CAVLC解码的一种有效方法

曹 宁 梅 侠
(河海大学电气工程学院, 南京 210098)

摘 要 已有的 CAVLC解码方法包括二叉树解码方法、全码表解码方法和 Hashemian解码方法等, 但是这些解码方法都只关注解码性能的一个方面: 解码速度或存储空间, 因而无法有效地提高整体性能。针对这一问题提出了一种快速的解码方法。该方法通过自动码表分配技术和码表地址转移技术来提高限定存储空间条件下的解码速度。实验结果表明, 使用相同的存储空间, 该方法的速度是传统解码方法的 1.5倍, 更加适用于 H.264标准。

关键词 CAVLC 二叉树 全码表 Hashemian方法 自动码表分配 码表地址转移
中图法分类号: TN919.81 **文献标识码**: A **文章编号**: 1006-8961(2008)02-0230-04

An Effective Way of CAVLC Decoding Methods

CAO Ning MEI Xia
(College of Electrical Engineering, Hohai University, Nanjing 210098)

Abstract CAVLC decoding methods known by people include bintree, full code table, and Hashemian decoding methods, etc. all of which lay emphasis on only one aspect of decoding, decoding speed or memory space, and thus the general capability can not be improved effectively. In view of that, the paper puts forward a new and fast decoding method which by means of the automatic distribution of the code table and the transfer of the code table address, improves the decoding speed in the limited memory space. As the experiment shows, in the same-size memory space, the speed of this new method is 1.5 times as fast as that of the tradition ways so that the new one is more suitable to H.264.

Keywords CAVLC, bintree, full code table, Hashemian method, the automatic distribution of the code table, the transfer of the code table address

1 引 言

ITU-T VCEG (video coding experts group)和 ISO/IEC MPEG (moving pictures experts group)共同组建的联合视频专家组 JVT于 2003年正式发布了新一代视频编码标准 H.264/AVC。H.264/AVC是在 H.26L的基础上推出的, 它的压缩效率是 MPEG-4或 H.263+的 2倍左右。H.264将主要用于视频会话、网络视频流和消费电子应用等。

在 H.264的 3个应用档次中都包含了 CAVLC编码(基于内容的自适应变长编码), 特别是基本档次和扩展档次只使用 CAVLC一种熵编码方式, 因此 CAVLC的编码码表和解码是现在研究的一个方向。对于 CAVLC的解码, 已经有了二叉树、全码表和 Hashemian等解码方法, 但是这些解码方法都只考虑了解码性能的一个方面, 即解码速度或者存储空间, 本文探讨一种基于 Hashemian方法的改进方法, 它使用自动化的方法来获得更好的解码码表分配。

2 CAVLC

H.264的早期草案中使用 UVLC(统一的可变长编码)对残差数据进行熵编码, 但是由于 UVLC没有利用相邻的 4×4变换块之间的关系, 因而编码效率不高, 于是在最终草案中就用 CAVLC来代替

收稿日期: 2006-06-07; 改回日期: 2006-10-17

第一作者简介: 曹宁(1962~), 男, 教授, 硕士生导师。1990年于东南大学无线电系获信号处理专业硕士学位, 长期从事多媒体图像压缩与传输、数字通信以及数字信号处理方面的研究和教学工作。E-mail: caoning@vip.163.com

UVLC。CAVLC具有以下的优点[1]:

(1) 基于上下文的自适应编码方法可全面提高编码的质量。

(2) 应用锯齿形扫描方法, 将非零系数值(Level)和零行程(Run)分开编码, 可提高编码的自适应性。

(3) 将非零系数值(Level)和零行程(Run)分开可降低存储数据的复杂度。

CAVLC对残差数据的编码过程如下[2]:

(1) 非零系数的数目(TotalCoeffs)以及拖尾系数的数目(TrailingOnes)进行编码。

(2) 对每个拖尾系数(Tls)的符号编码。

(3) 对除拖尾系数外的非零系数的幅值(Levels)编码。

(4) 对最后一个非零系数前的总的零的数目(TotalZeros)编码。

(5) 对每个非零系数前零的数目(RunBefore)编码。

编码过程实际上就是一个查表寻找码字的过程, 而解码过程则是一个根据码字反相查找的过程。由于码字不是统一长度的, 而是出现频率高的使用短码字, 出现频率低的则使用长码字, 这样在解码的时候就需要先确定码字的长度。

3　已经被提议的解码方法

VLC(可变长编码)广泛应用于各种视频压缩标准, 例如MPEG-2, MPEG-4, H. 26L等, 因此已经有很多VLC的解码方法, 下面介绍 3种已经被提议的方法。

3 1　二叉树解码方法

二叉树解码方法的原理就是将码表存储结构的码字转化为树形存储结构的码字, 即形成 1棵二叉解码树。解码过程实际上就是一个寻找叶节点的过程。图 1为 1棵简单的二叉解码树。

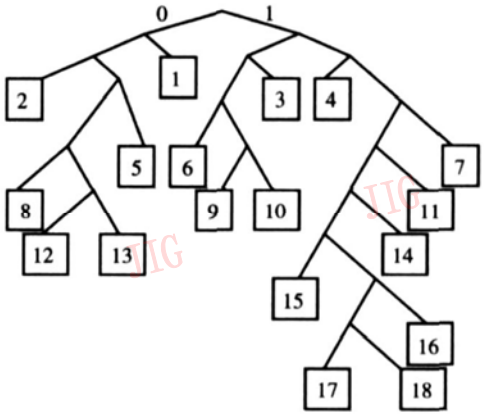

图 1　二叉解码树

Fig 1　Decoding bintree

二叉树方法充分利用了码字之间的相关性, 对存储空间的消耗非常少, 但是这种解码方法是以 1个比特作为比较对象的, 所以它的解码速度很慢。

3 2　全码表解码方法

由于编码码表中的码字是以非连续的方式存在的, 因此可以通过对每个码字的扩展来构建一张大小为 2^L (L 是编码码表中码字的最大长度)的连续的解码码表[3]。解码的时候只需要一次性的读取 L 位比特, 按其大小到解码码表中查找结果。

由于只需要比较一次就可以得到结果, 所以这种解码方法的速度非常快, 但是它要求有极大的存储空间, 所以只能用于 L 很小的场合。

3 3　Hashemian解码方法

Hashemian解码方法[4]是对二叉树和全码表解码方法的改进。它使用固定的长度 H 来分割解码二叉树, 对长度小于 H 的码字进行扩展。以图 1所示的二叉树码表为例, 当 H = 3时, 获得图 2所示的Hashemian解码码表。

码表 L0

码字	码长 D	结果 R
000	3	2
001	0	0
010	2	1
011	2	1
100	0	1
101	3	3
110	3	4
111	0	2

码表 L1

码字	码长 D	结果 R
000	2	8
001	2	8
010	3	12
011	3	13
100	1	5
101	1	5
110	1	5
111	1	5

码表 L3

码字	码长 D	结果 R
000	0	3
001	3	14
010	2	11
011	2	11
100	1	7
101	1	7
110	1	7
111	1	7

码表 L4

码字	码长 D	结果 R
000	1	15
001	1	15
010	1	15
011	1	15
100	3	17
101	3	18
110	2	16
111	2	16

码表 L2

码字	码长 D	结果 R
00	1	6
01	1	6
10	2	9
11	2	10

中转码表

转移码表码字	读取的长度	码表地址
0	3	L1
1	2	L2
2	3	L3
3	3	L4

图 2　Hashemian解码码表

Fig 2　Hashemian decoding table

解码过程就是读取固定长度的比特, 然后查表的过程。如果 $D>0$则找到结果 R; 如果 $D=0$, 则使用 R 作为转移码表码字到中转码表中查找下一个码表的入口地址。

Hashemian方法通过一次比较 H 个比特来提高解码速度, 同时使用码字分割技术来减少对存储空间的需求。但是 Hashemian方法也有其缺点, 一是对于一个给定大小的存储空间, 它无法尽可能地使用给定的存储空间以提高解码速度; 二是中转码表的使用使得可利用的存储空间进一步减少。

4 基于自动化码表分配的解码方法

平均解码码表的查找次数与解码速度有很大的关系, 解码速度的提高也就意味着平均的解码码表查找次数的减少。下面介绍平均解码码表查找次数的计算方法[5]。

$$\begin{aligned} T &= k_1P_1 + k_2P_2 + k_3P_3 + \cdots + k_1P_{n+1} + k_2P_{n+2} + \\ &\quad k_3P_{n+3} + \cdots + k_1P_{m+1} + k_2P_{m+2} + k_3P_{m+3} + \cdots \\ &= k_1(P_1 + P_{n+1} + P_{m+1} + \cdots) + k_2(P_2 + P_{n+2} + \\ &\quad P_{m+2} + \cdots) + k_3(P_3 + P_{n+3} + P_{m+3} + \cdots) + \cdots \\ &= k_1S_1 + k_2S_2 + k_3S_3 + \cdots \end{aligned} \tag{1}$$

$$N = \sum_{i=0}^{n-1} M_i \tag{2}$$

式中, T 是平均解码码表的查找次数。k_i 表示解码码表 t_i 中的码字需要查找的码表的个数。P_i 是第 i 个码字的概率。S_i 表示在 t_i 中最终可以查到结果的码字的概率和。N 表示所需要的内存空间的总和。M_i 表示每个解码码表所使用的存储空间。

对于一个给定大小的存储空间 A, 为了获得最快的解码速度, 则应在满足 $N \leqslant A$ 的条件下使得 T 最小。

T 的计算过程中需要用到每个码字的概率 P_i, 但是 H. 264标准只是提供了 CAVLC 的编码码表, 并没有提供每个码字的概率, 这样就不能用式(1)来计算 T。CAVLC 编码码表中的码字的分配是有一定的规则的, 短码字分配给使用频率高的符号, 长码字分配给使用频率低的符号。即

$$P(L_{\text{short}}) > P(L_{\text{long}}) \tag{3}$$

并且有 $\sum_{i=1}^{M} P_i = 1$, 其中, M 是码表中码字的最大数目。

根据上面的这个分配原则, 在限定存储空间为 A 的条件下, 使用下面的方法来代替式(1)获得更好的码表分配。

(1) 令 $k_1 = L$(编码码表中码字的最大长度), 获得 $M = 2^{k_1}$, 比较 M 与 A, 如果 $M > A$ 的话就逐渐减小 k_1 使得满足存储空间的要求, 如果 $M < A$ 则分配结束。

(2) 找出第 1层分割前缀码相同, 并且码长最短的一批码字, 获得这批码字的长度 L_1, 令 $K_2 = L_1 - k_1$ ($k_2 \leqslant k_1$), 同样通过减小 k_2 来满足存储空间的要求。

(3) 找出第 2层分割前缀码相同并且码长最短的一批码字, 获得这批码字的长度 L_2, 令 $k_3 = L_2 - k_2$ ($k_3 \leqslant k_2$), 同样通过减小 k_3 来满足存储空间的要求。

(4) 使用第 3步相同的方法获得 k_4、k_5 等, 直到分配完第 2步所获得的码字。

(5) 返回第 2步获得下一批前缀相同的码字, 然后使用上面的方法进行分配, 直到分配完所有的码字。

使用这种方法有以下的优点:

(1) 使用软件的方法来实现, 可以减少大量的手工计算。

(2) 不需要知道每个码字的概率也可以获得最小的平均解码码表查找次数。

(3) 不同层次使用不同长度的分割, 同一层次也使用不同长度的分割, 使得给定的存储空间得到最大的利用。

为了进一步减少对存储空间的消耗, 使用统一解码码表来代替 Hashemian解码码表。统一解码码表的关键就是使用了码表地址偏移技术。

统一解码码表中使用了变量 Y 和 Z。当 $Y>0$ 时, Y 和 Z 的定义与 Hashemian码表中 D 和 R 的定义相同, 当 $Y<0$ 时, Y 表示下一次需要读取的比特个数的负数, Z 表示从解码码表的起始位置往后移动的个数。以图 1为例, 令 $H=3$, 获得如表 1的统一解码码表。

其中坐标值等于偏移量 Z 加上读取值的大小。

使用统一解码码表来解 CAVLC 的过程如下:

(1) 从码流中读取长度为 k_1 的比特, 以其大小 b 作为坐标值到统一解码码表中查找结果。

(2) 如果 $Y>0$, 则 Z 即为所要的结果, 解码完成; 如果 $Y<0$, 则 $-Y$ 为下一次需要读取的比特个数, Z 为地址偏移。

表 1　统一解码码表

Tab. 1　Uniform decoding table

坐标值	Y	Z	坐标值	Y	Z	坐标值	Y	Z
0	3	2	12	1	5	24	1	7
1	- 3	8	13	1	5	25	1	7
2	2	1	14	1	5	26	1	7
3	2	1	15	1	5	27	1	7
4	- 2	16	16	1	6	28	1	15
5	3	3	17	1	6	29	1	15
6	3	4	18	2	9	30	1	15
7	- 3	20	19	2	10	31	1	15
8	2	8	20	- 3	28	32	3	17
9	2	8	21	3	14	33	3	18
10	3	12	22	2	11	34	2	16
11	3	13	23	2	11	35	2	16

(3) 从码流中读取 - Y个比特, 以其大小 b加上 Z作为坐标值来查找结果, 并把获得的结果返回第 2步进行比较。

5　性能分析

下面以 0≤NC < 2时解码总的非零系数个数(TotalCoeff)和拖尾系数的个数(TrailingOnes)为例来比较 Hashemian方法和我们所介绍的改进方法。当限制存储空间 A = 120的时候, Hashemian解码方法使用固定长度 H = 4分割编码码表, 而改进方法使用长度为 5, 4, 4, 3分割编码码表。两者的性能比较如表 2。

表 2　性能比较

Tab. 2　Performance comparing

码字长度	相同长度的个数	Hashemian方法	改进方法
1	1	1C	1C
2	1	1C	1C
3	1	1C	1C
4	0	NULL	NULL
5	1	3C　2S	1C
6	3	3C　2S	2C　1S
7	2	3C　2S	2C　1S
8	4	3C　2S	2C　1S
9	4	5C　4S	2C　1S
10	4	5C　4S	3C　2S
11	4	5C　4S	3C　2S
12	0	NULL	NULL
13	8	7C　6S	3C　2S
14	8	7C　6S	4C　3S
15	9	7C　6S	4C　3S
16	12	7C　6S	4C　3S

表 2中 C表示查表, 1C表示查一次表, S表示地址转移, 1S表示一次地址的偏移。查表的次数和转移的次数越少则解码的速度越快。如果查一次表的时间和一次地址转移的时间相同, 由上表可以知道, 在码字长度小于 4的时候, 两种方法的解码速度是相同的, 但是当码字长度大于 4的时候, 改进方法的解码速度就比 Hashemian方法要快, 特别是当码字长度为 5的时候, 改进方法的速度是 Hashemian方法的 5倍; 其他情况下速度的提升也在 2倍左右。因此, 相对于 Hashemian方法, 使用改进方法可以有效提高解码速度。但同时我们发现随着 A 的提高两者之间的性能差异逐渐减小。

限定存储空间 A = 120。在 JM 模型下, 使用 JVT提供的标准解码方法则解码器在 CAVLC上花费的时间为 16% 左右, 使用 Hashemian方法这个比例下降到 7.5% 左右, 而使用改进方法则只需要 5% 左右。可以看出改进方法具有明显的优势。

6　结　论

在限定的存储空间条件下, 改进方法通过使用更好的码表分配和码表地址偏移技术, 使得解码速度得到极大的提高, 从而为 H.264标准应用于实时环境提供了一种可行的解决方案。

参考文献 (References)

1　Bjontegaard Gisle, Lillevold Karl. Context-adaptive VLC (CVLC) coding of coefficients[R]. Fairfax Virginia: Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 3rd Meeting: 2002.

2　Bi Hou-jie. A Novel Video Coding Standard-H.264/AVC[M]. Beijing: Posts and Telecom Press, 2005: 119~121. [毕厚杰主编. 新一代视频压缩编码标准——H.264/AVC[M]. 北京: 人民邮电出版社, 2005: 119~121.]

3　Iyengar V, Chakrabarty K. An efficient finite-state machine implementation of Huffman decoders[J]. Information Processing Letters, 1997, **64**(6): 271~275.

4　Hashemian R. Memory efficient and high-speed search Huffman coding[J]. IEEE Transactions on Communications, 1995, **43**(1): 2576~2581.

5　Hong D, Eleftheriadis A. Automatic generation of C++/Java code for efficient VLC decoding[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2005, **21**(3): 253~260.