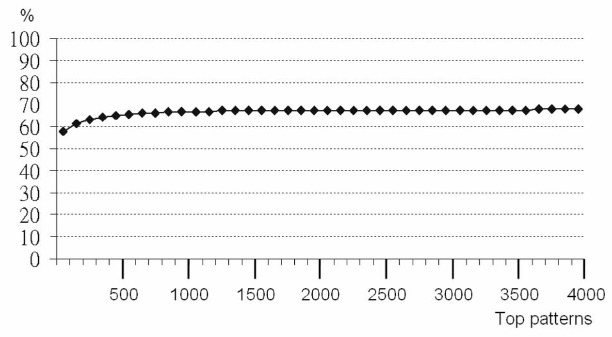

Fig. 3 Statistics of frequencies of patterns
(4,000 patterns arranged by frequency)

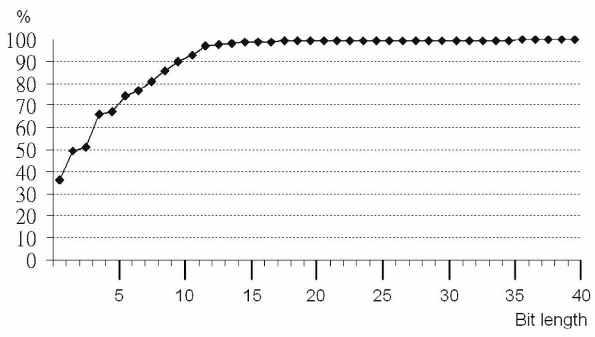

Fig. 5 Statistics of frequencies of patterns
(4,000 patterns arranged by bit length)

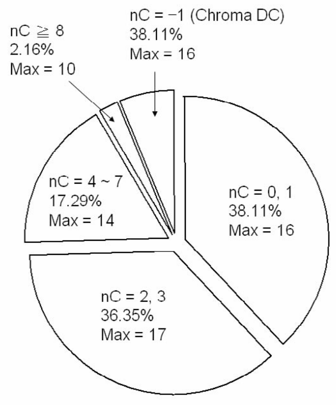

Fig. 4 Analysis of the top 500 frequent patterns

The 4,000 patterns which we sample have a variety of bit length from 1 to 77. We re-arrange the order of these 4,000 patterns according to their bit lengths, as shown in Fig. 5. There are 81.07% of patterns represented within 8 bits and 96.93% of patterns represented within 12 bits.

2.2. A pattern-search method before CAVLD

The pattern-search algorithm is shown in Fig. 6. We use a two-pass table look-up pattern-search method to reconstruct 4x4 (or 2x2) blocks directly. The first pass is reading 8 bits, and the second pass is reading 4 bits. If there is not any pattern matched in our look-up table, we have to reconstruct blocks by CAVLD.

According to our statistics, there are many patterns, but most of them only appear once. From this, our method only supports frequent patterns. The maximum size of our patterns is 12 bits because there are most of frequent patterns represented within 12 bits. However, a single 12-bit pattern look-up table is wasteful. The two-pass table look-up method is adopted to save memory space; we read 8 bits in the first pass and 4 bits in the second pass. It is efficient to reconstruct most of block one time only.

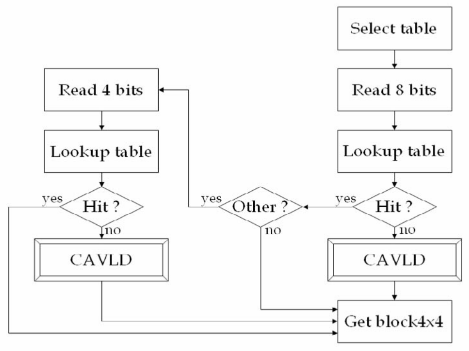

Fig. 6 Our pattern-search algorithm flow

There are four 8-bit pattern look-up tables and twenty-one 4-bit pattern look-up tables we build up. In the beginning of CAVLD, we choose 8-bit pattern look-up tables according to nC. Then, we use 8 bits to address the look-up table; the entry of the table has 32 bits which can tell us what is the length of the pattern, what block is reconstructed, how many non-zero coefficients are in the block, and what 4-bit pattern look-up table is chosen. If a pattern is matched in 8-bit pattern look-up table, we will decide to read 4 bits further or reconstruct a block directly. If there is not any pattern matched in the table, we have to use CAVLD to reconstruct a block. The 4-bit pattern look-up table is chosen by the entry of the 8-bit pattern look-up table. If a pattern is matched in 4-bit pattern look-up table again, a block is then reconstructed. Otherwise, we have to use CAVLD eventually; that is the worst case. Note that we do not build up a look-up table for nC ≧ 8 because the pattern-search method will be not more efficient than original procedure; we just use one pass for chroma DC because it is enough to cover most of frequent 2x2 blocks.

3. EXPERIMENTAL RESULTS

Because of our application target, we use ARMulator (ARM920T) to estimate our algorithm performance in an

Table II
Pattern-search hit rate of test sequences

	Decoded blocks	Matched patterns	Hit rate
Claire	29,198	21,563	73.85%
Highway	41,675	33,872	81.28%
Carphone	57,853	41,877	72.39%
Silent	62,253	42,127	67.67%
News	53,445	33,837	63.31%
Foreman	65,572	46,587	71.05%
Mobile	152,310	80,975	53.16%

ARM-based embedded system. There are seven test sequences, Mobile, Foreman, Carphone, Silent, News, Highway and Claire. All of them are 4:2:0 QCIF and 100 frames (IPPPP).

There are 41,675 blocks CAVLD needs to decode in Highway, and our pattern-search method can directly reconstruct 33,872 blocks, as shown in table II. The hit rate of Highway is 81.28%. The hit rate of Mobile is 53.16% because Mobile is a critical sequence, which has a variety of patterns. Since our algorithm can reduce memory access, it improves 14.86% performance in terms of processing cycles in Highway and 6.12% in Mobile, as shown table III.

4. CONCLUSION

We propose a pattern-search method before CAVLD. It can reconstruct a 4x4 (or 2x2) block directly without going through CAVLD. Number of table look-ups is less than two; thus, our pattern-search method can reduce memory access and speed up about 10% performance against the standard CAVLD procedure.

Moreover, we observe that behaviors of inverse transformation and inverse quantization are regular and confined to coefficients of the 4x4 block. Therefore, we can build new pattern-search tables to reconstruct inverse transformed 4x4 blocks to skip CAVLD and IT/IQ.

5. REFERENCES

[1] Joint Video team (JVT) of ISO/IEC MPEG and ITU-T VCEG, “Information Technology — Coding of Audio-visual Objects — Part 10: Advanced Video Coding”, *ISO/IEC 14496-10*, ISO/IEC, Switzerland, Dec. 2003.

[2] Iain E. G. Richardson, *H.264 and MPEG-4 Video Compression*, John Wiley & Sons, England, Sept. 2003.

[3] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, ”H.264/AVC Baseline Profile Decoder Complexity Analysis”, *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, vol. 13, pp. 704-716, July 2003

[4] X. Quan, L. Jilin, W. Shijie, and Z. Jiandong, “H.264/AVC baseline profile decoder optimization on independent platform”, *Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing,* IEEE, vol. 2, pp. 1253-1256, Sep. 2005.

Table III
Performance improvement of test sequences
(using ARMUlator (ARM920T) to count cycles)

	Entropy decoding with standard CAVLD	Entropy decoding with pattern-search method	Improvement
Claire	14,612,810	12,738,779	12.82%
Highway	20,639,203	19,572,458	14.86%
Carphone	29,777,102	25,990,674	12.72%
Silent	30,334,038	26,373,234	13.06%
News	30,725,393	28,079,039	8.61%
Foreman	34,192,569	29,959,594	12.38%
Mobile	89,094,039	83,645,173	6.12%

[5] S. Sriram and C. Y. Hung, “MPEG-2 Video Decoding on the TMS320C6X DSP Architecture”, *Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems & Computers*, IEEE, Pacific Grove, CA, vol. 2, pp. 1735-1739, Nov. 1998.

[6] D. Ishii, M. Ikekawa, and I. Kuroda, “Parallel variable length decoding with inverse quantization for software MPEG-2 decoders”, *IEEE Workshop on Signal Processing Systems*, IEEE, Leicester, pp. 500-509, Nov. 1997.

[7] S. Kato, K. Sugimoto, S. Adachi, and M. Etoh, “Structured "Truncated Golomb Code" for Context-Based Adaptive VLC”, *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, IEEE, vol. 1, pp. 323-326, Sept. 2003.

[8] W. Di, G. Wen, H. Mingzeng, and J. Zhenzhou, “A VLSI Architecture Design of CAVLC Decoder”, *Proceedings of 5th International Conference on ASIC*, IEEE, vol. 2, pp. 922-925, Oct. 2003.

[9] T. W. Chen, Y. W. Huang, T. C. Chen, Y. H. Chen, C. Y. Tsai, and L. G. Chen, “Architecture Design of H.264/AVC Decoder with Hybrid Task Pipelining for High Definition Videos”, *International Symposium on Circuits and Systems*, IEEE, vol. 3, pp. 2931-2934, May 2005.

[10] H. C. Chang, C. C. Lin, and J. I. Guo, “A Novel Low-cost High-performance VLSI Architecture for MPEG-4 AVC/H.264 CAVLC Decoding”, *International Symposium on Circuits and Systems*, IEEE, vol. 6, pp. 6110-6113, May 2005.

[11] Y. H. Moon, G. Y. Kim, and J. H. Kim, “An Efficient Decoding of CAVLC in H.264/AVC Video Coding Standard”, *IEEE Transactions on Consumer Electronics*, IEEE, vol. 51, pp. 933-938, Aug. 2005.

[12] S. Y. Tseng and J. G. Hsu, “The Profile of H.264”, *SoC Technical Journal*, STC/ITRI, Taiwan, vol. 3, pp. 111-119, Nov. 2005.