博士学位论文

视频图像压缩中熵编码技术研究

RESEARCH ON ENTROPY CODING FOR VIDEO AND IMAGE COMPRESSION

高敏

哈尔滨工业大学

2016 年 6 月

国内图书分类号：TP391.41　　学校代码：10213
国际图书分类号：681.39　　密级：公开

博士学位论文

视频图像压缩中熵编码技术研究

博士研究生：高敏
导　　师：高文教授
申请学位：工学博士
学科、专业：计算机应用技术
所在单位：计算机科学与技术学院
答辩日期：2016年6月
授予学位单位：哈尔滨工业大学

Classified Index: TP391.41
U. D. C.: 681.39

Dissertation for the Doctoral Degree in Engineering

RESEARCH ON ENTROPY CODING FOR VIDEO AND IMAGE COMPRESSION

Candidate : Gao Min
Supervisor : Prof. Gao Wen
Academic Degree Applied for : Doctor of Engineering
Specialty : Computer Application Technology
Affiliation : School of Computer Science and Technology
Date of Defence : Jun. 2016
Degree-Conferring-Institution : Harbin Institute of Technology

摘 要

在视频压缩标准中，熵编码模块在信源符号的压缩和码流的组织过程有着不可替代的作用。熵编码模块首先利用上下文建模技术来挖掘信源符号之间的统计冗余；然后使用熵编码引擎（算术编码技术或者变长编码技术）来去除统计冗余，进而产生紧凑的二进制码流，以供存储和传输。熵编码模块通常利用已经编码的符号来估计待编码符号的概率分布，从而能够高效地去除信源符号之间的冗余信息。然而，这种编码依赖关系却不利于熵编码模块的并行处理，降低了吞吐率。因此，如何平衡熵编码模块的编码效率和数据吞吐率已经成为衡量熵编码技术优劣的重要性能指标。随着高画质视频的逐步普及，未来的视频压缩标准将要处理数据量更加庞大的视频，因此继续提高熵编码模块的压缩效率仍然具有很重要的意义。

为了克服外界条件的限制，在目前最新的多媒体应用中，压缩感知技术正在被应用于采集数字图像/视频。当采用压缩感知技术对图像/视频进行采集时，得到的测量值（通常用测量值表示利用压缩感知采样得到的样本信息）和传统的图像/视频采集方法得到的像素值在本质上是不同的。测量值通常是利用随机高斯矩阵对原始信号进行投影操作而得到的，每个测量值都包含了原始信号的全局信息，并且各个测量值之间是相互独立的。所以，传统的图像/视频压缩标准技术不再适用于测量值的压缩。如何充分挖掘测量值的统计特性为其设计一个高效的熵编码器是测量值压缩中的一个新的挑战问题。

所以，在这种研究背景下，本文对 H.264/AVC，HEVC 和 AVS2 等视频压缩标准中的熵编码模块进行优化，来提高熵编码模块的压缩效率和数据吞吐率；另外，本文也为压缩感知中测量值的压缩设计了一个高效的熵编码器，来把测量值转化为紧凑的二进制码流，实现真正的数据压缩。因此，本文的研究工作主要包括以下四个部分：

第一，在 H.264/AVC 的上下文自适应二进制算术编码器（CABAC）的设计过程中，由于没有充分地考虑吞吐率这一指标，使得熵编码模块已经成为整个解码器的主要瓶颈之一。为了提高 H.264/AVC 中 CABAC 的吞吐率，本文首先根据预测残差的 DCT 系数的统计特性，提出了一个层次依赖上下文模型 HDCM (Hierarchical Dependency Context Model)。在 HDCM 中，DCT 系数块中的非零系数的个数和 DCT 系数的频域位置被用作为上下文，来挖掘 DCT 系数之间的统计冗余。然后，本文提出了一个基于层次依赖上下文模型

的二进制算术编码器 HDCMBAC 来编码 H.264/AVC 中的预测残差的 DCT 系数。为了高效地描述 DCT 系数块，并且降低语法元素之间的上下文依赖关系，HDCMBAC 重新设计了用于描述 DCT 系数块的语法元素。这些语法元素包括，DCT 系数块中非零系数的个数 *N*，用于指示每个位置上的 DCT 系数是否为非零系数的语法元素 *significant_flag* 和用于指示每个非零 DCT 系数的幅值的语法元素 *coeff_abs_level_minus1*。实验结果表明，与 H.264/AVC 中的 CABAC 相比，HDCMBAC 可以取得相似的编码效率，并且最大限度地降低了 DCT 系数之间的上下文依赖关系。

第二，为了继续提高视频编码标准中熵编码模块的编码效率和为下一代视频编码标准的制定做技术储备，本文以 HEVC 为基础，提出了一个内存消耗小并且编码效率高的熵编码方案，该方案包括变换系数的增强上下文建模方法和低内存消耗的二进制算术编码引擎。在变换系数的增强上下文建模方法中，本文采用当前变换系数的局部模板内非零变换系数的个数和当前变换系数的位置信息作为 *significant_coeff_flag*（用于指示当前的变换系数是否为非零系数的语法元素）的上下文；为了减少 *significant_coeff_flag* 的上下文模型的个数，变换系数块被分割为不同的区域，并且相同区域使用相同的上下文模型集合。在编码 *coeff_abs_greater1_flag*（用于指示当前的非零变换系数的绝对值是否大于 1 的语法元素）时，本文采用当前变换系数的局部模板内绝对值等于 1 和绝对值大于 1 的变换系数的个数作为其上下文；为了利用亮度分量中变换系数与其位置之间的相关性，变换系数的位置信息也被用作 *coeff_abs_greater1_flag* 的上下文。在编码语法元素 *coeff_abs_greater2_flag*（用于指示当前的非零变换系数的绝对值是否大于 2 的语法元素）时，本文采用当前变换系数的局部模板内绝对值大于 2 和绝对值大于 1 的变换系数的个数作为该语法元素的上下文。在低内存消耗的二进制算术编码引擎中，本文采用多参数的概率估计模型估计二进制符号的概率；在编码区间的分割过程中，本文提出了一个低位宽的乘法操作来代替传统的查表操作。如此设计之后，低内存消耗的二进制算术编码引擎在概率估计过程和编码区间分割过程中均不再需要大量的存储空间。实验结果表明，与 HEVC 中原始的熵编码方案相比，本文提出的熵编码方案具有更高的编码效率。

第三，在第二代中国国家视频压缩标准 AVS2 中，熵编码模块中存在着很强的顺序依赖关系，这些顺序依赖关系严重地制约着 AVS2 编解码器的吞吐率。这些顺序依赖关系主要来源于算术编码引擎的归一化过程和 bypass bin（概率等于 0.5 的二进制符号）的编码过程以及变换系数的上下文建模过

程。因此，本文从上述三个方面对 AVS2 的熵编码模块进行优化设计。具体来讲，本文首先提出了一种快速的，与标准兼容的算术编码引擎归一化方法。该方法简化了算术编码引擎的执行流程，减少调用归一化过程的次数。其次，本文提出了一个快速的 bypass bin 的编解码过程，使得 bypass bin 的编解码过程仅仅需要移位和加法操作即可完成，极大地降低了 bypass bin 的编解码复杂度。最后，本文改进了 AVS2 中变换系数的编码过程，降低变换系数之间的上下文依赖关系。实验结果表明，上述三个技术大幅度地提高 AVS2 中熵编码模块的吞吐率，同时性能损失也比较小。

第四，在图像的压缩感知采样中，为了提高测量值的压缩性能，差分脉冲预测（DPCM：Differential Pulse-Code Modulation）和均匀标量量化（SQ：uniform Scalar Quantization）被联合应用于测量值的压缩中。尽管如此，若想真正地实现测量值的压缩，即把测量值转化为紧凑的二进制码流，熵编码模块是一个必不可少的模块。为此，本文基于图像的差分脉冲预测和标量量化框架（DPCM-plus-SQ），为测量值的量化索引提出了一个高效的熵编码方案。在该熵编码方案中，本文分析了测量值的量化索引的统计特性，并且根据这些统计特性设计了相应地语法元素来描述测量值的量化索引。具体来说，本文首先使用语法元素 *significant_map* 来指示当前测量值的量化索引是否为非零；然后，对于非零的量化索引，使用语法元素 *abs_coeff_level_minus1* 和 *sign_flag* 来分别指示它的幅值和符号。为了挖掘这些语法元素的局部统计特性，本文采用自适应的算术编码引擎来编码这些语法元素，以期望去除它们的统计冗余从而产生紧凑的码流。实验结果表明，与测量值量化索引的 0 阶信息熵和 H.264/AVC 中 CABAC 的变换系数编码方法相比，本文提出的熵编码方案能够进一步提高测量值的编码效率。

关键词：视频编码；压缩感知；熵编码；上下文建模；算术编码；吞吐率

Abstract

As one of the most important modules in video/image codec, the entropy coding plays an indispensable role in compressing the sources symbols and organizing the bit-stream. The entropy coding first utilizes the context modeling scheme to exploit the statistical redundancy, and then utilizes the entropy coding engines i.e. variable length coding or arithmetic coding to remove the statistical redundancy and generate the compact bit-stream. The entropy coding usually uses the previously coded symbols to estimate the probability of the current symbol, thus they can efficiently remove the statistical redundancies. However, the coding dependencies make it difficult to improve the throughput of video codec. Therefore, how to banlance the throughput and coding efficiency has been the hot topic in video coding fields. With the explosion of the high quality video, the future video coding standards have to process the video with massive data, so it is still the main target to improve the coding efficiency when establishing the future video coding standards.

To overcome the restrictions of external conditions, the compressive sensing (CS) has been applied in the image/video acquisition in the emerging multimedia applications. The obtained measurements are different from the pixels in nature when applying CS into the acquisition of video/image. More specificially, the measurements are obtained via a linear projection into a lower-dimensional subspace chosen at random. Thus each measurement contains the information from all pixels in one image and one measurement is independent of others. So it is unsuitable to apply the conventional video/image coding tools to compress the measurements. Therefore, it is desirable to develop coding tools for compressing the CS measurements.

So, in this context described above, this dissertation focuses on entropy coding modules in the video coding standards including H.264/AVC, HEVC and AVS2 as well as the image compression based on compressive sensing.The main contributions are composed of the following four works:

First, since the throughput is not fully taken into account in CABAC within H.264/AVC, CABAC has been one of the most time-consuming modules in the

decoder. To improve the throughput of CABAC, a hierarchical dependency context model (HDCM) is firstly proposed to exploit the statistical correlations of DCT coefficients, in which the number of significant coefficients in a transform block and the scanned position are used to capture the magnitude varying tendency of transform coefficients. Then a new binary arithmetic coding using HDCM (HDCMBAC) is proposed. HDCMBAC associates HDCM with binary arithmetic coding to code the syntax elements for a transform block, which consist of the number of significant coefficients, significant flag and level information. Experimental results demonstrate that HDCMBAC can achieve similar coding performance as CABAC at low and high QPs. Meanwhile, the context dependency in the context modeling scheme can be reduced as much as possible.

Second, to further improve the coding efficiency of entropy coding module and prepare to establish the next generation video coding standard succeeding HEVC, an enhanced entropy coding scheme is proposed based on HEVC, which includes an improved context modeling scheme for transform coefficient levels and an binary arithmetic coding (BAC) engine with low memory requirement. In the improved context modeling scheme for transform coefficient levels, the contexts of *significant_coeff_flag* consist of the number of the significant transform coefficient levels in a local template and its position. To limit the total number of context models, TBs (TB: transform block) are split into different regions based on the coefficient positions. The same region in different TBs shares the same context model set. The context model index of *coeff_abs_greater1_flag* is determined by the number of transform coefficient levels in a local template with absolute magnitude equal to 1 and larger than 1. Moreover, TBs are also split into different regions to incorporate the coefficient position in luma in its context model selection. The context model index for *coeff_abs_greater2_flag* is determined by the number of transform coefficient levels in a local template with absolute magnitude larger than 1 and larger than 2. In the BAC engine with low memory requirement, the probability is estimated based on a multi-parameter probability update mechanism. Moreover, the multiplication with low bit capacities is used in the coding interval subdivision to substitute the large look-up table to reduce its memory consumption.

Experimental results demonstrate that the proposed two techniques can significantly improve the coding efficiency of the entropy coding module.

Third, there exist strong sequential dependencies in the entropy coding scheme within AVS2, which severely limit the throughput improvement of AVS2. These sequential dependencies mainly stem from normalization process and the bypass bins coding process in the arithmetic coding engine as well as the transform coefficients coding process. So, a fast and standard-complicant normalization is first proposed, which can simplify the arithmetic coding engine. Then, a fast coding process for bypass bins is proposed, in which only addition and shift operations are required for coding the bypass bins. Finally, the context modeling scheme for transform coefficients is modified to reduce the coding dependencies as much as possible. Experimental results demonstrate that the above three techniques can significantly improve the throughput of the entropy coding scheme in AVS2 by keeping the similar coding performance.

Fourth, Differential pulse-code modulation (DPCM) is recently coupled with uniform scalar quantization (SQ) to improve the rate-distortion performance for the block-based quantized compressive sensing of images. However, the entropy coding is still required to convert the quantization index of CS measurements into bitstream. Therefore, an arithmetic coding scheme is proposed for the quantization index within DPCM-plus-SQ framework by analyzing their statistics. In the proposed arithmetic coding scheme, the quantization index of the CS measurements is first represented by three syntax elements *significant_map*, *abs_coeff_level_minus1* and *sign_flag*, which denote the significance (i.e. non-zero), magnitude and sign flag of a quantization index, respectively. These syntax elements are then coded by an adaptive binary arithmetic coding engine M coder to remove the statistical redundancies and generate the bitstream, since M coder can capture the local statistics of these syntax elements. Compared with the zero order entropy of the quantization index and transform coefficient coding in CABAC within H.264/AVC, the proposed arithmetic coding scheme can further improve the coding efficiency.

Keywords: Video coding; compressive sensing; entropy coding; context modeling; arithmetic coding; throughput.

目 录

Contents

图表索引

List of Figures and Tables

第1章 绪　论

1.1 课题背景

由于图像和视频信号能够完整地，真实地为人类再现某一场景中的影像，所以人们可以直观地，高效地通过图像和视频信号来获取信息。在人类的日常生活中，图像/视频信号已经成为人类交流通信过程中不可缺少的信息载体。近些年来，随着信息技术的发展，高清和超高清的视频逐渐成为人类日常生活中的主流视频。另外，社交网络的全面兴起以及移动设备的普及也使得数字视频正在以前所未有的速度迅速增加。根据思科 2014-2019 年全球移动互联网发展趋势报告，2014 年全球网络数据达到 30 艾字节（Exabytes, 即 300 亿 GB），其中视频数据占据 55%。未来几年内，由于移动设备，可穿戴设备，在线视频以及物联网大爆发，到 2019 年，网络数据可能达到 300 艾字节，其中视频数据占据 72%。存储和传输数据量如此巨大的视频信息对视频编码标准提出了更高的要求，也使得视频编码标准持续成为国内外工业界和学术界的研究热点之一[1]。

由于数字视频信号的数据量十分庞大，如果不通过压缩处理则无法实施有效的存储和传输。例如，一路帧率为 30f/s 时间长度仅为一分钟的 1080p 的高清数字视频将需要 41.7GB 的存储空间；如果需要实时传输该视频，那么网络带宽需达到 593Mbps。显然，在实际应用中，如此巨大的存储空间和带宽要求是无法接受的。所以，为了有效的存储和传输数字视频，追求更高的压缩效率一直都是视频编码标准的主要目标。尽管如此，在诸如视频会议等需要实时解码的视频应用中，视频码流必须实时解码以保证视频的实时播放，这给编解码器的数据吞吐能力提出了更高的要求。 表 1-1 给出了采用 H.264/AVC 视频编码标准在不同的层级（level）下实时解码一帧图像需要处理二进制符号的速率峰值（peak bin rate）[2]，其中 Level 5.1 中的 peak bin rate 的单位是 Gbins/sec。假设每一帧图像需要 m 个二进制符号（m bins per frame）来表示，并且要求每秒解码 f 帧（即帧率为 f 帧每秒），则该二进制符号的速率为 $f \times m$。在硬件实现过程中，假设不采用并发技术，并且每个时钟周期解码 1 个二进制符号（1 bin per cycle），则编码器的工作频率需要达到 GHZ 量级。工作频率如此之高的编码器在硬件上是很难实现的，即使可以实现，这种编码器的功耗也将会很高。提高视频编码器的数据吞吐率可以在不显著

增加功耗的条件下，提高视频编码器的处理速度。因此，在制定视频压缩标准时，编码工具的数据吞吐率已经引起了人们的广泛关注。如何平衡视频压缩技术的编码效率和数据吞吐率已经成为视频编码标准制定过程中的研究热点问题。

表 1-1 在不同的层级（level）下实时解码一帧图像所需要的二进制符号的速率的峰值

Table 1-1 Peak bin rate requirements for real-time decoding of worst case frame at various high definition levels.

Level	Max frame rate	Max bins per frame	Max bit rate	Peak bin-rate
	fps	Mbins	Mbits/sec	Mbins/sec
3.1	30	4.0	17.5	121
3.2	60	4.0	25	242
4.0	30	9.2	25	275
4.1	30	17.6	50	527
4.2	60	17.6	50	1116
5.0	72	17.6	169	1261
5.1	26.7	17.6	300	2107

在传统的图像/视频压缩中，人们首先通过采集设备采集得到图像/视频数据（通常是$W \times H$的像素阵列，每个像素用 8 比特来表示），然后从数据本身的特性出发，挖掘和去除数据之间存在的冗余信息，最后把剩余的信息转化为二进制码流，以供存储和传输。这种传统的图像/视频压缩方式有两个特点：第一，只有图像/视频被完整的采集之后，图像/视频的压缩才能开始执行；第二，编码端需要具有强大的计算能力来完成图像/视频信号的压缩，而解码过程相比而言是比较简单的，并不需要很高的计算能力。因此，这种传统的图像/视频压缩方式适用于诸如电视广播等只需压缩一次，多次解码的应用场合。然而，在某些场合中，采集设备往往是廉价的，或者出于其他的考虑，采集设备无法完整地采集图像/视频信号；但是信息处理端却具备强大的计算能力，可以执行一些复杂的算法来完整地恢复图像/视频信号。例如，在侦察机的军事作业中，为了避免长时间地暴露在敌军雷达侦测范围内，侦察机需要在尽可能短的时间内搜集尽量多的信息；在医学领域中，为了避免病人长时间地暴露在辐射中，核磁共振成像仪被要求在尽可能短的时间生成清晰的诊断图像。在这种应用场合下，传统的图像和视频采集方式（即需要

完整地采集图像和视频信息）已经无法胜任。

在这种背景下，学术界提出了压缩感知（Compressed Sensing）理论[3]，即利用随机高斯矩阵把信号投影至一个维度更低的空间，也就是说仅仅采集信号的部分信息；如果信号在某些域中是稀疏的，那么接收端或解码端就可以利用这个低维度空间中的测量值来精确地恢复出原始信号。由于图像/视频信号在某些域（如变换域或者梯度域）中是稀疏的，因此，可以对图像/视频信号采用压缩感知采样，并且利用采样得到的测量值来精确地恢复图像/视频信号。尽管压缩感知采样可以把原始信号映射至维度更低的空间，即在采样的时候，对原始信号做了压缩操作。但是，从信息论[4,5]的角度来说，这种压缩并不是真正意义上的数据压缩，因为数据压缩通常是指信源编码[6]，即把任意类型的信号转化为紧凑的二进制码流。为了把测量值转化为紧凑的二进制码流，设计可以充分挖掘测量值统计特性的压缩工具已经成为测量值压缩的研究热点问题。

本章首先介绍视频压缩的关键技术和视频压缩标准的发展历程；其次介绍图像压缩感知采样的基本原理及其重构技术；然后对数据压缩中熵编码技术进行系统的介绍；最后给出本文的贡献以及整片论文的组织结构。

1.2 数字视频压缩

数字视频压缩之所以能够实现的根本原因是数字视频中存在着大量的冗余信息。这些冗余信息主要包括，空域冗余，时域冗余，视觉冗余和统计冗余。具体来说，空域冗余是指存在于静态图像中的冗余信息，例如图像中的前景物体或者背景区域的像素值是按照一定的规则分布于整幅图像之中，这些规则就是数字图像中的空域冗余；时域冗余是指视频序列中的相邻帧在亮度和色度空间中存在的相关性，例如同一视频序列中的相邻帧可能具有相同的背景区域和运动物体，唯一的区别可能是在不同帧中运动物体处在不同的空间位置上；视觉冗余是指人类视觉系统无法感知的信息，例如人类的视觉系统无法感知图像结构中的某些细微变化，即便是删除这些信息，人眼也不会察觉；统计冗余是指信源符号在信息表达上的表示冗余，例如某些信源符号出现的概率比较高，而有些信源符号出现的概率很低，那么可以根据出现概率的大小来为信源符号分配码字，从而降低信源的平均码字长度。

1.2.1 视频压缩关键技术

为了最大限度地去除数字视频中的这些冗余信息，学术界和工业界提出

了基于预测和变换的混合编码框架。图 1-1 描绘了视频压缩中混合编码框架示意图。输入的视频帧首先被划分成大小为 $N\times N$ 的编码单元，每个编码单元按照图 1-1 列出的过程依次进行处理。即首先利用帧内预测或者帧间预测技术来去除编码单元中的空域冗余或时域冗余，同时产生预测残差信号；然后预测残差信号经过变换和量化得到变换系数；之后对变换系数进行重组，最后送至熵编码模块去除其中的统计冗余并生成码流。由此可见，预测，变换，量化和熵编码等模块是组成混合编码框架的基本要素。下面本文将分别对这些编码模块进行简要的介绍。

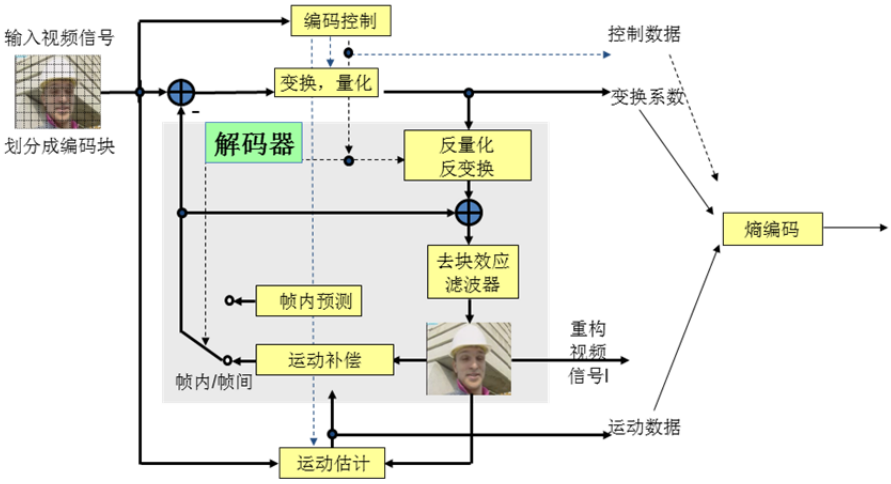


图 1-1 数字视频压缩的混合编码框架

Fig 1-1 The hybride video coding framework

预测技术包括帧内预测技术和帧间预测技术。帧内预测技术的目标是挖掘相邻像素和当前像素之间的相关性来去除视频帧中的空域冗余。它通过利用相邻像素的重构值来对当前像素进行预测，只记录当前像素值与其预测值之间的残差，从而去除静态图像中存在的空间冗余信息。最初的帧内预测技术是 DPCM[7]（Differential Pulse Code Modulation）技术，后来帧内预测技术发展为基于方向的帧内预测技术[8]。帧间预测技术通过利用相邻帧之间的相关性，根据前向或后向的重构帧来对当前帧进行预测，只记录当前帧与其预测值之间的残差，从而实现去除时域冗余的目的。最初的帧间预测技术也是基于 DPCM 技术来实现的，后来运动补偿预测[9]被用于视频的帧间预测模块。此后，运动补偿预测技术也在不断地发展着。例如，在预测方向的选择上，运动补偿技术从早期的单向预测发展为目前的多参考帧预测，加权预测；在

块划分方面，运动补偿技术从仅仅支持单一的块大小演变为目前的可变块大小的运动补偿预测；另外运动矢量的精度也从早期的整像素精度发展为目前的四分之一像素精度和八分之一像素精度。

变换技术是为了去除视频序列中的视觉冗余。预测残差经过变换以后，其中的大部分能量集中在少量的低频系数中。由于人眼对高频和低频系数的敏感程度不同，使得可以在编码的过程中引入适量的失真，也不会被人眼察觉出来。即根据变换系数的重要性，对其进行量化以实现数据的压缩。这里需要注意的是，变换技术并没有取得数据的压缩，它仅仅是把预测残差从空域转化到频域实现了去相关和能量集中，以方便后续的量化过程的处理。真正实现数据压缩的是量化模块。经过多年的发展，离散余弦变换[10]（DCT: Discrete Cosine Transform）因其较低的计算复杂度和较高的去相关能力以及较好的能量集中性能被广泛地应用在视频压缩中。后来，为了进一步降低 DCT 的计算复杂度和避免编解码端出现误差漂移现象，W. K. Cham 提出了整数 DCT[11]（ICT：Integer DCT）的概念。考虑到预测残差的多样性，率失真性能最优的变换技术[12]，模式依赖的方向变换[13]（MDDT：Mode Dependent Directional Transform）和离散正弦变换（DST：Discrete Since Transform）对某些特定类型的残差可以取得更好的去相关能力和能量集中性能。

量化技术是真正地实现了去除视觉冗余的编码技术，也是平衡码率和图像失真的主要方法。矢量量化和标量量化是信号处理领域经常使用的两种量化方式。由于矢量量化依赖于输入信号的统计特性，并且具有较高的计算复杂度，因此现有的视频压缩标准，诸如MPEG2，H.264/AVC，AVS和HEVC，并没有采用矢量量化。现有视频压缩标准通常采用包含死区（dead-zone）的均匀量化方法[14]。为了有效地利用人眼感知系统的特性，可以为处于不同频带位置的变换系数分配不同的量化权重，来提高压缩效率。MPEG2引入了两种加权量化矩阵[15]来分别处理帧内预测残差和帧间预测残差，并且每个加权量化矩阵将会为不同频域的变换系数分配不同的量化因子。H.264/AVC也利用人眼特性设计了一个加权量化缩放矩阵[16]，同时还允许用户根据自身需要来灵活地定义缩放矩阵。基于这种思想，有研究员提出了自适应量化矩阵选择技术[17]（AQMS: Adaptive Quantization Matrix Seletion），即根据变换系数的自身特性来自适应地选择量化矩阵。

熵编码技术是为了去除变换系数，运动矢量和控制信息等信源符号中存在的统计冗余，如图 1-1 所示。目前广泛使用的熵编码方案包括两类，一类

是变长编码[18~23]（VLC：Variable Length Coding），另一类是算术编码[24~30]（AC：Arithmetic Coding）。变长编码由于其较低的计算复杂度被普遍应用于早期的视频压缩标准。然而变长编码的压缩效率要低于算术编码。随着算术编码技术的计算复杂度越来越低，算术编码逐渐成为新兴视频编码标准中的主流熵编码技术。无论是变长编码还是算术编码，它们的基本思想都是根据信源符号的概率分布，为信源符号分配不同长度的码字。所以，用于估计信源符号概率分布的上下文建模技术在熵编码技术中有着不可忽视的作用。一个优秀的上下文建模过程可以充分挖掘信源符号的统计特性，因而可以使得熵编码取得更好的压缩性能。在 DCT 系数的变长编码和算术编码过程中，文献[31~41]利用 DCT 系数的统计特性来指导 DCT 系数的上下文模型，从而提高了 DCT 系数的编码效率。在视频压缩标准中，变长编码的典型技术是 H.264/AVC 中的基于上下文的变长编码技术[35]（CAVLC：Context-based Variable Length Coding），它采用 DCT 系数的统计特性来指导 DCT 系数的变长编码，提高了 DCT 系数的编码效率；算术编码的典型技术是 H.264/AVC 中的上下文自适应的二进制算术编码[40]（CABAC：Context-based Adaptive Binary Arithmetic Coding），它采用二进制算术编码引擎，并结合各个语法元素的统计特性极大地提高熵编码模块的编码性能。

1.2.2 数字视频压缩标准发展历程

在经济全球化的背景下，为了使得全球不同国家和地区的观众能够欣赏来自不同国家的影像，视频编码技术的标准化是多媒体应用中必不可少的的一部分。为此，国际标准化组织（ISO）及国际电工委员会（IEC）下的运动图像专家组（MPEG），国际电子联盟电信标准化局（ITU-T）下的视频编码专家组（VCEG）和中国数字音视频编解码标准工作组（AVS）都在致力于视频编码标准的制定工作。经过几十年的技术积累和发展，视频压缩标准也经过了几代的演变，其压缩性能得到了显著的提高。

（1）早期的视频压缩标准

早期的视频压缩标准包括 MPEG 组织制定的 MPEG-1[42]，MPEG-2[43]和 MPEG-4[44]和 ITU-T 制定的 H.261[45], H.263[46]等标准。其中 MPEG-1 制定于 1992 年，它的应用范围主要包括视频会议、影像电话及第一代的 CD-ROM。MPEG-2 制定于 1994 年，主要面向于数字多功能光盘（DVD）和数字视频广播等应用。MPEG-4 制定于 1998 年，该标准主要面向于低码率应用，诸如视频会议、影音邮件等，并且还支持影像物体的传输。ITU-T 制定的 H.261 标

准主要是为了在 ISDN 上实现可视电话、视频会议等应用。在 H.261 的基础上，ITU-T 制定了面向低码率的可视电话、视频会议等应用的 H.263 标准。早期的视频压缩标准在当时获得了广泛地应用，但是随着时间的发展，这些标准逐渐不能满足人们的要求了。因此，ITU-T VCEG 与 ISO/IEC MPEG 在 2001 年成立了联合视频组（JVT），联合制定了视频编码标准 H.264/AVC[47]。

（2）H.264/AVC 视频压缩标准

与早期的视频压缩标准相比，H.264/AVC 的压缩效率取得了极大的提高。实验结果表明，与 MPEG-4 相比，H.264/AVC 在相同的视频质量下可以节省 40%的码率。而且为了满足诸如监控视频，实时传输，有线广播等不同应用领域的需求，H.264/AVC 包括多个应用档次，例如基准档次（Baseline Profile）主要用于可视电话、会议电视、无线通信等实时视频通信；主要档次(Main Profile) 主要用于数字广播电视与数字视频存储；扩展档次(Extended Profile) 支持码流之间有效的切换（SP 和 SI 片）、 改进误码性能；高层档次(High Profile) 在主要档次的基础上增加了 8x8 内部预测、自定义量化、 无损视频编码和更多的 YUV 格式，多应用于广播电视和数字视频存储领域。

H.264/AVC 压缩性能的提高主要源于它采用了大量的创新技术[48]。例如，H.264/AVC 采用了基于方向的帧内预测技术[48]极大地提高了帧内编码的编码效率；在帧间预测方面，H.264/AVC 采用了更加高效的运动补偿技术，其中包括可变块大小的块分割技术，多参考帧技术[49]、长期参考帧[50]以及 1/4 精度的亮度运动补偿等技术；同时，H.264/AVC 在 B 帧中引入 direct 模式[51]，以适应于 B 帧的特性。除此之外，H.264/AVC 引入了自适应块大小的变换[52]和基于上下文的变长编码 CAVLC[35]以及上下文自适应的二进制算术编码 CABAC[40]等技术对预测残差进行高效地编码。为了提高重构图像的质量，H.264/AVC 在后处理阶段引入了自适应环路滤波器[53]。

（3）AVS-P2 视频压缩标准

由于我国对视频编码技术的研究工作起步较晚，使得我国没有及时地参与早期的国际视频编码标准的制定工作，导致我国在国际视频编码标准中缺乏自主知识产权。为了改变这种尴尬的境地和支持民族信息产业的发展，原国家信息产业部于 2002 年批准成立了 AVS 工作组，负责制定完全具备自主知识产权的音视频编码标准。经过多年的努力，AVS1.0 第二部分视频编码标准（AVS-P2）于 2006 年正式成为国家标准并开始实施。为了满足不同应用领域的要求，AVS-P2 也包含多个档次，如基准档次(Jizhun Profile) [54]、加强

档次(Jiaqiang Profile) [55]和增强档次(Zengqiang Profile)。其中，基准档次主要是面向于高清数字电视广播、HD-DVD 存储应用；加强档次的主要应用是定位于数字消费电影等多媒体应用；增强档次提供了一个编码性能更高的编码工具集合。

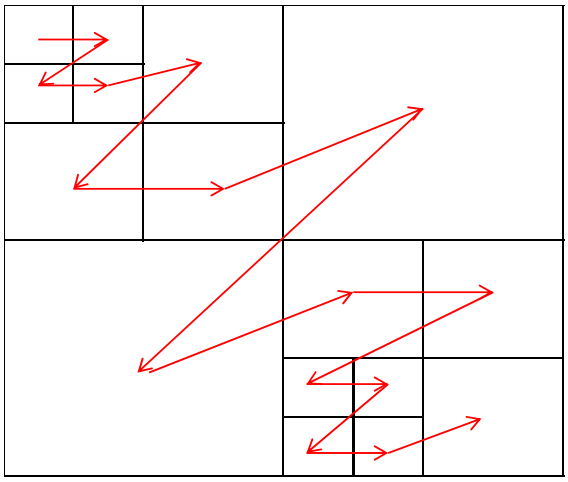
为了在复杂度和编码效率之间取得更好的平衡，AVS-P2[56]采用了与 H.264/AVC 不同的编码技术。首先，在帧内编码上，AVS-P2 采用 8x8 和 4x4 大小的帧内预测块，同时使用更少的预测方向来降低帧内预测的计算复杂度，而且相邻像素的选择和插值滤波器的设计也与 H.264/AVC 不同；在帧间预测上，AVS-P2 采用了最小块大小是 8x8 的运动预测块，运动补偿的精度为 1/4 精度[57]；在 B 帧的编码模式中，AVS-P2 创新地引入了对称模式(Symmetry mode)和改进的时空域结合的直接模式[58]，与 H.264/AVC 中的时空域直接预测模式相比，该方式可以取得 0.2dB 的编码增益。其次，AVS-P2 采用了 8x8 大小的整数变换[59]，并且在量化的过程中，除了与变换结合进行缩放之外，还采用了预缩放技术[60]。最后，AVS-P2 采用了基于上下文自适应的二维变长编码(C2DVLC)[36]和基于上下文二进制算术编码(CBAC)[41]来对变换系数进行高效的熵编码。C2DVLC 和 CBAC 使用不同与 CAVLC 和 CABAC 的上下文自适应模型来挖掘变换系数的统计特性，取得了与 CAVLC 和 CABAC 相当的编码效率。

（4）HEVC 视频压缩标准

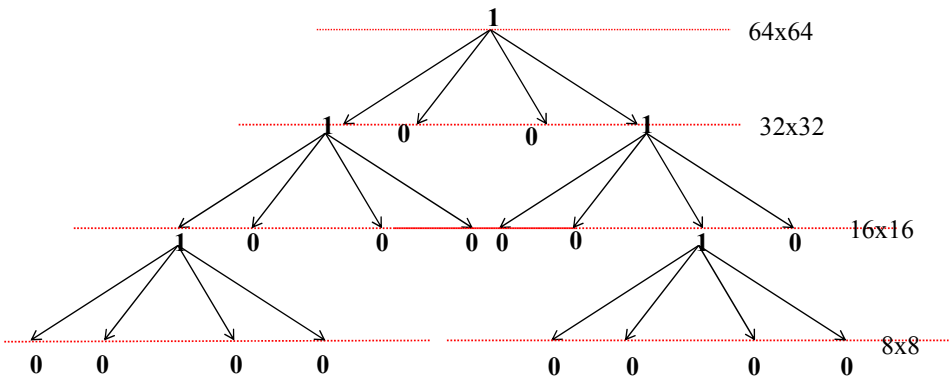
为了应对高清和超高清等视频应用的兴起和发展，MPEG 和 VCEG 联合成立的视频编码工作组 JCTVC 负责制定 H.264/AVC 的下一代视频编码标准 HEVC。与 H.264/AVC 相同，HEVC 也是基于混合编码框架的视频压缩标准。尽管如此，HEVC 在具体的编码工具上做出了很大的改进[61]。这些改进主要包括以下几个方面。

第一，HEVC 具有更加灵活地编码结构[61,62]。HEVC 中的基本编码单元是编码树单元（CTU：Coding Tree Unit），其最大尺寸可以为 64×64，最小尺寸为 8×8。在编码的过程中，每个 CTU 按照四叉树划分模式递归地划分为若干个编码单元（CU：Coding Unit）。假设 CTU 的尺寸为 $2N\times 2N$，其中 N 等于 8,16 或 32，如果不再继续对当前 CTU 进行划分，则当前 CTU 就是一个 $2N\times 2N$ 的 CU；如果继续对当前 CTU 进行划分，则按照四叉树的方式把当前 CTU 分割为 4 个尺寸为 $N\times N$ 的 CTU，然后继续判断每个尺寸为 $N\times N$ 的 CTU 是否需要分割，最终叶子节点处的 CTU 就组成了一个 CU。图 1-2（a）给出

了一个大小为 64×64 的 CTU 的划分过程示意图，图 1-2（b）列出了与其对应四叉树，其中 1 表示当前节点继续划分，0 表示当前节点是叶子节点。



（a）



（b）

图 1-2 大小为 64×64 的 CTU 划分

（a）CTU 划分示意图 （b）与该划分对应的四叉树

Fig 1-2 Partition of 64×64 CTU

(a) Illustration of the partition, (b) The corresponding quad-tree structure

第二，HEVC 采用了更加先进的预测方法[61]。与 H.264/AVC 中的宏块相似，HEVC 中的编码单元 CU 可以继续被划分为一个或多个预测单元（PU：Prediction Unit），并且每个 CU 只能采用帧内预测模式或者帧间预测模式。在帧内预测中，每个 CU 可以选择 PART_2N×2N 和 PART_N×N 两种模式，前者表示当前 CU 组成一个 PU，后者表示当前 CU 被划分为 4 个 PU。在帧

间预测中，每个 CU 有 8 种划分模式可供选择，其中 4 种是对称的 PU 划分 2N×2N、N×N、2N×N、N×2N，另外还有 4 种非对称的 PU 划分 PART_2N×nU、PART_2N×nD、PART_nL×2N、PART_nR×2N。图 1-3 给出了帧内和帧间预测的 CU 划分方式。除此之外，HEVC 定义了 35 种帧内编码预测模式来提升帧内预测的编码效率；在帧间预测上，HEVC 采用了 8-tap 的 1/2 像素插值滤波器，7-tap 的 1/4 精度的插值滤波器，并且在运动补偿的过程中采用了基于 DCT 的插值滤波器[63]。除此之外，HEVC 还使用了参考帧集合[65]（RPS：Reference Picture Set）和具有竞争机制的运动矢量预测技术[64]（AMVP：Advanced Motion Vector Prediction）等高效的编码工具大幅提升了帧间预测的编码效率。

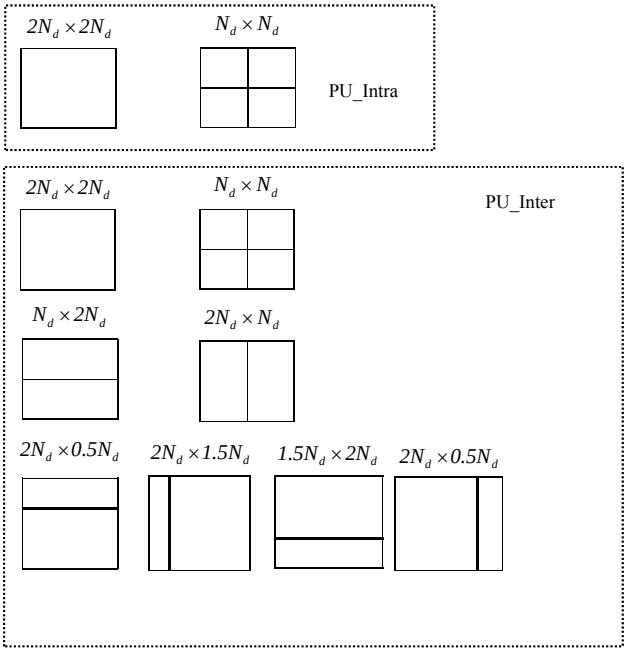


图 1-3 帧内和帧间预测模式下 CU 的划分方式

Fig 1-3 CU partition for Intra and Inter mode

第三，HEVC 采用了更加高效的变换矩阵和熵编码方法。在变换矩阵方面，HEVC 引入了大小为 32×32，16×16，8×8 和 4×4 的变换矩阵来对不同尺寸的预测残差块实施变换；另外 HEVC 还引入了 4×4 的 DST 变换来处理 4×4 的帧内预测残差块。在熵编码方面，HEVC 采用基于系数组（CG：Coefficient Group）的变换系数编码方案[66, 67]。该方法极大地提升 HEVC 中熵编码模块的数据吞吐率，使之更加适用于编码高清和超高器视频。

最后，HEVC 引入样本自适应偏移 (Sample Adaptive Offset, SAO)技术[68]，来进一步修正经过环路滤波模块处理后的重构图像。

（5）AVS2 视频压缩标准

为了应对高清和超高清等视频应用的兴起和发展，AVS 视频编码工作组于 2012 年开始制定 AVS-P2 的下一代视频压缩标准 AVS2。与现有的视频压缩标准一样，AVS2 依然采用混合编码框架，即使用预测、变换、量化和熵编码等技术对视频序列进行压缩。以 AVS-P2 为参考，AVS2 的压缩效率提高了 1 倍。AVS2 编码效率的提高主要源于 AVS2 采用了大量的高效的编码工具[69]。

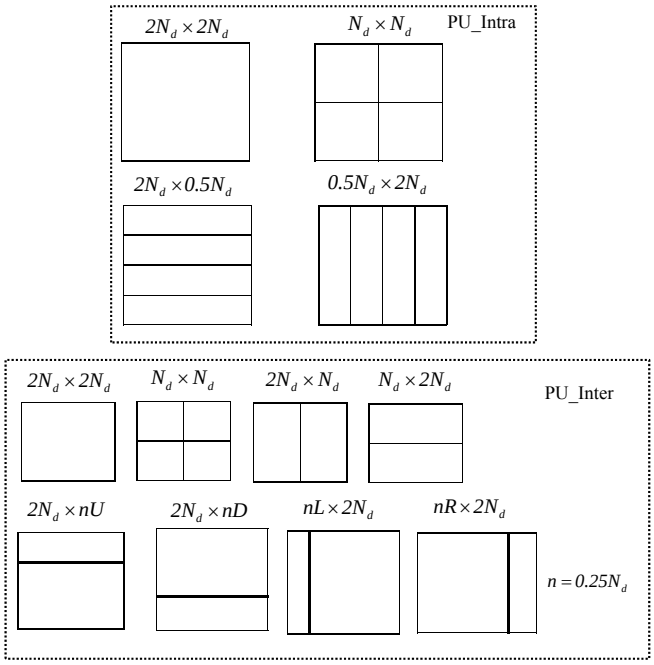


图 1-4 AVS2 中帧内预测和帧间预测中 CU 划分方式

Fig 1-4 CU partition for Intra and Inter mode in AVS2

第一，AVS2 采用了更加灵活地块划分方式。在编码单元划分过程中，AVS2 采用了基于四叉树的块划分方式；在预测单元划分过程中，AVS2 的帧内预测单元和帧间预测单元都采用了对称划分和非对称划分两种模式，如图 1-4 所示；在变换单元的选择过程中，AVS2 以率失真优化的方式为每个预测残差块选择最优的变换块大小。

第二，AVS2 采用了更加高效的帧内预测技术和帧间预测技术。在帧内预测方面，AVS2 使用了 33 种预测模式，并且采用了 4-tap 插值滤波器获取

帧内预测像素[70]；为了提升帧内预测的精度，AVS2 采用短距离帧内预测[71]（Short Distance Intra Prediction, SDIP）允许使用非正方形的划分方式，从而使得平均预测距离变短，提高预测精度；另外 AVS2 引入了 DM 模式[72]来提高色度空间的帧内预测精度，即如果采用帧内模式编码当前色度块时，AVS2 将该色度块所对应的亮度块的最优帧内预测模式作为该色度块的一种候选模式。在帧间预测方面，AVS2 引入了加权跳过模式[73]（weighted skip）和前向双假设预测帧 F 帧[74]，其中双假设预测包括空域双假设预测和时域双假设预测；另外 AVS2 还引入了渐进的运动矢量精度调整算法[75]，即根据运动矢量残差的大小来自适应地选择运动矢量精度。

第三，AVS2 采用更加灵活的变换方式和更加高效的熵编码方法。对于采用对称划分方式的预测单元，利用率失真优化的方式从 *2N*×*2N* 大小的变换单元或者 *N*×*N* 大小的变换单元中，选择一个最优的变换单元来对其进行变换；对于采用非对称划分方式的预测单元，首先采用 *2N*×*2N* 大小的变换单元对其进行变换，然后再使用基于四叉树划分的非方形变换[76]进行二次变换，从而使得变换系数更加集中。对于变换系数，AVS2 首先把变换系数块分成 4×4 大小的子块，然后以每个子块为单位进行熵编码，从而可以降低熵编码过程中的缓存消耗[77]。

最后，AVS2 引入样本自适应偏移 (Sample Adaptive Offset, SAO)技术[78]和自适应环路滤波[79]，来进一步提高重构图像的质量。

1.3 图像压缩感知

在某些应用场景中，由于外部条件的限制，信号采集设备无法完整地采集信号，而信号重构端却具有强大的计算能力，可以执行一些复杂的信号恢复算法来重构信号。考虑到这种应用场景的特点，学术界提出了压缩感知采样理论。

1.3.1 压缩感知基本理论

压缩感知作为一种新型的采样理论，在远小于 Nyquist 采样率的条件下，利用高斯随机矩阵来获取信号的测量值，每个测量值包含整个信号的全局信息。然后，重构端利用测量值采用非线性重建算法完美地重建信号。假设 $\mathbf{x}$ 表示维度为 N 的原始信号，$\mathbf{y}$ 表示通过随机采样获取的原始信号 $\mathbf{x}$ 的测量值，其中 $\mathbf{y}$ 的维度为 M 且 $M \ll N$。也就是说 $\mathbf{x}$ 与 $\mathbf{y}$ 之间满足如下关系，

$$\mathbf{y} = \mathbf{\Phi x} \tag{1-1}$$

其中，$\boldsymbol{\Phi}$ 是 $M \times N$ 的高斯随机矩阵，一般称之为采样矩阵，此时采样率 S（subrate: subsamplaing rate）为 $S = M / N$。由于 $\mathbf{y}$ 的维度远小于 $\mathbf{x}$ 的维度，因此传统的信号恢复方法是不能根据 $\mathbf{y}$ 恢复出 $\mathbf{x}$。但是如果 $\mathbf{x}$ 在某个域中是稀疏的，则压缩感知理论证明了可以利用 $\mathbf{y}$ 精确地恢复出 $\mathbf{x}$。

假设信号 $\mathbf{x}$ 经过变换矩阵 $\boldsymbol{\Psi}$ 作用后得到的变换系数向量是 $\overline{\mathbf{x}}$，如果按照幅值的降序顺序对 $\overline{\mathbf{x}}$ 中各个元素进行排序后，元素幅值是以指数形式递减的，则 $\overline{\mathbf{x}}$ 是稀疏的，其形式化描述如下：

$$\overline{\mathbf{x}} = \boldsymbol{\Psi}\mathbf{x} \quad and \ |\overline{x}_n| \geq |\overline{x}_{n+1}|, |\overline{x}_n| < Rn^{-r} \tag{1-2}$$

其中 $\overline{\mathbf{x}}$ 是 $\mathbf{x}$ 对应于 $\boldsymbol{\Psi}$ 的变换系数；$|\overline{x}_n|$ 是按照幅值的降序顺序对 $\overline{\mathbf{x}}$ 的各个元素进行排序后，处在位置 n 处的元素幅值；$r \geq 1$，$R < \infty$。假设只保留 $\overline{\mathbf{x}}$ 的 K 个最大幅值元素 $\overline{\mathbf{x}}_K$，则根据压缩感知理论，利用 $\mathbf{y}$ 得到的 $\mathbf{x}$ 重构值 $\hat{\mathbf{x}}$ 与原始信号 $\mathbf{x}$ 之间的误差满足下述关系：

$$\left\|\hat{\mathbf{x}} - \mathbf{x}\right\|_2 \leq C \frac{\left\|\mathbf{x} - \mathbf{x}_K\right\|_1}{\sqrt{K}} \tag{1-3}$$

其中，C 是一个常数，$\mathbf{x}_K = \boldsymbol{\Psi}^{-1}\overline{\mathbf{x}}_K$，$\boldsymbol{\Psi}^{-1}$ 是 $\boldsymbol{\Psi}$ 的逆矩阵。

为了从测量值 $\mathbf{y}$ 中恢复 $\mathbf{x}$，最直接的方法就是求解下述最优化方程：

$$\min \left\|\overline{\mathbf{x}}\right\|_0 \quad such\ that \ \ \mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\Psi}^{-1}\overline{\mathbf{x}} \tag{1-4}$$

得到 $\overline{\mathbf{x}}$，然后重建信号 $\hat{\mathbf{x}}$ 可用稀疏矩阵 $\boldsymbol{\Psi}$ 求得，即 $\hat{\mathbf{x}} = \boldsymbol{\Psi}^{-1}\overline{\mathbf{x}}$。尽管如此，求解 $\left\|\overline{\mathbf{x}}\right\|_0$ 是一个 NP 完全问题[121]。因此，在实际求解中[122]，往往会把 0 范数转化为 1 范数，即：

$$\min \left\|\overline{\mathbf{x}}\right\|_1 \quad such\ that \ \ \mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\Psi}^{-1}\overline{\mathbf{x}} \tag{1-5}$$

在实际应用中，测量值 $\mathbf{y}$ 不可避免地会感染噪声，即 $\mathbf{y} = \boldsymbol{\Phi}\mathbf{x} + \mathbf{n}$，其中 $\mathbf{n}$ 是噪声向量。在这种情况下，可以放宽公式（1-5）中的限制条件：

$$\min \left\|\overline{\mathbf{x}}\right\|_1 \quad such\ that \ \ \left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\Psi}^{-1}\overline{\mathbf{x}}\right\|_2 \leq \varepsilon \tag{1-6}$$

其中 ε 是任意小的正数。为了求解公式（1-6），可以采用拉格朗日乘数法进行求解，即

$$\min \left\|\overline{\mathbf{x}}\right\|_1 + \lambda \cdot \left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\Psi}^{-1}\overline{\mathbf{x}}\right\|_2^2 \tag{1-7}$$

其中 λ 是拉格朗日乘数因子，它的作用是平衡 $\overline{\mathbf{x}}$ 的稀疏度 $\left\|\overline{\mathbf{x}}\right\|_1$ 和保真项 $\left\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\Psi}^{-1}\overline{\mathbf{x}}\right\|_2^2$ 对重构信号的影响。

目前已经有很多算法可以求解公式（1-5），（1-6）和（1-7）等最优化问题，文献[123]对这些算法进行了总结。除了这些算法之外，还有基于梯度下降算法如 IST[124]（Iterative Splittling and Thresholding）和 SpaRSA[125]（sparse reconstruction via separable approximation）以及 GPSR[126]（gradient projection for sparse reconstruction）可以更快速地求解上述最优化问题。另外，贪心算法如匹配追踪[127]（MP：Match Pursuits），正交匹配追踪[128]（OMP：Orthogonal Matching Pursuits）也被用来求解上述最优化问题。

1.3.2 图像压缩感知采样及其重构算法

为了对一幅大小为 $N \times N$ 的图像实施压缩感知采样，最直接的方法是首先把该二维图像按照光栅扫描的方式转化为一维向量 $\mathbf{x}$，然后按照公式（1-1）对该向量进行采样得到测量值 $\mathbf{y}$，此时测量矩阵 $\mathbf{\Phi}$ 的维度是 $M \times N^2$。当利用 $\mathbf{y}$ 重建信号 $\mathbf{x}$ 时，信号重建的过程首先需要知道信号 $\mathbf{x}$ 的稀疏变换矩阵 $\mathbf{\Psi}$ 和测量矩阵 $\mathbf{\Phi}$，然后通过求解最优化问题（1-4）或（1-5）即可重建信号 $\mathbf{x}$。对于大小为 $N \times N$ 的图像，其稀疏矩阵 $\mathbf{\Psi}$ 的维度是 $N^2 \times N^2$。在这种情况下，当图像的分辨率越来越高时，存储测量矩阵 $\mathbf{\Phi}$ 和稀疏矩阵 $\mathbf{\Psi}$ 会消耗大量的内存空间，并且在信号重建的过程中，对维度如此之高的矩阵做乘法操作也将带来很高的计算负担。因此为了解决这些问题，有学者提出了基于块的压缩感知采样[129-133]（BCS：Block-based Compressive Sensing）。

在 BCS 中，图像首先被分成大小为 $B \times B$ 非重叠的图像块，然后对每个图像块做压缩感知采样。假设对于图像 $\mathbf{x}$ 中的第 j 个图像块，对其进行光栅扫描后得到的一维向量是 $\mathbf{x}_j$，则该图像块的其测量值 $\mathbf{y}_j$ 为

$$\mathbf{y}_j = \mathbf{\Phi}_B \mathbf{x}_j \tag{1-8}$$

其中，$\mathbf{y}_j$ 的长度为 M_B，$\mathbf{\Phi}_B$ 是维度为 $M_B \times B^2$ 的测量矩阵，其所对应的采样率是 M_B / B^2。

针对基于 BCS 的图像采样，图像重构端采取分块重构的策略，即先重构出每一个图像块，然后把重构的图像块拼接生成一幅完整的图像。但是这种重构方法仅仅考虑了块的稀疏度，而没有考虑当前图像块与整幅图像之间的关系，因此这种方法会使得重构图像中存在很严重的块效应。为了去除块效应并提高重构图像的质量，文献[129]提出了基于投影的 Landweber 迭代方法（BCS-SPL）。该方法首先重构每一个图像块并把重构的图像块拼接成一幅图像，然后以这幅图像为初始值进行迭代处理。在每次迭代过程中，首先利

用维纳滤波器对重构图像进行滤波来保持图像的平滑特性，在使用硬阈值操作来保证整幅图像的稀疏度。在此基础上，文献[130]采用方向性变换并且利用变换系数的统计特性来估计出一个阈值用于变换系数的缩放以保证整幅图像的稀疏度。文献[131]利用小波变换对图像进行变换，然后对不同频带位置的子图像使用不同的采样率进行采样，在相同采样率的条件下，这种采集方法可以取得更好的重构质量。文献[132]首先利用多假设预测方法对当前待处理的块进行预测，然后对该预测值进行压缩感知采样，并且与当前块的测量值相减得到测量值残差，接着使用 BCS-SPL[129]算法根据测量值的残差重构得到像素域的残差，最后，当前块的重构值就等于重构的像素域残差加上预测值。由于像素域的残差数据更加稀疏，因此该方法可以取得更好的图像重构质量。文献[133]对 BCS 的图像采样和恢复算法进行了总结。

1.3.3 图像压缩感知采样的压缩技术

压缩感知采样可以把原始信号映射至维度更低的空间，可以认为压缩感知技术在采样的同时对原始信号进行了压缩。但是从信息论[4,5]的角度来说，这种压缩并不是真正意义上的数据压缩，因为数据压缩通常是指信源编码[6]，即把任意类型的信号转化为紧凑的二进制码流。

为了把测量值转化为紧凑的二进制码流，量化模块是必不可少的一个模块。最简单的量化方式就是对测量值采用标量量化（SQ：Scalar Quatization）。但是文献[134]已经证明对测量值采用标量量化会取得很差的率失真性能。因此，为了提高量化模块的率失真性能，最近很多研究者都在研究适用于压缩感知的量化技术。文献[135,136]提出了一个均方误差（MSE：Mean Square Error）最优的非均匀量化方法。文献[137]提出了 BPDQ（Basis Pursuit Dequantizer）算法，可以利用经过标量量化的测量值恢复出高质量的图像。但是这个算法仅仅适用于测量值足够多的情况；当采集得到的测量值不能够捕捉图像稀疏特性时，该方法的重构效果会变差。文献[138]提出了一个渐进量化方法（PQ：Progressive Quantization），该方法把采集得到的测量值分成两部分，一部分是基本层，另一部分是精细层；该方法首先对基本层做粗量化，然后根据由基本层重构的图像再对精细层做细量化。对于一般的图像，PQ 算法可以取得较好的率失真性能，但是对于具有复杂纹理的图像，PQ 算法的效果并不好。文献[139]把均匀标量量化和 DPCM 预测技术结合在一起应用于 BCS 框架中，取得了与文献[137, 138]相当的率失真性能。但是该方法的计算复杂度要远低于文献[137, 138]中的算法。

1.4 视频/图像压缩中的熵编码技术

在视频和图像压缩系统中，预测残差经过量化后产生量化索引。由于预测，变换和量化技术并不能完全去除视频和图像中的冗余信息，因此量化索引还存在着一些冗余信息，这些冗余信息通常被称之为统计冗余。熵编码技术正是为了去除信源符号中的统计冗余而被提出的。

1.4.1 信息熵和自信息

定义 $\boldsymbol{\mathcal{F}}=\{\mathcal{F}_n\}$ 表示一个信源，其中 $\mathcal{F}_n$ 是一个随机变量，表示该信源输出的第 n 个字符。定义 $\mathcal{F}_n$ 在某一实例的具体取值记为 f_n，其取值集合为 $\boldsymbol{\mathcal{A}}=\{a_1,a_2,...,a_L\}$，假设用 $p_{\mathcal{F}}(f)$ 表示信源的概率质量函数(pmf: probability mass function)，则当信源输出一个符号 f，该符号所包含的信息，即该符号的自信息被定义为：

$$I(f)=\log\frac{1}{p_{\mathcal{F}}(f)} \tag{1-9}$$

其中 log 表示以 2 为底的对数，这样计算得到的自信息的单位是 bit/符号。

自信息 $I(f)$ 是信源输出符号 f 时所包含的信息量。当信源输出不同的符号时，通常情况下，这些符号具有不同的自信息。为了衡量该信源所包含的信息量，信息论引入了信息熵的概念，即所有符号的自信息的平均值，其计算公式如下，

$$H_1(\boldsymbol{\mathcal{F}})=E\left[\log\frac{1}{p_{\mathcal{F}}(f)}\right]=-\sum_{f\in\boldsymbol{\mathcal{A}}}p_{\mathcal{F}}(f)\log p_{\mathcal{F}}(f) \tag{1-10}$$

信息熵 $H_1(\boldsymbol{\mathcal{F}})$ 从平均意义上度量了某一信源所包含的信息量，其物理意义为该信源每输出一个符号所能提供的平均信息量。

信息熵 $H_1(\boldsymbol{\mathcal{F}})$ 也被称为一阶熵。通常情况下，信源输出的符号序列，则信源的 N 阶熵被定义如下形式，

$$\begin{aligned}H_N(\boldsymbol{\mathcal{F}})&=H(\boldsymbol{\mathcal{F}}_1,\boldsymbol{\mathcal{F}}_2,...,\boldsymbol{\mathcal{F}}_N)\\&=-\sum_{[f_1,f_2,...,f_N]\in\boldsymbol{\mathcal{A}}^N}p(f_1,f_2,...,f_N)\log p(f_1,f_2,...,f_N)\end{aligned} \tag{1-11}$$

其中，$\boldsymbol{\mathcal{A}}^N$ 是集合 $\boldsymbol{\mathcal{A}}$ 的 N 阶笛卡尔积，$p(f_1,f_2,...,f_N)$ 是 N 个连续符号的联合概率。此外，信息论还引入了信源的 $\boldsymbol{N}$ 阶条件熵，其定义为，

$$H_{C,N}(\boldsymbol{\mathcal{F}}) = H(\boldsymbol{\mathcal{F}}_{N+1} \mid \boldsymbol{\mathcal{F}}_1, \boldsymbol{\mathcal{F}}_2, \cdots, \boldsymbol{\mathcal{F}}_N) = -\sum_{[f_1, f_2, \ldots, f_N] \in \boldsymbol{\mathcal{A}}^N} p(f_1, f_2, \ldots, f_N) H(\boldsymbol{\mathcal{F}}_{N+1} \mid f_1, f_2, \cdots, f_N) \quad (1\text{-}12)$$

其中，$\boldsymbol{\mathcal{F}}_1, \boldsymbol{\mathcal{F}}_2, \cdots, \boldsymbol{\mathcal{F}}_N$ 称为 $\boldsymbol{\mathcal{F}}_{N+1}$ 的上下文，并且 $H(\boldsymbol{\mathcal{F}}_{N+1} \mid f_1, f_2, \cdots, f_N)$ 的定义如下，

$$H(\boldsymbol{\mathcal{F}}_{N+1} \mid f_1, f_2, \cdots, f_N) = -\sum_{f_{N+1} \in \boldsymbol{\mathcal{A}}} p(f_{N+1} \mid f_1, f_2, \cdots, f_N) \log p(f_{N+1} \mid f_1, f_2, \cdots, f_N) \quad (1\text{-}13)$$

信息论中已经证明 $H_N(\boldsymbol{\mathcal{F}})/N$ 和 $H_{C,N}(\boldsymbol{\mathcal{F}})$ 都是变量 N 的非递增函数，并且当 N 趋向于无穷大时，它们的极值都存在且相等。这个极值被定义为**极限熵**（entropy rate），记为 $\bar{H}(\boldsymbol{\mathcal{F}})$。而且 $H_N(\boldsymbol{\mathcal{F}})/N$，$H_{C,N}(\boldsymbol{\mathcal{F}})$，$H_1(\boldsymbol{\mathcal{F}})$ 和 $\bar{H}(\boldsymbol{\mathcal{F}})$ 之间存在如下关系，

$$\bar{H}(\boldsymbol{\mathcal{F}}) \le H_{C,N}(\boldsymbol{\mathcal{F}}) \le \frac{1}{N} H_N(\boldsymbol{\mathcal{F}}) \le H_1(\boldsymbol{\mathcal{F}}) \quad (1\text{-}14)$$

其中当信源是无记忆信源 i.i.d（*independent and identically distributed*）时，公式（1-14）中的等号才成立。因此，如果信源输出的连续符号之间存在相关性，可以通过利用联合熵和条件熵来获得更高的编码效率，即通过多个符号联合编码方法和上下文建模技术来提高熵编码的编码效率。

1.4.2 熵编码引擎

熵编码引擎是按照一定的规则把信源符号转化为能够唯一译码的码字，其主要目标是去除信源符号中的统计冗余。目前广泛使用的熵编码引擎包括变长编码和算术编码两种编码方法。

变长编码 VLC 是把一个信源符号或者多个信源符号转化为一个码字的熵编码方法。目前视频和图像压缩系统中广泛使用的变长编码方法包括 Huffman 编码[18]，指数哥伦布编码[21]（Exp-Golomb Code）和哥伦布莱斯码（Golomb-Rice Code）编码[22]以及一元码（Unary Code）[23]。Huffman 编码的基本思想是根据信源符号的概率分布为信源符号分配码字，具体来说就是，为出现概率高的信源符号分配短码字，而为出现概率低的信源符号分配长码字。由于 Huffman 编码需要在编解码端存储 Huffman 树，因此当信源符号增多时，Huffman 编码将会消耗更多的内存空间并且使用一个体积很大的 Huffman 树进行编解码也会增加编解码的计算复杂度。为了解决 Huffman 编码的这些缺点，研究人员提出了指数哥伦布码，哥伦布莱斯码和一元码。这

三种编码方法具有规则的码字结构。对于指数哥伦布编码方法，其码字是由前缀和后缀组成。当使用 k 阶指数哥伦布码编码正整数 n 时，产生码字的前缀由整数 $y=\left\lfloor \log_2(\frac{n}{2^k}+1) \right\rfloor$ 的一元码组成，后缀由 $z=n+2^k(1-2^y)$ 的长度为 $k+y$ 的定长码组成；当使用 k 阶哥伦布莱斯码编码正整数 n 时，产生码字的前缀由整数 $\mathrm{y}=\left\lfloor \frac{n}{2^k} \right\rfloor$ 的一元码组成，后缀由 $z=n-y\cdot 2^k$ 的长度为 k 的定长码组成。指数哥伦布码和哥伦布莱斯码对于概率分布满足几何分布的信源可以取得最优的编码效率。当使用一元码编码正整数 n 时，它产生的码字是由 n 个 1 加一个 0 组成（亦或有 n 个 0 加一个 1 来组成）。一元码对概率分布满足指数分布的信源可以取得最优的编码效率。

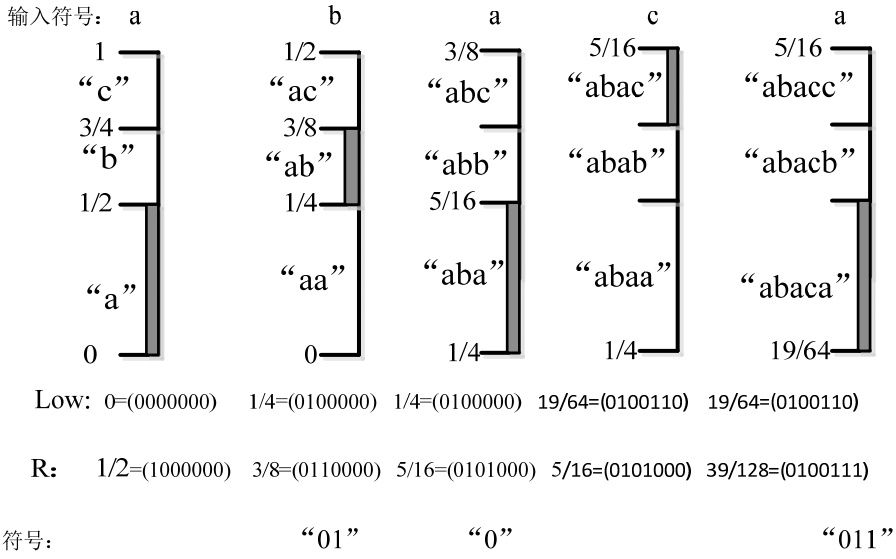


图 1-5 符号串“abaca”的算术编码过程

Fig 1-5 The arithmetic coding procedure for encoding “abaca”

算术编码 AC 是把整个输入信源符号转化为一个或多个码字的熵编码方法。香农信息论最早提出了算术编码的思想。文献[24, 25]根据这个思想提出了可实际应用的算术编码方法。简单来说，算术编码过程是递归地对编码区间进行划分的过程，它包含三个步骤。第一，按照信源符号概率分布对编码区间进行分割，每个信源符号对应其中一个子区间；第二，根据当前要编码的信源符号选择与其对应的子区间，作为编码下一个信源符号的编码区间；第三，重复上述两个步骤直至编码结束所有的信源符号。图 1-5 给出了符号

串“abaca”的算术编码过程。

1.4.3 上下文建模技术

从 1.4.1 节可知，利用信源符号的条件熵可以获得更高的编码效率。基于这种思想，视频和图像编码系统都采用基于上下文的熵编码技术，即使用先前已经编码过的信源符号来估计当前待编码的信源符号的概率，进而熵编码引擎使用这个条件概率来对当前符号实施编码。

为某一信源选择合适的上下文模型主要涉及到两个问题，第一是从先前已经编码的符号中选择哪些符号作为上下文；第二是如何利用设计好的上下文准确地估计条件概率。针对第一个问题，Rissanen 最早提出了上下文状态树方法[80]，该方法把历史信息的不同排序用上下文状态树的形式来表示，在编码时上下文状态的决策就是对上下文状态树的裁剪。由于这种方法在图像和视频等二维信号的上下文建模中会产生上下文稀疏问题[81]，即没有足够多的训练样本来获取稳定的概率分布，因此这种方法不适用于图像和视频的熵编码设计。所以图像和视频压缩系统一般都采用待编码符号的先验知识来指导上下文建模。尽管如此，在图像和视频的上下文建模过程中，仍然会产生上下文稀疏问题。为了解决上下文设计过程中产生的上下文稀疏问题，有学者提出了上下文量化技术[82, 83]，即把概率分布相同的上下文合并为一个上下文。针对第二个问题，文献[84]提出的 scaled-count 方法，该方法对最近出现符号给予更大权重；文献[85]对概率估计过程中的概率更新速度进行了研究。

1.5 本文课题的提出及其主要贡献

根据应用场景的不同，数字图像和视频可以利用传统的采样方式或者压缩感知采样方式获得样本。传统的采样方式利用 Nyquist 采样定理把数字图像/视频采集成$W \times H$的像素阵列，并且每个像素通常用 8 比特来表示。压缩感知采样通常利用随机高斯矩阵对原始信号进行投影操作而得到测量值，每个测量值包含了原始信号的全局信息，并且各个测量值之间是相互独立的。所以，传统采样得到的像素值与压缩感知采样得到的测量值在本质上是不同的。因此，在不同的采样方式下，需要使用不同的压缩算法对样本进行压缩。通常情况下，数字视频压缩标准被用于压缩通过传统采样方式获得的像素值；测量值压缩算法被用于压缩通过压缩感知采样方式获得的测量值。在数据压缩系统中，熵编码模块的主要作用是去除信源符号之间的统计冗余，同时生成码流，以供存储和传输。因此，熵编码模块是数据压缩系统中的很重要的

模块。

在 H.264/AVC 的制定过程中，标清视频是其要处理的主流视频信号，因此 CABAC 的吞吐率并没有引起人们的注意，从而导致 CABAC 中存在着大量的数据依赖关系。随着高清和超高清视频的兴起，人们逐渐意识到 CABAC 已经成为 H.264/AVC 解码器的主要瓶颈，严重地限制着解码器速度的提高和功耗的降低。因此，如何在保证编码性能的前提下，提高 H.264/AVC 中 CABAC 的吞吐率是视频压缩技术研究过程中的一个很重要的问题。

HEVC 是 H.264/AVC 的下一代视频编码标准，也是目前最新的国际视频压缩标准。相比于 H.264/AVC，HEVC 采用了更加高效的预测和变换技术，使得预测残差的变换系数块更加稀疏；此外，由于 HEVC 的主要应用领域是高清和超高清视频的压缩，因此 HEVC 还需要提高变换系数编码方案的吞吐率。由于这些原因，H.264/AVC 中的变换系数编码方案已经无法用于编码 HEVC 中的变换系数。为了充分挖掘变换系数块的统计特性和提高变换系数编码模块的吞吐率，HEVC 重新设计了变换系数块的编码流程。尽管如此，随着数字视频的分辨率越来越高，未来的视频压缩标准将要处理数据量更加庞大的数字视频；并且，随着科技的发展，硬件设备的成本将会越来越低，它们的性能将会越来越高。因此，在不显著增加硬件设计复杂度的条件下，继续提高 HEVC 中熵编码模块的编码效率依然是熵编码技术研究的热点问题。

AVS2 是我们国家制定的最新的视频压缩标准，它的主要应用领域也是实现对高清和超高清视频的压缩。为了避免专利纠纷和保护民族信息产业，AVS2 大量地采用了具有自主知识产权的编码技术。在相同的重构视频质量下，AVS2 能够比 AVS-P2 节省接近一半的码率。尽管如此，由于设计上的缺陷，AVS2 中的熵编码模块中存在着很强的顺序依赖关系，这些顺序依赖关系严重地制约着 AVS2 解码器的吞吐率。因此，在保证编码性能的前提下，提高 AVS2 中熵编码模块的吞吐率对 AVS2 标准的硬件实现有着十分重要的意义。

当利用压缩感知采样技术对数字视频/图像进行采样时，需要使用测量值压缩算法来把采集得到的测量值转化为紧凑的二进制码流，以供存储和传输。在目前的测量值压缩算法中，还没有一个高效的熵编码方案来去除测量值之间的统计冗余，同时把测量值转化为紧凑的二进制码流。因此，如何根据测量值的统计特性为其设计高效的熵编码方案也是一个亟需解决的问题。

针对以上叙述的问题，本文的研究工作围绕着视频压缩标准和测量值压缩算法中熵编码技术展开的，其主要目标是提高数字视频/图像编码系统中熵

编码模块的吞吐率和编码性能。本文研究内容的组织结构如图 1-6 所示。

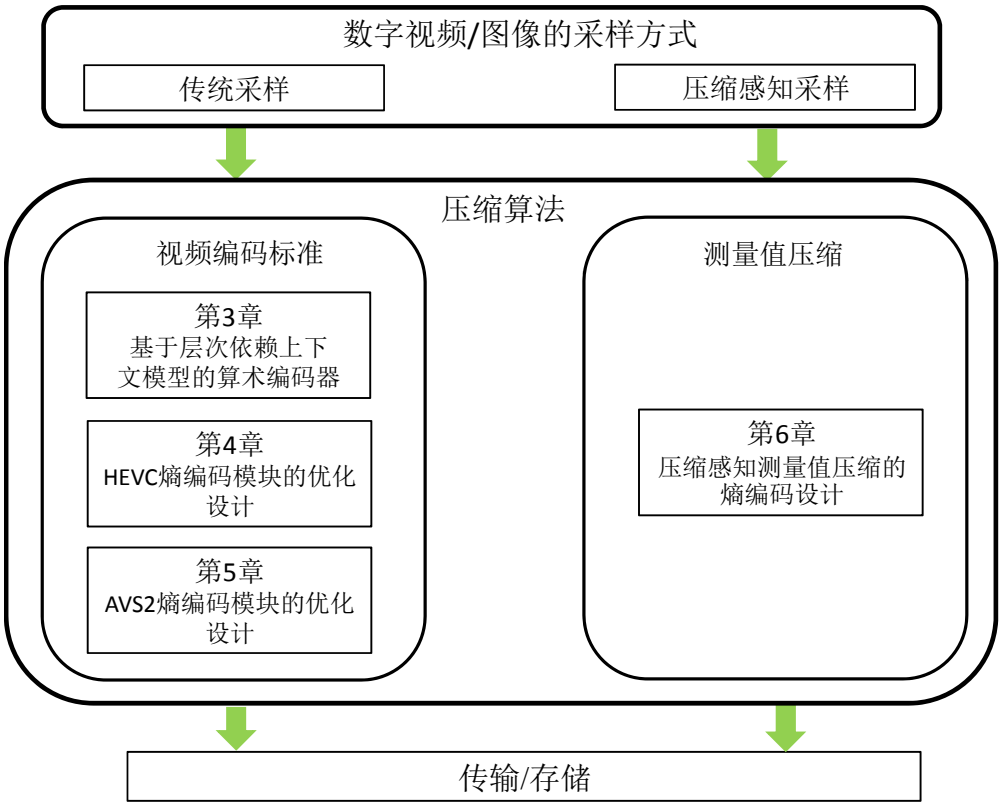


图 1-6 本文研究内容组织结构示意图

Fig 1-6 The organization framework of this thesis

具体来讲，本文的研究内容包括以下四个部分：

（1） 提出了一个基于层次依赖上下文模型的算术编码方案。为了提高 H.264/AVC 中 CABAC 的数据吞吐率，本文提出了层次依赖上下文模型 HDCM (Hierarchical Dependency Context Model) 来挖掘 DCT 系数之间的统计冗余。基于这个上下文模型，本文设计了一个基于层次依赖上下文模型的二进制算术编码器 HDCMBAC 来编码预测残差的 DCT 系数。在 HDCMBAC 中，同类语法元素之间不再产生上下文依赖关系，因此 HDCMBAC 最大限度地降低了 DCT 系数之间的上下文依赖关系，同时可以取得与 CABAC 相似的编码效率。

（2） 提出了一个增强的变换系数的上下文建模过程和低内存消耗的算术编码引擎。为了提高 HEVC 中的变换系数的压缩效率，本文针对变换系数

的统计特性提出了一个增强的上下文建模方法，该上下文建模方法采用当前变换系数的一个局部模板内变换系数的幅值信息和当前变换系数的位置信息作为它的上下文；为了降低上下文模型的个数，同时为了避免产生上下文稀疏问题，本文把变换系数块分割为不同的区域，并且规定相同区域内的变换系数共享一个上下文模型集合。另外，为了提高 HEVC 中算术编码引擎的编码效率和降低内存消耗，本文基于多参数的概率估计模型提出了一个低内存消耗的自适应的二进制算术编码引擎。该算术编码引擎使用多参数的概率估计模型来估计二进制符号的概率，并且在编码区间分割过程中采用了一个低位宽的乘法操作来代替传统的查表操作。如此设计之后，该算术编码引擎将不再需要大量的内存资源来预存信息。与 HEVC 中原有的熵编码方案相比，本文提出的熵编码方案具有更高的编码效率和更低的内存消耗。

（3） 提出了 AVS2 熵编码模块的优化设计方案。为了提高 AVS2 中熵编码模块的数据吞吐率，本文从三个方面对 AVS2 的熵编码模块进行了优化设计。首先，本文提出了一种快速的，并且与标准兼容的算术编码引擎归一化方法；其次，本文提出了一个快速的 bypass bin（概率等于 0.5 的二进制符号）的编解码过程；最后，本文改进了 AVS2 中变换系数的编码过程，来最大限度地降低变换系数之间的编码依赖关系。本文提出的这三个技术可以大幅度地提高 AVS2 中的熵编码模块的数据吞吐率，同时性能损失也比较小。

（4） 提出了一个适用于图像压缩感知采样中测量值压缩的算术编码方案。为了进一步提高测量值的压缩性能，本文基于图像的差分脉冲预测和标量量化框架（DPCM-plus-SQ）,为测量值的量化索引提出了一个高效的算术编码方案。在该编码方案中，本文首先分析了测量值的量化索引的统计特性；然后设计了相应地语法元素来描述测量值的量化索引。具体来说，首先使用语法元素 *significant_map* 来指示当前测量值的量化索引是否为非零，然后使用语法元素 *abs_coeff_level_minus1* 来指示当前测量值的量化索引的幅值，最后使用 *sign_flag* 来指示当前测量值的量化索引的符号。实验结果表明，与测量值的量化索引的 0 阶信息熵和 H.264/AVC 中 CABAC 相比，本文提出的熵编码方案可以进一步提高测量值的压缩性能。

本文内容的具体组织结构安排如下：

第 1 章，本章首先介绍视频编码技术和视频编码标准的发展历程；其次介绍了压缩感知采样的基本理论和图像压缩感知采样的相关技术以及图像压缩感知采样的编码技术；然后介绍了熵编码的基本理论和目前广泛使用的熵

编码引擎以及上下文建模技术；最后给出本文的主要贡献和内容安排。

第 2 章，本章按照视频压缩标准的发展历程对其中的熵编码技术进行总结，主要包括早期的视频压缩标准中的熵编码技术，H.264/AVC 的熵编码技术及其后来的改进，AVS-P2 的熵编码技术和 HEVC 中的熵编码技术。

第 3 章，本章首先介绍了 H.264/AVC 中的变换系数编码方法，然后根据变换系数的统计特性提出了层次依赖关系的上下文模型，并且利用这个新的上下文模型重新设计和实现变换系数的熵编码模块。最后对本文提出的熵编码方法从编码性能和并行性等方面做了详细的分析。

第 4 章，本章首先介绍了 HEVC 中变换系数的编码方案和算术编码引擎的流程，然后详细介绍本文提出的变换系数的增强上下文建模方法和低内存消耗的自适应二进制算术编码引擎。最后对本文提出的熵编码方案在编码性能和编码时间两个方面进行了分析评价。

第 5 章，本章首先介绍了 AVS2 的算术编码引擎和变换系数的上下文模型，并指出其中存在的问题；然后详细介绍本文提出的解决方案，主要包括算术编码引擎的快速归一化方法和 bypass bin 的快速编解码方法以及变换系数的上下文建模过程的优化方案。最后分别对本文提出的编码技术在编码性能和编码时间两个方面进行测试分析。

第 6 章，本章首先介绍基于 DPCM 和标量量化的测量值压缩框架，然后对该框架中的测量值残差的量化索引的统计特性进行分析，进而为测量值残差的量化索引设计相应的语法元素和上下文模型，接着采用二进制算术编码引擎对这些语法元素进行压缩，最后对这个编码方案的编码性能进行了验证。

第2章 视频压缩中熵编码技术的研究现状

如何提高熵编码模块的编码效率一直都是视频和图像压缩系统的关键问题。为了实现这个目标，熵编码引擎从最初的变长编码发展到目前的算术编码；上下文模型也从最初的静态模型发展为目前的具有自适应更新特性的动态模型。近些年来，随着新型多媒体应用的出现，数据吞吐率逐渐成为衡量熵编码技术优劣的重要指标。本章将按照视频编码标准的发展历程来介绍国内外熵编码技术的研究现状。

2.1 早期视频编码标准中的熵编码技术

早期的视频压缩标准如 MPEG-1 和 MPEG-2 都是采用非自适应的 Huffman 编码作为熵编码方法。其中 MPEG-1 仅仅定义了一个 Huffman 码表用于编码变换系数的量化索引。在视频序列中，由于不同帧具有不同的统计特性，待编码数据的统计特性也会随着编码帧的不同而发生变化；另外帧内编码的帧和帧间编码的帧也具有不同统计特性。因此，这种简单的熵编码方案具有很低的编码效率。所以，MPEG-2 就引入了另外一个 Huffman 表来编码帧内宏块的变换系数。尽管如此，MPEG-2 中的熵编码仍然无法根据待编码数据的统计特性自适应地切换码表。后来的视频编码标准 H.263 同样采用非自适应的 Huffman 作为熵编码方法，但是采用不同的 Huffman 码表来编码运动矢量和变换系数。除了变长编码之外，H.263 还提出了基于语法的算术编码 SAC[86]（Syntax-based Arithmetic Coding）作为熵编码方法。考虑到帧内宏块和帧间宏块具有不同的统计特性，SAC 分别为帧内宏块和帧间宏块定义了不同的概率模型；另外在编码变换系数时，SAC 还使用变换系数的位置作为上下文来挖掘变换系数与其位置之间的相关性。与 H.263 中的变长编码相比，SAC 可以取得 5%的编码增益。

2.2 H.264/AVC 中的熵编码技术

与 H.263 类似，H.264/AVC 也同时支持变长编码和算术编码两种熵编码方法，其中变长编码被称为通用变长编码（UVLC：Universal Variable Length Coding），算术编码被称为上下文自适应的二进制算术编码（CABAC：Context-based Adaptive Binary Arithmetic Coding）。与初期视频编码标准中

的熵编码技术相比，UVLC 和 CABAC 大大地提高了视频编码性能。在 H.264/AVC 的推广过程中，人们又陆续提出了一些方法来进一步提高 UVLC 和 CABAC 的编码效率。后来随着高清视频的出现，人们发现 UVLC 和 CABAC 的数据吞吐率很低，以至于很难满足高清视频的实时编解码。为此，人们提出了很多改进方案来提高 UVLC 和 CABAC 的数据吞吐率。

（1）通用变长编码 UVLC

由于不同的语法元素具有不同的统计特性，并且它们在码流中所占的比例也不同，因此 UVLC 采用了不同的变长编码方法来编码不同的语法元素，包括非自适应的 0 阶指数哥伦布码（0 order Exponential-Golomb）和上下文自适应的变长编码方法（CAVLC：Context-based Variable Length Coding）。

表 2-1 部分 0 阶指数哥伦布码

Table 2-1 0 order Exponential-Golomb codes for some integers

码字索引 *CodeNumber*（$n = 0 \sim 6$）	0 阶	
	前缀	后缀
0	1	-
1	01	0
2	01	1
3	001	00
4	001	01
5	001	10
6	001	11

由 1.4.2 节可知，对于概率分布呈几何分布的语法元素，0 阶指数哥伦布码可以取得接近最优的编码性能。在码字结构上，0 阶指数哥伦布码由前缀和后缀两部分构成，表 2-1 列出了部分 0 阶指数哥伦布码的码字。从中可以看到 0 阶指数哥伦布码的前缀部分一直比后缀部分多 1 位，这种规则的码字结构极大地降低了 0 阶指数哥伦布码的编解码复杂度。由于 0 阶指数哥伦布码具有这些优点，UVLC 使用 0 阶指数哥伦布码来编码运动矢量残差和预测模式等控制信息。

由于变换系数在整个码流中占据一半以上的数据量，因此 UVLC 采用了 CAVLC 来对其进行编码。对于变换系数来说，由于图像中存在空域相关性，

当前变换系数块和其相邻的变换系数块之间存在着很强的相关性；另外，变换系数经过 zig-zag[87]扫描以后，高频位置的系数取值经常为 0，1 或者-1，并且变换系数的幅值按照扫描顺序呈下降趋势。考虑到变换系数块的这些统计特性，CAVLC 定义了一系列的语法元素来描述一个变换系数块，如表 2-2 所示。

表 2-2 H.264/AVC 中 CAVLC 所定义的语法元素

Table 2-2 Syntax elements in CAVLC within H.264/AVC

编码元素	定义
N	一个 DCT 系数块中非零系数个数
L_i	非零系数，$i=0,1,\cdots,N-1$，按照 zig-zag 扫描顺序索引
T	非零系数序列 $L_0,L_1,\cdots,L_{N-1}$ 末尾幅值为 ±1 的非零系数个数
Z	沿扫描路径最后一个非零系数前零系数的个数
R_i	沿扫描路径 L_i 前连续零系数个数，$i=0,1,\cdots,N-1$，$R_i\geq0$

在编码过程中，CAVLC 对 N 和 T 采用联合编码方式，即先用一个符号 *coeff_token* 来表示（N,T），然后对 *coeff_token* 分配一个合适的码字。CAVLC 为 *coeff_token* 定义了 4 个码表，在编码时，根据上下文 $\hat{N}$ 来自适应地选择码表。其中，$\hat{N}$ 被定义为 $\hat{N}=(N_U+N_L)/2$，N_U 和 N_L 分别表示当前变换系数块上侧及左侧变换系数块的非零系数个数。

编码结束（N,T）以后，CAVLC 开始编码非零系数的幅值 L_i。由于非零系数的幅值沿着 zig-zag 扫描顺序是呈递减趋势。因此，CAVLC 按照逆向 zig-zag 扫描方式来编码非零系数序列，即先编码 L_{N-1}，最后编码 L_0。在编码 L_i 时，其上下文状态 $C(L_i)$ 计算过程可以迭代地描述为

$$C(L_i)=\begin{cases}C(L_{i+1})+1, & \text{if } \mathrm{abs}(L_{i+1})>Th[C(L_i)]\\ C(L_{i+1}), & \text{otherwise}\end{cases} \qquad (2\text{-}1)$$

其中 $C(L_{i+1})$ 表示编码 L_{i+1} 时的上下文状态；$C(L_{N-1})$ 为 0；Th 是一个包含 7 个元素的阈值数组，即 $Th[0..6]=\langle0,3,6,12,24,48,\infty\rangle$。最后 CAVLC 采用哥伦布莱斯码（Golomb-Rice）来生成码字，其阶数记为 $C(L_i)$。

接着 CAVLC 开始编码 Z。在 CAVLC 中，编码 Z 的码表共有 15 个，码表的选择由 N 来决定。最后，CAVLC 开始编码 R_i 序列。R_i 序列的编码也是逆序的，即先编码 R_{N-1}，最后编码 R_0。编码 R_i 时的上下文状态 $C(R_i)$ 可由公式（2-2）来计算，其中 $C(\bullet)=0$ 表示已经编码完所有的零系数，不再需要对

后面的R_i进行编码，因此$C(R_i)$的取值范围是 1~7，分别对应着相应的 7 个变长码表。

$$C(R_i)=\begin{cases}7, & if\ (Z-\sum_{j=i+1}^{N-1}R_j)>6\\ Z-\sum_{j=i+1}^{N-1}R_j, & if\ (Z-\sum_{j=i+1}^{N-1}R_j)\le 6\end{cases} \quad (2\text{-}2)$$

（2）上下文自适应的二进制算术编码 CABAC

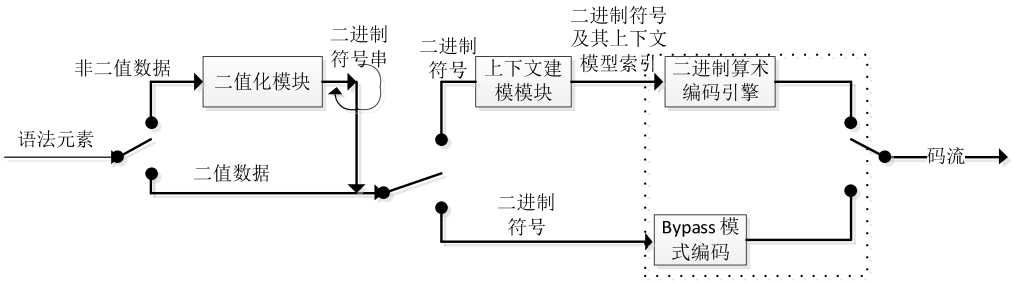


图 2-1 CABAC 编码流程示意图

Fig 2-1 Illustration of encoding procedure in CABAC

CABAC 由三个功能模块组成：二值化模块，上下文建模模块和二进制算术编码引擎，如图 2-1 所示。由于二进制算术编码引擎所能处理的信源符号只能是 0 或者 1，所以二值化模块的主要作用是把非二值的数据转化为二进制序列，以方便二进制算术编码引擎的处理；上下文建模模块的主要作用是根据上下文信息为二进制符号选择合适的上下文模型；二进制算术编码引擎的作用则是利用选中的上下文模型对当前二进制符号进行编码并产生码流。接下来，本文将简要介绍 CABAC 中的这三个功能模块。

二值化模块：为了降低需要编码的二进制符号的个数，二值化过程需要根据语法元素的统计特性来选择相应的二值化方案。因此 CABAC 规定了 5 种二值化方案：一元码（Unary Code），截断一元码（Truncated Unary Code），*k* 阶指数哥伦布码，定长码和 Huffman 码。为了高效地对变换系数和运动矢量残差进行二值化，CABAC 把截断一元码和 *k* 阶指数哥伦布码（UEG*k*）联合在一起作为它们的二值化方案。具体来说，CABAC 把截断一元码和 0 阶指数哥伦布码（UEG0）联合作为变换系数的二值化方法，把截断一元码和 3 阶指数哥伦布码（UEG3）联合作为运动矢量残差的二值化方法。

上下文建模模块：上下文建模模块一共定义了 399 个上下文模型。每个

上下文模型都可以使用二进制算术编码引擎中定义的有限状态机（FSM：Finite State Machine）来自适应地更新概率[88]。除此之外，上下文建模模块还采用了自适应的上下文模型选择算法为当前的语法元素选择一个合适的上下文模型。这种自适应的概率更新和上下文模型选择算法可以更加充分地挖掘语法元素之间的统计相关性。比如，在 CABAC 中，用于编码 Intra 帧的宏块类型 *mb_type* 的上下文模型一共有 8 个，即从标号为 3 的上下文模型到标号为 10 的上下文模型；当编码某个宏块的 *mb_type* 时，上下文建模模块首先根据当前宏块的上侧和左侧宏块的 *mb_type* 的取值，来从该上下文模型集合中选择一个合适的上下文模型，假设其编号为 6，然后二进制算术编码引擎就用标号为 6 的上下文模型中存放的概率来对当前宏块的 *mb_type* 进行编码，之后二进制算术编码引擎会根据当前宏块的 *mb_type* 的取值来更新标号为 6 的上下文模型中所存放的概率。

二进制算术编码引擎：CABAC 采用的二进制算术编码引擎是 M-coder。在 M-coder[30, 90]中，二进制符号的概率是用二元组 (p_{LPS}, V_{MPS}) 来表示的，其中 p_{LPS} 表示 LPS（Least Probable Symbol）的概率，V_{MPS} 表示 MPS（Most Probable Symbol）的取值；而且 M-coder 还采用了文献[27]中的归一化方法实时地向码流中输出比特，这样就可以保证编码区间的长度 R 一直处在一个合理的范围之内，即 $R_{\min} \le R < R_{\max}$。为了删除归一化过程的条件判断，同时避免按位调用归一化过程，文献[140]提出了一个快速归一化方案，即通过查表和缓存比特的方式来降低归一化过程的执行时间。

为了避免在编码区间分割过程中引入乘法操作，M-coder 把区间 $[R_{\min}, R_{\max})$ 均匀地分成 4 子区间，并且利用公式（2-3）把 p_{LPS} 的取值范围 $(0, 0.5]$ 离散化为一个集合 $S = \{p_0, p_1, ..., p_{63}\}$。

$$p_s = \alpha \cdot p_{s-1}, \quad for \ all \ \ s = 1, 2, ..., 63 \tag{2-3}$$

其中 s 表示集合 S 中的每个概率的索引，$p_0 = 0.5$，$\alpha = (\frac{0.01875}{0.5})^{1/63}$。这样设计之后，M-coder 就可以通过查表操作来实现编码区间的分割，即

$$R_{LPS} = R \cdot p_{LPS} \approx TabLPSRange[s][\Delta] \tag{2-4}$$

其中 Δ 表示 R 所在子区间的索引，$\Delta \in \{0, 1, 2, 3\}$。当编码完一个二进制符号之后，M-coder 还定义了一个概率更新模块来自适应地根据已经编码的二进制符号估计信源的概率分布。这个概率更新模块是基于文献[88]的更新规则采用有限状态机 FSM 的方式实现的，即根据时间 t 时 LPS 的概率取值 p_{LPS}^t 以及

此时编码的是 LPS 或 MPS，来决定时间 *t+1* 时的 LPS 的概率取值 p_{LPS}^{t+1}，如公式（2-5）所示

$$p_{LPS}^{t+1}=\begin{cases}\alpha\cdot p_{LPS}^{t}, & \text{if an MPS is occurced}\\ \alpha\cdot p_{LPS}^{t}+(1-\alpha), & \text{otherwise}\end{cases} \tag{2-5}$$

由于 p_{LPS} 的取值范围 $(0,0.5]$ 已经被离散化为一个集合 $S=\{p_0,p_1,...,p_{63}\}$，假设 p_{LPS}^{t} 对应着集合 S 中的元素 p_s，那么如果当前编码的是 MPS，则 p_{LPS}^{t+1} 就对应着 p_{s+1}；如果当前编码的是 LPS，则 p_{s+1} 就对应着与 $\alpha\cdot p_s+(1-\alpha)$ 最接近的 S 中的元素。因此公式（2-5）的概率更新过程可以通过查表 *TransStateLPS[s]* 和 *TransStateMPS[s]* 来实现。

与图像压缩标准 JBIG 和 JPEG 中的 QM-coder[89]以及 JBIG2 和 JPEG2000 中的 MQ-coder[89]相比，M-coder 在编码性能和计算复杂度两个方面都取得最好的性能。具体来说，M-coder 在数据吞吐率方面比 MQ-coder 提高了 5%~18%，在编码性能上比 MQ-coder 提高了 2%~4%。在视频编码中，M-coder 与采用乘法和除法操作的算术编码器可以取得相似的编码性能。尽管如此，与 VLC 相比，M-coder 的计算复杂度还是增加了很多，这主要是因为在调用 M-coder 之前 CABAC 需要调用上下文模型选择算法为待编码的二进制符号选择一个合适的上下文模型，并且 M-coder 自身还要调用概率更新模块来自适应地更新当前上下文模型所存储的概率。所以为了进一步降低 CABAC 的计算复杂度，CABAC 引入了 bypass 编码模式的算术编码方法，即概率等于 0.5 的算术编码方法。由于概率等于 0.5，bypass 编码模式的算术编码只需移位操作即可完成，也就是说使用 bypass 编码模式编码一个二进制符号时，只需要向码流中输出一个比特即可。在编码性能方面，与 UVLC 相比，CABAC 可以进一步降低 9%~14%的码率。

（3）UVLC 和 CABAC 的改进

在 CABAC 中，上下文建模模块和 M-coder 中的概率更新模块对编码性能有着很重要的影响。因此，为了进一步提升 CABAC 的编码性能，有学者对这两个模块进行了深入的研究。

考虑到计算复杂度的问题，CABAC 中的上下文建模模块仅仅使用了当前宏块的上侧和左侧宏块的信息作为上下文的来源。为了提升上下文建模模块的编码性能，文献[92]使用更多的相邻宏块的编码信息作为当前宏块的上下文来源，并且采用 GRASP（Growing，Reordering and Selection by Pruning）算法估计当前宏块中待编码符号的概率。在 GRASP 中，相邻宏块的编码信

息被插入到二叉上下文树中，该二叉上下文树通过收集每个节点的概率分布进而计算出给定上下文下的码字长度，最后 GRASP 选择码字长度最短的上下文最为最终的上下文。与 CABAC 相比，GRASP 可以提升 3%的编码效率，但是同时也大幅度地增加了计算复杂度。基于类似的想法，在编码运动矢量的垂直分量时，文献[93]就把水平分量的取值作为上下文来指导垂直分量的编码；在编码变换系数时，文献[94]把相邻变换块中的对应位置处的变换系数作为上下文用于指导当前变换系数的编码。

为了提高算术编码引擎中的概率更新模块的编码性能，文献[95]提出了基于虚拟滑动窗（VSW：Virtual Sliding Window）的概率更新方法，该方法采用不同长度的滑动窗口对具有不同统计特点的信源实施概率估计。不同长度的窗口对应着不同的概率更新速率（在刚开始编码时，以较大的速率进行更新，从而可以快速地达到稳定的概率；当概率已经稳定时，则以较小的速率进行更新，来抑制噪声的影响），从而提高了概率更新模块的编码性能。基于相似的思想，文献[96]提出了多参数的概率估计方案，该方案用两个不同的速率对概率进行更新，然后把两种速率更新得到的概率均值作为最终的概率。文献[97]则是采用加权上下文树（CTW：Context Tree Weighting）方法实施概率更新。

由于在设计 H.264/AVC 的熵编码时没有过多的考虑数据吞吐率的问题，因此其中的熵编码模块逐渐成为整个解码器的瓶颈模块。为了提高熵编码模块的数据吞吐率使之适合于高清视频的实时编解码，人们提出了很多的改进方案。文献[98,99]在硬件结构设计中考虑了如何提高 CAVLC 的数据吞吐率。文献[100]提出了一种名为 NBAC（N bins per cycle Binary Arithmetic Coding）的算术编码引擎。在编码/解码过程中，NBAC 会保存每一个 bin 的概率并且为当前要编码的 bin 序列构造一个概率分布表，因此，NBAC 可以同时对多个 bin 进行编码。文献[101]引入了一种名为 PIPE（Probability Interval Partitioning Entropy）的熵编码技术。该技术用一个概率集合来表示一个固定的概率区间，每一个 bin 的概率都可以使用该集合中的一个元素来表示，因此 PIPE 采用一种类似于 Huffman 编码的思想，把多个 bin 映射为一个码字，实现了同时对多个 bin 进行编码。为了能够在多核处理器平台上采用多线程技术并行地对多个 bin 进行编码，文献[102]提出了一种语法元素分割技术，把要编码的语法元素分为三组，并且为每一组分配一个进程。由于组与组之间没有相关性，因此这三个进程可以并行地对其组内的编码元素进行编码。利用语法元素分割的思想，文献[103]针对 CABAC 的解码过程提出了一个多

线程并行处理方案。

2.3 AVS-P2 中熵编码技术

AVS-P2 也支持两种熵编码方案，一种是低复杂度的二维变长编码 2DVLC[36]（Two-dimensional Variable Length Coding），另一种是编码性能更高的增强算术编码 EAC[41]（Enhanced Arithmetic Coding）。

（1）二维变长编码 2DVLC

2DVLC 采用定长码或指数哥伦布码来编码所有的语法元素。2DVLC 根据语法元素的概率分布来从中选择一个编码方案来生成码字。对于变换系数，2DVLC 采用基于上下文的二维变长编码方案 C2DVLC（Context-based 2D-VLC）来生成码字。

C2DVLC 首先利用 zig-zag 扫描把二维的变换系数块转化为一维的变换系数数组，然后把这个一维数组转化为 $(Run, Level)$ 二元组序列，其中 Run 表示两个非零系数之间零系数的个数，$Level$ 表示非零系数的幅值。接下来，C2DVLC 将对 Run 和 $Level$ 进行联合编码以利用它们之间的相关性。C2DVLC 为 $(Run, Level)$ 二元组序列一共定义了 5 个码表，每个码表对应着 $(Run, Level)$ 的一个典型概率分布。为了在编码过程中实现码表的自适应切换，C2DVLC 利用已编码的 $Level$ 的最大值来决定当前上下文状态以及对应的码表。每个码表存放的是指数哥伦布码的阶数 k，最后 C2DVLC 利用阶数为 k 的指数哥伦布码来生成最终的码字。

（2）增强的算术编码 EAC

EAC 的编码框架与 CABAC 类似，也包含三个功能模块：二值化模块，上下文建模模块和算术编码模块。

EAC 中的二值化模块包含四种二值化方案，它们分别是一元码（Unary Code），截断一元码（Truncated Unary Code），k 阶指数哥伦布码和定长码。对于每个非二值的语法元素，EAC 都会根据给定语法元素的概率分布从这些二值化方案中选择一种对其进行二值化。

EAC 中的上下文建模模块为每个二进制符号定义了一个或者多个上下文模型。EAC 采用自适应的上下文模型选择算法为给定语法元素选择合适的上下文模型。对于某些语法元素如运动矢量残差，它们的上下文模型是根据相邻块的语法元素取值来决定的；而对于其他的语法元素如宏块分割类型，它们的上下文模型是根据已经编码的该语法元素的二进制符号来确定的。对

于变换系数，EAC 采用了基于上下文的二进制算术编码 CBAC（Context-based Binary Arithmetic Coding）方法，该方法的技术特点包括，第一：它的编码语法元素与 C2DVLC 一样都是 $(Run, Level)$ 二元组序列；第二：它采用了一元码对 $(Run, Level)$ 进行二值化；第三：它根据已经编码的所有 Level 的最大值来设计上下文模型；最后，它采用了上下文加权技术进一步提高上下文模型的准确性。

EAC 中的二进制算术编码引擎是基于对数域的二进制算术编码引擎[104]。与 M-coder 不同，EAC 中的二进制算术编码引擎为了避免乘法和除法操作，把运算域从实数域转化为对数域，因为实数域中的乘法和除法操作在对数域可以通过加法和减法来完成。在 EAC 中，二进制算术编码引擎保存的是 MPS 概率的对数 LG_p_{MPS}，因此其中的编码区间分割可以通过公式（2-6）来实现，

$$\begin{cases} LG\_R = LG\_R + LG\_p_{MPS}, & if\ \ MPS\ \ is\ occured \\ rLPS = R_1 - R_2, & otherwise \end{cases} \tag{2-6}$$

其中 R 表示与 MPS 对应的编码区间长度，LG_R 是 R 的对数，$rLPS$ 是与 LPS 对应的编码区间长度，R_1 是编码 LPS 之前与 MPS 对应的编码区间，R_2 是编码 LPS 之后的与 MPS 对应的编码区间。根据公式（2-6）可知，在编码 LPS 时，需要把编码区间从对数域转换到实数域，即 $LG\_R \to R$。为了解决这个问题，EAC 采用了公式（2-7）来实现对数域到实数域的转换。

$$R = 2^{LG\_R} = 2^{s+t} = 2^s \cdot 2^t \approx 2^s \cdot (1+t) \tag{2-7}$$

其中 s 和 t 分别表示 LG_R 的整数部分和小数部分，公式（2-7）的推导过程采用了公式（2-8）中的近似关系，

$$2^x \approx 1+x,\ \ 0 \le x \le 1 \tag{2-8}$$

由于 $R_1 > R_2 > 0.5 \cdot R_1$，因此 $LG\_R_1 > LG\_R_2 > LG\_R_1 - 1$，所以 $s_1 = s_2$ 或者 $s_1 = s_2 + 1$。所以，公式（2-6）中的 $rLPS$ 可以用公式（2-9）来计算得到，

$$rLPS = R_1 - R_2 = \begin{cases} t_1 - t_2, & if\ \ s_1 == s_2 \\ (t_1 << 1) - t_2, & otherwise \end{cases} \tag{2-9}$$

另外 EAC 采用如公式（2-10）所示的概率更新方式来更新 LG_p_{MPS}，

$$\begin{cases} LG\_p_{MPS} = LG\_p_{MPS} + LG\_f, & if\ \ LPS\ \ is\ occured \\ LG\_p_{MPS} = LG\_p_{MPS} - (LG\_p_{MPS} >> cw), & otherwise \end{cases} \tag{2-10}$$

其中 $f = {(2^{cw} - 1)}/{2^{cw}}$，$cw$ 是一个可调参数，来控制概率更新的速度。

2.4 HEVC 中的熵编码技术

由于在 H.264/AVC 视频编码标准制定的过程中，视频编码技术的数据吞吐率并没有引起人们的关注，而且文献[100~103]所提出的并行处理技术都是在标准确定后才提出来的，因此这些并行处理技术和 H.264/AVC 中的 CABAC 是无法兼容的，即采用上述视频编码技术压缩得到的码流不能被 H.264/AVC 的 CABAC 解码。因此在 HEVC 的制定过程中，CABAC 的并行处理技术被充分的考虑进来。

考虑到编解码器的硬件实现复杂度，HEVC 仅仅定义了一个熵编码方案即 CABAC。它的编码框架与 H.264/AVC 中的 CABAC 是一样的，如图 2-1 所示。它们之间的区别是 HEVC 中的 CABAC 大量地采用了并行处理技术来提高 CABAC 的数据吞吐率。这些并行处理技术大致可以分为以下几类。

1） 减少使用上下文模型编码的二进制符号的个数：CABAC 之所以能够取得很高的压缩效率，主要是由于其中的上下文建模模块可以精确地估计出每一个二进制符号的概率分布。CABAC 中的上下文建模模块为不同类型的二进制符号定义了不同的上下文模型。在编码的过程中，每一个上下文模型要使用 M-coder 中定义的有限状态机来自适应地更新概率，即根据当前二进制符号的取值更新当前上下文模型中的概率取值，为编码下一个二进制符号做准备。另外，对于每个使用上下文模型的二进制符号，上下文建模模块还需要调用上下文选择算法为其选择合适的上下文模型。由此可见，使用上下文模型编码的二进制符号之间存在着很强的上下文依赖关系，所以减少这种类型的二进制符号可以提高 CABAC 的数据吞吐率[105~107]。

2） 把采用 bypass 编码模式的二进制符号组织在一起：由于编码一个采用 bypass 编码模式的二进制符号仅仅需要移位操作即可完成，即只需向码流中输出一个比特。因此把这种类型的二进制符号组织在一起编码可以提高 CABAC 的数据吞吐率[108~110]。比如，只需向码流中输出 n 个比特就完成了对 n 个使用 bypass 编码模式的二进制符号的编码。

3） 把使用同一个上下文模型的二进制符号组织在一起：对于每个使用上下文模型的二进制符号，CABAC 中的上下文建模模块需要调用上下文模型选择算法为其选择合适的上下文模型。如果相邻的二进制符号使用的是不同上下文模型，那么上下文建模模块需要考虑所有的上下文模型组合情况才能为每个二进制符号选择出正确的上下文模型，并

且这些组合情况会随着并行度的增加而呈指数形式增长。因此，把使用相同上下文模型的二进制符号组织在一起会降低上下文模型的组合数，在并行编码多个二进制符号的时候，会使得编解码器使用较少的推理计算，所以提高了 CABAC 的数据吞吐率[111,112]。

4） 降低上下文模型选择算法中的依赖关系：上下文模型选择算法是根据已经编码的语法元素来为待编码的语法元素选择合适上下文模型。因此，降低上下文模型选择算法中的依赖关系可以在并行处理中降低编解码器的推理计算，进而提高 CABAC 的数据吞吐率[113,114]。

5） 降低要编码的二进制符号：最直接的提高熵编码的数据吞吐率的技术就是降低 CABAC 要编码的二进制符号的总数。为了达到这种效果，可以引入高层语法等技术[115]。

6） 降低内存消耗：访问内存也经常会增加关键路径的延迟，进而导致数据吞吐率的降低。所以，降低内存消耗也会增加 CABAC 的数据吞吐率[116,117]。

为了进一步提高 HEVC 中熵编码模块的编码效率，文献[118]对 HEVC 中的变换系数的上下文建模模块做了修改。在文献[118]中，当前变换数的局部模板内的变换系数信息和当前变换系数的频域位置被作为上下文来源来对变换系数进行上下文建模。这种上下文建模思想最早出现于文献[119]中，但是与文献[119]不同的是，文献[118]只采用当前扫描过程中已经编码的二进制符号作为上下文，也就是说在当前扫描过程中编码的二进制符号不会参考先前扫描过程中编码的二进制符号，这样可以打破不同扫描过程之间的数据依赖关系，提高熵编码的数据吞吐率。为了挖掘不同尺寸的变换系数块的统计特性，文献[118]把变换系数块的大小作为一个额外的上下文，也就是说不同尺寸的变换系数块使用不同的上下文模型集合。

2.5 本章小结

本章按照视频编码标准的发展历程对熵编码技术的研究现状进行了系统的分析与总结。首先介绍了早期视频压缩标准中的熵编码技术；然后着重介绍 H.264/AVC 视频编码标准中的熵编码技术及其改进方案；接着介绍了国内视频压缩标准 AVS-P2 中的熵编码技术；最后总结了 HEVC 中的熵编码技术与 H.264/AVC 中的熵编码技术的区别。

第3章 基于层次依赖上下文模型的算术编码器

由于在视频编码标准 H.264/AVC 的制定过程中，没有充分考虑 CABAC 的吞吐率，使得 CABAC 成为 H.264/AVC 解码器的主要瓶颈之一，严重地制约着 H.264/AVC 解码器的吞吐率的提高。提高 CABAC 的并行性是提高其吞吐率的有效方法。在编码变换系数时，CABAC 利用已经编码的变换系数来估计当前变换系数的概率分布，这样的设计在变换系数之间引入了上下文依赖关系，不利于 CABAC 的并行处理。本章的研究目标是，首先为变换系数寻找一些合理的上下文，它们既能准确地预测变换系数的统计特性，又能避免在变换系数之间引入上下文依赖关系；然后利用这些上下文设计高效的上下文模型，进而指导算术编码引擎的编码。因此，本章通过分析变换系数的统计特性提出了一个层次依赖的上下文模型 HDCM (Hierarchical Dependency Context Model)，然后设计了一个基于层次依赖上下文模型的二进制算术编码器 HDCMBAC 来编码 H.264/AVC 中预测残差的 DCT 系数。HDCM 在充分挖掘变换系数的统计特性的同时，可以使得上下文建模模块和算术编码引擎并行执行，从而在保证压缩效率的同时提高解码吞吐率。

3.1 H.264/AVC 中变换系数的上下文建模过程

下面将介绍 H.264/AVC 中变换系数的上下建模过程。从中可以看到，变换系数的上下文模型选择算法中存在着上下文依赖关系，使得上下文建模模块与算术编码引擎只能串行处理。

表 3-1 H.264/AVC 中 CABAC 编码变换系数所使用的语法元素

Table 3-1 Syntax elements used in CABAC within H.264/AVC

编码元素	定义
significant_map	指示当前扫描位置上的变换系数是否为非零系数
last_significant_map	指示当前非零变换系数是否是当前变换系数块中的最后一个非零系数
coeff_abs_level_minus1	表示当前非零变换系数的绝对值减 1
coeff_sign_flag	表示当前非零变换系数的符号

CABAC 设计了四个语法元素来编码变换系数，如表 3-1 所示。在解码过程中，CABAC 首先解码当前位置的 *significant_map*，如果 *significant_map*

等于 1 表示当前位置处存在非零变换系数，否则表示该位置不存在非零变换系数。如果 *significant_map* 等于 1，则需要继续解码 *last_significant_map* 来判断当前位置处的非零系数是否是变换系数块中的最后一个非零系数，如果 *last_significant_map* 等于 1，则表示当前位置处的非零变换系数是最后一个非零变换系数，否则表示当前位置处的非零变换系数不是最后一个非零变换系数。如果 *significant_map* 等于 0，则处理下一个扫描位置。解码完所有扫描位置的 *significant_map* 和 *last_significant_map* 之后，如果某一扫描位置处的 *significant_map* 等于 1，则继续解码该位置处非零变换系数的幅值 *coeff_abs_level_minus1* 和符号 *coeff_sign_flag*。

对于 *significant_map* 和 *last_significant_map*，CABAC 的上下文建模模块把变换系数的扫描位置作为它们的上下文，它们的上下文模型选择过程描述如下，

$$\chi_{SIG}(coeff[i]) = \chi_{LAST}(coeff[i]) = i \qquad (3\text{-}1)$$

其中 $coeff[i]$ 表示扫描位置 i 处的变换系数，$\chi_{SIG}(\bullet)$ 表示 *significant_map* 的上下文模型的索引，$\chi_{LAST}(\bullet)$ 表示 *last_significant_map* 的上下文模型的索引。在变换系数的解码过程中，任何非零变换系数的 *significant_map* 和 *last_significant_map* 的解码过程都是交叉执行的，即只有当 *significant_map* 解码结束后，解码器才能判断是否需要解码 *last_significant_map*，如果需要解码，则执行 *last_significant_map* 的上下文建模和熵编码过程，否则解码下一个扫描位置的 *significant_map*。

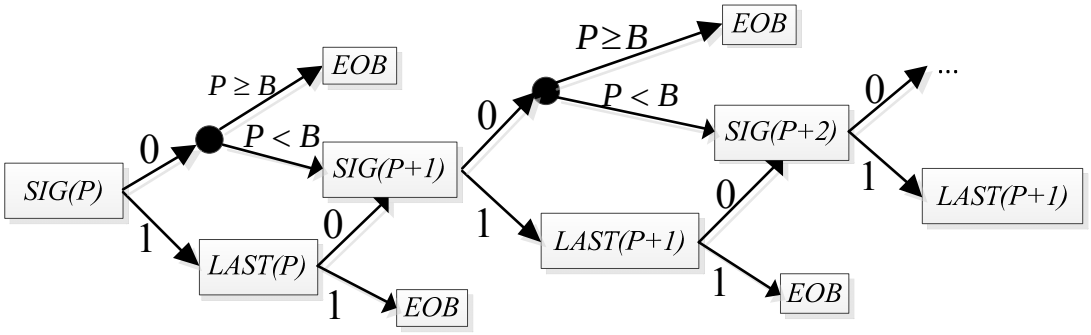


图 3-1 并行解码 *significant_map* 和 *last_significant_map* 时推导计算的示意图

Fig 3-1 Speculative computations required for decoding *significant_map* and *last_significant_map* in parallel

由于 *significant_map* 和 *last_significant_map* 采用的是不同的上下文模型

集合，这种交叉解码非常不利于它们的并行处理。如果期望并行解码 *significant_map* 和 *last_significant_map*，则需要利用推导计算（speculative computation）预先计算解码过程中可能使用的上下文模型索引值。如图 3-1 所示，为了并行解码位于 *P*，*P*+1 和 *P*+2 处的 *significant_map* 和 *last_significant_map*，则需要首先采用推导计算得到图 3-1 中的每个节点的上下文模型索引。

对于 *coeff_abs_level_minus1*，CABAC 采用已经编码的幅值等于 1 的变换系数的个数 $NumT1$ 和幅值大于 1 的变换系数的个数 $NumLgt1$ 作为上下文。由于它不是二值数据，因此 CABAC 首先调用二值化方案 UEG0 来把它转化为二进制符号序列。对于其中的第一个二进制符号，即索引为 0 的二进制符号(本文采用 *bin0* 表示)，它的上下文模型选择算法如公式（3-2）所示；对于索引取值落在区间 $[1,13]$ 内的二进制符号（本文采用 *bin13* 来表示），它们的上下文模型选择算法如公式（3-3）所示；对于剩余的二进制符号，CABAC 采用 bypass 编码模式来对它们进行编码。

$$\chi_{AbsCoeff}(i,bin0)=\begin{cases}4, if\ \ NumLgt1(i)>0\\ \max(3,NumT1(i)), otherwise\end{cases} \qquad (3\text{-}2)$$

$$\chi_{AbsCoeff}(i,bin13)=5+\max(4,NumLgt1(i)) \qquad (3\text{-}3)$$

其中 *coeff_abs_level_minus1* 是按照扫描顺序的逆序来进行编码的，在刚开始编码非零系数时，$NumT1$ 和 $NumLgt1$ 都被初始化为 0。在编码过程中，如果 *bin0* 等于 1，则表示该变换系数是大于 1，否则表示该变换系数是等于 1。

从公式（3-2）和（3-3）可以发现，如果先解码所有的 *bin0*，则在 *bin13* 的上下文模型选择过程中，就不会存在上下文依赖关系，这是因为当所有的 *bin0* 被解码后，就可以计算出 *NumLgt1(i)*。但是，在 *bin0* 的上下文模型选择过程中，当前 *bin0* 的上下文模型依赖于已经编码过的 *bin0* 的取值，所以 *bin0* 之间存在上下文依赖关系，不利于上下文建模模块和熵编码过程的并行执行。如果期望并行解码 *bin0*，则同样需要使用推导计算预先计算解码过程中可能使用的上下文模型的索引值。图 3-2 给出了并行解码 *bin0* 时所需要的推导计算示意图，其中 $C_{L_{bin0}^{P\_i}}$ 表示位于 P_i 处的变换系数 $coeff[P\_i]$ 所使用的上下文模型的索引值。从图 3-2 可见，为了并行解码位于 *P_0*，*P_1*, *P_2* 和 *P_3* 处的 *bin0*，则需要首先采用推导计算得到图 3-2 中的每个节点的上下文模型索引。

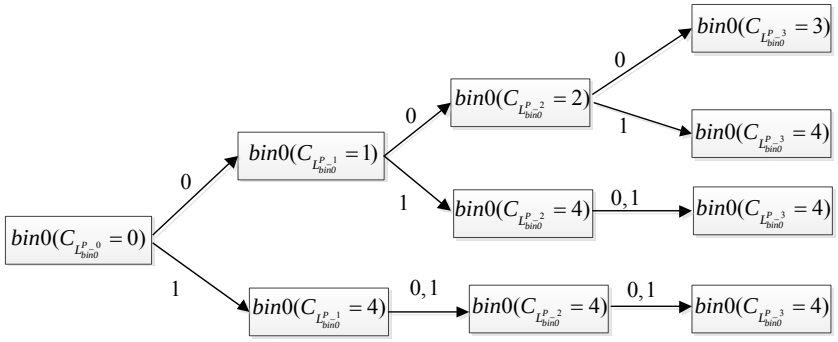


图 3-2 并行解码 *bin0* 时推导计算的示意图

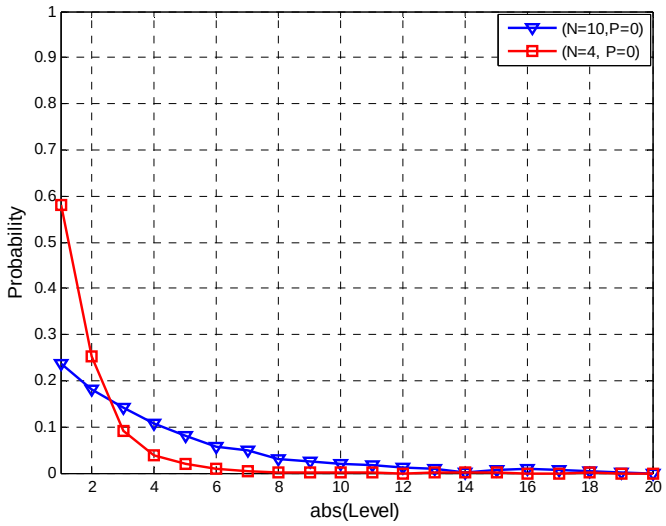
Fig 3-2 Speculative computation required for decoding *bin0s* in parallel.

3.2 层次依赖上下文模型

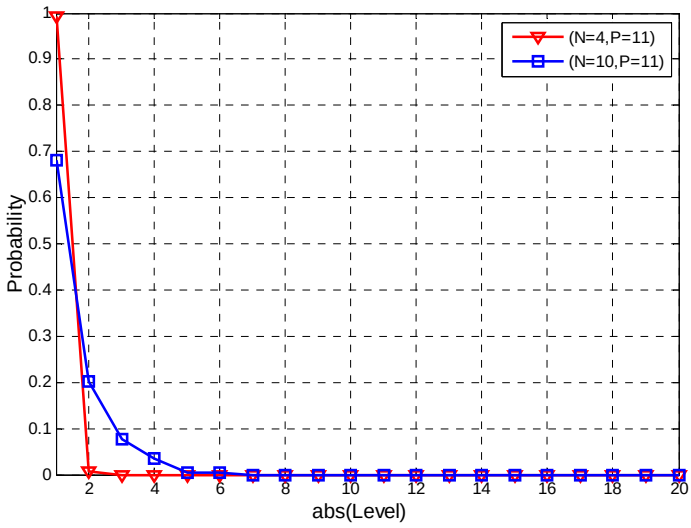
为了打破 CAVLC 中存在的上下文依赖关系，王强博士针对变长编码首次提出了 DCT 系数的层次依赖上下文模型[38]来捕捉语法元素 *Level* 和 *Run* 的统计特性，其中 *Level* 表示非零 DCT 系数的幅值，*Run* 表示两个相邻的非零 DCT 系数之间的幅值为零的变换系数的个数。在这个上下文模型中，*Level* 的上下文包括待编码的 DCT 系数的频域位置和当前 DCT 块中非零系数的个数；*Run* 的上下文包括当前待编码的 *Run* 沿 zig-zag 扫描路径的出现次序和当前 DCT 块中非零系数的个数；由于 DCT 系数的频域位置需要根据先前已经编码的 *Run* 才能计算得到，因此 *Level* 的上下文建模过程依赖于先前已经编码的 *Run*，但是由于 *Run* 和 *Level* 分别处在不同的数据层次上，所以这种上下文依赖关系并不影响上下文建模过程和变长编码过程的并行执行，从而使得这两个模块能够并行处理。最后，哥伦布莱斯码被用于编码 *Level* 和 *Run*，其阶数由当前待编码语法元素的上下文来确定。

为了打破 CABAC 中变换系数之间的上下文依赖关系，本文针对变换系数的算术编码提出了层次依赖上下文模型。该上下文模型不但能够挖掘变换系数的统计特性，还能避免引入上下文依赖关系。根据文献[120]可知，位于不同扫描位置的变换系数具有不同的概率分布。因此，扫描位置 P 可以被用作一个上下文来挖掘变换系数的统计特性，例如 H.264/AVC 就把扫描位置 P 作为 *significant_map* 的上下文。由于预测残差的多样性，扫描位置 P 并不能非常准确地反映变换系数的取值概率分布，例如平滑区域中某一位置的变换系数和复杂纹理区域中同一位置的变换系数的概率分布是不同的。为了进一步区分具有不同统计特性的概率分布，本文把变换系数块中非零系数的个数 N 作为额外的上下文，这是因为变换系数块中非零系数的个数 N 在一定程度

上反映了编码区域的纹理复杂性和残差能量的大小，例如 N 越小表示编码区域越平滑或者残差能量越小，N 越大表示编码区域的纹理越复杂或者残差能量越大。



（a）扫描位置 P 等于 0



（b）扫描位置 P 等于 11

图 3-3 在 Football@CIF 视频上当 N=4 和 N=10 时非零系数取值的概率分布

Fig 3-3 The probability distribution of abs(*Level*) for Football for N = 4 and N = 10

图 3-3 给出了非零系数在 Football@CIF 序列上的概率分布，其中 $N=4$ 和 $N=10$，P 分别为 0 和 11。可以发现，变换系数块中非零系数的个数 N 可以进一步区分同一扫描位置上非零系数取值的概率分布，比如在 P=0 的情况下，当 N=4 时，*bin0* 等于 1 的概率是 0.6；当 N=10 时，*bin0* 等于 1 的概率是 0.23。所以，为了打破 CABAC 中变换系数之间的上下文依赖关系，本文采用 DCT 系数的频域位置 P 和当前 DCT 块中非零系数的个数 N 来作为 *significant_map* 和 *bin0* 的上下文。对于 *significant_map*，它的上下文组织方式如公式（3-4）所示，

$$C_{SIG}(P,N)=P+(N-1)\times B \tag{3-4}$$

其中 *SIG* 表示 *significant_map*，B 是一个常数，表示一个变换系数块的大小，比如对于 4×4 大小的变换系数块，B 的取值为 16。对于 *bin0*，它的上下文组织方式如公式（3-5）所示，

$$C_{bin0}(P,N)=P+(N-1)\times B \tag{3-5}$$

对于 *bin13*，它的上下文以及上下文模型选择过程与 CABAC 一样。这是因为当所有的 *bin0* 被解码后，则能够准确地确定解码 *bin13* 所需要的上下文模型，所以 *bin13* 之间不存在上下文依赖关系。

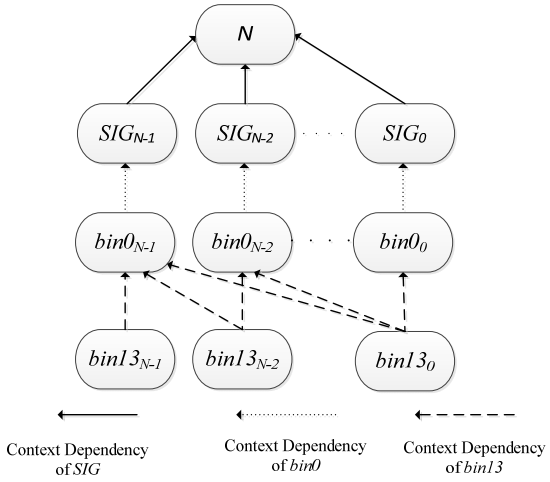


图 3-4 层次依赖上下文模型

Fig 3-4 Hierarchical Dependency Context Model

由于该模型中的上下文依赖关系仅存在于不同类型的语法元素，即 *significant_map* 依赖非零系数个数 N；*bin0* 依赖 *significant_map* 和 N；*bin13* 依赖 *bin0*，如图 3-4 所示。从中可见，该上下文模型中的上下文依赖关系呈层次状，所以称之为层次依赖上下文模型（HDCM：Hierarchical Dependency

Context Model）。由于该层次依赖上下文建模方案打破了同类语法元素之间的上下文依赖关系，所以解码得到上一层语法元素之后，即可准确得到下一层语法元素的上下文模型，从而在并行解码 *significant_map* 或 *bin0* 时不再需要推导计算。因此，该上下文建模方案使得上下文建模模块与算术编码模块的并行执行成为可能。

3.3 基于层次依赖上下文模型的算术编码 HDCMBAC

利用上一节描述的层次依赖上下文模型 HDCM，本文为变换系数设计了一个基于层次依赖上下文模型的算术编码方案 HDCMBAC。接下来，本小节将详细描述 HDCMBAC 的设计流程。

3.3.1 HDCMBAC 编码方案

由于 *significant_map* 和 *last_significant_map* 的交叉解码过程非常不利于 CABAC 的并行处理，因此 HDCMBAC 把变换系数块中的非零系数的个数 N 作为一个语法元素，用于指示最后一个非零系数的位置。因此，HDCMBAC 的语法元素包括 N，*significant_map*，*bin0* 和 *bin13*。由于在这些编码元素之间存在着上下文依赖关系，因此它们的编码顺序是 $N \rightarrow$*significant_flag* $\rightarrow$*bin0* $\rightarrow$*bin13*。即先编码 N；其次编码 *significant_map*；接着编码 *bin0*；最后编码 *bin13*。

对于 N，由于它不是二值数据，所以 HDCMBAC 首先使用一元码把 N 二值化为二进制符号序列。对于每一个二进制符号，它的索引 *binIdx* 和 $\hat{N}$ 被用作上下文，其中 $\hat{N} = (N_U + N_L)/2$，N_U 和 N_L 分别是位于当前变换块上方和左方的变换块中非零系数的个数。*binIdx* 和 $\hat{N}$ 的组织方式如公式（3-6）所示，

$$C_{Nbin}(binIdx, \hat{N}) = binIdx + (\hat{N} - 1) \times B \tag{3-6}$$

根据 N，*significant_map* 和 *bin0* 的上下文组织方式可见，如果按照这种组织方式来设计上下文模型，那么每一个语法元素将会有高达 B^2 个上下文模型，以 4×4 的变换系数块为例，每个语法元素将会有 256 个上下文模型。如此多的上下文模型将会增加上下文建模模块的内存消耗，同时也会产生上下文稀疏问题。为了解决这个问题，本文采用上下文量化方法把具有相同概率分布的上下文模型合并成一个上下文模型。

因此，对于 N 的每个二进制符号，它的上下文模型选择过程如公式（3-7）所描述，

$$C_{Nbin}(binIdx, \hat{N}) = k_N[\Delta_{\hat{N}}][binIdx] \tag{3-7}$$

其中 $k_N[\Delta_{\hat{N}}][binIdx]$ 存放 N 二值化产生的二进制符号的上下文模型索引号，$\Delta_{\hat{N}}$ 的取值为 0，1，2 和 3，分别对应着四个不同的 $\hat{N}$ 区间 $[0,2)$，$[2,4)$，$[4,8)$ 和 $[8,16)$。对于 *significant_map*，它的上下文模型选择过程如公式（3-8）所描述，

$$C_{SIG}(P,N)=k_{SIG}[P][\Delta_N] \tag{3-8}$$

其中 $k_{SIG}[P][\Delta_N]$ 是存放 *significant_map* 的上下文模型索引号的数组，Δ_N 的取值为 0，1，2 和 3，分别对应着四个不同的 N 区间 $[0,3)$，$[4,5)$，$[5,10)$ 和 $[10,16)$。对于 *bin0*，它的上下文模型选择过程如公式（3-9）所描述，

$$C_{bin0}(P,N)=k_{bin0}[P][N] \tag{3-9}$$

其中 $k_{bin0}[P][N]$ 是存放 *bin0* 的上下文模型索引号的数组，它的取值范围是 0，1,2 和 3。 表 3-2 通过一个例子展示了 HDCMBAC 中上下文建模模块的执行过程。在这个例子中，扫描位置 5 到 15 之间的变换系数都为零系数。

表 3-2 HDCMBAC 中上下文建模模过程的示例

Table 3-2 Example for context modeling scheme in HDCMBAC

扫描位置 P	0	1	2	3	4
变换系数	9	0	3	-1	1
significant_map	1	0	1	1	1
C_{SIG}	$k_{SIG}[0][1]$	$k_{SIG}[1][1]$	$k_{SIG}[2][1]$	$k_{SIG}[3][1]$	$k_{SIG}[4][1]$
bin0	1		1	0	0
C_{bin0}	$k_{bin0}[0][4]$		$k_{bin0}[2][4]$	$k_{bin0}[3][4]$	$k_{bin0}[4][4]$

最终，HDCMBAC 采用 M-coder 来编码所有的二进制符号，根据第二章的描述，每个上下文模型需要保存二元组 (p_{LPS},V_{MPS})，其中 p_{LPS} 表示 LPS（least probable symbol）的概率，V_{MPS} 表示 MPS（most probable symbol）的取值。算法 3-1 给出了 HDCMBAC 的解码流程的描述，其中 getCtxforN()，getCtxforSig()，getCtxforBin0() 和 getCtxforBin13() 分别表示对 N，*significant_map*，*bin0* 和 *bin13* 的上下文建模过程，decodebin()表示二进制算术解码过程。

算法 3-1 HDCMBAC 的解码端算法描述

Algorithm 3-1 The description of HDCMBAC decoding scheme

输入：码流 str
输出：N, Significant_map[0…15], Bin0[0…15]和 Bin13[0…15]
初始化：N=1, SigNums=0, NumLgt1[0,..,15]=0, Bin0[0,…,15]=0,
Significant_map[0,...,15]=0, Bin13[0,…,15];

```
For binidx←0 to 16 do
    C_Nbin = getCtxforN(binidx,N̂);
    bin    = decodebin(str, C_Nbin);
    If bin==0 then break;
    N++;
End For
For P←0 to 16 do
    C_Sig = getCtxforSig(P, N);
    Sig = decodebin(str, C_Sig);
    Significant_map[P]=Sig;
    If Significant_map[P]==1 then SigNums++;
    If SigNums==N then break;
End For
For P←15 downto 0 do
    If significant_map[P]==1 then
       C_Bin0 = getCtxforBin0(P, N);
       Bin0[P] = decodebin(str, C_Bin0);
       If Bin0[P]==1 then NumLgt1[P] = NumLgt1[P+1]+1;
    End If
End For
For P←15 downto 0 do
    If Bin0[P]==1 then
       C_Bin13 = getCtxforBin13(NumLgt1[P]);
       Bin13[P] = decodebin(str, C_Bin13);
    End If
End For
```

3.3.2 上下文建模模块和算术编码模块的并行组织方式

对于 N 的每个二进制符号，由于二进制符号的索引被用于上下文模型的选择过程，因此只有当前的二进制符号解码结束后，才能判断 N 的解码过程是否结束。在解码 N 时，为了使上下文建模模块和算术编码模块能并行执行，上下文建模模块一直为 N 的每个二进制符号选择上下文模型并把选中的上下文模型的索引传给算术编码模块。如果解码出的二进制符号的取值指示 N 的解码过程已经结束，则把算术编码的状态恢复到开始状态。图 3-5 给出了 N

的上下文建模模块和算术编码模块并行组织方式，其中 C_{Nbin} 表示 *N* 中每一个二进制符号的上下文模型选择过程；C_{SIG} 表示 *significant_map* 的上下文模型选择过程；*R&D* 表示算术编码过程；*Undo* 表示把算术编码的状态恢复至开始状态。

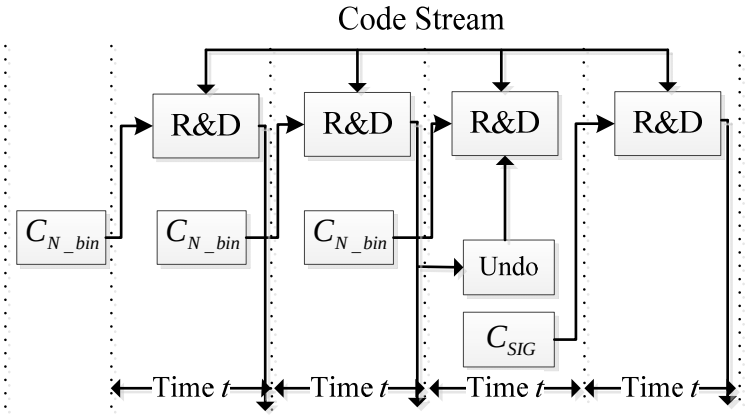


图 3-5 HDCMBAC 编码非零系数 *N* 时上下文建模模块和算术编码模块的并行组织方式

Fig 3-5 Parallel organization between context modeling and binary arithmetic decoding in HDCMBAC

对于 *significant_map*，HDCMBAC 需要根据它的取值来判断 *significant_map* 的解码过程是否结束。为了在编码 *significant_map* 时使上下文建模模块和算术编码模块并行执行，上下文建模模块一直为 *significant_map* 选择上下文模型并把选中的上下文模型的索引传给算术编码模块。如果当前解码的 *significant_map* 的取值表示其解码过程已经结束，则把算术编码的状态恢复到开始状态。图 3-6 给出了解码 *significant_map* 和 *bin0* 以及 *bin13* 时的上下文建模模块和算术编码模块并行执行的组织方式，其中 $C_{L^i_{bin0}}$ 和 $C_{L^i_{bin13}}$ 分别表示对处于扫描位置 *i* 的非零系数的 *bin0* 和 *bin13* 的上下文模型选择过程。

通过图 3-5 和图 3-6 的并行组织方式，HDCMBAC 的上下文建模模块和算术编码模块可以并行执行。例如，当解码器对 *coeff_abs_level_minus1[1]*的 *bin0* 进行算术编码时，可以同时对 *coeff_abs_level_minus1[0]*的 *bin0* 进行上下文建模过程。

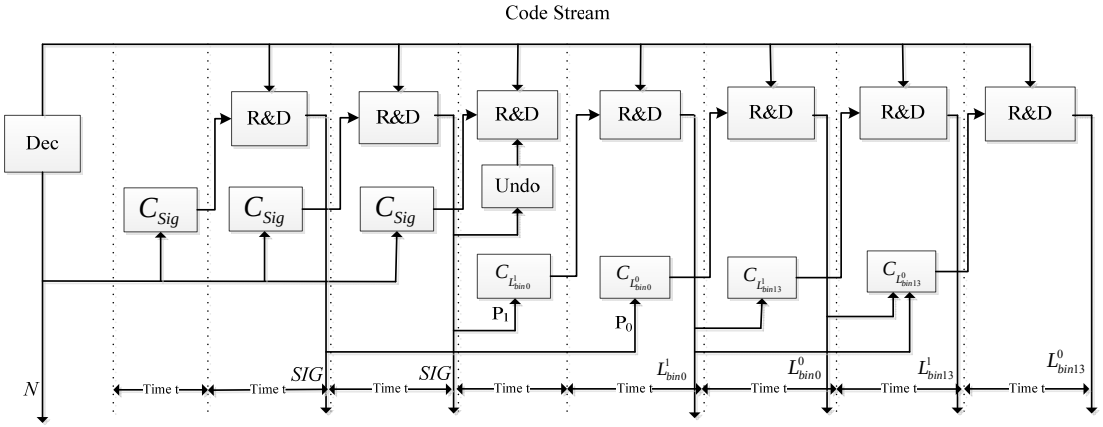


图 3-6 HDCMBAC 解码非零系数时上下文建模模块和算术编码模块的并行执行方式

Fig 3-6 Parallel organization between context modeling and arithmetic decoding in HDCMBAC

3.4 实验结果

本节从编码性能，数据吞吐率，运算操作数以及内存消耗等方面来验证 HDCMBAC 的有效性。本文以 H.264/AVC 的参考软件 JM15.1 为实验平台。测试条件包括，4×4 的 DCT 变换量化块，5 个参考帧，1/4 像素间的预测精度，运动估计的范围是+/-32 像素，量化参数 QP 是 16，20，24，28，32 和 36。测试的序列包括 CIF 格式的 *Mobile*，*Foreman* 和 *Paris*；4CIF 格式的 *City*，*Crew* 和 *Ice* 序列；720p 格式的 *City*，*Bigships* 和 *Cyclist*；分辨率为 1920×1080 的序列 *BasketballDrive*，*BQTerrace* 和 *Cactus*；分辨率为 2560×1600 的序列 *PeopleOnStreet* 和 *Traffic*。

3.4.1 编码效率比较

为了衡量 HDCMBAC 的编码效率，本文在 All Intra（AI），Low Delay（LD）和 Random Access（RA）三个配置文件下对 HDCMBAC 的编码性能做了测试。表 3-3 给出了 HDCMBAC 相对于 CABAC 的 BD-Rate，其中 *LowQP* 表示此时的 BD-Rate 是在 QP 等于 16，20，24 和 28 的条件下计算得到的，*HighQP* 表示此时的 BD-Rate 是在 QP 等于 24，28，32 和 36 的条件下计算出来的。当 BD-Rate 为负值时表示 HDCMBAC 的编码性能比 CABAC 好，当 BD-Rate 为正值时表示 HDCMBAC 的编码性能比 CABAC 差。

表 3-3 HDCMBAC 相对于 CABAC 的 BD-Rate[%]

Table 3-3 BD-Rate[%] of HDCMBAC relative to CABAC

Sequences	BD-Rate[%]					
	Low QP			High QP		
	AI	LD	RA	AI	LD	RA
Mobile@CIF	-0.5	0.2	0.1	-0.2	0.3	0.4
Paris@CIF	-1.2	-0.7	-0.8	-0.7	0.1	0.0
Foreman@CIF	-0.2	-0.1	0.0	0.0	-0.2	0.2
City@4CIF	-0.2	0.4	0.4	-0.1	-0.2	0.1
Crew@4CIF	-0.3	0.2	0.3	-0.5	-0.2	0.0
Ice@4CIF	-1.1	-0.7	-0.5	-0.1	0.0	0.1
City@720p	-0.1	0.4	0.4	0.1	0.0	0.4
Bigships@720p	-0.1	0.7	0.6	0.1	0.2	0.2
Cyclist@720p	0.0	-0.1	0.1	0.0	-0.3	0.1
BasketballDrive@1920x1080	0.2	0.3	0.5	0.0	-0.1	0.0
BQTerrace@1920x1080	-0.5	0.9	0.9	-0.7	0.3	0.2
Cactus@1920x1080	0.0	0.8	0.7	0.0	0.1	-0.1
PeopleOnStreet@2560x1600	-0.2	-0.3	-0.3	-0.1	-0.1	0.0
Traffic@2560x1600	-0.6	-0.5	-0.5	-0.2	0.1	0.0
Average	**-0.343**	**0.107**	**0.136**	**-0.171**	**0.000**	**0.114**

由表 3-3 可知，HDCMBAC 可以取得和 CABAC 相似的编码性能。具体来说，HDCMBAC 在 AI 配置下可以取得比 CABAC 更好的编码性能，在 LD 和 RA 配置下的编码性能比 CABAC 略低；并且低 QP 下的编码增益或编码损失比高 QP 下的大。为了解释其中的原因，本文用互信息 $I(X;Y)$ 来衡量编码元素 X 与其上下文模型 Y 之间的相关性，其中 $I(X;Y)$ 的计算方式如公式（3-10）所示，

$$I(X;Y)=\sum_{y\in Y}\sum_{x\in X}p(x,y)\cdot\log(\frac{p(x,y)}{p(x)\,\mathrm{p}(y)}) \qquad (3\text{-}10)$$

其中 $p(x,y)$ 为随机变量 X 和 Y 联合概率分布，$p(x)$ 为随机变量 X 的概率分布，$p(y)$ 为随机变量 Y 的概率分布。一般来讲，互信息 $I(X;Y)$ 越大说明上下文模型 Y 与编码元素 X 之间的相关性越大，则用 Y 来指导 X 编码就能取得更高的编码性能。因此，本文分别计算了 HDCMBAC 和 CABAC 中的上下文模型与其编码元素之间的互信息，如表 3-4 所示。其中 *LAST* 表示 *last_significant_map* 的互信息，*SIG* 被标示 *significant_map* 的互信息。

表 3-4 HDCMBAC 和 CABAC 中语法元素与其上下文模型的互信息

Table 3-4 Mutual information of syntax elements and their context models in CABAC and HDCMBAC

Sequences	Mutual Information					
	CABAC			HDCMBAC		
	LAST	*SIG*	*Bin0*	*N*	*SIG*	*Bin0*
Mobile@CIF	0.282	0.102	0.278	0.164	0.285	0.287
City@4CIF	0.322	0.127	0.202	0.233	0.257	0.224
Bigships@720p	0.240	0.236	0.181	0.199	0.262	0.187
BQTerrace@1920x1080	0.311	0.067	0.176	0.188	0.208	0.196
PeopleOnStreet@2560x1600	0.157	0.146	0.306	0.202	0.212	0.271
Average	**0.2624**	**0.1356**	**0.2286**	**0.1972**	**0.2448**	**0.2330**

通过表 3-4 可知，CABAC 中 *last_significant_map* 的互信息比 HDCMBAC 中 *N* 的互信息要大；CABAC 中 *significant_map* 的互信息要比 HDCMBAC 中 *significant_map* 的互信息低；CABAC 中 *bin0* 的互信息比 HDCMBAC 中 *bin0* 的互信息差不多。由于在 LD 和 RA 配置下，非零变换系数的个数比较少，因此编码 *significant_map* 节省的比特无法弥补编码 *N* 所多消耗的比特，所以在 LD 和 RA 下 HDCMBAC 比 CABAC 的编码性能略低一些；但是在 AI 配置下，非零系数的个数比较多，因此编码 *significant_map* 节省的比特可以弥补编码 *N* 所多消耗的比特，所以在 AI 下 HDCMBAC 比 CABAC 的编码性能要高一些。

3.4.2 HDCMBAC 中并行性分析

按照 2.3.2 节所列出的并行组织方案，HDCMBAC 在解码时几乎可以像没有上下文建模模块一样快速地解析码流。为了量化地给出 HDCMBAC 相对于 CABAC 的并行性的提高，本节采用 HDCMBAC 相对于 CABAC 的加速比（Speedup）作为衡量标准来分析 HDCMBAC 的并行性。

由于 HDCMBAC 和 CABAC 都可以用一些基本操作来实现，这些基本操作包括加法（Add），减法（Subtract），移位（Shift），比较（Compare），逻辑运算（Logical）和访存（Move）操作。这些基本操作所需要的时钟周期是一样的，因此 HDCMBAC 相对于 CABAC 的加速比可以按照公式（3-11）来计算。

$$S_p = \frac{T_{CABAC}}{T_{HDCMBAC}} \tag{3-11}$$

其中 p 表示所使用的处理器的个数，T_{CABAC} 表示 CABAC 所需要的时钟周期，

$T_{HDCMBAC}$表示 HDCMBAC 所需要的时钟周期。

表 3-5 HDCMBAC 相对于 CABAC 的加速比

Table 3-5 Speedup of HDCMBAC relative to CABAC

QP / Sequence	16	20	24	28	32	36
Mobile@CIF	1.17	1.18	1.18	1.18	1.18	1.18
Paris@CIF	1.17	1.17	1.17	1.15	1.18	1.13
Foreman@CIF	1.16	1.16	1.18	1.19	1.17	1.18
City@4CIF	1.19	1.19	1.18	1.17	1.17	1.17
Crew@4CIF	1.19	1.18	1.16	1.15	1.14	1.13
Ice@4CIF	1.19	1.17	1.17	1.17	1.17	1.16
City@720p	1.19	1.19	1.18	1.17	1.16	1.15
Bigships@720p	1.19	1.18	1.18	1.17	1.17	1.15
Cyclist@720p	1.19	1.17	1.16	1.15	1.16	1.13
BasketballDrive@1920x1080	1.19	1.18	1.18	1.16	1.16	1.16
BQTerrace@1920x1080	1.18	1.19	1.19	1.19	1.17	1.16
Cactus@1920x1080	1.19	1.19	1.18	1.16	1.16	1.15
PeopleOnStreet@2560x1600	1.18	1.15	1.12	1.13	1.12	1.15
Traffic@2560x1600	1.19	1.18	1.18	1.17	1.16	1.15
Average	**1.184**	**1.177**	**1.172**	**1.165**	**1.162**	**1.154**

由于 HDCMBAC 和 CABAC 的区别在于变换系数的解码过程，因此本节只关注与变换系数相关的语法元素，即 CABAC 中的 *significant_map*，*last_significant_map*，*bin0* and *bin13* 和 HDCMBAC 中的 *N*，*significant_map*，*bin0* 和 *bin13*。CABAC 的串行特性使得上下文建模模块和算术解码模块必须串行执行，因此T_{CABAC}是上下文建模模块和算术编码模块所消耗的时钟周期之和。根据 3.4.2 节所列出的并行组织方案，使用 2 个处理器即可实现 HDCMBAC 中的上下文建模模块和算术解码模块的并行处理。此时$T_{HDCMBAC}$只是算术编码模块所消耗的时钟周期。表 3-5 列出了 HDCMBAC 相对于 CABAC 的加速比。使用 2 个处理器时，在不同的 QP 下，HDCMBAC 相对于 CABAC 的平均加速比是 1.154~1.184。这个加速比并不是十分显著，这主要是因为 HDCMBAC 和 CABAC 中的算术编码模块即 M-coder 在解码过程中占主导地位，并且 M-coder 解码一个二进制符号所需要的时间是上下文建模模块的 4 倍。如果使用文献[100]所设计的算术编码器作为算术编码模块，则 HDCMBAC 相对于 CABAC 的加速比将会达到 1.423。

在硬件设计中，由于 HDCMBAC 的 *significant_map* 和 *bin0* 的上下文建模过程打破了同一类型语法元素之间的上下文依赖关系，所以，在并行解码

significant_map 和 *bin0* 时，不再需要推导计算和存储单元来计算和保存解码过程中可能使用的上下文模型。

3.4.3 HDCMBAC 的运算操作数和内存消耗

本节将使用基本操作查表（Look-up table），比较（Compare）和加法（Add）的个数来衡量 HDCMBAC 和 CABAC 的上下文建模模块的运算操作数。在 HDCMBAC 中，*N* 和 *significant_map* 都需要 1 次比较操作和一个查表操作来获取二元组 (σ,V_{MPS})；对于 *bin0*，它首先需要一次查表操作来获取上下文模型的索引，然后再需要一次查表操作来获取二元组 (σ,V_{MPS})。在 CABAC 中，*significant_map* 和 *last_significant_map* 都需要一次查表操作来获取二元组 (σ,V_{MPS})；另外 *last_significant_map* 还需要一次比较操作来确定是否需要解码 *last_significant_map*；对于 *bin0*，它首先需要一次比较操作和一次加法来计算 *NumT1* 和 *NumLgt1*，然后需要 1 次或者 2 次比较操作来获取上下文模型的索引，最后需要一次查表操作来获取二元组 (σ,V_{MPS})。表 3-6 给出了 HDCMBAC 和 CABAC 中上下文建模模块所需要的操作数目。从中可见，HDCMBAC 的运算操作数要比 CABAC 低。

表 3-6 HDCMBAC 和 CABAC 中上下文建模模块所需要的运算操作数目

Table 3-6 The number of operations cost by context modeling in HDCMBAC and CABAC

	语法元素	操作数目		
		查表	比较操作	加法
HDCMBAC	*N*	1	1	
	significant_map	1	1	
	bin0	2		
CABAC	*significant_map*	1		
	last_significant_map	1	1	
	bin0	1	3 或 2	1

在 HDCMBAC 和 CABAC 中，有 5 中变换系数类型，即 *Luma_DC*，*Luma_AC*，*Luma*_4×4，*Chroma_DC* 和 *Chroma_AC*。HDCMBAC 和 CABAC 都要为每一个变换系数类型存储 *significant_map*，*bin0* 和 *bin13* 的上下文模型。CABAC 要额外为每个变换系数类型存储 *last_significant_map* 的上下文模型，而 HDCMBAC 要额外为每个变换系数类型存储 *N* 的上下文模型。由于存储二元组 (σ,V_{MPS}) 将需要 7 个比特，公式（3-9）中的映射关系的每一个条目都需要 2 个比特。所以表 3-7 列出了 HDCMBAC 和 CABAC 中内存消耗。

从中可见，HDCMBAC 需要跟多的内存空间。

表 3-7 HDCMBAC 和 CABAC 的内存消耗

Table 3-7 Memory requirement of HDCMBAC and CABAC

	语法元素	表格大小 (bits)	内存消耗 (bits)
HDCMBAC	*N*	5×4×16×7	5307
	significant_flag	5×4×16×7	
	bin0	16×16×2+5×4×7	
	bin13	5×5×7	
CABAC	*significant_flag*	5×16×7	1470
	last_significant_flag	5×16×7	
	bin0	5×5×7	
	bin13	5×5×7	

3.5 本章小结

本章针对变换系数提出了基于层次依赖上下模型的算术编码方案 HDCMBAC。HDCMBAC 通过打破变换系数之间的上下文依赖关系和设计新的语法元素，使得在编码变换系数时上下文建模模块和算术编码模块可以并行执行，从而提高数据吞吐率。在硬件设计时，为了实现语法元素的并行解码，HDCMBAC 不再需要使用推导计算和存储单元来计算和保存解码过程中可能使用的上下文模型。同时 HDCMBAC 可以取得与 CABAC 相似的编码性能。

第4章 HEVC 中熵编码模块的优化设计

HEVC 是目前最新的视频压缩标准，它的编码效率相比于 H.264/AVC 提高了一倍。但是，随着人们对视频质量的要求越来越高，视频信号的数据量将会越来越大，那么未来的视频压缩标准将会处理数据量更加庞大的视频，因此，继续提高视频压缩标准的压缩效率依然是未来视频压缩标准的主题之一。作为视频压缩标准中核心模块之一的熵编码模块，在不显著增加硬件设计复杂度的前提下，如何提高其编码效率依然是视频压缩领域的研究热点。为此，本章以 HEVC 为基础，通过重新设计变换系数的上下文建模方案和算术编码引擎来提升它的编码效率。

4.1 HEVC 中熵编码模块的概述

上下文自适应的二进制算术编码 CABAC 是 HEVC 中唯一的熵编码方法。如 2.2 小节所描述，HEVC 中的 CABAC 包括二值化过程，上下文建模过程和二进制算术编码引擎。算术编码引擎包含两个编码模式，一个是 regular 模式，即使用上下文模型的编码方式，另一个是 bypass 模式，即概率等于 0.5 的编码模式。上下文建模模块的主要功能是为采用 regular 模式的二进制符号提供上下文模型。因此，本小节将给出变换系数的上下文建模过程和 regular 模式下的算术编码引擎的编码流程。

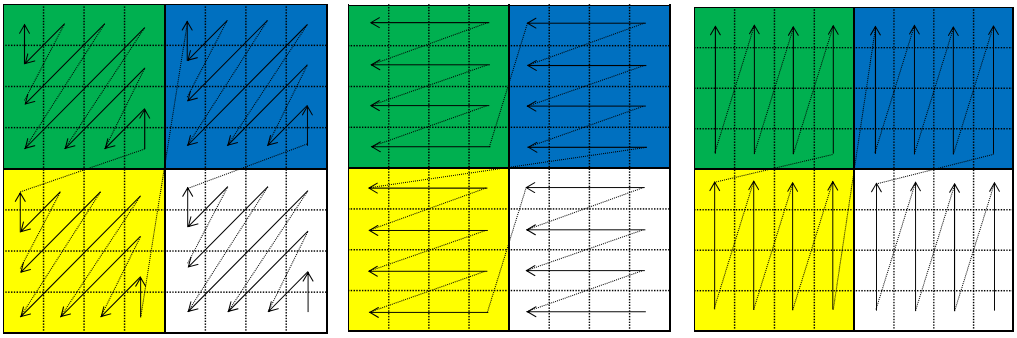
4.1.1 HEVC 变换系数的编码流程

（1）变换系数编码流程概述

HEVC 中变换系数块（TB：Transform Block）的大小可以是 4×4，8×8，16×16 和 32×32。为了编码不同大小的变换系数块，HEVC 采用了基于系数组（CG：Coefficient Group）的编码方案。在这种编码方案中，尺寸大于 4×4 的变换系数块被分割为不重叠的 4×4 大小的子块，每个子块中的系数按照给定的扫描顺序组成了一个 CG。图 4-1 给出了一个尺寸为 8×8 的变换系数块在不同扫描方式下的 CG 划分示意图，其中不同的颜色代表不同的 4×4 子块。

经过预测，变换和量化之后，变换系数块中的大部分系数都是零系数，甚至在有些情况下整个变换系数块都是零系数，因此可以认为变换系数块具

有稀疏特性。为了充分利用这种稀疏特性，HEVC 首先编码语法元素 *CBF*（*coded_block_flag*）来表示当前变换系数块是否包含非零系数。如果当前变换系数块包含非零系数即 *CBF* 等于 1，则继续编码最后一个非零系数的位置信息，用来指示当前变换系数块的哪个区域包含非零系数。最后一个非零系数的位置信息是按照某一扫描顺序计算得到，最后编码的信息是最后一个非零系数相对于直流系数（DC 系数）的（X,Y）坐标。



（a）　　（b）　　（c）

图 4-1 8×8 大小的变换系数块中 CG 划分示意图

（a）对角线扫描，（b）水平扫描，（c）竖直扫描

Fig 4-1 Illustration of CG partition within 8×8 transform block

(a) Diagnal, (b) Horizontal, (c) Vertical

接着，HEVC 从最后一个非零系数开始按照逆向扫描顺序来编码每个变换系数。具体来讲，HEVC 首先为每个 CG 编码一个语法元素 *CSBF*（*coed_sub_block_flag*）来指示该 CG 是否包含非零系数，其中 *CSBF* 等于 1 表示该 CG 包含非零系数，否则 *CSBF* 为 0；由于包含最后一个非零系数的 CG 的 *CSBF* 已经确定为 1，因此不再为该 CG 编码 *CSBF*；另外由于 DC 系数在很多情况下都为非零系数，因此也不为包含 DC 系数的 CG 编码 *CSBF*。对于 *CSBF* 等于 1 的 CG，HEVC 使用语法元素 *significant_coeff_flag* 来指示该 CG 内每一个位置处的变换系数是否是非零系数；对于非零系数即 *significant_coeff_flag* 等于 1 的系数，使用语法元素 *coeff_abs_greater1_flag* 来指示当前非零系数是否大于 1；对于绝对值大于 1 的非零系数即 *coeff_abs_greater1_flag* 等于 1 的系数，使用语法元素 *coeff_abs_greater2_flag* 来指示当前非零系数是否大于 2；对于绝对值大于 2 的非零系数即 *coeff_abs_greater2_flag* 等于 1 系数，使用语法元素 *coeff_abs_level_remaining* 来表示其绝对值中剩余的数值；最后使用语法元素 *coeff_sign_flag* 指示每个

非零系数的符号。为了提高编码变换系数的数据吞吐率，HEVC 在不同的扫描过程编码不同的语法元素。即首先编码当前 CG 内所有变换系数的 *significant_coeff_flag*，然后编码该 CG 内所有的 *coeff_abs_greater1_flag*，接着编码该 CG 内所有的 *coeff_abs_greater2_flag*，最后编码该 CG 内所有的 *coeff_abs_level_remaining*；除此之外，HEVC 仅仅对一个 CG 内前八个变换系数的 *coeff_abs_greater1_flag* 和第一个绝对值大于 1 的变换系数的 *coeff_abs_greater2_flag* 采用 regular 模式编码；对于 CG 内剩余变换系数，HEVC 则使用语法元素 *coeff_abs_level_remaining* 来编码它们的幅值，并且 *coeff_abs_level_remaining* 和 *coeff_sign_flag* 采用 bypass 模式来编码。

（2）HEVC 中上下文建模过程

在编码 *significant_coeff_flag* 时，HEVC 为 4×4 大小的变换系数块采用了基于位置的上下文模型，即按照图 4-2 为每个位置选择上下文模型。而对于尺寸大于 4×4 的变换系数块，HEVC 首先根据当前 CG 的右侧和下侧 CG 的 *CSBF* 值 s_r 和 s_l 为当前 CG 选择一个模板，如图 4-3 所示；然后根据当前变换系数在当前 CG 内的位置通过选中的模板来选择上下文模型。

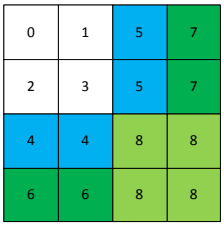


图 4-2 4×4 大小的变换系数块中 *significant_coeff_flag* 上下文模型选择方案

Fig 4-2 The context model selection for *significant_coeff_flag* in 4×4 TB in HEVC

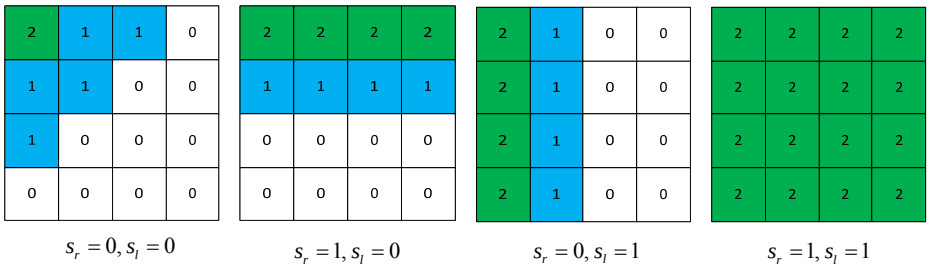


图 4-3 用于编码尺寸大于 4×4 的变换系数块中 *significant_coeff_flag* 的模板

Fig 4-3 Templates for coding *significant_coeff_flag* in TB larger than 4×4

当编码 *coeff_abs_greater1_flag* 时，HEVC 首先根据已经编码的 CG 内是否包含绝对值大于 1 的系数来选择上下文模型集合，然后再利用当前 CG 内

已经编码的绝对值等于 1 的变换系数个数 *NumEqu1* 和绝对值大于 1 的变换系数个数 *NumGre*1 从这个上下文模型集合中选择合适的上下文模型。在编码 *coeff_abs_greater2_flag* 时，HEVC 直接根据已经编码的 CG 内是否包含绝对值大于 1 的变换系数来为当前变换系数选择上下文模型。

4.1.2 HEVC 中算术编码引擎

与 H.264/AVC 相同，HEVC 采用的算术编码引擎也是 M-coder。关于 M-coder 的实现原理，本文在 2.3 小节做了详细的介绍，这里将不再赘述。算法 4-1 给出了使用 M-coder 编码一个二进制符号的编码流程，其中“>>”表示右移，bin 表示当前要编码的二进制符号，L 表示编码区间的下边界，R 表示编码区间的长度，s 表示的是 LPS 概率 p_{LPS} 的索引，其取值范围是 $0 \le s \le 63$，*TabLPSRange*[s][Δ] 是实现编码区间分割的数组，*TransStateLPS*[s] 和 *TransStateMPS*[s]是实现概率更新的数组。

算法 4-1 M-coder 编码一个二进制符号的执行流程

Algorithm 4-1 Procedure for encoding a binary symbol using M-coder

输入： bin, s, R 和 L
输出： s, R 和 L

Δ = (R >> 6) & 3;
R_{LPS} = TabLPSRange[s][Δ];
$R = R - R_{LPS}$;
If bin==LPS then
    $L = L + R$;
    $R = R_{LPS}$;
    s = *TransStateLPS*[s];
Else
    s = *TransStateLPS*[s];
End If
调用归一化过程输出比特

4.2 HEVC 中熵编码过程的优化方案

通过上面的描述可以发现，HEVC 中变换系数的上下文建模过程都是从相邻 CG 中获取上下文信息。这种方式虽然可以减少变换系数之间的上下文依赖关系，但是同时也会使得上下文建模过程无法充分挖掘变换系数之间的统计相关性。另外，在 M-coder 的设计中，为了能够采用查表操作来完成编码区间分割和概率自适应更新，概率区间 $(0.0, 0.5]$ 被离散化为 64 个典型的概率，如公式（2-3）所示。在这个离散化过程中，接近 0.5 的概率区间是以很

粗糙的粒度来实现离散化的，这对其编码效率会带来不利影响；同时二进制算术编码引擎中的 *TabLPSRange*[][]，*TransStateLPS*[]和 *TransStateMPS*[]会增加硬件实现时的内存消耗。因此，为了进一步提高 HEVC 的编码效率和降低其内存消耗，本文提出了一个变换系数的增强上下文建模过程和低内存消耗的二进制算术编码引擎。

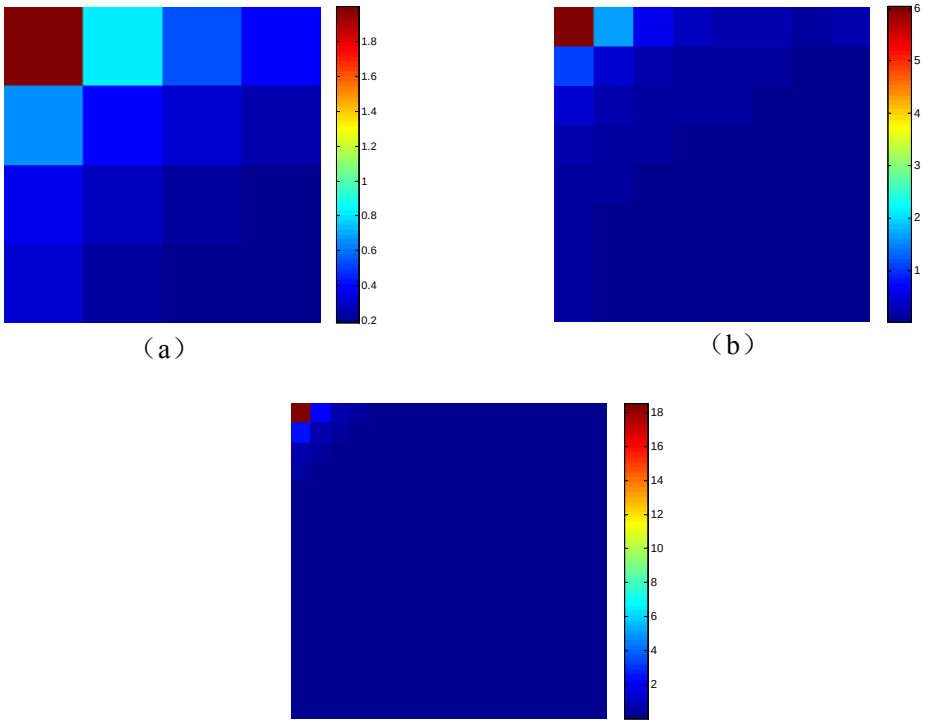
4.2.1 变换系数的增强上下文建模过程

（1）变换系数的统计特性

在第三章中，为了提高 H.264/AVC 中 CABAC 的数据吞吐率，变换系数块中非零变换系数的个数和变换系数的频域位置被用作上下文来挖掘变换系数之间的统计相关性，并且根据第三章的描述可知，上下文模型的个数将会随着变换系数块的尺寸而呈指数级增长。在这种情况下，在编码较大尺寸的变换系数块（如 8×8，16×16 和 32×32 大小的变换系数块）时，其中所涉及的上下文模型的个数将会远远地超出可以接受的范围。除此之外，对于尺寸较大的变换系数块，整个变换系数块中的非零变换系数的个数可能无法准确地捕捉某一变换系数的概率分布，这是因为距离当前变换系数较远的变换系数不但无法提高上下文模型的准确性，反而会增加上下文模型的不确定性。所以，第三章中的上下文模型将不再适用于编码 HEVC 的变换系数。尽管如此，正如 3.3 节的分析所示：非零变换系数的个数在一定程度上反映了编码区域的纹理复杂性。那么，是否可以采用距离当前变换系数最近的几个变换系数中的非零变换系数的个数来捕捉变换系数之间的统计相关性？基于这个想法，本文对 HEVC 中的变换系数的统计特性进行了详细地分析。

众所周知，处在低频位置的变换系数的概率分布和处在高频位置的变换系数的概率分布是不同的，图 4-4 给出了 HEVC 中不同大小的变换系数块中每个位置上变换系数概率分布的标准差。从中可以看到，按照从高频到低频的顺序，变换系数概率分布的标准差是逐渐递增的；并且在高频区域中，不同位置处的变换系数概率分布的标准差比较接近。

除了变换系数的频率位置以外，当前待编码的变换系数和已经编码的变换系数之间也存在着相关性。比如，如果已经编码的变换系数中存在非零系数，则当前待编码的变换系数很有可能也是非零系数。为了从已经编码的变换系数中选取合适数量的变换系数来预测当前变换系数的概率分布，本文采用随机变量 X 和 Y 之间的互信息 $I(X;Y)$ 来衡量它们之间的相关性，互信息的计算方法如公式（4-1）所示。



（a）　　（b）

（c）

图 4-4 HEVC 中不同大小的变换系数块中每个位置的概率分布标准差

（a）4×4，（b）8×8，（c）16×16

Fig 4-4 The standard variance of transform coefficient level at each position in TBs

(a) 4×4, (b) 8×8, (c) 16×16

$$I(X;Y)=\sum_{y\in Y}\sum_{x\in X}p(x,y)\cdot\log_2\left(\frac{p(x,y)}{p(x)\cdot p(y)}\right) \qquad (4\text{-}1)$$

在公式（4-1）中，$p(x,y)$是随机变量 X 和 Y 之间的联合概率分布；$p(x)$和 $p(y)$分别表示随机变量 X 和 Y 的边缘概率。表 4-1 给出了在 8×8 大小的变换系数块中 *significant_coeff_flag* 与 *numSigs* 之间的互信息，其中 *numSigs* 表示的是被某一模板所覆盖的变换系数中非零系数的个数，这个模板可能是 $\{x_0,x_1\}$，$\{x_0,...,x_4\}$ 和 $\{x_0,...,x_8\}$ 中的任何一个。其中 $\{x_0,x_1\}$，$\{x_0,...,x_4\}$ 和 $\{x_0,...,x_8\}$的分布如图 4-5 所示。在表 4-1 中，*tmp1*，*tmp2* 和 *tmp3* 分别表示 $\{x_0,x_1\}$，$\{x_0,...,x_4\}$ 和 $\{x_0,...,x_8\}$三种模板；当前变换系数的位置是通过该系数相对于 DC 系数的(X,Y)坐标来表示的。

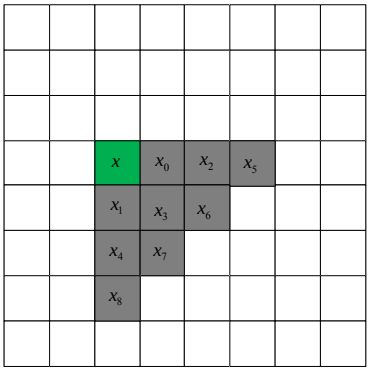


图 4-5 表 4-1 中当前变换系数 *x* 的不同模板示意图

Fig 4-5 Illustration of the templates for the current transform coefficient *x* in Table 4-1

如果随机变量 X 和 Y 之间的互信息 $I(X;Y)$ 越大，则 X 和 Y 之间的相关性就越大，因此用 Y 来指导 X 的上下文建模就能取得更好的编码性能。通过表 4-1 可以发现，*significant_coeff_flag* 和模板 *tmp2* 中的 *numSigs* 之间的互信息要一直大于 *significant_coeff_flag* 和模板 *tmp1* 中的 *numSigs* 之间的互信息，而且有时会大于 *significant_coeff_flag* 和模板 *tmp3* 中的 *numSigs* 之间的互信息。这是因为仅仅使用很少一部分的相邻变换系数则不能完全挖掘变换系数之间的相关性；但是如果使用太多的相邻变换系数则可能会产生上下文稀疏问题。因此与文献[119]一样，本文采用模板 *tmp2* 作为最终的模板来挖掘变换系数之间的相关性。

表 4-1 8×8 大小的变换系数块中的 *significant_coeff_flag* 与 *numSigs* 之间互信息

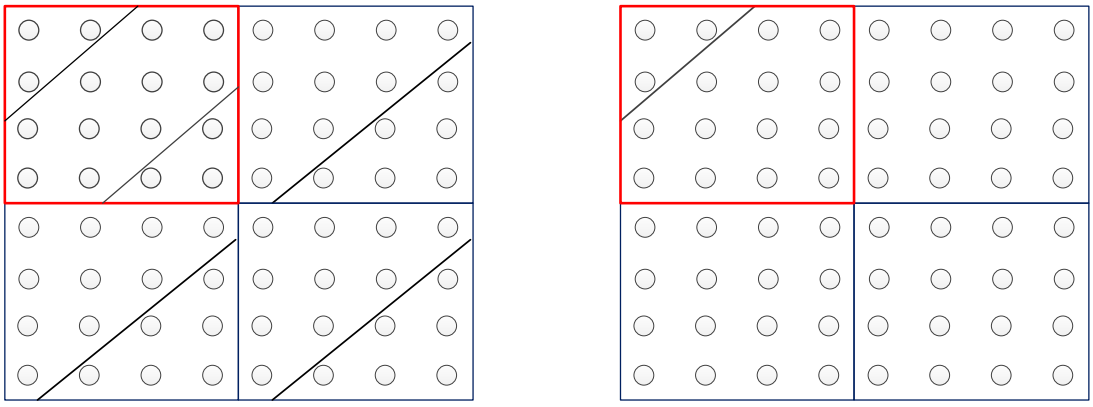
Table 4-1 Multual information between *significant_coeff_flag* and *numSigs* in 8×8 TB

模板 位 置	tmp1	tmp2	tmp3
(0, 0)	0.190	0.244	0.323
(1, 0)	0.113	0.143	0.145
(0, 1)	0.125	0.159	0.158
(3, 0)	0.092	0.127	0.135
(0, 3)	0.115	0.159	0.169
(5, 0)	0.102	0.141	0.118
(0, 5)	0.087	0.125	0.111

（2）变换系数的增强上下文建模过程

通过上一小节所描述的 HEVC 中变换系数的统计特性，本文决定采用当前变换系数的频域位置和其局部模板所覆盖的变换系数作为它的上下文来源。在解码当前变换系数的 *significant_coeff_flag* 时，当前变换系数在整个变换系

数块内的频域位置(x,y)和其局部模板内的非零系数的个数*numSigs*被用作上下文。如果简单地把(x,y)和*numSigs*组合在一起，则需要很多的上下文模型。这种情况不但会增加内存消耗，同时也会导致上下文稀疏问题。以 8×8 大小的变换系数块为例，因为$0 \le x \le 7$，$0 \le y \le 7$和$0 \le numSigs \le 5$，这种简单的组合方式将会产生 384 个上下文模型。



（a）　　（b）

图 4-6 编码 *significant_coeff_flag* 时亮度和色度分量中变换系数块的分割方式

（a）亮度分量，（b）色度分量

Fig 4-6 Splitting method of TBs in Luma and Chroma components for coding *significant_coeff_flag*

(a) Luma, (b) Chroma

为了解决这个问题，本文把变换系数块按照频域位置(x,y)分成不同的区域，每个区域内的 *significant_coeff_flag* 的上下文模型根据*numSigs*来选择。对于亮度变换系数块，左上角 CG 按照公式（4-2）被分成了 3 个区域

$$RegIdx = \begin{cases} 0, & if \ \ x_{InCG} + y_{InCG} < 2 \\ 1, & if \ \ x_{InCG} + y_{InCG} \ge 2 \ \ and \ \ x_{InCG} + y_{InCG} < 5 \\ 2, & if \ \ x_{InCG} + y_{InCG} \ge 5 \end{cases} \tag{4-2}$$

其余的 CG 按照公式（4-3）被分成 2 个区域

$$RegIdx = \begin{cases} 3, & if \ \ x_{InCG} + y_{InCG} < 4 \\ 4, & otherwise \end{cases} \tag{4-3}$$

对于色度分量中的变换系数块，按照公式（4-4）把它分成 2 个区域，

$$RegIdx = \begin{cases} 0, & if \quad x + y < 2 \\ 1, & otherwise \end{cases} \tag{4-4}$$

这些公式中 *RegIdx* 是区域的索引，(x_{InCG}, y_{InCG})是当前变换系数在当前 CG 内的位置，它们可以按照公式（4-5）来计算，

$$\begin{cases} x_{InCG} = x - ((x >> 2) << 2) \\ y_{InCG} = y - ((y >> 2) << 2) \end{cases} \tag{4-5}$$

图 4-6 给出了以上描述的变换系数块的分割方式。基于这个变换系数块的分割，*significant_coeff_flag* 的上下文模型选择过程可以用公式（4-6）来描述，

$$Ctx_{Sig} = RegIdx \times 6 + numSigs \tag{4-6}$$

其中 Ctx_{Sig} 表示选中的上下文模型的索引。

一般情况下，亮度分量包含着图像的纹理信息，因此亮度分量中的非零变换系数的位置分布更加广泛，即绝大部分的非零系数仍然位于低频位置，其余位置有时也会包含非零变换系数。所以，本文对左上角 CG 划分成三个区域来提高上下文模型的准确性，而对于其余 CG 划分成 2 各区域来减少上下文建模过程的复杂度。与亮度分量相比，色度分量的平滑特性更加突出，经过预测变换和量化之后，只有左上角位置上的几个低频系数是非零系数，其余的变换系数几乎都为零系数。在这种情况下，使用变换系数在变换系数块中的位置(x, y)将会把具有相似统计特性的变换系数划分在同一个区域内。因此，本文对亮度分量和色度分量分布采用了图 4-6 所示的划分方式。

综上所述，在编码变换系数的 *significant_coeff_flag* 时，首先根据当前 CG 的位置以及当前变换系数在当前 CG 内的位置选择上下文模型集合，或者根据当前变换系数在变换系数块中的位置选择上下文模型集合；然后再根据局部模板所覆盖的非零系数的个数从该上下文模型集合中选择最终要使用的上下文模型。算法 4-2 给出了该语法元素的上下文模型选择过程，其中(x, y)表示当前变换系数在变换系数块内的位置；eType==1 表示当前编码的是亮度分量，否则表示色度分量；“<<”和“>>”分别表示左移和右移操作。

算法 4-2 本文提出的 *significant_coeff_flag* 的上下文模型选择过程

Algorithm 4-2 Context model seclection for *significant_coeff_flag* in the proposed method

输入：(x, y), numSigs, eType
输出：Ctx_{Sig}

```
If eType == 1 then
    xCG = x >> 2;
    yCG = y >> 2;
    xInCG = x – (xCG << 2);
    yInCG = y – (yCG << 2);
    If xCG + yCG == 0 then
      If xInCG + xInCG < 2 then
        RegIdx = 0;
      Else If 2 <= xInCG + xInCG < 5 then
            RegIdx = 1;
          Else
              RegIdx = 2;
          End If
      End If
    Else
      If xInCG + xInCG < 4 then
        RegIdx = 3;
      Else
          RegIdx = 4;
      End If
    End If
Else
    If x + y < 2 then
        RegIdx = 0;
    Else
        RegIdx = 1;
    End If
End If
CtxSig = 6×RegIdx + numSigs;
```

在编码变换系数的 *coeff_abs_greater1_flag* 时，由于此时所有变换系数的 *significant_coeff_flag* 已经解码完毕，并且当前变换系数的局部模板内的变换系数的 *coeff_abs_greater1_flag* 也已经解码完毕，因此当前变换系数的局部模板内绝对值等于 1 的变换系数个数 *NumEqu1* 和绝对值大于 1 的变换系数的个数 *NumGre1* 可以被用作上下文。另外，为了挖掘变换系数与其频域位置之间的相关性，本文把频域位置 (x, y) 作为亮度分量中变换系数的上下文。为了合

理地在上下文模型选择过程中使用频域位置(x,y)，本文把变换系数块按照公式（4-7）分成 3 个区域，如图 4-7 所示。

$$RegIdx=\begin{cases}0,\ if\ \ x+y<3\\ 1,\ if\ \ x+y\geq 3\ and\ \ x+y<10\\ 2,\ if\ \ x+y\geq 10\end{cases} \tag{4-7}$$

然后对于每个区域，按照公式（4-8）为 *coeff_abs_greater1_flag* 计算上下文模型索引。

$$Ctx_{greater1}=7\times RegIdx+\begin{cases}\min(NumGre1-1,3),\ if\ \ NumGre1>0\\ \min(NumEqu1,2)+4,\ otherwise\end{cases} \tag{4-8}$$

图 4-7 编码亮度分量中的 *coeff_abs_greater1_flag* 时变换系数块的分割方式

Fig 4-7 Splitting method of TB for coding *coeff_abs_greater1_flag* in Luma

由于只有在编码 *coeff_abs_greater1_flag* 等于 1 的变换系数时，才需要编码 *coeff_abs_greater2_flag*；并且在编码变换系数的 *coeff_abs_greater2_flag* 时，所有变换系数的 *coeff_abs_greater1_flag* 都已经解码完毕，同时当前变换系数的局部模板内的变换系数的 *coeff_abs_greater2_flag* 也已经解码完毕。所以当前变换系数的局部模板内绝对值大于 1 的变换系数的个数 *NumGre1* 和绝对值大于 2 的变换系数的个数 *NumGre2* 都可以被用作上下文。另外，由于 *coeff_abs_greater2_flag* 在编码变换系数中所占的比例较少，因此，为了降低上下文模型的个数，本文不再对变换系数块进行区域划分。所以，*coeff_abs_greater2_flag* 的上下文模型索引可以直接按公式（4-9）来计算。

$$Ctx_{greater2}=\begin{cases}0,\ \ if\ \ \ NumGre2>0,\\ 1,\ else\ \ if\ \ NumGre1>0\\ 2,\ otherwise\end{cases} \tag{4-9}$$

算法 4-3 编码 *coeff_abs_greater1_flag* 时上下文模型选择方法

Algorithm 4-3 The context model selection for coding *coeff_abs_greater1_flag*

输入：(x, y), NumEqu1, NumGre1, eType
输出：$Ctx_{greater1}$

```
If eType==1 then
  If 当前变换系数是最后一个非零系数 then
    Ctx_greater1 = 21;
  Else
    If x + y < 3 then
      RegIdx = 0;
    Else If x + y < 10 then
        RegIdx = 1;
      Else
        RegIdx = 2;
      End If
    End If
    If NumGre1 > 0 then
      Ctx_greater1 = 7×RegIdx + min(NumGre1 – 1, 3);
    Else
      Ctx_greater1 = 7×RegIdx + min(NumEqu1, 2) + 4;
    End If
  End If
Else
    If 当前变换系数是最后一个非零系数 then
      Ctx_greater1 = 7;
    Else
      If NumGre1 > 0 then
        Ctx_greater1 = min(NumGre1 – 1, 3);
      Else
        Ctx_greater1 = min(NumEqu1, 2) + 4;
      End If
    End If
End If
```

对于变换系数块中的最后一个非零系数，它的 *coeff_abs_greater1_flag* 和 *coeff_abs_greater2_flag* 的上下文模型是单独的一个上下文模型。即这个上下文模型是为最后一个非零系数专门设置的，不会被其他的变换系数所使用。所以，在编码 *coeff_abs_greater1_flag* 时，首先根据当前变换系数在变换系数块中的位置选择上下文模型集合，然后再根据局部模板内的 *NumEqu1* 和

NumGre1 从该上下文模型集合中选择最终要使用的上下文模型；而在编码 *coeff_abs_greater2_flag* 时，则直接根据局部模板内的 *NumGre1* 和 *NumGre2* 来选择最终需要使用的上下文模型。因此，改进后的 *coeff_abs_greater1_flag* 的上下文模型选择过程如算法 4-3，其中(x, y)表示当前变换系数在变换系数块内的位置；eType==1 表示亮度分量，否则表示色度分量。算法 4-4 给出了改进后的 *coeff_abs_greater2_flag* 的上下文模型选择过程，其中(x, y)表示当前变换系数在变换系数块内的位置，并且亮度分量和色度分量的 *coeff_abs_greater2_flag* 的上下文模型的选择过程是一样的，但使用的是不同的上下文模型集合。

算法 4-4 编码 *coeff_abs_greater2_flag* 时上下文模型选择方法

Algorithm 4-4 The context model selection for coding *coeff_abs_greater2_flag*

输入：(x, y), NumGre1, NumGre2
输出：$Ctx_{greater2}$

```
If 当前变换系数是最后一个非零系数 then
    Ctx_greater2 = 3;
Else
    If NumGre2 > 0 then
       Ctx_greater2 = 0;
    Else If NumGre1 > 0 then
            Ctx_greater2 = 1;
         Else
            Ctx_greater2 = 2;
         End If
    End If
End If
```

4.2.2 内存消耗的二进制算术编码引擎

HEVC 中的二进制算术编码引擎 M-coder 采用指数形式的概率分割方式把概率区间(0,0.5]离散化为 64 个典型概率。如果编码的符号是 MPS（Most Probable Symbol），则这种方式的概率离散化在概率更新过程中不会产生误差，因为此时概率正好是从 p_n 更新到 p_{n+1}，是一个点到点的更新过程；但是如果编码的是 LPS（Least Probable Symbol）则这种方式的概率离散化在概率更新过程中就会产生误差，因为此时要从这 64 个典型概率中选择一个与更新得到的概率最相近的那个典型概率作为最终的概率，这时就会产生误差。并且这个误差在概率接近 0.5 是会变大。为了解决这个问题，文献[96]提出了一个多参数概率估计方法。在这个方法中，概率 p（$0.0 < p \le 1.0$）是通过一个

位宽为 15 的整数来表示的，并且使用不同的更新速度来更新 p，如公式（4-10）所示，

$$\begin{aligned} & p_{new} = (p^0_{new} + p^1_{new} + 1) >> 1, \\ & with \begin{cases} p^0_{new} = \bar{\alpha}_0 \cdot y + (1 - \bar{\alpha}_0) \cdot p^0_{old} \\ p^1_{new} = \bar{\alpha}_1 \cdot y + (1 - \bar{\alpha}_1) \cdot p^1_{old} \end{cases} \end{aligned} \tag{4-10}$$

其中，“>>”表示向右移位符号，p^0_{old} 和 p^1_{old} 是更新前的概率；p^0_{new} 和 p^1_{new} 是更新后的概率；$\bar{\alpha}_0$ 和 $\bar{\alpha}_1$ 是两个不同的更新速度，分别等于 $1/16$ 和 $1/128$；p_{new} 是最终的更新概率。

在编码区间分割过程中，文献[96]采用了查表的方法。由于此时的概率取值范围是(0.0, 1.0]，这个表格的大小将达到 288Kb。如此大的表格将会消耗更多的内存，从而给硬件设计带来困难。但是如果直接采用乘法操作来实现编码区间的分割，将会引入一个位宽为 15×9 的乘法器。位宽如此高的乘法器也将会增加硬件设计的复杂度。

为了降低算术编码引擎的内存消耗，本文使用一个低位宽的乘法器来实现编码区间的分割。为此，本文把编码区间长度 R 的取值区间 $[2^{b-1}/2, 2^{b-1})$ 给均匀地划分成 32 子区间，因此编码区间分割过程可以按照公式（4-11）来完成，

$$\begin{aligned} R &= p \cdot R \\ &\approx \frac{(p << (b-1)) + p \cdot \frac{1}{32} \cdot \Delta \cdot 2^{b-1}}{2} \\ &= \frac{(p << (b-1)) + 2^{b-6} \cdot p \cdot \Delta}{2} \\ &\approx \frac{(p << (b-1)) + 2^{b} \cdot (p >> 6) \cdot \Delta}{2} \end{aligned} \tag{4-11}$$

其中，“<<”和“>>”分别表示向左和向右移位符号，b 是存储 R 的寄存器的位宽；Δ 是子区间的索引，其取值范围是 0~31。另外为了提高公式（4-11）的近似精度，本文采用二元组 (p_{LPS}, V_{MPS}) 来表示概率 p。这样设计之后，只需位宽为 9×5 的乘法器即可完成编码区间的分割。算法 4-5 给出了本文提出的低内存消耗的自适应二进制算术编码器的编码流程。

算法 4-5 本文提出的低内存消耗的自适应二进制算是编码器的编码流程

Algorithm 4-5 The procedure of the proposed binary arithmetic coding engine

输入： bin, p_0, p_1, R 和 L

输出： p_0, p_1, R 和 L

$\Delta = (R - 2^{(b-2)}) >> 3$;
$p_{LPS} = (p_0 + p_1) >> 1$;
$p_{LPS} = ((1 << k) - p_{LPS}, p_{LPS})$;
$R_{LPS} = (p_{LPS} << (b - 1) + ((\Delta \times (p_{LPS} >> 6)) << b)) + (1 << (k - 1)) >> k$;
$R_{LPS} = (R_{LPS} + 1) >> 1$;
$R = R - R_{LPS}$;
If bin==LPS then
　　L = L + R;
　　R = R_{LPS};
　　调用归一化过程；
Else
　　调用归一化过程；
End If
If bin==1 then
　　$p_0 = p_0 + (1 << (k - 4)) + (p_0 >> 4)$;
　　$p_1 = p_1 + (1 << (k - 7)) + (p_1 >> 7)$;
Else
　　$p_0 = p_0 - (p_0 >> 4)$;
　　$p_1 = p_1 - (p_1 >> 7)$;
End If

4.3 实验结果与分析

为了验证本文提出的熵编码优化方案的有效性，本文在 HEVC 参考软件平台 HM14.0 上进行了测试实验，配置条件为 HM Main Profile 下的三组配置条件 All Intra (AI)，Low Delay (LD)，Random Access (RA)。本文一共测试了 20 个通用的测试序列，这些测试序列按照分辨率和内容可以分成 6 类，分别是 Class A (2K)，Class B (1080p)， Class C (WVGA)， Class D (WQVGA)，Class E (720p) 和 Class F。其中 Class F 是屏幕序列，即其内容主要是通过电脑截屏产生的。另外，本文还在 5 个 UHD（超高清视频：分辨率是 3840×2160）视频序列上进行了测试，这些 UHD 序列包括 *Fountains*, *Runners*, *Rushour*, *Trafficflow* and *Campfireparty*。测试的 QP 点包括 22,27,32 和 37，编码性能通过 BD-Rate 来衡量。

4.3.1 变换系数的增强上下文模型

为了衡量变换系数的增强上下文模型的编码性能，本文把变换系数的增强上下文模型和 HEVC 中原始上下文模型以及文献[118]所提出的上下文模型做了比较。接下来，本文首先给出变换系数的增强上下文模型的整体性能，然后分别给出非零系数标志位（*significant_coeff_flag*）和非零系数幅值信息（*coeff_abs_greater1_flag* 和 *coeff_abs_greater2_flag*）的上下文模型的性能。在下面的表格中 ***Overall(HD)***表示的是在测试序列 Class A, ClassB 和 UHD 上的平均 BD-Rate。

（1）变换系数的增强上下文模型的整体性能

表 4-2 给出了变换系数的增强上下文模型相比于 HEVC 中原始上下文模型的性能比较。从中可以看出，与 HEVC 中原始上下文模型相比，变换系数的增强上下文模型平均可以取得 0.4%~0.8%的性能增益。

表 4-2 变换系数的增强上下文模型相对于 HEVC 中原始上下文模型的 BD-Rate[%]

Table 4-2 BD-Rate[%] of the improved context modeling scheme over that in HEVC

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.9	-0.6	-0.7	-0.5	-0.3	-0.2			
Class B	-0.8	-0.6	-0.7	-0.5	-0.4	-0.6	-0.3	-0.1	-0.1
Class C	-0.6	-0.2	-0.3	-0.6	-0.4	-0.3	-0.3	0.4	0.2
Class D	-0.6	-0.4	0.0	-0.6	0.5	-0.4	-0.4	-1.2	-0.1
Class E	-1.0	-0.6	-0.5				-0.6	-1.0	-0.2
Overall	**-0.8**	**-0.5**	**-0.4**	**-0.6**	**-0.2**	**-0.4**	**-0.4**	**-0.4**	**0.0**
Class F	-1.5	-1.5	-1.5	-1.3	-1.3	-1.2	-1.3	-1.5	-1.8
UHD	-0.7	-0.8	-0.9	-0.5	-0.2	0.1	-0.3	-0.2	-0.1
Overall(HD)	**-0.8**	**-0.7**	**-0.8**	**-0.5**	**-0.3**	**-0.2**	**-0.3**	**-0.1**	**-0.1**
Enc Time		107%			100%			100%	
Dec Time		101%			100%			100%	

表 4-3 给出变换系数的增强上下文模型相比于文献[118]提出的上下文模型的性能比较。从中可以看出，与文献[118]提出的上下文模型相比，变换系数的增强上下文模型平均可以取得 0.1%~0.2%的性能增益；在高分辨率的视频序列上，变换系数的增强上下文模型平均可以取得 0.2%~0.4%的性能增益。在接下来的描述中，本文将会解释为什么本文提出的上下文模型在高分辨率的视频上可以取得更好的编码性能。

表 4-3 变换系数的增强上下文模型相对于文献[118]中上下文模型的 BD-Rate[%]

Table 4-3 BD-Rate[%] of the improved context modeling scheme over that in *Ref*[118]

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.2	-0.1	-0.1	-0.1	0.1	-0.1			
Class B	-0.4	-0.2	-0.2	-0.3	0.1	0.1	-0.2	0.3	-0.4
Class C	-0.1	0.1	0.0	-0.1	-0.3	0.1	-0.1	0.2	-0.3
Class D	-0.1	0.1	0.2	-0.1	0.7	0.2	0.0	-1.0	-0.3
Class E	-0.2	0.2	0.0				0.0	-0.6	0.4
Overall	**-0.2**	**0.0**	**0.0**	**-0.2**	**0.1**	**0.1**	**-0.1**	**-0.2**	**-0.2**
Class F	0.1	0.1	0.1	0.1	0.1	0.0	-0.2	-0.5	0.0
UHD	-0.3	-0.1	-0.2	-0.6	0.0	0.1	-0.2	0.0	-0.1
Overall(HD)	**-0.3**	**-0.1**	**-0.2**	**-0.4**	**0.0**	**0.0**	**-0.2**	**0.0**	**-0.2**
EncTime		101%			100%			100%	
DecTime		100%			100%			100%	

表 4-4 给出了变换系数的增强上下文建模方法，HEVC 中上下文建模方法和文献[118]中上下文建模方法中的上下文模型的个数。从中可以看到，对于 *significant_coeff_flag*，变换系数的增强上下文建模方法所需要的上下文模型个数与 HEVC 相同，并且远低于文献[118]中的上下文建模方法；从上下文模型总个数上来看，变换系数的增强上下文建模方法所需要的上下文模型个数要大于 HEVC，但是仍然低于文献[118]中的上下文建模方法。

表 4-4 变换系数的增强上下文模型，HEVC 中上下文模型和文献[118]中上下文模型中所需要的上下文模型个数

Table 4-4 The number of the context models in the context modeling scheme in HECV, *Ref*[118] and the proposed method.

	HEVC		Method in *Ref*[118]		The proposed method	
	Luma	Chroma	Luma	Chroma	Luma	Chroma
Significant_coeff_flag	27	15	54	12	30	12
Coeff_abs_greater1_flag	16	8	16	6	22	8
Coeff_abs_greater2_flag	4	2	0	0	4	4
All	**72**		**88**		**80**	

（2）非零系数标志位的上下文模型的编码性能

表 4-5 给出了本文提出的 *significant_coeff_flag* 的上下文模型相对于 HEVC 中的上下文模型的 BD-Rate。从中可以看出，与 HEVC 中的 *significant_coeff_flag* 的上下文模型相比，本文提出的 *significant_coeff_flag* 的上下文模型平均可以取得 0.3%~0.4%的性能增益。

表 4-5 本文提出的 *significant_coeff_flag* 上下文模型相对于 HEVC 的 BD-Rate[%]

Table 4-5 BD-Rate[%] of the proposed context modeling scheme for *significant_coeff_flag* over that in HEVC

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.5	-0.1	-0.3	-0.4	-0.2	-0.3			
Class B	-0.5	-0.2	-0.4	-0.3	-0.3	-0.7	-0.2	-0.6	-0.2
Class C	-0.3	0.0	0.1	-0.4	-0.1	-0.3	-0.3	0.0	-0.1
Class D	-0.4	-0.2	0.1	-0.5	0.4	-0.4	-0.4	-0.4	-0.3
Class E	-0.5	-0.4	-0.3				-0.5	0.2	-0.3
Overall	**-0.4**	**-0.2**	**-0.2**	**-0.4**	**-0.1**	**-0.5**	**-0.3**	**-0.3**	**-0.2**
Class F	-1.5	-1.3	-1.6	-1.3	-1.5	-1.1	-1.3	-1.7	-1.7
UHD	-0.2	-0.4	-0.5	-0.2	0.2	-0.5	-0.2	-0.3	-0.5
Overall(HD)	**-0.4**	**-0.3**	**-0.4**	**-0.3**	**-0.2**	**-0.5**	**-0.2**	**-0.5**	**-0.3**
Enc Time		102%			102%			101%	
Dec Time		101%			100%			100%	

表 4-6 给出了本文提出的 *significant_coeff_flag* 的上下文模型相对于参考文献[118]中的上下文模型的 BD-Rate。从中可以看出，相比于文献[118]中的上下文模型，本文提出的上下文模型在使用更少的上下文模型的情况下比参考文献[118]中的上下文模型取得了更好或者相似的编码性能。接下来，本文将对 HEVC，文献[118]和本文提出的上下文建模方案进行分析。

表 4-6 本文提出的 *significant_coeff_flag* 上下文模型相对于文献[118]中方法的 BD-Rate[%]

Table 4-6 BD-Rate[%] of the proposed context modeling scheme for *significant_coeff_flag* over that in *Ref*[118]

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	0.0	0.0	0.0	0.0	0.3	0.2			
Class B	-0.1	-0.1	0.0	-0.1	0.1	0.1	-0.1	0.0	0.2
Class C	0.0	-0.1	0.0	0.0	0.1	0.0	-0.1	0.0	-0.3
Class D	0.0	-0.2	0.2	-0.2	0.6	0.1	-0.2	-0.2	-0.2
Class E	0.0	-0.1	-0.2				0.1	0.4	0.3
Overall	**0.0**	**-0.1**	**0.0**	**-0.1**	**0.3**	**0.1**	**-0.1**	**0.1**	**0.0**
Class F	0.1	0.1	0.0	-0.1	-0.1	0.0	0.3	0.2	0.2
UHD	-0.1	-0.1	0.0	0.0	0.1	-0.3	-0.1	0.0	-0.3
Overall(HD)	**-0.1**	**-0.1**	**0.0**	**0.0**	**0.1**	**0.0**	**-0.1**	**0.0**	**0.0**
Enc Time		100%			100%			101%	
Dec Time		100%			100%			100%	

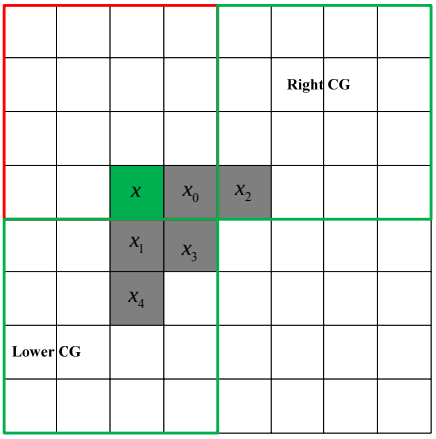


图 4-8 本文提出的上下文模型和 HEVC 中的上下文模型中所使用的相邻系数示意图

Fig 4-8 Illustration of the neighboring coefficients used in this paper and HEVC

HEVC，文献[118]和本文提出的上下文建模方案都把变换系数的频域位置作为一个上下文来挖掘变换系数的统计相关性。在不同的变换系数块中，同一个频域位置处的变换系数的概率分布可能会有很大的不同，例如平滑区域处的变换系数的概率分布不同于非平滑区域处的变换系数的概率分布。所以，除了频域位置之外，还需要其他的信息作为上下文来区分变换系数的概率分布。为此，文献[118]和本文提出的上下文建模方案把当前变换系数的局部模板所覆盖的变换系数作为上下文，而 HEVC 采用的当前 CG 右侧和下侧 CG 的 *CSBF* 取值为上下文。由于当前变换系数的局部模板所覆盖的变换系数距离当前待编码的系数的距离更近，如图 4-8 所示。换句话说，距离当前变换系数较远的变换系数将不再被用作当前变换系数的上下文。因此，文献[118]和本文提出的上下文建模方案取得了更好的编码效率。

为了挖掘不同尺寸的变换系数块中的统计特性，文献[118]把变换系数块的大小也作为一个上下文，这意味着不同大小的变换系数块将使用不同的上下文模型集合。一方面，变换系数块的大小能够以更加精细化的方式来区分 *significant_flag* 的概率分布，所以它能提高上下文建模方案的准确性；另一方面，它将会增加 *significant_flag* 的上下文模型的个数，可能会导致上下文稀疏问题。与文献[118]不同的是，本文把变换系数块分成了不同的区域，每个区域共用一个上下文模型集合。因此，本文提出的上下文建模方案可以避免上下文稀疏问题，从而可以取得更好的编码性能。

（3）非零系数幅值信息的上下文模型的编码性能

表 4-7 本文提出的变换系数幅值的上下文模型相对于 HEVC 的 BD-Rate[%]

Table 4-7 BD-Rate[%] of the proposed context modeling scheme for level information over that in HEVC

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.5	-0.6	-0.5	-0.2	-0.2	0.0			
Class B	-0.4	-0.3	-0.4	-0.2	0.1	-0.4	-0.1	0.0	0.2
Class C	-0.3	-0.4	-0.4	-0.1	0.1	-0.1	0.0	0.1	0.3
Class D	-0.2	-0.4	-0.1	-0.1	0.7	-0.1	-0.1	-0.1	-0.1
Class E	-0.5	-0.6	-0.4				-0.3	0.3	0.3
Overall	**-0.4**	**-0.4**	**-0.3**	**-0.2**	**0.2**	**-0.2**	**-0.1**	**0.0**	**0.2**
Class F	-0.1	-0.2	-0.3	0.1	0.1	0.2	0.0	-0.1	-0.3
UHD	-0.5	-0.5	-0.6	-0.3	0.3	0.2	-0.2	-0.1	0.1
Overall(HD)	**-0.5**	**-0.4**	**-0.5**	**-0.3**	**0.1**	**-0.1**	**-0.1**	**-0.1**	**0.1**
Enc Time		105%			100%			101%	
Dec Time		101%			100%			100%	

表 4-7 给出了本文提出的变换系数幅值（*coeff_abs_greater1_flag* 和 *coeff_abs_greater2_flag*）的上下文模型相对于 HEVC 中的上下文模型的 BD-Rate。从中可以看到，与 HEVC 中的上下文模型相比，本文提出的上下文模型平均可以取得 0.1%~0.4%的性能增益。这主要是因为本文把当前变换系数的局部模板所覆盖的变换系数作为上下文来挖掘变换系数之间的相关性。

表 4-8 本文提出的变换系数幅值的上下文模型相对于文献[118]中方法的 BD-Rate[%]

Table 4-8 BD-Rate[%] of the proposed context modeling scheme for level information over that in *Ref*[118]

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.3	-0.3	-0.1	-0.2	0.0	-0.1			
Class B	-0.3	-0.1	-0.2	-0.2	0.2	-0.2	-0.2	0.0	-0.3
Class C	-0.1	-0.3	0.0	-0.1	-0.1	-0.2	0.0	-0.2	-0.3
Class D	0.0	0.0	-0.1	0.0	1.5	-0.1	0.2	0.1	-0.2
Class E	-0.3	-0.1	-0.3				0.1	0.8	1.9
Overall	**-0.2**	**-0.2**	**-0.1**	**-0.1**	**0.3**	**-0.1**	**0.0**	**0.1**	**0.1**
Class F	0.0	0.0	-0.3	-0.2	0.0	-0.2	0.2	0.3	0.0
UHD	-0.3	-0.2	-0.3	-0.6	-0.2	-0.3	-0.1	-0.2	-0.1
Overall(HD)	**-0.3**	**-0.2**	**-0.2**	**-0.3**	**0.0**	**-0.2**	**-0.2**	**-0.1**	**-0.2**
Enc Time		100%			100%			101%	
Dec Time		100%			100%			100%	

表 4-8 给出了本文提出的变换系数幅值（*coeff_abs_greater1_flag* 和

coeff_abs_greater2_flag）的上下文模型相对于文献[118]中的上下文模型的BD-Rate。从中可以看到，与文献[118]中的上下文模型相比，本文提出的上下文模型平均可以取得 0.0%~0.2%的性能增益；在高分辨率的视频序列上，平均的性能增益是 0.2%~0.3%。具体来说，本文提出的上下文模型在 AI 和 RA 配置下以及 LD 配置下的高分辨率视频上可以取得更高的编码性能；在 LD 配置下的低分辨率的视频上，本文提出的上下文模型取得较差的编码性能。这主要是因为本文提出的上下文模型在编码 *coeff_abs_greater1_flag* 时把 *NumEqu1* 作为额外的上下文，在编码 *coeff_abs_greater2_flag* 时把 *NumGre*1 作为额外的上下文。因此，在编码 LD 配置下的低分辨率视频时，由于缺少训练样本可能会产生上下文稀疏问题，进而使得编码性能变差。但是，在编码 AI 和 RA 配置下，由于存在足够多的训练样本使得每个上下文模型能够达到概率稳定状态，所以本文提出的上下文模型取得了更高的编码性能。

虽然本文提出的 *coeff_abs_greater1_flag* 和 *coeff_abs_greater2_flag* 上下文建模方案引入了额外的上下文依赖关系，即 *coeff_abs_greater1_flag* 依赖于 *significant_flag*，*coeff_abs_greater2_flag* 依赖于 *coeff_abs_greater1_flag*。但是根据公式（4-8）和（4-9）可知，在编码 *coeff_abs_greater1_flag* 时，只有在 *NumGre1* 等于 0 时，才会使用 *NumEqu1*；在编码 *coeff_abs_greater2_flag* 时，只有在 *NumGre2* 等于 0 时，才会使用 *NumGre1*。因此本文提出上下文建模方案可以尽量地降低这种上下文依赖关系。

4.3.2 低内存消耗的二进制算术编码引擎

表 4-9 本文提出的算术编码引擎相对于 M-coder 的 BD-Rate[%]

Table 4-9 BD-Rate[%] of the proposed binary arithmetic coding engine over M-coder

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.8	-1.2	-0.8	-0.8	-0.9	-0.5			
Class B	-0.7	-0.8	-0.6	-0.7	-0.3	-0.2	-0.7	-0.2	0.2
Class C	-0.7	-0.8	-0.7	-0.5	-0.4	-0.4	-0.4	0.0	0.2
Class D	-0.6	-0.7	-0.4	-0.4	-0.5	-0.3	-0.4	-0.6	0.5
Class E	-0.7	-1.3	-0.8				-0.5	-1.0	0.1
Overall	**-0.7**	**-0.9**	**-0.6**	**-0.6**	**-0.5**	**-0.3**	**-0.5**	**-0.4**	**0.3**
Class F	-0.5	-0.5	-0.4	-0.2	0.0	0.2	-0.4	-0.8	-0.9
UHD	-0.8	-0.7	-0.7	-0.8	-0.1	-0.4	-0.8	-0.3	-0.6
EncTime		108%			103%			103%	
DecTime		102%			100%			101%	

为了衡量本文提出的算术编码引擎的编码性能，本小节通过比较本文提

出的算术编码引擎，M-coder 和文献[96]所提出的算术编码引擎来衡量本文提出的算术编码引擎的编码性能。文献[96]中的算术编码引擎包括基于查表实现的算术编码引擎和基于位宽等于 15×9 的乘法操作的算术编码引擎。

表 4-9 给出了本文提出的算术编码引擎相对于 M-coder 的 BD-Rate。从中可以看到，相比于 M-coder，本文提出的算术编码引擎平均可以取得 0.5%~0.7%的性能增益。这主要是因为本文提出的算术编码引擎采用多参数的概率估计模型使得概率更新过程更加准确。

表 4-10 本文所提出的算术编码引擎相对于文献[96]中基于查表实现的算术编码引擎的 BD-Rate[%]

Table 4-10 BD-Rate[%] of the proposed binary arithmetic coder over that in *Ref*[96] with look-up table.

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01			
Class B	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02
Class C	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.02	-0.02	-0.02
Class D	-0.03	-0.02	-0.02	-0.02	-0.02	-0.02	-0.03	-0.03	-0.03
Class E	-0.04	-0.03	-0.03				-0.03	-0.03	-0.03
Overall	**-0.02**	**-0.02**	**-0.02**	**-0.02**	**-0.02**	**-0.01**	**-0.02**	**-0.02**	**-0.02**
Class F	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.00	0.00	0.00
UHD	-0.01	0.00	0.00	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
EncTime		93%			98%			99%	
DecTime		96%			99%			99%	

表 4-10 给出了本文提出的算术编码引擎相对于文献[96]中基于查表操作的算术编码引擎的 BD-Rate。从中可以看到，相比于文献[96]中基于查表操作的算术编码引擎，本文提出的算术编码引擎可以取得略好的编码性能。这说明了本文提出的低位宽的乘法可以取得比较准确的编码区间分割精度。在编解码时间上，本文提出的算术编码引擎要低于文献[96]中基于查表操作的算术编码引擎。这是因为文献[96]中基于查表操作的算术编码引擎需要一个尺寸很大的表格来实现编码区间分割，在编解码过程中，处理器无法把该表中的所有内容都预取到缓存，因此，如果处理器想要访问的内容不在缓存中，就需要从内存重新取到缓存，这将消耗更多的时间。另外由于 AI 配置只允许使用帧内预测模式，因此，变换系数所占的比例更大，进而调用算术编码引擎的次数更多；加之 AI 配置下的编解码时间比较短，所以在 AI 配置下，编解码时间的下降的比例要大于 LD 和 RA 配置下。

表 4-11 给出了本文提出的算术编码引擎相对于文献[96]中基于位宽为 15×9 的乘法器实现的算术编码引擎的 BD-Rate。从中可以看到，本文提出的算术编码引擎可以取得与这个算术编码引擎相似的编码性能。这说明本文提出的位宽为 9×5 的乘法在编码区间分割过程可以取得位宽为 15×9 的乘法相似的精度。

表 4-11 本文提出的算术编码引擎相对于文献[96]中基于位宽为 15×9 乘法器的算术编码引擎的 BD-Rate[%]

Table 4-11 BD-Rate[%] of the proposed arithmetic coder over that in *Ref*[96] with bit capacity equal to 15×9

	All Intra (AI)			Random Access (RA)			Low Delay (LD)		
	Y	U	V	Y	U	V	Y	U	V
Class A	0.00	0.00	0.00	0.00	0.00	0.00			
Class B	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Class C	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Class D	-0.01	-0.01	-0.01	0.00	0.00	0.00	0.00	0.00	0.00
Class E	-0.01	0.00	-0.01				0.02	0.02	0.02
Overall	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Class F	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.03	0.03
UHD	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
EncTime		98%			97%			98%	
DecTime		100%			99%			99%	

4.3.3 优化后的熵编码模块的整体性能

表 4-12 优化后的熵编码模块相对于 HEVC 的 BD-Rate[%]

Table 4-12 BD-Rate[%] of the proposed two techniques over that in HEVC

	All Intra (AI)			Low Delay(LD)			Random Access (RA)		
	Y	U	V	Y	U	V	Y	U	V
Class A	-1.7	-1.7	-1.5				-1.4	-1.1	-1.0
Class B	-1.5	-1.5	-1.5	-1.0	-0.6	-0.4	-1.2	-0.9	-0.9
Class C	-1.3	-1.1	-1.1	-0.9	-0.1	-0.3	-1.1	-0.8	-0.8
Class D	-1.2	-0.9	-0.6	-1.0	-0.6	-0.3	-1.0	0.2	-0.8
Class E	-1.7	-1.9	-1.5	-0.9	-1.9	-0.6			
Overall	**-1.4**	**-1.4**	**-1.2**	**-0.9**	**-0.7**	**-0.4**	**-1.1**	**-0.6**	**-0.9**
Class F	-2.0	-2.0	-2.0	-1.6	-1.8	-2.3	-1.9	-1.5	-1.3
UHD	-1.4	-1.6	-1.6	-1.3	-0.2	-0.3	-1.2	-0.1	-0.3
EncTime		119%			106%			107%	
DecTime		104%			102%			101%	

由于变换系数的增强上下文模型是通过引入额外的上下文更加高效地挖

掘变换系数之间的统计相关性，而低内存消耗的二进制算术编码引擎是通过使用多参数概率估计模型和低位宽的乘法操作来分别提高概率估计模型和编码区间分割的准确性。因此，这两个技术的编码性能增益是可以叠加的，即同时使用这两个技术可以进一步提高熵编码模块的编码性能。表 4-12 给出同时使用变换系数的增强上下文模型和低内存消耗的二进制算术编码引擎的熵编码模块相对于 HEVC 中原始的熵编码模块的编码性能。从中可以看到，与 HEVC 中原始的熵编码模块相比，同时采用这两个技术的熵编码模块平均可以取得 0.9%~1.4%的编码性能增益，该实验结果表明这两个技术的编码性能增益是可以叠加的。

4.4 本章小结

本章以 HEVC 中熵编码模块为基础，提出了一个变换系数的增强上下文模型和一个低内存消耗的算术编码引擎。前者采用当前变换系数的局部模板所覆盖的变换系数和当前变换系数的频域位置作为上下文，提高了变换系数的上下文建模模块的编码性能。后者采用了多参数的概率更新模型来提高概率更新模块的准确性，并且在编码区间分割中引入了一个低位宽的乘法器来降低算术编码引擎的内存消耗。这两个技术可以分别提高 HEVC 的编码效率，并且它们的编码增益是可以叠加的。

第5章 AVS2 中熵编码模块的优化设计

在我国第二代视频压缩标准 AVS2 中，熵编码模块存在着很强的顺序依赖关系，这些顺序依赖关系严重地制约着 AVS2 编解码器的并行性设计。这些顺序依赖关系主要来源于算术编码引擎的归一化过程和 bypass bin(概率等于 0.5 的二进制符号)的编码过程以及变换系数的上下文建模过程。为了解决这些问题，本章从上述三个方面对 AVS2 的熵编码模块进行了优化设计，来提高 AVS2 的熵编码模块的数据吞吐率。

5.1 AVS2 中熵编码模块的概述

本小节将详细介绍 AVS2 中的熵编码技术：包括二进制算术编码引擎的编解码流程以及变换系数的上下文建模方案，然后分析指出其中存在的设计缺点对编解码器的硬件实现带来的影响。

5.1.1 AVS2 中二进制算术编码引擎

AVS2 的算术编码引擎是基于对数域的算术编码引擎，其中编码区间的长度 R 和 MPS 的概率 p_{MPS} 都是以对数的形式来存储的，即 LG_R 和 LG_p_{MPS}。LG_R 又被分成整数部分 s 和小数部分 t。也就是说，AVS2 的算术编码引擎所存储的数据是 s，t 和 LG_p_{MPS}。根据 2.4 节关于该算术编码引擎的描述，算法 5-1 给出了它的编码区间分割过程。其中“>>”表示向右移位操作，bin 是当前要编码的二进制符号，R_1 是编码之前编码区间的长度，s_1 和 t_1 分别是 LG_R_1 的整数部分和小数部分，R_2 是编码之后与 MPS 对应的编码区间的长度，s_2 和 t_2 分别是 LG_R_2 的整数部分和小数部分，rLPS 表示与 LPS 对应的编码区间的长度。由算法 5-1 可知，如果 bin 是 MPS，则下一次编码的编码区间就是此时与 MPS 对应的编码区间，因此只需把 s_2 和 t_2 分别幅值给 s_1 和 t_1；如果 bin 是 LPS，则下一次编码的编码区间就是与 LPS 对应的编码区间，其长度为 $rLPS$。这时候需要先求解 $rLPS$ 的取值，然后再求解 $rLPS$ 的对数来实现 s_1 和 t_1 赋值。为了实现这个目的，该过程首先根据公式（2-6）和公式（2-9）来计算得到 $rLPS$；然后调用归一化过程 *Normalization*()来向码流中输出比特，同时调整 $rLPS$ 的取值；最后求解 $rLPS$ 的对数实现对 s_1 和 t_1 赋值。为了实现从实数域到对数域的转换，AVS2 中的算术编码引擎使用了公式（5-1）中的近似关系来实现这种转换，其中，$\log(\bullet)$ 表示以 2 为底的对数。

$$\log(1+x) \approx x, \quad 0 \le x \le 1 \tag{5-1}$$

算法 5-1 AVS2 的算术编码引擎中编码区间分割过程

Algorithm 5-1 The partition of coding interval in the arithmetic coding in AVS2

输入： bin, LG_p_{MPS}, s_1 和 t_1

输出： s_1 和 t_1

```
If t1 >= (LG_pMPS >> 2), then
    s2 = s1;
    t2 = t1 - (LG_pMPS >> 2);
    s_flag = 0;
Else
    s2 = 1 + s1;
    t2 = 256 + t1 - (LG_pMPS >> 2);
    s_flag = 1;
End If
If bin==MPS, then
    s1 = s2;
    t1 = t2;
Else
    If s_flag == 0, then
        rLPS = LG_pMPS >> 2;
    Else
        rLPS = t1 + (LG_pMPS >> 2);
    End If
    rLPS = Normalization(rLPS);
    s1 = 0;
    t1 = rLPS & 0xff;
End If
```

基于公式（5-1），AVS2 中的算术编码引擎规定 256 对应[0, 1]区间中的 1，并且调用归一化过程之后 $rLPS$ 满足 $rLPS \geq 256$。因此 $rLPS$ 的对数可以通过公式（5-2）来近似求得，

$$\log(rLPS) = \log(256 + \Delta_{rLPS}) \approx \Delta_{rLPS} \qquad (5\text{-}2)$$

其中 Δ_{rLPS} 对应 $\log(rLPS)$ 的小数部分，它的取值范围满足 $0 \leq \Delta_{rLPS} \leq 255$；此时由于 $rLPS < 512$ 即 $rLPS < 2.0$，所以 $\log(rLPS)$ 整数部分为 0。因此，LPS 编码结束后，$s_1 = 0$ 且 $t_1 = rLPS \,\&\, 0\text{xff}$。

图 5-1 给出了 AVS2 的算术编码引擎编码一个二进制符号的详细流程。随着编码区间的递归分割，编码区间将越来越小。为了避免编码区间发生下溢，当 $rLPS$ 小于 0x100 时，算术编码引擎会使用 while 循环来扩展 $rLPS$ 的长度同时增加 s_2 的取值；在向码流中输出比特时，算术编码引擎通过 s_2 来控

制向码流中输出比特的个数，并且每次循环只移出一个比特；最后，算术编码引擎还需要一个 while 循环来恢复编码区间的边界 low 的取值。这个过程对应着算法 5-1 中的归一化过程 *Normalization*()。

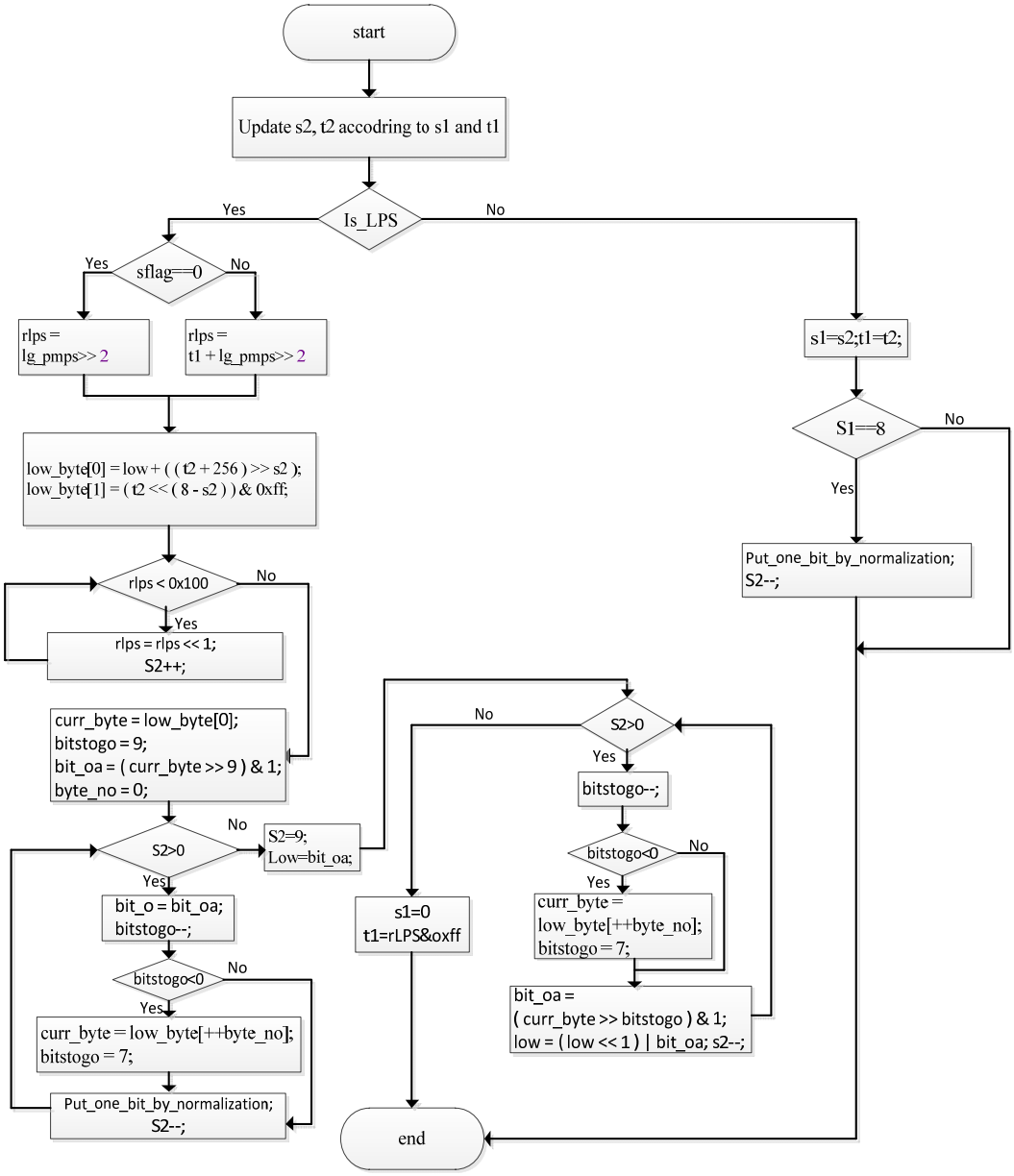


图 5-1 AVS2 中算术编码引擎的编码流程图

Fig 5-1 The encoding scheme of binary arithmetic coding engine in AVS2

图 5-2 给出了 AVS2 中的解码端的详细流程图。从中可以看到，解码端

的归一化过程也是通过 while 循环来调整 rLPS 的取值。

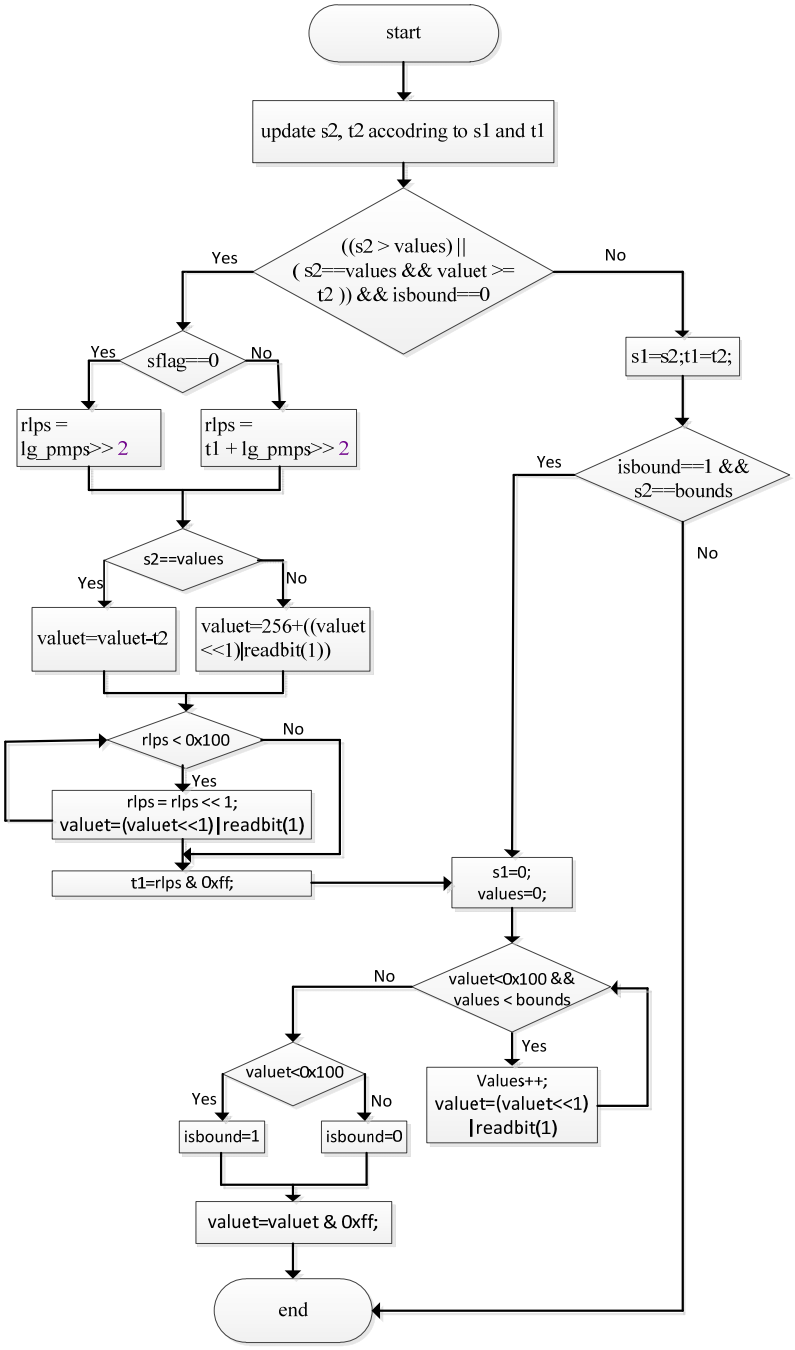


图 5-2 AVS2 中算术编码引擎的解码流程图

Fig 5-2 The decoding scheme in AVS2 binary arithmetic decoding

在图 5-1 和图 5-2 中，由于 while 循环需要不断地执行条件判断来确定循环是否结束，所以这种操作不利于并行处理。另外，每当需要向码流中输出

比特时，AVS2 的算术编码引擎就会调用归一化过程。这种按位操作的归一化过程不但严重地制约着 AVS2 的算术编码引擎的吞吐率的提高，而且也增加了算术编码引擎的硬件设计复杂度。

由于 bypass bin 的概率等于 0.5（其信息熵为 1），所以，只需从码流中读出一个比特同时保持编码区间不变，即可完成 bypass bin 的解码。但是 AVS2 的算术编码引擎在解码 bypass bin 时产生了不确定性，无法确定从码流中读入的比特数，因此 AVS2 的算术编码引擎必须调用图 5-1 或图 5-2 的编解码过程来编码或解码 bypass bin。具体来讲，在 AVS2 的算术编码引擎中，$LG_p_{MPS}=1023$表示 $p_{MPS}=0.5$，并且t_1的取值小于等于 255。因此，在编码 bypass bin 时，若t_1等于 255 时，根据算法 5-1 可知s_1等于s_2，由图 5-2 可知，此时解码器可能会读入比特，也可能不会读入比特；若t_1小于 255 时，根据算法 5-1 可知s_1等于s_2+1，由图 5-2 可知，此时解码器可能读入 1 个比特（$t_1>0$）或读入 2 个比特（$t_1==0$）。正是由于这种不确定性，使得 AVS2 中的算术编码引擎必须调用图 5-1 或图 5-2 的编解码过程来编码或解码 bypass bin。这种设计严重地制约着 AVS2 中算术编码引擎的吞吐率的提高。

5.1.2 AVS2 中变换系数的上下文建模方案

在 AVS2 中，变换系数是以$(Run, Level)$二元组的形式来编码的，其中 *Level* 表示非零系数的幅值，*Run* 表示两个连续非零系数之间零系数的个数。它们的编码顺序是，先编码 *Level*，其次编码 *Level* 的符号 *Sign*，最后编码 *Run*。由于 AVS2 中变换系数块包括 4×4,8×8,16×16 和 32×32 的变换系数块。因此 AVS2 采用基于系数组（CG：Coefficient Group）变换系数编码方案，即变换系数块首先被分成 4×4 大小的 CG，然后按照逆向扫描顺序来处理每个 CG。每个 CG 内的$(Run, Level)$二元组也是按照逆向扫描顺序来编码。

在编码 *Run* 和 *Level* 之前，AVS2 采用一元码把它们二值化成二进制符号序列。在编码 *Level* 的每个二进制符号时，考虑到变换系数的频域位置与变换系数之间的统计相关性以及变换系数之间的统计相关性，在 AVS2 中，*Level* 的上下文包括当前 CG 的索引 *CGIdx*，当前 *Level* 在当前 CG 内的位置 *pos* 和当前 CG 内已经编码的$(Run, Level)$二元组的个数 *pairsInCG* 以及已经编码过的 *Level* 的最大值 *Lmax*。根据 *CGIdx* 和 *pos*，AVS2 把变换系数块分成两个区域，不同的区域使用不同的上下文模型集合，如此设计是为了利用变换系数的频域位置与变换系数之间的统计相关性；在选中上下文模型集合之后，AVS2 使用当前 CG 内已经编码的非零系数的个数即 *pairsInCG* 和 *Lmax* 从该

上下文模型集合中选择最终要使用的上下文模型，这样就可以利用变换系数之间的统计相关性。算法 5-2 给出了 *Level* 的上下文模型选择过程，其中亮度分量和色度分量的上下文选择过程一样，但是使用的上下文模型集合是不同的。

算法 5-2 AVS2 中 *Level* 的上下文模型选择算法

Algorithm 5-2 Context modeling for *Level* in AVS2

输入：CGIdx, pos, pairsInCG 和 Lmax
输出：CtxInc

```
If CGIdx==0 && pos < 2, then
    regionIdx = 0;
Else
    regionIdx = 1;
End If
pairsInCGIdx = min((pairsInCG + 2) / 2, 2);
If Lmax == 0, then
    rank = 0;
Else If Lmax == 1, then
      rank = 1;
    Else If Lmax == 2, 则
          rank = 2;
        Else If Lmax == 3 or Lmax == 4, then
              rank = 3;
            Else
              rank = 4;
            End If
        End If
    End If
End If
CtxInc = 10×regionIdx + min(rank, pairsInCGIdx + 2) + (5×pairsInCGIdx)/2;
```

在编码 *Run* 的每个二进制符号时，它的上下文包括先前已经编码的 6 个 $(Run, Level)$ 二元组中的 *Level* 的绝对值之和 *absSum* 以及当前 $(Run, Level)$ 二元组中 *Level* 的绝对值 *absLevel*，当前二进制符号的索引 *binidx*，当前 CG 的索引 *CGIdx* 和当前 Level 在当前 CG 内的位置 *pos*。AVS2 选择这些信息作为 *Run* 的上下文也是为了挖掘变换系数的频域位置与变换系数之间的统计相关性以及变换系数之间的统计相关性。根据 *CGIdx*，*pos* 和 *binidx*，AVS2 利用 *Run* 的每个二进制符号（对应着取值为零的变换系数）的频域位置把它们划分在不同的区域中，并且规定不同的区域使用不同的上下文模型集合；然后 AVS2 利用已经编码过的非零系数的绝对幅值从选中的上下文模型集合中选

择最终要使用的上下文模型。算法 5-3 给出了 *Run* 的上下文模型选择过程，其中 *IsLuma==true* 表示当前编码的是亮度分量，否则表示当前编码的是色度分量。

算法 5-3 AVS2 中 *Run* 的上下文模型选择算法

Algorithm 5-3 Context modeling for *Run* in AVS2

输入：CGIdx, binidx, pos, absSum, absLevel 和 IsLuma
输出：CtxInc

```
If CGIdx==0 then
   If binidx+pos==14 then
      regionIdx = 0;
   Else If 9<=binidx+pos<14 then
            regionIdx = 1;
        Else
            regionIdx = 2;
        End If
   End If
Else
      If binidx+pos>=9 then
            regionIdx = 3;
      Else
            regionIdx = 4;
      End If
End If
If IsLuma==true then
      CtxInc=3×regionIdx+min(2, (absLevel+absSum)/2);
Else
      CtxInc=3×min(2,regionIdx)+min(2, (absLevel+absSum)/2);
End If
```

通过算法 5-2 可知，*Level* 的上下文模型选择需要 *pos* 来做上下文，而 *pos* 是通过 *Run* 计算得到的，因此 *Level* 的上下文模型选择要依赖于 *Run*。另外由算法 5-3 可知，*Run* 的上下文选择需要 *absSum* 和 *absLevel*，而这两个上下文的计算都依赖于 *Level*，所以 *Run* 的上下文模型选择又依赖于 *Level*。因此，*Run* 和 *Level* 的编码过程是交叉的。由于 *Run* 和 *Level* 使用的是不同的上下文模型集合，所以，为了并行解码 *Run* 和 *Level*，需要大量的推导计算来计算解码过程可能需要的上下文模型索引，并且还需要大量的存储空间来保存这些可能被使用的上下文模型索引，所以，这种交叉的编码方式不利于熵编码模块的并行设计。

5.2 AVS2 中熵编码器的优化设计

针对 AVS2 中算术编码引擎存在的问题，本文提出了一个二进制算术编码引擎的快速归一化方法和 bypass bin 的快速编解码方法来提高 AVS2 中算术编码引擎的数据吞吐率和降低其硬件实现的复杂度。另外针对 AVS2 中变换系数的交叉编解码方法，本文提出一种 *Run* 和 *Level* 的层次编解码方案来降低其硬件设计复杂度。

5.2.1 AVS2 中算术编码引擎的优化设计

（1）二进制算术编码引擎的快速归一化方法

在 AVS2 的算术编码引擎中，只有当 $rLPS$ 处于 $1 \le rLPS \le 255$ 这个范围时，才需要调用图 5-1 中的 while 循环来调整 $rLPS$ 的取值。因此，可以定义一个包含 128 个元素的表格 *norm* 来存储不同 $rLPS$ 所需要移出的比特数，

$$nl = norm[rLPS >> 1] \tag{5-3}$$

其中 nl 为编码区间长度等于 $rLPS$ 时所移出的比特数，例如当 $rLPS = 1$ 时 nl 等于 8；当 $rLPS = 32$ 时 nl 等于 3。这样设计后，只需通过查表即可知道需要移出的比特数，而不再需要 while 循环了。

图 5-1 所示编码端的归一化过程可以认为是在对数域执行的，因为它是根据 s_2 和 t_2 来控制比特的输出，而不是通过编码区间的边界 *low*；同理，图 5-2 所示解码端的归一化过程也可以认为是在对数域中执行的，因为它也是通过 *values* 和 *valuet* 来控制比特读入，而不是实数域中的 *value*。所以，本文提出的快速归一化方法首先把对数域转化为实数域，然后在实数域执行归一化过程，即在编码端通过编码区间的下边界 *low* 来控制比特输出，在解码端通过 *value* 来控制比特读入。

由图 5-1 所示的编码端，如果当前编码的是 MPS，则只有在 s_1 等于 8 时才会调用归一化过程，向码流中输出比特；同理如图 5-2 所示的解码端，如果当前解码的是 MPS，则只有在某些条件下才会从码流中读入比特。这样设计可以减少调用归一化的次数，从而降低程序执行时间。尽管如此，这样的设计仅仅在编码 MPS 时降低了调用归一化的次数，在编码 LPS 的时候仍然需要调用归一化过程；并且这样的设计使得 s_1 的取值可以是[0, 7]的任意整数，因而导致编码器和解码器的执行流程变得非常复杂，比如编码端在编码 LPS 时需要额外的 while 循环来恢复编码区间的下边界 *low*，解码端也需要额外的变量 *bounds* 来控制比特的读入。所以，为了简化对数域到实数域之间的转化

并且降低 *low* 和 *value* 的位宽，本文对此作了修改，即如果当前编码的是 MPS，则 s_1 等于 1 时就会调用归一化过程，向码流中输出比特。此时 *low* 和 *value* 的位宽均为 11 位。为了避免按位调用归一化过程，本文引入了 *BitsLeft* 来表示编码端/解码端需要向码流中输出/读入的比特数，即 *BitsLeft* 减 1 表示要向码流中输出一个比特。同时，本文提出的快速归一化方法是当缓存的比特数等于 16 时才真正地调用一次归一化过程向码流中输出 16 个比特。为了实现这个目的，本文把编码区间的边界 *low* 的位宽从 11 位扩展为 11+16 位，解码端的 *value* 的位宽也从 11 为扩展为 11+16 位。

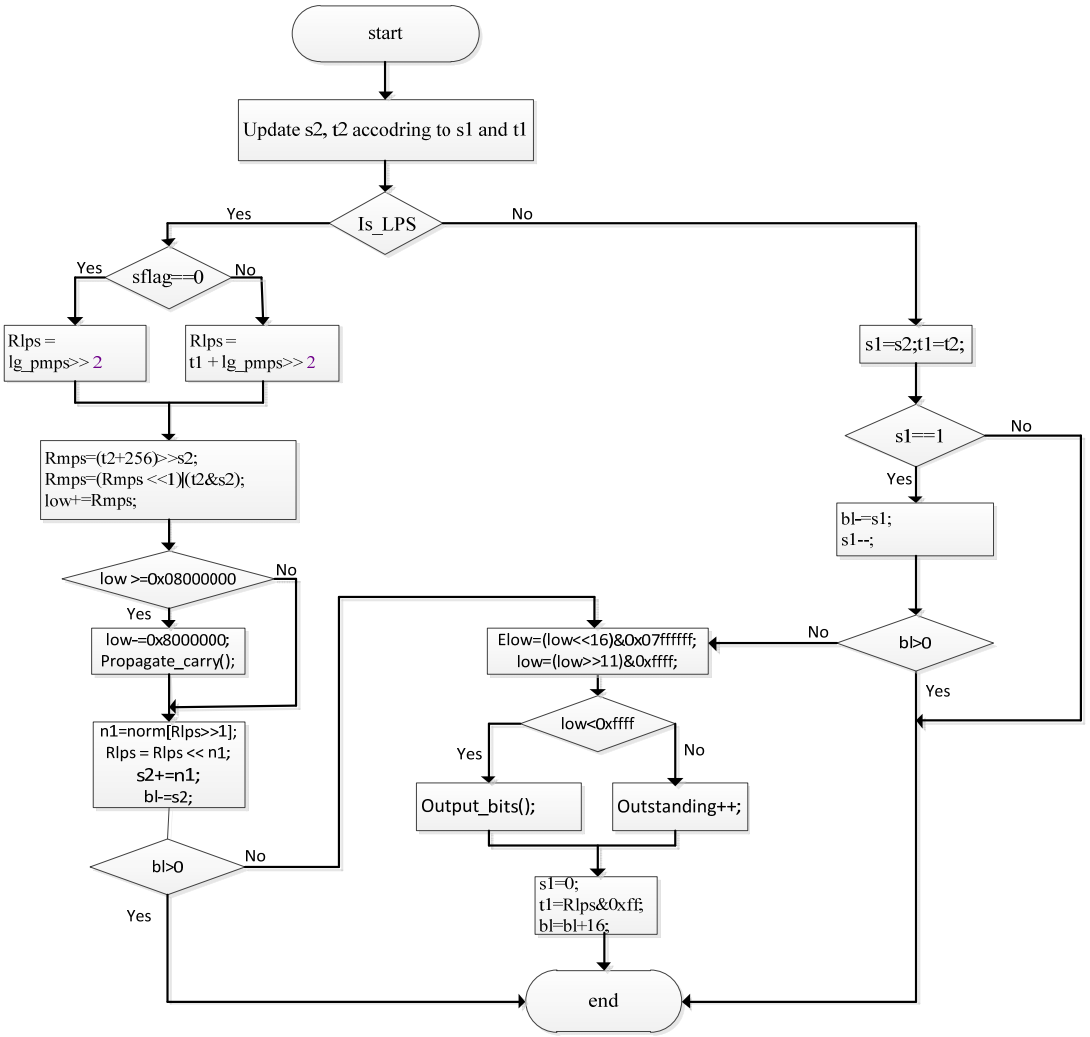


图 5-3 改进后的 AVS2 中算术编码引擎的编码端执行流程图。

Fig 5-3 The encoding scheme of the improved binary arithmetic coding engine in AVS2

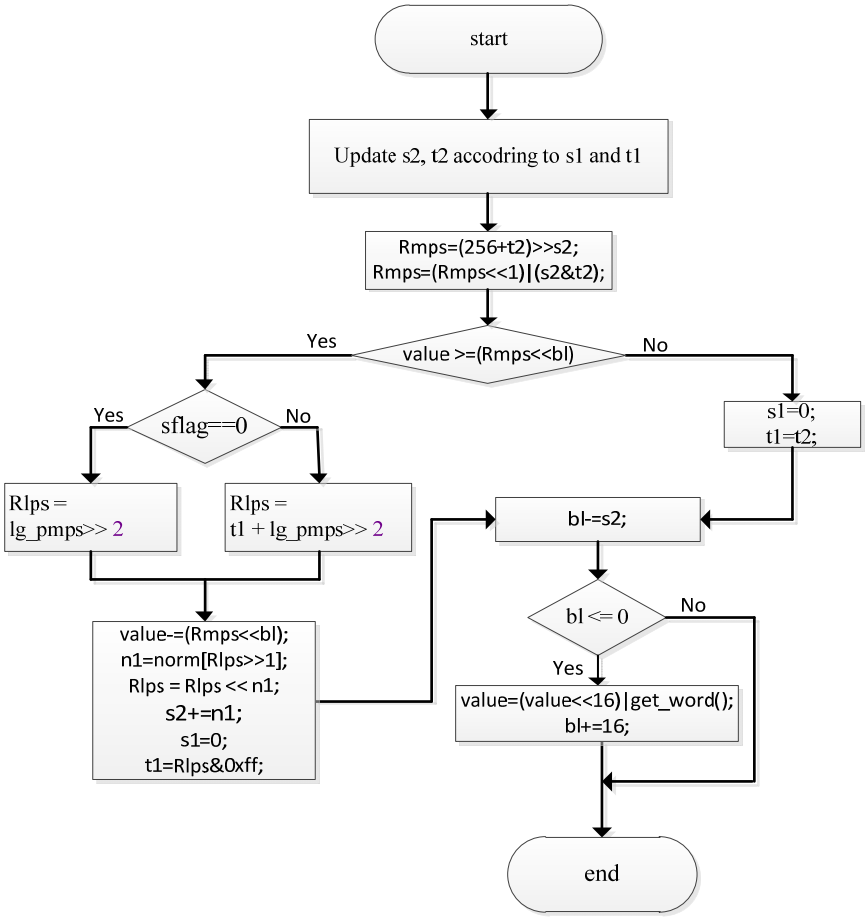


图 5-4 改进后的 AVS2 中算术编码引擎的解码端执行流程图

Fig 5-4 The decoding scheme of the binary arithmetic coding engine in AVS2

图 5-3 列出了改进后的 AVS2 中算术编码引擎的编码端执行流程图，其中 *bl* 表示 *BitsLeft*，*Rmps* 和 *Rlps* 分别表示与 MPS 和 LPS 对应的编码区间的长度。图 5-4 列出了改进后的 AVS2 中算术编码引擎的解码端执行流程图。

（2）Bypass Bin 的快速编解码算法

5.1.1 节已经分析了为什么 AVS2 中算术编码引擎必须调用图 5-1 和图 5-2 所示的编码或解码流程来处理 bypass bin。为了解决这个问题，本文重新定义表示编码区间长度的 s 和 t，即 $s \geq 0$ 和 $t \leq 254$。在实现过程中 s 和 t 的初始值分别为 0 和 254，并且在算法 5-1 所示的编码区间分割流程中，当 $t_1 < LG\_p_{MPS}$ 时，s_2 和 t_2 的计算方式修改为如下方式，

$$\begin{cases} s_2 = s_1 + 1 \\ t_2 = 255 + t_1 - LG\_p_{MPS} \end{cases} \tag{5-4}$$

在这种情况下，R_{LPS} 的计算方式也需要做些修改来保证 R_{LPS}+R_2==R_1，如公式（5-5）所示，

$$R_{LPS}=1+t_1+LG\_p_{MPS} \tag{5-5}$$

这样修改以后，当编码 bypass bin 时，会保证 $s_2=s_1+1$，并且 $R_{LPS}=1+t_1+255\geq 256$，根据图 5-3 的编码流程可知，此时编码器只向码流中输出一个比特。同理，在解码端，如图 5-4 的解码流程可知，此时解码器也只从码流中读入一个比特。所以此时 bypass bin 的编解码过程仅仅需要移位和加法操作即可完成，如图 5-5 所示。如此设计之后，LG_p_{MPS} 仍然只需 10 比特来表示，即 $LG_p_{MPS}=1023$仍表示 p_{MPS}=0.5。

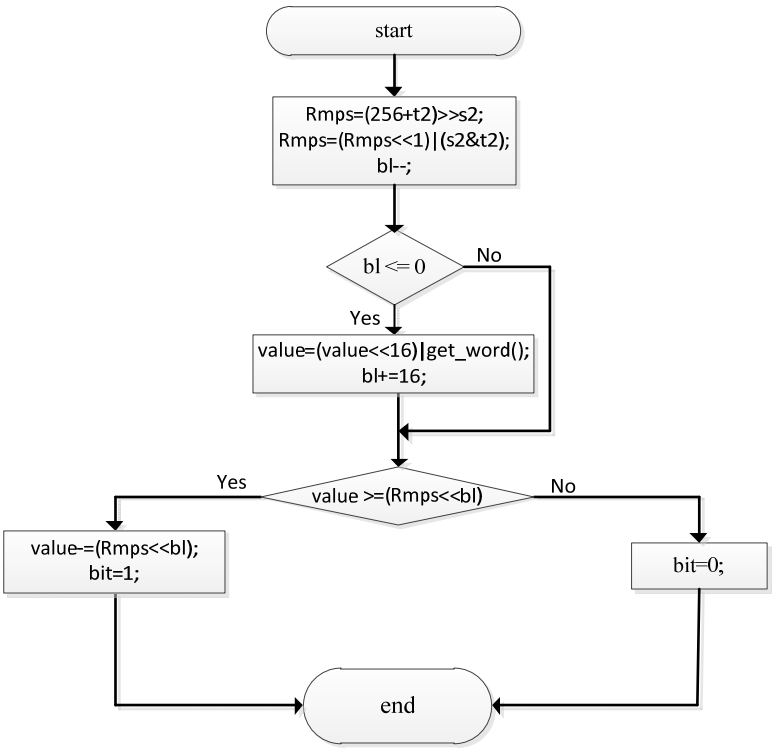


图 5-5 改进后 AVS2 中的 bypass bin 的解码流程

Fig 5-5 The decoding scheme for bypass bin in the improved AVS2 arithmetic coding engine

5.2.2 AVS2 中变换系数的上下文建模过程的优化

为了打破 *Run* 和 *Level* 以及 *Sign* 之间的交叉编码，本文首先针对 *Run* 提出了一个新的上下文模型选择方案，然后修改了 *Run*，*Level* 和 *Sign* 的编码顺序。

本文仍然采用一元码来对 *Run* 进行二值化。在编码 *Run* 的每个二进制符号时，它的上下文包括当前二进制符号的索引 *binIdx*，当前 (*Run*,*Level*) 二元组中 *Level* 在 CG 中的位置 *pos*，当前 CG 的索引 *CGIdx* 和同一个 CG 内已经编码的非零系数的个数 *numSigs*。*Run* 的每个二进制符号的上下文模型选择过程如算法 5-4 所示，

算法 5-4 本文提出的 AVS2 中 *Run* 的上下文模型选择算法

Algorithm 5-4 The proposed context modeling for *Run* in AVS2

输入：CGIdx, binIdx, pos, numSigs 和 IsLuma
输出：CtxInc

```
px = AVS_SCAN4x4[pos+binIdx][0];
py = AVS_SCAN4x4[pos+binIdx][1];
If IsLuma==true then
   If CGIdx==0 then
      regionIdx = Temp1[py][px];
   Else If CGIdx==1 then
            regionIdx = Temp2[py][px];
         Else
            regionIdx = 4;
         End If
   End If
   CtxInc = 4×regionIdx+min(3,numSigs/2);
Else
      If CGIdx==0 then
         regionIdx = Temp3[py][px];
      Else
         regionIdx = 2;
      End If
      CtxInc = 4×regionIdx+min(3,numSigs/2);
End If
```

其中 *IsLuma* 等于 true 表示当前编码的是亮度分量，否则当前编码的是色度分量；AVS_SCAN4x4 是按照 4×4 大小块把 zig-zag 扫描位置转化为(x,y)坐标位置；*Tempt1~Temp3* 分别是一些模板，它们的内容为如图 5-6 所示。

对 *Run* 的上下文模型选择过程做如此修改之后，*Run* 的编码过程将不再依赖于 *Level*，因此 *Run*，*Level* 和 *Sign* 可以分层进行编码，即先编码所有的 *Run*，然后编码所有的 *Level*，最后编码所有的 *Sign*。改进后的 AVS2 中变换系数的编码过程如算法 5-5 所示。

算法 5-5 改进后 AVS2 熵编码模块的解码算法描述

Algorithm 5-5 The description of the improved AVS2 entropy coding scheme

输入：码流 str, CGIdx, numOfCoeffInCG, pos, IsLuma
输出：DCT_Level[0…15], DCT_Run[0…15]

```
初始化：pairsInCG=0, posRun=pos, posLevel=pos;
For i←0 to numOfCoeffInCG do
    symbol = 0;
    binIdx = 0;
    numSigs = getNumSigs(i);
    ctxInc = getCtxforRun(binIdx, CGIdx, posRun, numSigs, IsLuma);
    bin = decode_bin(str, ctxInc);
    while bin == 0 do
        symbol++;
        binIdx++;
        ctxInc = getCtxforRun(binIdx, CGIdx, posRun, numSigs, IsLuma);
        bin = decode_bin(str, ctxInc);
    End while
    DCT_Run[i] = symbol;
    posRun = UpdatePos(DCT_Run[i]);
End For
Lmax = 0;
For i←0 to numOfCoeffInCG do
    symbol = 0;
    ctxInc = getCtxforLevel(CGIdx, posLevel, pairsInCG, Lmax);
    bin = decode_bin(str, ctxInc);
    while bin == 0 do
        symbol++;
        bin = decode_bin(str, ctxInc);
    End while
    DCT_Level[i] = symbol + 1;
    posLevel = UpdatePos(DCT_Run[i]);
    Lmax = UpdateLmax(DCT_Level[i]);
End For
For i←0 to numOfCoeffInCG do
    bin = decode_bin_eq_prob(str);
    If bin == 1 then
      DCT_Level[i] = - DCT_Level[i];
    End If
End For
```

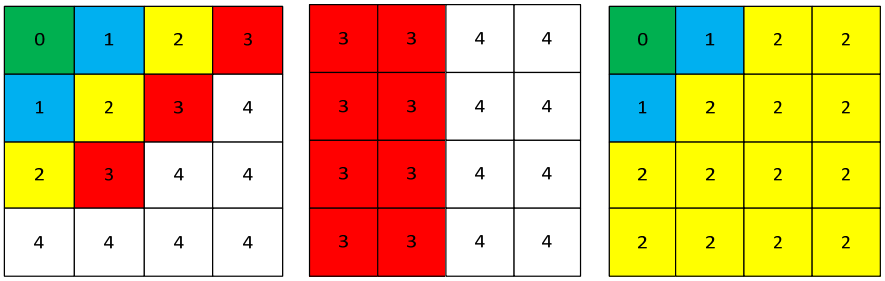


图 5-6 模板 *Tempt1~Tempt3*

Fig 5-6 Illustration of *Tempt1~Tempt3*

在算法 5-5 中，函数 decode_bin()和 decode_bin_eq_prob()分别表示采用上下文模型和等概率编码方式的编码函数，函数 getCtxforRun()和 getCtxforLevel()的作用分别是为 *Run* 和 *Level* 中的二进制符号来选择合适的上下文模型；函数 getNumSigs()是为了计算 *numSigs*，函数 UpdatePos()和 UpdateLmax()分别是为了更新 *pos* 和 *Lmax*。由于同类语法元素（*Run* 或者 *Level*）采用的是同一个上下文模型集合，所以，这样分层的编码流程能够把同类语法元素组织在一起来编码，因而可以最大限度地减少并行解码过程中所需要的推导计算，更适合编解码器的并行设计。

5.3 实验结果

本节通过比较优化后的 AVS2 的熵编码方案与原始的熵编码方案来评价优化后的熵编码方案的编码性能和编码时间。本实验采用 AVS2 参考软件 RD8.0 作为测试平台，测试条件是 AVS2 工作组制定的通用测试条件，包括全帧内编码（AI：All Intra），低延迟编码（LD：Low Delay）和随机访问编码（RA：Random Access）。本文在 20 个标准测试序列上进行了测试，它们的分辨率是从 WQVGA 到 UHD，每个测试序列用 4 个 QP 点进行编码，它们是 27，32，38 和 45。

5.3.1 优化后的二进制算术编码引擎

（1）算术编码引擎的快速归一化方法

本文所提出的算术编码引擎的快速归一化方法是一个与标准兼容的方法，因此，它的编码性能与原始算术编码引擎一样。由于算术编码引擎的快速归一化算法的主要目的是减少编码过程中调用归一化过程的次数，进而减少算术编码引擎消耗的时间。在 720p，1080p 和 UHD 三类测试序列上，本文提

出的算术编码引擎快速归一化方法平均可以把熵编码模块的执行时间降低25%。另外，该快速归一化方法能够简化算术编码引擎的执行流程，因而更加适合编码器的硬件设计，降低硬件实现复杂度。

（2）Bypass Bin 的快速编解码算法

表 5-1 给出了本文提出的 bypass bin 的快速编码方法相对于原始的编码方法的编码性能和编解码时间的比较。从中可以看到，本文提出的 bypass bin 的快速编码方法与原始的算术编码引擎相比，平均码率增加仅为 0.1%，这表明本文提出的 bypass bin 的快速编码方法可以取得与原始编码算法相似的编码性能。在编解码时间上，本文提出的 bypass bin 的快速编解码算法平均能够把编码 bypass bin 的编码时间降低 58%。除此之外，由于改进后的算术编码引擎能够在编码 bypass bin 时保证只向码流移出一个比特同时保证编码区间不变，因此，改进后的算术编码引擎能够在一个时钟周期内编码或解码多个 bypass bin，因而可以极大地提高熵编码模块的数据吞吐率。

表 5-1 Bypass bin 的快速编解码算法相对于原始算术编码引擎的 BD-Rate[%]

Table 5-1 BD-Rate[%] of the proposed bypass bin coding relative to the original method

	AI			LD			RA		
	Y	U	V	Y	U	V	Y	U	V
UHD	0.0	0.0	-0.1	-0.1	0.0	-0.1	0.0	0.2	0.2
1080p	0.0	0.0	0.0	0.0	-0.4	0.3	0.0	0.1	0.6
WVGA	0.0	0.0	0.2	0.0	0.0	0.3	0.0	0.1	-0.1
WQVGA	-0.1	0.3	0.0	0.0	-0.1	0.3	0.0	-0.2	0.0
720p	-0.1	0.0	0.1	0.1	0.3	0.4	0.4	-0.4	-0.2
Average	**0.0**	**0.1**	**0.0**	**0.0**	**0.0**	**0.3**	**0.1**	**0.0**	**0.1**
EncTime		90%			98%			98%	
DecTime		95%			98%			99%	

5.3.2 优化后的变换系数的上下文建模过程

表 5-2 给出了本文提出的变换系数的上下文模型选择算法与原始的上下文模型选择算法的性能比较。从中可见，本文提出的上下文模型选择算法的平均码率增加为 0.1%~0.2%。由于本文提出的变换系数上下文模型选择算法打破了 *Run* 和 *Level* 之间的上下文依赖关系，并且采用分层编码方式来编码 *Run*，*Level* 和 *Sign*，即先编码 CG 内所有的 *Run*，然后编码 CG 内所有的 *Level*，最后编码 CG 内所有的 *Sign*。所以，本文提出的上下文模型选择方法能够降低并行解码过程中所需要的推导计算和存储可能使用的上下文模型所需要的存储空间，因而，本文提出的上下文模型选择方法更加适合于编码器的并行

设计。

表 5-2 本文提出的变换系数上下文模型选择算法相对于原始算法的 BD-Rate[%]

Table 5-2 BD-Rate[%] of the proposed context modeling of transform coefficients relative to the original one

	AI			LD			RA		
	Y	U	V	Y	U	V	Y	U	V
UHD	0.1	0.1	0.1	0.0	-0.1	0.0	0.0	0.0	0.0
1080p	0.3	-0.2	-0.1	0.1	-0.3	-0.5	0.1	-0.7	-0.1
WVGA	0.1	0.1	0.0	0.0	0.0	-0.5	0.1	0.1	-0.1
WQVGA	0.1	-0.1	0.0	0.1	-0.2	0.4	0.1	-0.1	0.2
720p	0.2	0.2	0.3	0.1	-0.3	0.7	0.3	-0.7	-0.6
Average	**0.2**	**0.0**	**0.1**	**0.1**	**-0.2**	**0.0**	**0.1**	**-0.3**	**-0.1**
EncTime		100%			100%			99%	
DecTime		99%			100%			100%	

5.3.3 优化后的熵编码模块的整体性能

表 5-3 优化后的熵编码模块相对于原始的熵编码模块的 BD-Rate[%]

Table 5-3 BD-Rate[%] of the optimized entropy coding module relative to the original one

	AI			LD			RA		
	Y	U	V	Y	U	V	Y	U	V
UHD	0.2	0.4	0.2	0.2	-0.1	0.0	0.0	-0.1	0.0
1080p	0.3	0.4	0.3	0.0	-0.1	0.8	0.1	-0.4	0.1
WVGA	0.2	0.0	0.1	0.0	-0.3	-0.1	0.2	-0.1	0.4
WQVGA	0.1	0.0	0.1	0.0	-0.4	0.6	0.2	0.0	0.6
720p	0.3	0.1	0.0	0.0	-0.3	0.2	0.1	-0.8	0.3
Average	**0.2**	**0.2**	**0.2**	**0.0**	**-0.2**	**0.3**	**0.1**	**-0.3**	**0.2**
EncTime		93%			96%			94%	
DecTime		97%			98%			97%	

表 5-3 给出了优化后的熵编码模块的整体性能，即二进制算术编码引擎的快速归一化方法，Bypass bin 的快速编解码算法和改进后的变换系数的上下文建模方法联合使用时的编码性能。从表 5-3 可见，三种方法联合使用时最大的平均码率增加仅为 0.2%，但是可以大幅度地降低熵编码模块的硬件设计复杂度和提高熵编码模块的数据吞吐率。

5.4 本章小结

本章针对 AVS2 中熵编码模块提出了三个优化方案。它们分别是算术编码引擎的快速归一化，bypass bin 的快速编解码方案和变换系数上下文模型

选择的优化方案。这三个技术降低了 AVS2 中熵编码模块的串行依赖关系，提高了熵编码模块的数据吞吐率，更加有利于 AVS2 编解码器的硬件设计。

第6章 压缩感知测量值压缩的熵编码设计

本文在绪论部分比较了基于压缩感知的图像压缩和传统的视频/图像压缩标准之间的区别，并且指出它们各自的优缺点和所适用的场景。在视频压缩标准中，采集设备首先需要完整地采集得到视频信号，然后编码器利用预测，变换，量化和熵编码技术挖掘视频信号的冗余信息来实现数据压缩。由于视频信号的空域像素值和帧间像素值之间存在着强烈的相关性，因此，经过预测，变换和量化之后，变换系数之间仍然存在着一些统计冗余，比如频域位置在一定程度上可以反映变换系数的概率分布，或者相邻位置的变换系数也可以反映当前变换系数的概率分布。所以，视频压缩标准中的熵编码模块通过建立复杂的上下文建模机制来挖掘变换系数之间的统计冗余，然后利用熵编码引擎去除这些统计冗余。因此，本文在前几章中通过设计更加高效的上下文建模方法和熵编码引擎来提高熵编码模块的压缩效率和数据吞吐率。然而，在图像压缩感知中，为了克服外界条件的限制，采集设备通过简单的随机投影来得到测量值，接收端通过利用非线性的信号重建算法来恢复原始信号。编码端得到的测量值之间是相互独立的，并且每个测量值都包含了原始信号的全局信息。因此，视频压缩标准中的熵编码技术不再适用于测量值压缩，因而需要根据测量值的统计特性来设计相应的熵编码器，以便去除测量值之间的统计冗余。所以，本文以文献[139]中的 DPCM-plus-SQ 的编码框架为基础，为测量值的量化索引设计了一个熵编码方案来去除量化索引之间的统计冗余性，把量化索引转化为紧凑的二进制码流，从而实现真正意义上的数据压缩。

6.1 基于 DPCM 的标量量化方案

在传统的压缩感知采样中，随着图像的分辨率越来越高，采样矩阵和稀疏矩阵会消耗大量的内存空间，同时在信号重建过程中，对维度如此之高的矩阵做乘法操作也将带来很高的计算负担。因此为了解决这些问题，有学者提出了基于块的压缩感知采样（BCS：Block-based Compressive Sensing），即在 BCS 中，图像首先被分成非重叠的图像块，然后对每个图像块做压缩感知采样。

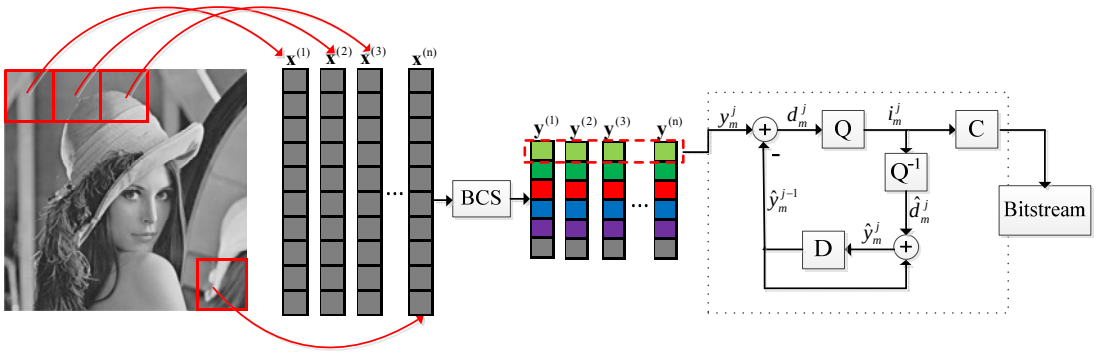


图 6-1 BCS 中基于 DPCM 的标量量化方案框架图

Q：标量量化，C：熵编码模块，D：DPCM 模块

Fig 6-1 Framework of DPCM-plus-SQ of BCS.

Q: uniform SQ; C: entropy coded module; D is DPCM prediction module

由于图像相邻块之间存在相关性，因此在 BCS 中从相邻块采样得到的测量值也具有相关性。所以在编码当前块的测量值时，可以把相邻块的重构测量值作为当前块测量值的预测值，用于减少当前块测量值的信息熵。基于这种思想，有学者提出了基于 DPCM 的标量量化方案[139]。如图 6-1 所示的 BCS 中基于 DPCM 的标量量化方案的框架图，一幅输入图像首先被分成互不重叠的大小为 $B\times B$ 图像块。按照光栅扫描方式，其中每个图像块用 $\mathbf{x}^{(j)}$ 来表示，其中 $j=1,2,...,n$；然后对每个图像块 $\mathbf{x}^{(j)}$ 进行压缩感知采样得到测量值向量 $\mathbf{y}^{(j)}$，在编码的时候，测量值向量 $\mathbf{y}^{(j)}$ 中的第 m 个测量值 y_m^j 通过 $\mathbf{x}^{(j-1)}$ 的重构测量值向量 $\hat{\mathbf{y}}^{(j-1)}$ 的第 m 个重构测量值 $\hat{y}_m^{j-1}$ 来预测；得到的测量值残差 $d_m^j=y_m^j-\hat{y}_m^{j-1}$ 通过标量量化生成量化索引 $i_m^j=Q[d_m^j]$；最后把这个量化索引送入熵编码模块中进行编码并产生码流；在编码端为了给下一次预测做准备，需要得到 y_m^j 的重构值 $\hat{y}_m^j$，此时需要对量化索引 i_m^j 进行反量化得到测量值残差的重构值 $\hat{d}_m^j$，因此 y_m^j 的重构值 $\hat{y}_m^j=\hat{d}_m^j+\hat{y}_m^{j-1}$。

在目前的参考文献中，量化索引 i_m^j 并没有被真正地进行熵编码，而是采用了量化索引的 0 阶信息熵来作为最终的码率输出。量化索引的 0 阶信息熵可以通过公式（6-1）来计算得到。

$$E=-\sum_{y}p(y)\cdot\log(p(y)) \qquad (6\text{-}1)$$

其中 $p(y)$ 是量化索引的取值概率分布，$\log(\bullet)$ 是以 2 为底的对数。

6.2 基于 DPCM 的标量量化索引的算术编码方案

在编码量化索引向量 $\mathbf{i}^{(j)}=[i_1^j,...,i_m^j,...,i_{M_B}^j]^T$ 时，本文发现量化索引向量中的非零元素的分布是独立于其位置信息的。图 6-2 给出了分辨率为 512x512 的 *Lena* 图像，在采样率为 0.08 和 0.10 下，按照大小为 16×16 的 BCS 采样，经过 DPCM 和标量量化处理后，非零量化索引的概率分布示意图。从图 6-2 可以看到，非零量化索引的概率分布是独立于其位置信息的。这可能是因为测量值是通过随机矩阵投影得到的，导致量化索引向量中的非零元素的分布与位置无关。另外通过量化索引向量的统计信息，本文发现量化索引向量中的最后一个非零元素往往出现在该向量的最后一个位置。换句话说，量化索引向量中的非零元素是随机地分布于整个向量中，所以为了高效地编码量化索引向量，本文首先用一个语法元素 *significant_map* 来表示每个位置上的元素是否为非零元素；然后对于每个非零元素，本文再使用语法元素 *abs_coeff_level_minus1* 和 *sign_flag* 来表示非零元素的绝对值减 1 和非零元素的符号信息。

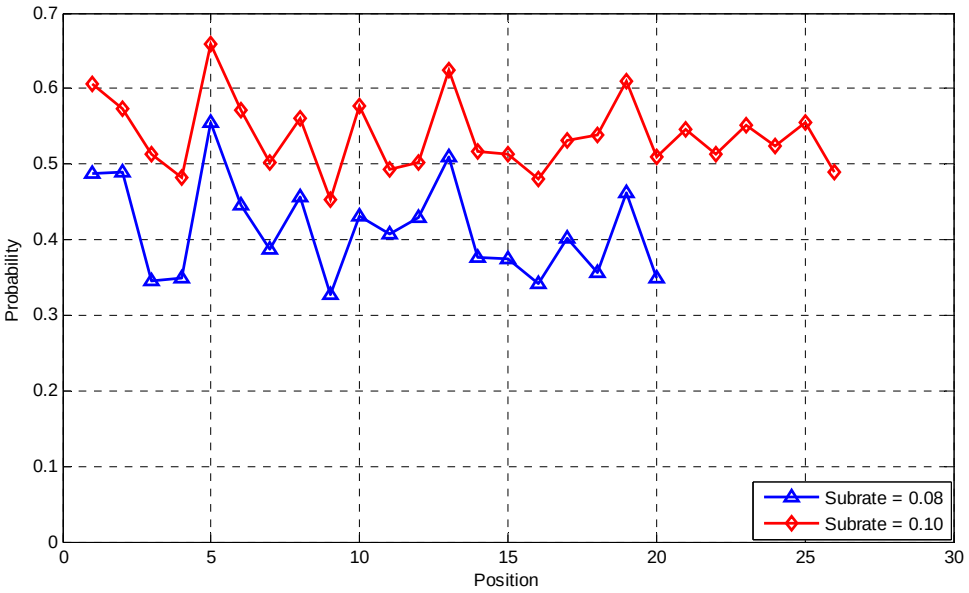


图 6-2 *Lenna* 图像中量化索引向量中每个位置上的非零元素出现的概率

Fig 6-2 Probability of the significant component in a quantization index vector for *Lenna*.

算法 6-1 本文提出的量化索引向量的编码算法描述

Algorithm 6-1 The description of the proposed decoding scheme

输入：码流 str

输出：significant_map[0…M_B-1], abs_coeff_level_minus1[0…M_B-1], sign_flag[0…M_B-1]

初始化：ctxSig=0, ctxLevel=0, significant_map[0,…,M_B-1]=0, abs_coeff_level_minus1[0,…,M_B-1]=0, sign_flag[0,…,M_B-1]=0;

```
For m ← 0 to M_B – 1 do
    significant_map[m] = decode_bin(str, ctxSig);
End For
For m ← 0 to M_B – 1 do
    If significant_map[m] == 1 then
       bin = decode_bin(str, ctxLevel);
       If bin == 0 then
          abs_coeff_level_minus1[m] = 0;
       Else
          symbol = 1;
          ctxLevel++;
          bin = decode_bin(str, ctxLevel);
          binIdx = 2;
          while bin != 0 && binIdx != 13
                bin = decode_bin(str, ctxLevel);
                binIdx++;
                symbol++;
          End while
          If bin != 0 then
             symbol += exp_golomb_decode_eq_prob(str) + 1;
          End If
          abs_coeff_level_minus1[m] = symbol;
       End If
    End If
End For
For m ← 0 to M_B – 1 do
    If significant_map[m] == 1 then
       sign = decode_bin_eq_prob(str);
       If sign == 1 then
         sign_flag[m] = 1;
       End If
    End If
End For
```

基于上面所设计的语法元素，编码一个量化索引向量的流程如下所示：首先，为量化索引向量中的每个位置编码 *significant_map* 来表示当前位置上的元素是否为非零元素；其次，为量化索引向量中的每个非零系数，编码语法元素 *abs_coeff_level_minus1* 来表示非零元素的绝对值减 1；最后，为量化索引向量中的每个非零元素编码语法元素 *sign_flag* 来表示非零元素的符号信息。

由于语法元素 *abs_coeff_level_minus1* 不是二值数值，因此本文利用 UEG0[39]（即截断值为 14 截断一元码和 0 阶指数哥伦布码的联合方式）来对 *abs_coeff_level_minus1* 来进行二值化，把它转化为二进制符号序列，其中前 14 个二进制符号（即 0≤binIdx≤13 的二进制符号）采用上下文模型编码方式，剩余的二进制符号序列（即 binIdx≥14 的二进制符号）采用概率等于 0.5 的编码方式。在上下文建模模块中，由于量化索引向量中的非零元素的分布与其位置是独立的，因此本文编码 *significant_map* 时只采用了一个上下文模型；另外，由于压缩感知是通过随机矩阵投影而得到的，因此当前位置的测量值的取值分布与其他位置上的测量值取值也是独立的，也就是说已经编码的测量值并不能准确地预测当前位置的测量值取值，所以本文在编码 *abs_coeff_level_minus1* 时也只采用了一个上下文模型。整个编码流程的伪代码算法 6-1 所示，其中 decode_bin()表示采用上下文模型的二进制符号解码函数，decode_bin_eq_prob()表示采用概率等于 0.5 的二进制符号解码函数，exp_golomb_decode_eq_prob()表示解码 *abs_coeff_level_minus1* 中 binIdx 大于 14 的二进制符号的函数，M_B 表示量化索引向量的长度，它由图像块的大小 B^2 和采样率 S 来决定。

为了高效地编码上述语法元素，本文采用二进制算术编码引擎 M-coder 作为使用上下文模型的二进制符号解码函数 decode_bin()。M-coder 是一个没有乘法和除法操作的算术编码引擎，并且其中的概率更新模块可以根据已经编码的二进制符号来自适应的更新概率从而可以捕捉信源符号的局部统计特性。由于 M-coder 中的概率是以二元组 (p_{LPS}, V_{MPS}) 来表示的，其中 p_{LPS} 是 LPS（Least Probable Symbol）的概率，V_{MPS} 是 MPS（Most Probable Symbol）的取值。另外 p_{LPS} 是通过典型概率索引 σ 来表示，其中 $0 \le \sigma \le 63$，每个索引分别对应典型概率集合 $S_p = \{p_0, p_1, ..., p_{63}\}$ 中的典型概率。所以，使用 M-coder 作为算术编码引擎来编码某个信源，只需确定两个参数 (σ, V_{MPS}) 即可。

本文提出的这种编码方案与 H.264/AVC 中的 CABAC 关于变换系数编码的方案类似，但是它们之间有两个不同点，正是这两个不同点使得本文提出的编码方案在编码测量值的量化索引时能够取得更好的编码效率。这两个不同点包括：第一，本文提出的编码方案删除了语法元素 *last_significant_map*，这主要是因为测量值的量化索引向量中的最后一个非零元素往往出现在该向量的最后一个位置，在这种情况下，就没有必要编码 *last_significant_map*；第二，本文提出的编码方案为 *significant_map* 和 *abs_coeff_level_minus1* 只设计了一个上下文模型，这主要是因为测量值的量化索引向量中的元素与其位置之间没有相关性并且量化索引向量中的元素之间也没有相关性，在这种情况下，CABAC 中的上下文模型将会导致上下文稀疏问题。

6.3 实验结果

表 6-1 在 *Lenna* 和 *Barbara* 测试图像上 Meth1，Meth2,和 Meth3 所需要的比特率

Table 6-1 Bit-rate achieved by Meth1, Meth2 and Meth3 on *Lenna* and *Barbara* images

	Lenna					Barbara				
	Meth1 [bpp]	Meth2 [bpp]	Meth3 [bpp]	$BR_{1,3}$ [%]	$BR_{2,3}$ [%]	Meth1 [bpp]	Meth2 [bpp]	Meth3 [bpp]	$BR_{1,3}$ [%]	$BR_{2,3}$ [%]
1	0.150	0.144	0.137	**8.67**	**4.86**	0.159	0.156	0.152	**4.40**	**2.56**
2	0.258	0.248	0.234	**9.30**	**5.65**	0.245	0.241	0.231	**5.71**	**4.15**
3	0.313	0.302	0.278	**11.18**	**7.95**	0.308	0.302	0.288	**6.49**	**4.64**
4	0.418	0.400	0.374	**10.53**	**6.50**	0.386	0.378	0.360	**6.74**	**4.76**
5	0.554	0.527	0.493	**11.01**	**6.45**	0.551	0.536	0.513	**6.90**	**4.29**
6	0.607	0.577	0.538	**11.37**	**6.76**	0.603	0.586	0.560	**7.13**	**4.44**
7	0.686	0.651	0.613	**10.64**	**5.84**	0.717	0.696	0.660	**7.95**	**5.17**
8	0.820	0.776	0.730	**10.98**	**5.93**	0.764	0.742	0.702	**8.12**	**5.39**
9	0.884	0.835	0.781	**11.65**	**6.47**	0.878	0.851	0.810	**7.74**	**4.82**
10	0.994	0.938	0.884	**11.07**	**5.76**	0.935	0.907	0.862	**7.81**	**4.96**
11	1.106	1.044	0.993	**10.22**	**4.89**	1.050	1.015	0.969	**7.71**	**4.53**
12	1.247	1.172	1.103	**11.55**	**5.89**	1.149	1.112	1.056	**8.09**	**5.04**
13	1.388	1.305	1.241	**10.59**	**4.90**	1.227	1.184	1.130	**7.91**	**4.56**
14	1.456	1.368	1.302	**10.58**	**4.82**	1.511	1.457	1.398	**7.48**	**4.05**
15	1.601	1.501	1.421	**11.24**	**5.33**	1.458	1.407	1.340	**8.09**	**4.76**
16	1.687	1.583	1.505	**10.79**	**4.93**	1.551	1.496	1.430	**7.80**	**4.41**
17	1.867	1.766	1.704	**8.73**	**3.51**	1.645	1.586	1.521	**7.54**	**4.10**
18	1.741	1.631	1.544	**11.32**	**5.33**	1.739	1.679	1.614	**7.19**	**3.87**
19	1.834	1.720	1.635	**10.85**	**4.94**	1.836	1.773	1.710	**6.86**	**3.55**
20	1.928	1.811	1.731	**10.22**	**4.42**	1.933	1.870	1.808	**6.47**	**3.32**
AVG	**1.077**	**1.015**	**0.962**	**10.62**	**5.56**	**1.032**	**0.999**	**0.956**	**7.21**	**4.37**

表 6-2 在 *Goldhill* 和 *Peppers* 测试图像上 Meth1，Meth2,和 Meth3 所需要的比特率

Table 6-2 Bit-rate achieved by Meth1, Meth2 and Meth3 on *Goldhill* and *Peppers* images

	Goldhill					Peppers				
	Meth1 [bpp]	Meth2 [bpp]	Meth3 [bpp]	$BR_{1,3}$ [%]	$BR_{2,3}$ [%]	Meth1 [bpp]	Meth2 [bpp]	Meth3 [bpp]	$BR_{1,3}$ [%]	$BR_{2,3}$ [%]
1	0.152	0.153	0.152	**0.00**	**0.65**	0.156	0.152	0.148	**5.13**	**2.63**
2	0.244	0.245	0.240	**1.64**	**2.04**	0.246	0.238	0.227	**7.72**	**4.62**
3	0.391	0.390	0.380	**2.81**	**2.56**	0.381	0.369	0.348	**8.66**	**5.69**
4	0.417	0.413	0.402	**3.60**	**2.66**	0.458	0.443	0.421	**8.08**	**4.97**
5	0.512	0.507	0.494	**3.52**	**2.56**	0.514	0.495	0.470	**8.56**	**5.05**
6	0.578	0.569	0.553	**4.33**	**2.81**	0.638	0.612	0.580	**9.09**	**5.23**
7	0.659	0.650	0.634	**3.79**	**2.46**	0.750	0.720	0.688	**8.27**	**4.44**
8	0.784	0.775	0.758	**3.32**	**2.19**	0.780	0.747	0.710	**8.97**	**4.95**
9	0.898	0.881	0.857	**4.57**	**2.72**	0.931	0.888	0.844	**9.34**	**4.95**
10	0.958	0.939	0.912	**4.80**	**2.88**	1.053	1.006	0.961	**8.74**	**4.47**
11	1.022	1.003	0.974	**4.70**	**2.89**	1.121	1.068	1.022	**8.83**	**4.31**
12	1.267	1.242	1.211	**4.42**	**2.50**	1.179	1.125	1.080	**8.40**	**4.00**
13	1.252	1.227	1.189	**5.03**	**3.10**	1.271	1.207	1.148	**9.68**	**4.89**
14	1.559	1.529	1.493	**4.23**	**2.35**	1.477	1.405	1.349	**8.67**	**3.99**
15	1.644	1.616	1.581	**3.83**	**2.17**	1.477	1.410	1.358	**8.06**	**3.69**
16	1.717	1.687	1.649	**3.96**	**2.25**	1.563	1.485	1.419	**9.21**	**4.44**
17	1.793	1.762	1.722	**3.96**	**2.27**	1.643	1.564	1.500	**8.70**	**4.09**
18	1.888	1.860	1.821	**3.55**	**2.10**	1.882	1.793	1.727	**8.24**	**3.68**
19	1.982	1.960	1.921	**3.08**	**1.99**	2.061	1.990	1.937	**6.02**	**2.66**
20	1.872	1.839	1.797	**4.01**	**2.28**	1.953	1.856	1.779	**8.91**	**4.15**
AVG	**1.079**	**1.062**	**1.037**	**3.66**	**2.37**	**1.077**	**1.029**	**0.986**	**8.36**	**4.35**

为了验证本文提出的编码方案的编码性能，本文以文献[139]所提出的基于 DPCM 的标量量化方案为实验框架。在这个框架中，每幅输入图像都被分成大小为 16x16 的非重叠的图像块，每个图像块采用高斯随机矩阵进行采样，采样率和量化步长是通过搜索而找到的最优组合方式。

本文一共测试了 4 幅分辨率为 512x512 的灰度图像，包括 *Lenna*，*Barbara*，*GoldHill* 和 *Peppers*。在接下来的描述中，AC-DPCM-SQ 表示采用本文提出的算术编码方案来编码的方法，即在这个方法中，量化索引向量被传递给本文提出的算术编码模块，由算术编码模块来把它们转化为码流并计算产生的比特数；ORG-DPCM-SQ 表示文献[139]所提出编码方案，在这个方案中，量化索引并没有被真正的熵编码，而是把它的 0 阶信息熵作为最终的比特数；CABAC-DPCM-SQ 表示采用 CABAC 中编码变换系数的方案来编码量化索引的方法，即量化索引被传递给 CABAC 中的变换系数的编码模块中，由该模

块把它们转化为二进制码流并计算产生的比特数。表 6-1 和表 6-2 分别给出了 ORG-DPCM-SQ，CABAC-DPCM-SQ 和 AC-DPCM-SQ 三种编码方案在 4 幅图像上所产生的比特总数，其中 Meth1 表示 ORG-DPCM-SQ 所产生的比特率，Meth2 表示 CABAC-DPCM-SQ 所产生的比特率，Meth3 表示 AC-DPCM-SQ 所产生的比特率；$BR_{1,3}$ 表示 Meth3 相对于 Meth1 的比特率下降的百分比；$BR_{2,3}$ 表示 Meth3 相对于 Meth2 的比特率下降的百分比。从表 6-1 和表 6-2 中可以看出，本文提出的算术编码方案比原始的编码方案平均节省 3.65%~10.63%的码率；相比于 CABAC 中变换系数的编码方案，本文提出的编码方案可以节省 2.37%~5.56%的码率。

图 6-3~图 6-6 分别给出了这三种编码方案在 0.0~1.0 的码率区间上的率失真曲线，其中 SQ 表示直接对测量值进行标量量化的编码方案。在这些曲线中，SQ 和 ORG-DPCM-SQ 中的比特率是通过公式（6-1）计算得到的。由于熵编码方法是无损的编码方案，所以本文只采用了 SPL[129]这一种图像恢复算法来利用解码得到的测量值重构原始图像，进而计算重构图像和原始图像的失真。

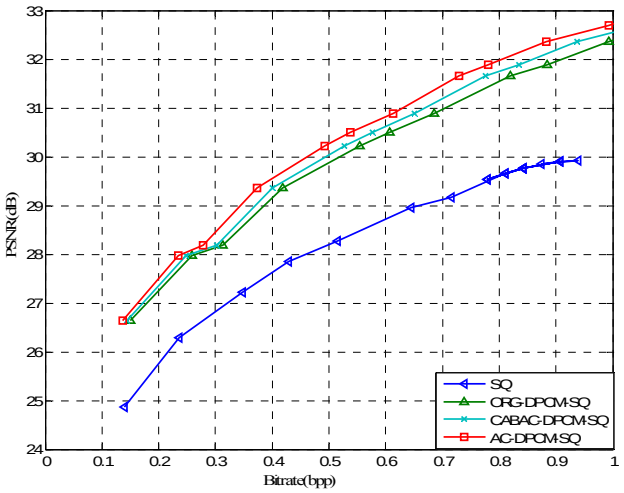


图 6-3 *Lenna* 图像的率失真曲线

Fig 6-3 RD curve for *Lenna*

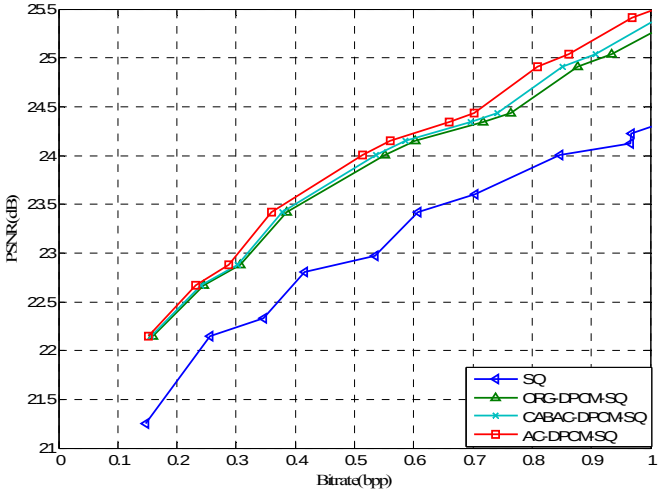


图 6-4 *Barara* 图像的率失真曲线

Fig 6-4 RD curve for *Barara*

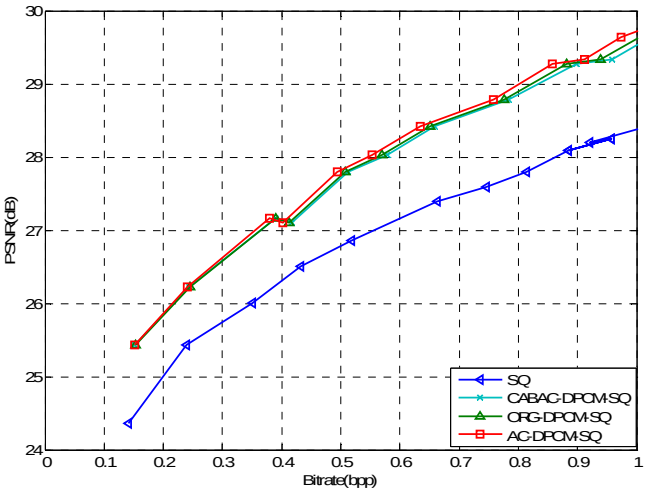


图 6-5 *Goldhill* 图像的率失真曲线

Fig 6-5 RD curve for *Goldhill*

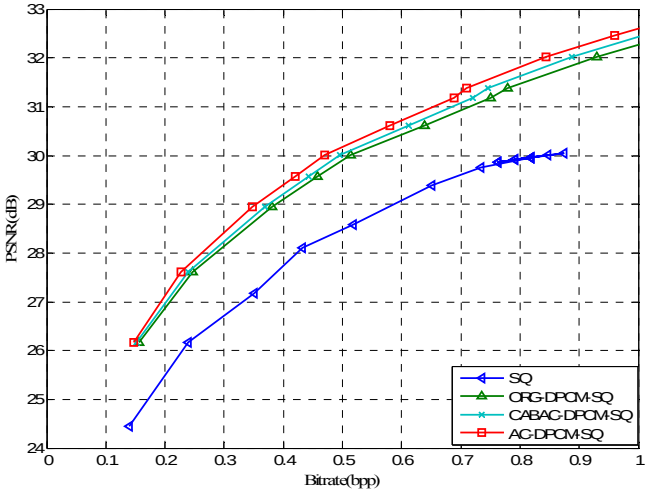


图 6-6 *Peppers* 图像的率失真曲线

Fig 6-6 RD curve for *Peppers*

6.4 本章小结

在基于块的图像压缩感知采样中，采用 DPCM 的标量量化方案能够以较低的计算复杂度下取得较高的率失真性能。为了进一步提升基于图像压缩感知的测量值编码的压缩效率，并且把测量值残差的量化索引转化为紧凑的二进制码流，本章基于测量值残差的量化索引的统计特性为它设计了一个算术编码方案。与测量值残差的量化索引的 0 阶信息熵以及 CABAC 中变换系数的编码方案相比，本章提出的算术编码方案可以进一步提高编码性能。

结　论

熵编码技术是视频和图像压缩系统中的核心模块之一，它在去除信源符号的统计冗余性和码流组织中有着无法替代的作用。考虑到高清和超高清视频应用的兴起，熵编码模块的吞吐率已经成为设计熵编码模块必须要考虑的性能指标之一；如何在吞吐率和编码性能之间取得一个最优的平衡已经成为视频编码技术研究中的热点问题。随着电子信息技术的发展趋势，未来的视频压缩标准将要处理数据量更大的视频信号，因此提高熵编码技术的编码性能依然是视频压缩标准的主要目标。在未来几年中，软硬件设备的计算能力将越来越强，而造价会越来越低，这使得将来的视频压缩标准可以在不显著降低吞吐率的前提下采用更加复杂的编码技术来提高编码性能。另外随着压缩感知这一新型的采样技术在图像和视频采集中的应用，传统的视频和图像压缩中的熵编码技术已经无法取得最优的编码效率。基于这些问题，本文对H.264/AVC，AVS2 和 HEVC 中的熵编码模块进行优化设计来提高它们的编码性能或数据吞吐率，同时也为图像压缩感知采样中的测量值压缩提出了一个高效的熵编码方案。具体地，本文的主要研究成果包括：

（1）提出了一个基于层次依赖上下文模型的算术编码器 HDCMBAC 来高效地编码 H.264/AVC 中的变换系数。HDCMBAC 采用层次依赖上下文模型 HDCM 来挖掘变换系数之间的统计相关性，并且重新设计了用于描述变换系数的语法元素。通过合理地组织上下文建模模块和熵编码模块的执行过程，HDCMBAC 中的上下文建模模块和算术编码模块可以并行地执行，使得 HDCMBAC 就像没有上下文建模模块一样。实验结果表明，HDCMBAC 可以取得与 H.264/AVC 中的 CABAC 相当的编码效率。

（2）提出 HEVC 视频编码标准中的熵编码模块的优化方案。为了提高 HEVC 中熵编码模块的编码性能，本文提出一个变换系数的增强上下文模型和一个低内存消耗的二进制算术编码引擎。在变换系数的增强上下文模型中，本文把变换系数的频域位置和当前变换系数的局部模板内的变换系数作为上下文的来源；在低内存消耗的二进制算术编码引擎中，本文使用多参数的概率更新模型来提高概率更新的精度，并且在编码区间分割过程中引入了一个低位宽的乘法操作来降低编码区间的内存消耗。实验结果表明这两种方案都可以提高 HEVC 中熵编码模块的编码性能，并且这两个编码技术的编码增益

是可以叠加的。

（3）提出了 AVS2 视频编码标准中熵编码模块的优化设计方案。为了提高 AVS2 中熵编码模块的数据吞吐率，本文从三个方面对 AVS2 中的熵编码模块进行了优化。首先，本文提出了一个与标准兼容的快速的算术编码引擎的归一化方案，该方法从实数域来执行归一化过程，并且每隔 16 比特才调用一次归一化过程，减少了调用归一化过程的次数；其次，本文提出了一个快速的 bypass bin 的编解码方法，使得只需一次加法和移位便可完成 bypass bin 的编解码过程；最后，本文改进了 AVS2 中变换系数的上下文建模过程，来最大限度地降低变换系数之间的上下文依赖关系。实验结果表明，上述三个技术可以大大地提高 AVS2 中的熵编码模块的数据吞吐率，同时性能损失也比较小。

（4）提出了图像压缩感知中测量值压缩的熵编码方案。为了提高图像压缩感知中测量值压缩的率失真性能并且把测量值残差的量化索引转化为紧凑的二进制码流，本文根据测量值残差的量化索引的统计特性提出了一个高效的熵编码方案。该方案为量化索引向量中的每个位置定义了语法元素 *significant_map* 来表示当前位置的元素是否为非零元素；其次，为每个非零元素，定义了 *abs_coeff_level_minus1* 和 *sign_flag* 来表述非零系数的幅值信息。考虑到测量值量化索引向量中的元素之间的相关性，本文为 *significant_map* 和 *abs_coeff_level_minus1* 分别设计了一个上下文模型。与测量值残差的量化索引的 0 阶信息熵以及 CABAC 中的变换系数的编码方案相比，本文提出的熵编码方案可以取得更好的编码性能。

熵编码模块是视频和图像压缩中最重要的模块之一。提高熵编码模块的吞吐率和编码效率的方法一般包括改进其中的上下文建模过程和熵编码引擎。本文对视频压缩标准中熵编码模块的研究主要包括，在二维视频压缩的混合编码框架下，通过改进变换系数的上下文建模过程和算术编码引擎来提高它的吞吐率或编码性能；而对图像压缩中的熵编码技术的研究工作主要是在图像压缩感知的框架下，针对测量值量化索引的统计特性设计了一个熵编码方案。该方案不但可以去除测量值量化索引的统计冗余提高了编码效率，还把测量值量化索引转化为了二进制码流。随着技术的进步和发展，未来的视频压缩标准可能需要处理诸如 3D 视频，全景视频或者分辨率更高的二维视频。这些视频序列中存在着更多的统计依赖关系，如何利用这个统计依赖关系来提高熵编码模块的编码性能和数据吞吐率将会是一个新的研究热点。另外随

着图像压缩感知采样技术的发展，新型的压缩感知采样技术已经提出来了，这些采样技术不但保留了传统图像压缩感知采样的优点，还使得测量值更加容易压缩。针对这些新型的压缩感知采样技术来设计合理的熵编码方案，将会进一步提高压缩感知测量值的率失真性能。

参考文献

[1] 高文，赵德斌，马思伟，“数字视频编码技术原理[M]”，北京：科学出版社，2010.

[2] V. Sze and P. Chandrakasan, “Joint Algorithm-Architecture Optimization of CABAC[J],” Journal of Signal Processing Systems, vol. 69, no. 3, pp. 239-252, Dec 2012.

[3] R. G. Baraniuk, E. Candes, R. Nowak, and M. Vetterli, “Compressive Sampling[J],” IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 12–13, March 2008.

[4] C. E. Shannon, “A Mathematical Theory of Communication[J],” The Bell System Technical Journal, vol. 27, pp. 379–423, 623–656, July, October 1948.

[5] C. E. Shannon, “Coding Theorems for a Discrete Source with a Fidelity Criterion[J],” IRE International Convention Record, Part 4, vol. 7, pp. 142–163, 1959.

[6] A. Gersho and R. M. Gray, “Vector Quantization and Signal Compression[M],” Kluwer Academic Publishers, Norwell, MA, 1992.

[7] C. C. Culter. Differential Quantization of Communication Signals. US Patent 2605631. July 1952.

[8] G. Bjontegaard, “Response to Call for Proposals for H.26L[C],” ITU-T/Study Group 16/ Video Coding Experts Group, document Q15-F-11, Nov. 1998.

[9] A. N. Netravali and J. D. Robbins, “Motion-compensated Television Coding: Part 1[J],” The Bell System Technical Journal. 1979, 58(3).

[10] N. Ahmed, T. Natarajan and K. Rao, “Discrete Cosine Transform[J],” IEEE Transactions on Computers, vol. C-23, no. 1, Jan 1974.

[11] W. K. Cham, “Development of integer cosine transforms by the principle of dyadic symmetry[C],” Proceedings of IEE Communications, Speech and Vision 1999, vol. 136, pp. 276-282, 1999.

[12] X. Zhao, L. Zhang, S. W. Ma and W. Gao, “Video Coding with

Rate-Distortion Optimized Transform[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 1, pp. 138-151, Jan 2012.

[13] Y. Ye and M. Karczewicz, “Improved H.264 intra coding based on bidirectional intra prediction, directional transform, and adaptive coefficient scanning[C],” In Proceedings of IEEE International Conference on Image Processing 2008.

[14] W. H. Chen and W. K. Pratt, “Scene adaptive coder[J],” IEEE Transactions on Communications, vol. 32, no. 3, pp. 225-232, Mar 1984.

[15] J. Lee, “Rate distortion optimization of parameterized quantization matrix for MPEG-2 encoding[C],” In Proceedings of IEEE International Conference on Image Processing 1998.

[16] J. Lu, T. Chen, Y. Kashiwagi and S. Kadono, “Proposal of quantization weighting for H.264/MPEG-4 AVC Professional Profiles[C],” JVT documents, JVT-K029r1, Mar.2004.

[17] A. Tanizawa and T. Chujoh, “Adaptive quantization matrix selection on KTA software[C],” ITU-T Q6/SG16 VCEG-AD06, Oct 2006.

[18] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes[C],” Proceedings of the IRE, vol. 40, no. 9, pp. 1098-1101, Sep 1952.

[19] R. G. Gallager, and D. Van Voorhis, “Optimal Source Codes for Geometrically Distributed Integer Alphabets[J],” IEEE Transactions on Information Theory, vol. 21, no. 2, pp. 228-230, Mar 1975.

[20] R. G. Gallager, “Variations on a Theme by Huffman[J],” IEEE Transactions on Information Theory, vol. 24, no. 6, pp. 668-674, Nov 1978.

[21] S. W. Golomb, “Run-Length Encoding[J],” IEEE Transactions on Information Theory, vol. 12, no. 3, pp. 399-401, Jul 1966.

[22] R. F. Rice, “Some Practical Universal Noiseless Coding Techniques,” Jet Propulsion Laboratory, Pasadena, California, JPL Publication 79-22, Mar 1979.

[23] D. Salomon, “Variable-Length Codes for Data Compression[M],” Springer-Verlag, 2007.

[24] R. Pasco, “Source Coding Algorithms for Fast Data Compression[D],” Doctoral Dissertation, Stanford University, 1976.

[25] J. J. Rissanen, “Generalized Draft Inequality and Arithmetic Coding[J],” IBM Journal of Research and Development, vol. 20, no. 3, pp. 198-203, May 1976.

[26] J. J. Rissanen, and G. G. Langdon, “Arithmetic Coding[J],” IBM Journal of Research and Development, vol. 23, no. 2, pp. 149-162, Mar 1979.

[27] I. H. Witten, J. G. Cleary, and R. Neal, “Arithmetic Coding for Data Compression[J],” Communications of ACM, vol. 30, no. 6, pp. 520-540, Jun 1987.

[28] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, “An Overview of the Basic Principles of the Q-coder Adaptive Binary Arithmetic Coder[J],” IBM Journal of Research and Development, vol. 32, no.6, pp. 717–726, Nov 1988.

[29] J. Rissanen and K. M. Mohiuddin, “A Multiplication-Free Multialphabet Arithmetic Code[J],” IEEE Transactions on Communications, vol. 37, no. 2, pp. 93–98, Feb. 1989.

[30] D. Marpe, and T. Wiegand, “A Highly Efficient Multiplication-Free Binary Arithmetic Coder and Its Application in Video Coding[C],” In Proceedings of IEEE International Conference on Image Processing 2003.

[31] J. Apostolopoulos, A. Pfajfer, H. M. Jung, and J. S. Lam, “Position-Dependent Encoding[C],” In Proceedings of IEEE Acoustics, Speech, and Signal Processing 1994.

[32] E. C. Reed and J. S. Lim, “Efficient Coding of DCT Coefficients by Joint Position-Dependent Encoding[C],” In Proceedings of IEEE Acoustics, Speech, and Signal Processing 1998.

[33] G. Lakhani, “Modified JPEG Huffman Coding[J],” IEEE Transactions on Image Processing, vol. 12, no. 2, pp. 159-169, Feb. 2003.

[34] G. Lakhani, “Optimal Huffman Coding of DCT Blocks[J],” IEEE Transactions on Circuits System and Video Technology, vol. 14, no. 4, pp. 522-527, April 2004.

[35] G. Bjntegaard and K. Lillevold, “Context-adaptive VLC (CVLC) coding of coefficients[C],” JVT-C028, 3rdMeeting: Fairfax, Virginia, USA, 6-10 May, 2002.

[36] Q. Wang, D. Zhao, W. Gao, “Context-Based 2D-VLC Entropy Coder in

AVS Video Coding Standard[J],” Journal of Computer Science & Technology. vol. 21, no.3, pp. 315-322, May 2006.

[37] Q. Wang, D. Zhao, S. Ma, Y. Lu, Q. Huang, W. Gao, “Context-Based 2D-VLC for Video Coding[C],” In Proceedings of IEEE International Conference on Multimedia and Expo 2004.

[38] 王强，基于上下文的变长编码技术研究[D]，博士学位论文，哈尔滨工业大学，2009.

[39] C. Tu and T. D. Tran, “Context-Based Entropy Coding of Block Transform Coefficients for Image Compression[J],” IEEE Transactions on Image Processing, vol. 11, no. 11, pp. 1271-1283, Nov. 2002.

[40] D. Marpe, H. Schwarz, and T. Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp.620-636, July 2003.

[41] L. Zhang, Q. Wang, N. Zhang, D. Zhao, X. Wu and W. Gao, “Context-Based Entropy Coding in AVS Standard[J],” Signal Processing: Image Communication, vol. 24, no. 4, pp. 263-276, April 2009.

[42] MPEG. “Coding of Moving Pictures and Associated Audio -- For Digital Storage Media at up to about 1.5 Mbit/s - Part 2: Video,” ISO/IEC 11172-2 (MPEG-1), 1993.

[43] ITU-T. “Information Technology - Generic Coding of Moving Pictures and Associated Audio Information: Video,” ITU-T Recommendation H.262 | ISO/IEC 13818-2 (MPEG-2), 1995.

[44] MPEG. “Information Technology - Coding of Audio-Visual Objects - Part 2: Visual,” ISO/IEC 14496-2 (MPEG-4), 2002.

[45] ITU-T. “Video Codec for Audio Visual Services at px64 kbit/s,” ITU-T Recommendation H.261, 1993.

[46] ITU-T. “Video Coding for Low Bitrate Communication,” ITU-T Recommendation H.263, 1998.

[47] ITU-T. “Advanced Video Coding for Generic Audio Visual Services,” ITU-T Rec.H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC 1, Version 1, May 2003.

[48] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, “Overview of the

H.264/AVC Video Coding Standard[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no.7, pp.560-576, July 2003.

[49] T. Wiegand and B. Girod, “Multi-Frame Motion-Compensated Prediction for Video Transmission[M],” Boston, MA: Kluwer Academic Publisers, 2001.

[50] T. Wiegand, X. Zhang and B. Girod, “Long-term Memory Motion-compensated Prediction[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 9, no. 1, pp. 70-84, Feb. 1999.

[51] A. M. Tourapis, F. Wu and S. Li, “Direct mode coding for bipredictive slices in the H.264 standard[J], ” IEEE Transactions on Circuits and Systems for Video Technology, vol. 15, no. 1, pp. 119-126, Jan. 2005.

[52] M. Wien, “Variable Block-size Transforms for H.264/AVC[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 604-613, July 2003.

[53] P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz, “Adaptive Deblocking Filter[J],” IEEE Transactions Circuits System for Video Technology, vol. 13, no. 7, pp. 614–619, July 2003.

[54] AVS 工作组, “《AVS 视频》(20070321)，” N1370，北京，2009 年 3 月.

[55] AVS 工作组, “AVS1-P2 JQP 标准文本（FCD）,” N1600，杭州，2009 年 3 月.

[56] L. Yu, S. Chen and J. Wang, “Overview of AVS-video Coding Standards[J],” Signal Processing: Image Communication, vol. 24, no. 4, pp. 247-262, Apr 2009.

[57] R. Wang, C. Huang, J. Li and Y. Shen, “Sub-pixel Motion Compensation Interpolation Filter in AVS[C],” Proceedings of IEEE International Conference on Multimedia and Expo, 2004.

[58] X. Ji, D. Zhao, F. Wu, Y. Lu and W. Gao, “B-Picture Coding in AVS Video Compression Standard[J],” Signal Processing: Image Communication, vol. 23, pp.31-41, Jan. 2008.

[59] S. Ma and W. Gao, “Low Complexity Integer Transform and Adaptive Quantization Optimization[J],” Journal of Computer Science & Technology, vol. 21, no. 3, pp. 354-359, May 2006.

[60] C. Zhang, J. Lou, L. Yu and J. Dong, “The Technique of Pre-scaled Integer

Transform[C],” Proceedings of IEEE International Symposium on Circuits and Systems 2005.

[61] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard[J],” IEEE Transactions Circuits System for Video Technology, vol. 22, no. 12, pp. 1649-1668, Dec 2012.

[62] I. Kim, J. Min, T. Lee, W. Han, and J. Park, “Block Partitioning Structure in the HEVC Standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1697 - 1706, Sep. 2012.

[63] U. Kemal, A. Alshin, E. Alshina, F. Bossen, W. Han, J. Park and J. Lainema, “Motion Compensated Prediction and Interpolation Filter Design in H.265/HEVC[J],” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 946-956, Dec. 2013.

[64] J. Lin, Y. Chen, Y. Huang and S. Lei, “Motion Vector Coding in the HEVC Standard[J],” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 957-968, Dec. 2013.

[65] R. Sjoberg,Y. Chen, A. Fujibayashi, M. Hannuksela, J. Samuelsson, T. Tan, Y. Wang, and S. Wenger, “Overview of HEVC High-Level Syntax and Reference Picture Management[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1858-1870, Dec. 2012.

[66] J. Sole, R. Joshi, N. Nguyen, T. Ji, M. Karczewicz, G. Clare, F. Henry and A. Duenas, “Transform Coefficient Coding in HEVC[J],” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1765-1777, Dec. 2012.

[67] T. Nguyen, P. Helle, M. Winken, B. Bross, D. Marpe, H. Schwarz and T. Wiegand, “Transform Coding Techniques in HEVC[J],” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 978-989, Dec 2013.

[68] C. Fu, E. Alshina, A. Alshin, Y. Huang, C. Chen, and C. Tsai, C. Hsu, S. Lei, J. Park and W. Han, “Sample Adaptive Offset in the HEVC Standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1755-1764, Dec. 2012.

[69] S. Ma, T. Huang, C. Reader, and W. Gao, “AVS2—Making Video Coding Smarter[J],” IEEE Signal Processing Magazine, vol.32, no. 2, pp. 172-183, Mar 2015.

[70] Y. Piao, S. Lee, and C. Kim, “Modified intra mode coding and angle adjustment[C],” AVS_M3304, 48th AVS Meeting, Beijing, China, Apr 2014.

[71] Q. Yu, X. Cao, W. Li, Y. Rong, Y. He, X. Zheng, and J. Zheng, “Short distance intra prediction[C],” AVS_M3171, 46th AVS Meeting, Shenyang, China, Sept. 2013.

[72] Y. Piao, S. Lee, I.-K. Kim, and C. Kim, “Derived mode (DM) for chroma intra prediction[C],” AVS_M3042, 44th AVS Meeting, Luoyang, China, Mar. 2013.

[73] Z. Shao and L. Yu, “Multi-hypothesis skip/direct mode in P frame[C],” AVS_M3256, 47th AVS Meeting, Shenzhen,China, Dec. 2013.

[74] Y. Lin and L. Yu, “F Frame CE: Multi Forward Hypothesis Prediction[C],” AVS_M3326, 48th AVS Meeting, Beijing, China, Apr. 2014.

[75] J. Ma, S. Ma, J. An, K. Zhang, and S. Lei, “Progressive Motion Vector Precision,” AVS_M3049, 44th AVS Meeting, Luoyang, China, Mar. 2013.

[76] W. Li, Y. Yuan, X. Cao, Y. He, X. Zheng, and J.Zhen, “Non-Square Quad-tree Transform,” AVS_M3153, 45th AVS Meeting, Taicang, China, June 2013.

[77] J. Wang, X. Wang, T. Ji, and D. He, “Two-Level Transform Coefficient Coding,” AVS_M3035, 43rd AVS Meeting, Beijing, China, Dec. 2012.

[78] J. Chen, S. Lee, C. Kim, C.-M. Fu, Y.-W. Huang, and S. Lei, “Sample Adaptive Offset for AVS2,” AVS_M3197, 46th AVS Meeting, Shenyang, China, Sept. 2013.

[79] X. Zhang, J. Si, S. Wang, S. Ma, J. Cai, Q. Chen, Y.-W. Huang, and S. Lei, “Adaptive Loop Filter for AVS2,” AVS_M3292, 48th AVS Meeting, Beijing, China, Apr. 2014.

[80] J. Rissanen, “A Universal Data Compression System[J],” IEEE Transactions Information Theory, vol. 29, no. 5, pp. 656-664, Sep. 1983.

[81] M. J. Weinberger, J. Rissanen, and R. B. Arps, “Applications of Universal Context Modeling to Lossless Compression of Gray-Scale Images[J],” IEEE Transactions Image Processing, vol. 5, no. 4, pp. 575-586, Apr. 1996.

[82] X. Wu, “Context Quantization with Fisher Discriminant for Adaptive Embedded Wavelet Image Coding[C],” In Proceedings of IEEE Data

Compression Conference 1999.

[83] S. Forchhammer, X. Wu, and J. D. Andersen, “Optimal Context Quantization in Lossless Compression of Image Data Sequences[J],” IEEE Transactions Image Processing, vol. 13, no. 4, pp. 509-517, Apr 2004.

[84] D. L. Duttweiler and C. Chamzas, “Probability Estimation in Arithmetic and Adaptive-Huffman Entropy Coders[J],” IEEE Transactions Image Processing, vol. 4, no. 3, pp. 237-246, Mar. 1995.

[85] A. Zandi, G. G. Langdon, “Adaptation for Non-Stationary Binary Sources for Data Compression[C],” In Proceedings of Signals, Systems and Computers 1995.

[86] X. Ran, and C. Choo, “Syntax-Based Arithmetic Video Coding for Very Low Bitrate Visual Telephony,” In Proceedings of IEEE International Conference on Image Processing 1995.

[87] A. G. Tescher and R. V. Cox, “An Adaptive Transform Coding Algorithm,” In Proceedings of IEEE International Conference on Communications 1976.

[88] P. G. Howard, and J. S. Vitter, “Practical Implementations of Arithmetic Coding. Image and Text Compression[M],” Storer, Ed., pp. 85-112, Kluwer Academic, 1992

[89] D. Taubman and M.W. Marcellin, “JPEG2000 Image Compression: Fundamentals,Standards and Practice[M],” Boston, MA: Kluwer, 2002.

[90] D. Marpe, “Adaptive Context-Based and Tree-Based Algorithms for Image Coding and Denoising[D],” Doctoral Dissertation, Universitat Rostock, Rostock 2004.

[91] D. Marpe, G. Marten, and H. L. Cycon, “A Fast Renormalization Technique for H.264/MPEG4-AVC Arithmetic Coding[C],” 51st Internationales Wissenschaftliches Kolloquium Technische Universitat Ilmenau, September 2006.

[92] M. Mrak, D. Marpe, and T. Wiegand, “A Context Modeling Algorithm and Its Application in Video Compression[C],” In Proceedings of IEEE International Conference on Image Processing 2003.

[93] M. Ghandi, M. M. Ghandi, and M. B. Shamsollahi, “A Novel Context Modeling Scheme for Motion Vectors Context-Based Arithmetic Coding[C],” In Proceedings of IEEE Canadian Conference on Electrical &

Computer Engineering 2004.

[94] S. Milani, and G. A. Mian, “An Improved Context Adaptive Binary Arithmetic Coder for the H.264/AVC Standard[C],” In Proceedings of European Signal Processing Conference 2006.

[95] E. Belyaev, A. Turlikov, K. Egiazarian, M. Gabbouj, “An Efficient Adaptive Binary Arithmetic Coder With Low Memory Requirement[J],” IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 1053-1061, Jun, 2013.

[96] A. Alshin, E. Alshina, “Multi-Parameter Probability Up-Date for CABAC[C],” JCTVC-F254, 6th Joint Collaborative Team on Video Coding (JCTVC) Meeting, Torino, Italy, 14-22 July, 2011.

[97] D. Hong, M. Schaar, and B. Popescu, “Arithmetic Coding with Adaptive Context-Tree Weighting for the H.264 Video Coders[C],” In Proceedings of Visual Communications and Image Processing 2004.

[98] D. Wu, W. Gao, M. Hu, and Z. Ji, “A VLSI Architecture Design of CAVLC Decoder[C],” In Proceedings of International Conference on ASIC 2003.

[99] H. Chang, C. Lin, and J. Guo, “A Novel Low-Cost High-Performance VLSI Architecture for MPEG-4 AVC/H.264 CAVLC Decoding[C],” In Proceedings of IEEE International Symposium on Circuits and Systems 2005.

[100] V. Sze and M. Budagavi, “Parallel CABAC for Low Power Video Coding[C],” In Proceedings of International Conference on Image Processing 2008.

[101] D. Marpe, H. Schwarz,and T. Wiegand, “Entropy Coding In Video Compression Using Probability Interval Partitioning[C],” In Proceedings of Picture Coding Symposium 2010.

[102] S. Chen, S. Chen,and S. Sun, “P3-CABAC: A Nonstandard Tri-Thread Parallel Evolution of CABAC in the Manycore Era[J],” IEEE Transactions on Circuits System and Video Technology, vol. 20, no 6, pp. 920-924, June 2010.

[103] W. J. Kim, K. Cho and K. S. Chung, “Multi-Threaded Syntax Element Partitioning for Parallel Entropy Decoding[J],” IEEE Transactions on Consumer Electronics, vol. 57, no. 2, pp. 897-905, June 2011.

[104] Y. Wei, P. Yang and Y. He, “Arithmetic Codec on Logarithm Domain[C],” In Proceedings of Picture Coding Symposium 2006.

[105] T. Nguyen, “CE11: Coding of Transform Coefficient Levels With Golomb-Rice Codes[C],” document JCTVC-E253, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[106] T. Nguyen, D. Marpe, H. Schwarz, and T. Wiegand, “Modified Binarization and Coding of MVD for PIPE/CABAC[C],” document JCTVC-F455,Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[107] J. Chen, W. J. Chien, R. Joshi, J. Sole, and M. Karczewicz, “Non-CE1: Throughput Improvement on CABAC Coefficients Level Coding[C],” document JCTVC-H0554, Joint Collaborative Team on Video Coding (JCT-VC), Feb. 2012.

[108] V. Sze and M. Budagavi, “Parallel Context Processing of Coefficient Level[C],” document JCTVC-F130, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[109] H. Sasai and T. Nishi, “Modified MVD Coding for CABAC[C],” document JCTVC-F423, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[110] W.-J. Chien, J. Chen, M. Coban, and M. Karczewicz, “Intra Mode Coding for INTRA_NxN[C],” document JCTVC-I0302, Joint Collaborative Team on Video Coding (JCT-VC), Apr. 2012.

[111] M. Budagavi and M. U. Demircin, “Parallel Context Processing Techniques for High Coding Efficiency Entropy Coding in HEVC[C],” document JCTVC-B088, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2010.

[112] J. Sole, R. Joshi, and M. Karczewicz, “CE11: Parallel Context Processing for the Significance Map in High Coding Efficiency[C],” document JCTVCE338, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[113] V. Sze and M. Budagavi, “Parallelization of HHI_TRANSFORM_CODING[C],” document JCTVC-C227, Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2010.

[114] A. Cheung and W. Lui, “Parallel Processing Friendly Simplified Context

Selection of Significance Map[C],” document JCTVC-D260, Joint Collaborative Team on Video Coding (JCT-VC), Jan. 2011.

[115] M. Zhou, V. Sze, and Y. Mastuba, “A Study on HEVC Parsing Throughput Issue[C],” document JCTVC-F068, Joint Collaborative Team on Video Coding (JCT-VC), Jul. 2011.

[116] V. Sze, “Reduction in Contexts Used for significant_coeff_flag and Coefficient Level[C],” document JCTVC-F132, Joint Collaborative Team on Video Coding (JCT-VC), Jun. 2011.

[117] V. Sze and A. P. Chandrakasan, “Joint Algorithm-Architecture Optimization of CABAC[C],” document JCTVC-E324, Joint Collaborative Team on Video Coding (JCT-VC), Mar. 2011.

[118] J. Chen, W. Chien, M. Karczewicz, X. Li, H. Liu, A. Said, L. Zhang, X. Zhao, “Further improvements to HMKTA-1.0[C],” VCEG-AZ07, 2015.

[119] T. Nguyen, D. Marpe, and T. Wiegand, “Non-CE11: Proposed Cleanup for Transform Coefficient Coding[C],” JCTVC-H0288, 8th Joint Collaborative Team on Video Coding (JCT-VC) Meeting, San Jose, USA, Feb, 2012

[120] E. Lam and J. Goodman, “A mathematical Analysis of the DCT Coefficient Distributions of Images[J],” IEEE Transactions on Image Process, vol. 9, no. 10, pp. 1661-1666, Oct 2000.

[121] E. J. Candes and T. Tao, “Decoding by Linear Programming[J],” IEEE Transactions on Information Theory, vol. 51, no. 12, pp. 4203–4215, Dec 2005.

[122] S. S. Chen, D. L. Donoho, and M. A. Saunders, “Atomic Decomposition by Basis Pursuit[J],” SIAM Journal on Scientific Computing, vol. 20, no. 1, pp. 33–61, Aug 1998.

[123] J. A. Tropp and S. J.Wright, “Computational Methods for Sparse Solution of Linear Inverse Problems[J],” Proceedings of the IEEE, vol. 98, no. 6, June 2010, pp. 948–958.

[124] I. Daubechies, M. Defrise, and C. De Mol, “An Interative Thresholding Algorithm for Linear Inverse Problems with a Sparsity Constraint[J],” Communications on Pure and Applied Mathematics, vol. 57, no. 11, pp. 1413–1457, Nov 2004.

[125] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo, “Sparse

Reconstruction by Separable Approximation[J],” IEEE Transactions on Signal Processing, vol. 57, no. 7, pp. 2479–2493, July 2009.

[126] M. A. T. Figueiredo, R. D. Nowak, and S. J.Wright, “Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems[J],”IEEE Journal on Selected Areas in Communications, vol. 1, no. 4, pp. 586–597, Dec 2007.

[127] S. Mallat and Z. Zhang, “Matching Pursuits with Time-Frequency Dictionaries[J],” IEEE Transactions on Signal Processing, vol. 41, no. 12, pp. 3397–3415, Dec 1993.

[128] J. Tropp and A. Gilbert, “Signal Recovery from Random Measurements Via Orthogonal Matching Pursuit[J],” IEEE Transactions on Information Theory, vol. 53, no.12, pp. 4655–4666, Dec 2007.

[129] L. Gan, “Block Compressed Sensing of Natural Images[C],” In Proceedings of the International Conference on Digital Signal Processing, 2007.

[130] S. Mun and J. E. Fowler, “Block Compressed Sensing of Images Using Directional Transforms[C],” In Proceedings of the International Conference on Image Processing, 2009.

[131] J. E. Fowler, S. Mun, and E. W. Tramel, “Multiscale Block Compressed Sensing with Smoother Projected Landweber Reconstruction[C],” In Proceedings of the European Signal Processing Conference 2011.

[132] C. Chen, E. W. Tramel, and J. E. Fowler, “Compressed-Sensing Recovery of Images and Video Using Multihypothesis Predictions[C],” In Proceedings of the 45th Asilomar Conference on Signals, Systems, and Computers 2011.

[133] J. E. Fowler, S. Mun, and E. W. Tramel, “Block-Based Compressed Sensing of Images and Video[J],” Foundations and Trends in Signal Processing, vol. 4, no. 4, pp. 297–416, Mar 2012.

[134] V. K. Goyal, A. K. Fletcher, and S. Rangan, “Compressive Sampling and lossy Compression[J],” IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 48–56, Mar 2008.

[135] W. Dai, H. V. Pham, and O. Milenkovic, “Distortion-Rate Functions for Quantized Compressive Sensing[C],” In Proceedings of IEEE Information Theory Workshop on Networking and Information Theory 2009.

[136] J. Z. Sun and V. K. Goyal, “Optimal Quantization of Random Measurements In Compressed Sensing[C],” In Proceedings of IEEE International Symposium on Information Theory 2009.

[137] L. Jacques, D. K. Hammond, and J. M. Fadili, “Dequantizing Compressed Sensing: When Oversampling and Non-Gaussian Constraints Combine[J],” IEEE Transactions on Information Theory, vol. 57, no. 1, pp. 559–571, Jan 2011.

[138] L. Wang, X. Wu, and G. Shi, “Binned Progressive Quantization for Compressive Sensing[J],” IEEE Transactions on Image Processing, vol. 21, no. 6, pp.2980–2990, June 2012.

[139] S. Mun, and J. E. Fowler, “DPCM for quantized block based compressive sensing of images[C],” In Proceedings of European Signal Processing Conference 2012.

[140] D. Marpe, H. Kirchhoffer, and G. Marten, “Fast renormalization for H.264/MPEG4-AVC arithmetic coding[C],” In Proceedings of European Signal Processing Conference 2006.

攻读博士学位期间发表的学术论文及研究成果

国际期刊：

1 **Min Gao**, Qiang Wang, Debin Zhao, and Wen Gao, “Arithmetic Coding Using Hierarchical Dependency Context Model for H.264/AVC Video Coding,” **Multimedia Tools and Application**s, vol. 75, no. 12, pp. 7351-7370, June 2016. (SCI检索，IF: 1.346)

2 **Min Gao**, Xiaopeng Fan, Debin Zhao, and Wen Gao, “An Enhanced Entropy Coding Scheme for HEVC”, **Signal Processing: Image Communication**, vol. 44, pp. 108-123, May 2016. (SCI 检索，IF: 1.462)

国际会议：

1 **Min Gao**, Xiaopeng Fan, Qiang Wang, Debin Zhao, and Wen Gao, “A Parallel Context Model for Level Information in CABAC,” in Proc. VCIP11, Tainan, Taiwan, Nov 2011.

2 **Min Gao**, Siwei Ma, Debin Zhao, and Wen Gao, “A Spatial-Interview Auto-Regressive Supper-Resolution Scheme for Multi-view Image via Scene Matching Algorithm”, in Proc. ISCAS 2013, Beijing, China, May 2013.

3 **Min Gao**, Xiaopeng Fan, Tao Zhang, Debin Zhao, and Wen Gao, “An Error Robust Distortion Model for Depth Map Coding in Error Prone Network”, in Proc. ICME2013 Workshop, San Jose, USA, Jul 2013.

4 **Min Gao**, Xiaopeng Fan, Feng Jiang, Debin Zhao and Wen Gao, “Estimation of End-to-end Distortion of Virtual View for Error-Resilience Depth Map Coding”, in Proc ICIP2013. Melbourne, Australia, Sept 2013.

5 **Min Gao**, Qiang Wang, Debin Zhao, and Wen Gao, “Hierarchical Dependency Context Model Based Arithmetic Coding for DCT Video Compression”, in Proc ICIP2014, Paris, France, Oct 2014.

6 **Min Gao**, Burak, Cizemeci, Michael Eiler, Eckehard Steinbach, Debin Zhao, and Wen Gao, “Macroblock Level Rate Control for Low Delay H.264/AVC based Video Communication”, in Proc PV2015, Cairns, Australia, May 2015.

7 Xinwei Gao, Xiaopeng Fan, **Min Gao**, Debin Zhao, “Directional Intra Frame Interpolation for HEVC Compressed Video”, in Proc. ICIP2014, Paris, France, Oct 2014.

8 **Min Gao**, Debin Zhao, Wen Gao, “An Arithmetic Coding Scheme for Block-based Compressive Sensing of Images”, Submitted to VCIP 2016.

AVS 提案：

1 **高敏**，范晓鹏，赵德斌，马思伟，高文，AVS2 算术编码引擎快速归一化方法，AVS_M3496, 第 50 次会议，南京， 2014 年 9 月.

2 **高敏**，范晓鹏，赵德斌，马思伟，高文，AVS2 中 bypass bin 的快速解码方法，AVS_M3497, 第 50 次会议，南京， 2014 年 9 月.

3 **高敏**， 范晓鹏，赵德斌，马思伟，高文，一种 Run-Level 分层编码方法，AVS_M3505, 第 50 次会议，南京， 2014 年 9 月.

H266 提案：

1 **MinGao**, Xiaopeng Fan, Debin Zhao, Improved Context Modeling Scheme for Transform Coefficient Coding，VC-05-061, Da Lian, Sep. 2015.

2 **MinGao**, Xiaopeng Fan, Debin Zhao, Modifications for Binary Arithmetic Coding Engines with Adaptive Probability Update parameter，VC-05-060, Da Lian, Sep. 2015.

3 **MinGao**, Xiaopeng Fan, Debin Zhao, An Enhanced Entropy Coding Scheme for Future Video Coding Standard，VC-06-074, Beijing, Nov. 2015.

专利：

1 赵德斌, **高敏**，范晓鹏，王强, 刘绍辉. 用于二进制算术编码可并行的非零系数上下文建模方法. 专利号 ZL 2011 1 0172229.X, 2013 年 6 月。

2 范晓鹏，**高敏**, 赵德斌, 刘绍辉. 视频压缩中变换系数的上下文建模方法. 专利申请号 2015105473237, 2015 年 8 月。

致 谢

时光飞逝，转眼之间已经过去五年了。回顾这五年攻读博士的光阴，我经历过兴奋，开心，失望，悲伤和彷徨。庆幸的是我在各位老师的悉心指导和同学的无私帮助下找到了自己的研究方向，同时也收获了属于自己的研究成果。正是这样一段复杂的历程，让我慢慢地成长起来，体味到什么是付出，什么是收获，更体会到什么是集体。值此论文完成之际，我要向大家致以诚挚的谢意。

感谢我的导师高文教授，高文教授作为 JDL 实验室的开创者和领导者，用他的辛勤劳动为我们搭建了一个良好的平台，为我们的学术研究和交流提供了优越的环境。实验室浓厚的的学术氛围，高涨的科研热情，老师与同学之间那深厚的感情，都让我受益匪浅。高文教授渊博的知识、严谨的作风和广阔的视野也给我留下极其深刻的印象，高文教授总能用朴素的语言把研究问题描述的如此清晰。

感谢我的副导师赵德斌教授，感谢赵老师多年来在学业上对我不辞辛苦地培养，感谢赵老师多年来在生活上对我如长辈般的关怀。赵老师严谨的治学风格，平易近人的品质和积极乐观的人生态度使我获益良多。攻读博士期间，每当我在学术研究上遇到挫折时，赵老师总是耐心地为我进行学术规划。尤其难忘的是，当我开始写学术文章时，您总是逐句地帮我修改文章，不仅让我学会了做研究的思路，懂得写文章时的表达技巧，更让我感受到了您那一丝不苟的治学精神及对学生无私的关怀。

感谢范晓鹏教授，感谢范老师每次都不厌其烦地为我解答问题，即使我所问的是一个非常简单的问题，您也总是耐心地给我讲解，在您的帮助下，我少走了很多弯路。

感谢实验室的刘绍辉老师和姜峰老师，感谢您们在生活和学术上对我的帮助。每当我们在生活和学习上遇到困难时，我们总是去麻烦您们，您们总是能帮我们解决这些问题。您们为我们的付出，我们都记在了心里，永远不能忘记。

感谢王强师兄和张莉师姐，感谢你们在我初入熵编码这个领域时对我的帮助。还记得硕士期间向你们请教熵编码的问题，你们耐心的解答和中肯的建议使我受益匪浅。

感谢实验室的赵欣师兄，王悦师兄，新峰师兄，苫设师兄，张健师兄和齐峰师兄，谢谢你们在学习生活中对我的无私帮助，你们身上很多优秀的品质都对我产生了深深的影响。感谢北京大学数字媒体所的师弟师妹们，感谢哈工大的师弟师妹们，和你们在一起的日子是那么的美好，至今还时时记起当初的快乐时光。感谢欣玮和赵亮，我们三个一起走过了 7 年时光，7 年间我们相互扶持，相互帮助，此情此景令我难忘。

感谢生我养我的父母，感谢我的哥哥和嫂子，感谢我的妻子，你们的爱是战胜一切苦难的动力之源，也是我心灵的港湾和感情的归宿。

最后，感谢百忙之中抽出时间审阅论文的各位老师！感谢所有关心过我、帮助过我的人！