

# 基于 CAVLC 解码算法优化的研究

李 芬, 包晓敏

(浙江理工大学信息学院, 杭州 310018)

**摘 要:** 由于 CAVLC(基于上下文的自适应变长编码)码字长度的不固定,对解码速度的影响很大,成为限制其广泛应用的重要因素之一。针对这一问题,文章研究了 H. 264 标准中 CAVLC 解码算法,在分析了 CAVLC 码表特征后,提出了一种改进的 CAVLC 表格查找算法,该算法将码表通过适当分块来缩小其查表范围,避免对整个码表进行查找,从而提高解码器在解码过程中的解码速度。实验结果表明,所提出的优化算法对于每帧图像解码速度提高了 1.55% 左右,最终到达了一定的解码优化效果。

**关键词:** H. 264 标准; CAVLC; 码表分块; 优化

**中图分类号:** TP391 **文献标志码:** A

## 0 引 言

MPEG (moving picture experts group) 和 VCEG (video coding experts group) 已经联合开发了一种比早期研发的 MPEG 和 H. 263 性能更好的视频压缩编码标准即 H. 264 标准<sup>[1]</sup>。它优异的压缩性能在数字电视广播、视频实时通信、网络视频流媒体传递以及多媒体短信等各个方面发挥重要作用。但其高性能是与其编解码的复杂度的提高为代价的,而解码算法的复杂度直接影响到解码的效率。

在熵编码方面, H. 264 使用了 CABAC 和 CAVLC 两种不同的编码方式。CABAC 熵编码是基于区间划分的算术编码方式,这种方式的效率很高,但算法相对复杂。CAVLC 是一种可变长编码,它根据已编码的句法元素动态调整编码中使用的码表。在编码过程中有些语法元素是组合编码的,为满足实时性的要求,有必要对码表结构进行调整及程序结构进行重新设计,以实现快速高效的搜索。

## 1 CAVLC

### 1.1 CAVLC 基本原理

在 H. 264 的 CAVLC(基于上下文自适应的可

变长编码)中,根据 coeff\_token(句法元素)动态地查找编码表中的码表,因此,具有极高的压缩比<sup>[2]</sup>。

CAVLC 主要用于亮度和色度残差数据的编码。残差经过变换量化后的数据有如下的特点:  $4 \times 4$  块数据经过预测、变换、量化后,非零系数主要集中在低频部分,高频部分系数大部分为零;量化后的数据经过 zig-zag 扫描,DC 系数附近的非零系数值较大,而高频位置上的非零系数值大部分是 +1 和 -1;相邻的  $4 \times 4$  块的非零系数的数目是相关的。CAVLC 充分利用残差经过整数变换、量化后数据的特性进行压缩,进一步减少数据中的冗余信息,为 H. 264 卓越的编码效果奠定了基础。

### 1.2 CAVLC 解码算法描述

CAVLC 解码需要以下 5 个步骤:

a) 解析出非零系数数目 (TotalCoeffs) 和拖尾系数数目 (TrailingOnes), 如果 +/ - 1 的个数大于 3 个, 之后最后 3 个被视为拖尾系数, 其余被视为普通非零系数。根据变量 NC (Number Current, 当前块) 的值从 4 个变长表格中选择要查找的表格, 变长表格存储结构为二维结构, 存储的内容为码字, 二维下标分别是 TotalCoeffs 和 TrailingOnes。NC 的值是基于上下文求得的, 根据当前块左边  $4 \times 4$  块的非

零系数数目(NA)和当前块上面  $4 \times 4$  块的非零系数数目(NB)求得。

b) 解析拖尾系数的符号,由于在编码时,对于每个拖尾系数( $\pm 1$ )只需要指明其符号,其符号用一个比特表示(+用 0 表示,一用 1 表示)编码的顺序是从高频数据开始,按照反向扫描的顺序。因此,解码时也是按照反向扫描顺序,+用 0 表示,一用 1 表示。

c) 解析除拖尾系数之外非零系数的幅值。

d) 解析最后一个非零系数前零的数目(TotalZeros),TotalZeros 的值是通过入口参数 TotalCleffs 查表求得的。

e) 解析每个非零系数前零的个数(RunBefore),它是按照反向扫描的顺序来解码的,即从最高频的非零系数开始。RunBefore 的值是查表求得的,表格的入口参数是 ZeroLeft,其表示当前非零系数之前零的总个数。

## 2 已经被提议的解码方法

CAVLC 广泛应用于各种视频编码标准,例如 MPEG-2、MPEG-4、H. 263 和 H. 264 等,因此有很多解码方法。下面介绍其中的 3 种方法。

### 2.1 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  $2^L$  ( $L$  是编码码表中码字最大的长度)的连续的解码码表<sup>[3]</sup>。解码时只需要一次性地读取  $L$  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,但它要求有极大的存储空间,所以只能用于  $L$  很小的场合。

### 2.2 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,即形成一棵二叉解码树。解码过程实际上就是一个寻找叶节点的过程。为使二叉树结构清晰,图 1 中首先定义了 3 种不同类型的子树 T1、T2 和 T3,其叶子节点“0”代表 CoeffToken 码树的叶子。将树根“G”放在最左侧,整棵树的深度为 15,叶子节点数为 62。从左至右各叶子节点对应的码字序号分别为 0, 1, 2, 3, ..., 61, 相应的二进制码字序列为 1, 01, 001, 000100, 000101, 00011, ..., 0000 0000000 0001。

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小,但是这种解码方法是以一个比特作为比较对象的,所以解码速度很慢。

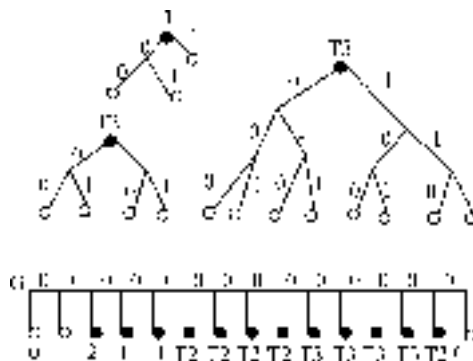


图1 CoeffToken 系数码表二叉树结构

### 2.3 Hashemian 解码方法

Hashemian 解码方法<sup>[4]</sup>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,对于长度小于  $H$  的码字进行扩展。以图 1 所示的二叉树码表为例,当  $H=4$  时,获得图 2 的 Hashemian 解码码表。

0001	0010	0011	0100	0101	0110	0111	1000
0000	0001	0010	0011	0100	0101	0110	0111
0010	0011	0100	0101	0110	0111	1000	1001
0011	0100	0101	0110	0111	1000	1001	1010
0100	0101	0110	0111	1000	1001	1010	1011
0101	0110	0111	1000	1001	1010	1011	1100
0110	0111	1000	1001	1010	1011	1100	1101
0111	1000	1001	1010	1011	1100	1101	1110
1000	1001	1010	1011	1100	1101	1110	1111
1001	1010	1011	1100	1101	1110	1111	1111
1010	1011	1100	1101	1110	1111	1111	1111
1011	1100	1101	1110	1111	1111	1111	1111
1100	1101	1110	1111	1111	1111	1111	1111
1101	1110	1111	1111	1111	1111	1111	1111
1110	1111	1111	1111	1111	1111	1111	1111
1111	1111	1111	1111	1111	1111	1111	1111

图2 Hashemian 解码码表

图 2 的每一项包含 3 个信息位:标志位;十进制表示的码字长度/待查比特数;十六进制表示的码字序号/转移地址。若标志位为 0,则表示该项为一叶节点,第 2 个信息为该码字的长度;第 3 个信息则为所解析数的具体的码字序号;若标志位为 1,则表示该项为下一级子树的根节点,当前仍然没有查到具体码字,需要转到该子树所对应的下一级表中继续查找,第 2 个信息为解下一级所需要的比特数,即下一

次要查找的比特数,第3个信息则为下一级表的入口地址。现举例说明当接收端接收到 0000 0001 11011...码字序列时的解码过程:从左往右第 0~3 比特为 0000,查图 2 的第 1 格,查到的值为  $(1/4)/14$ ,标志位为 1,表示此时没有得到相应的码字,只是得到下一次查表的入口地址 14H;继续检查码字序列的第 4~7 比特 0001,其值为 1H,查图 2 中的  $14H+1H=15H$ ,15H 地址对应的值为  $(1/2)/28$ ,标志位为 1,表示还没有解码成功,继续检查码字序列的第 8~11 比特 1101,从图 2 的 15H 地址开始往后移动 13 格,查到图 2 中相应的值为  $(0/11)/15$ ,标志位为 0,表示解码成功,码长 11 位,码字序号为 15H,即 0000 0001 110。

Hashemian 解码方法通过一次比较  $H$  个比特来提高解码速度,但是 Hashemian 解码方法也有其缺点:对于一个给定大小的存储空间,无法尽可能地使用给定的存储空间以提高解码速度。

### 3 改进的 CAVLC 解码方法

#### 3.1 码表的分割

在已经提议的 3 种解码方法中,每种方法都有所不足。为此,笔者提出了一种改进的 CAVLC 解码方法。在研究过程中,针对其码表中码字前缀的不同,采取把整个码表分成若干个子表的方法,缩小其查找范围来提高查表效率<sup>[5]</sup>。在 H. 264 的码表中,每个码字有确定长度的零前缀,而且成逐步递增的规律,根据这一特点,可以把原码表分成 3 个部分,每一部分的码字具有相同长度的零前缀<sup>[6]</sup>。(具体分法如图 3 所示)

原码表	1	5	7	7	7	7	15	11	8	15	11	15	11	15	11	7	4
	0	1	4	6	6	6	6	14	10	14	10	14	0	1	14	10	6
	0	0	1	5	5	5	5	5	13	9	13	9	13	9	13	9	5
	0	0	0	3	3	4	4	4	4	4	12	12	8	12	8	12	8
子表 1	5	5	7	7	7	7	0	0	0								
	0	1	4	6	6	6	0	0	0								
	0	0	1	5	5	5	5	0	0								
	0	0	3	3	4	4	4	4	4								
子表 2	7	15	11	8	15	11	0	0									
	0	1	4	10	14	10	0	0									
	0	0	5	3	9	13	9	0									
	0	0	0	0	4	12	12	8									
子表 3	15	11	15	11	7	4											
	14	10	1	14	10	6											
	0	13	9	13	9	5											
	0	0	12	8	12	8											

图 3 码表的分割

原表通过特定的分割,变为 3 个子表,在解码过程中,通过一定的判断条件来选择所查码表,可以明显提高查表的效率。

#### 3.2 程序算法流程

程序流程如图 4 所示。

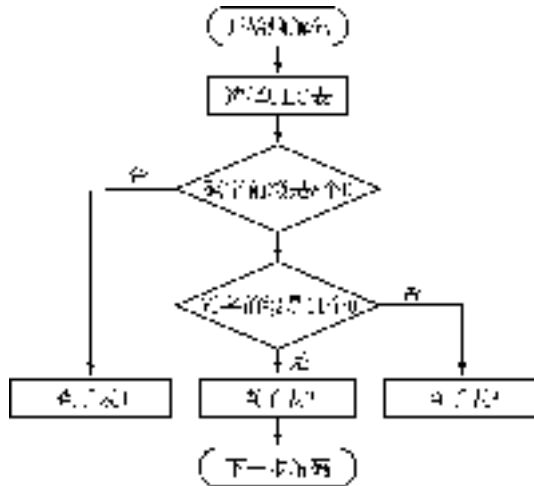


图 4 程序流程

优化后的解码过程:

步骤 1:根据码流的头信息选择码表。

步骤 2:从码流中读取 8 比特数据,对读入的数据进行判断,以选择所选子表号。判断为非 0 时,选择子表 1,跳至步骤 3。否则进行下一步判断,重新读取 9 比特数据,如果读入 1,选择子表 2,否则选择子表 3。

步骤 3:返回语法元素或出错处理。

步骤 4:继续下一步解码处理。

从图 4 可以看出,经过适当分割后的码表其大小已显著减小,理论上相对于原算法效率可以提高 3 倍。同时对 coeff\_token 码表中的子表 2、子表 3 的码字和码长作相应的分割处理来进一步优化。

上面主要是从程序结构上优化解码程序,程序细节方面的优化对解码速度的提高也非常重要,比如:通过选择静态变量来提高编译目标代码的质量,尽量减少对内存的访问和修改。

#### 3.3 实验结果

通过对手机、汽车电视序列进行测试(测试长度为 10 帧,实验平台为 inter 23 GHz, 2G 内存, Windows7 系统),相比标准 H. 264 解码算法, H. 264 解码器经过 CAVLC 码表分块解码优化后,程序执行速度大为提升。测试结果如下表 1 所示。

表 1 原算法与优化算法解码时间对比表

	手机	汽车电视
原算法/s	66.065	65.875
优化算法/s	56.898	57.364

从表1测试结果可以看出,对于每帧图像解码速度提高了1.55%左右。但是,解码速度的提升是以增加内存空间和增加代码复杂度为代价的。如果进一步划分码表,由于增加了条件的判断,性能提升不明显。

## 4 结 语

H. 264 是现在视频编解码领域研究的热点也是未来的发展方向,它将代替 MPEG-2 成为主流的信源压缩标准,将 H. 264 的编解码速度尽可能地提高,降低成本,就可以在更多的领域中应用。为此,本文对 H. 264 中 CAVLC 解码算法提出了一种改进算法,利用其码字前缀的不同划分码表,缩小了查表的范围,避免了对整个码表的遍历,也减少了对长码字解码的时间,在解码效率上得到了很大的提高。经测试,改进后的算法比原算法在每帧解码速度上提高了1.55%左右,具有明显的实用意义。

## 参考文献:

- [1] 张 杰. 视频编解码标准 H. 264/AVC 中重要技术[J]. 现代电子技术, 2004, 27(6): 101-103.
- [2] 毕厚杰. 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.
- [3] Iyengar V, Chakrabarty K. An efficient finite-state machine implementation of Huffman and decoders[J]. Information Processing Letters, 1997, 64(6): 271-275.
- [4] Hashemian R. Design and hardware implementation of a memory efficient Huffman and decoding [J]. IEEE Transactions Consumer Electronics, 1994, 40(3): 345-352.
- [5] 宋宝良, 戴学丰. H. 264 解码器中 CAVLC 码表查找算法的分析与优化[J]. 现代电子技术, 2009, 16(3): 155-157.
- [6] 薛 全, 张 颖, 刘济林. 基于变步长分组的 H. 264 系数码表优化[J]. 电路与系统学报, 2006, 11(3): 115-117.

# Research on CAVLC Decoding Algorithm Optimization

LI Fen, BAO Xiao-min

(School of Information Science and Technology, Zhejiang Sci-Tech University,  
Hangzhou 310018, China)

**Abstract:** Unfixed code word length of CAVLC(context-based adaptive variable length coding) having big influence on the decoding speed becomes one of the important factors restricting its wide application. In allusion to this problem, this paper studies CAVLC decoding algorithm in H. 264 standard and puts forward an improved CAVLC table lookup algorithm after analyzing CAVLC code table features. This algorithm reduces the table lookup scope through appropriate partitioning of code table, thus avoiding searching the whole code table and improving the decoding speed of decoder in the decoding process. The experimental result shows that the optimization algorithm put forward improves the decoding speed of each frame of image by about 1.55% and finally achieves a certain decoding optimization effect.

**Key words:** H. 264 standard; CAVLC; code table block; optimization

(责任编辑: 陈和榜)