METHOD FOR PERFORMANCE-COMPLEXITY ANALYSES IN SOC-BASED DESIGNS

SHYAM PARIKKAL KRISHNAMURTHY

(B.E., ANNA UNIVERSITY)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND

COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

ACKNOWLEDGEMENTS

I would like to dedicate this thesis to my family, especially my parents. I am extremely grateful for their understanding and support during the period of my Masters Program.

I would like to express my heartfelt gratitude to my supervisor, Dr Le Minh Thinh, for his valuable guidance and support in my research work. He has provided various constructive suggestions and recommendations for my research work.

I would also like to express my sincere thanks to my colleagues, Tian Xiaohua and Hong Zhiqian for all their help throughout my research work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

B&CM	Binarization & Context Modeling
CABAC	Context Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CIF	Common Intermediate Format
FSM	Finite State Machine
GOP	Group of Pictures
IS	Interval Subdivision
ISA	Instruction Set Architecture
LPS	Least Probable Symbol
MPEG	Moving Picture Expert Group
MPS	Most Probable Symbol
NRDSE	Non-residual Data Syntax Element
QCIF	Quarter Common Intermediate Format
RDO	Rate Distortion Optimization
RDSE	Residual Data Syntax Element
Y-PSNR	Peak Signal-to-Noise Ratio of the Luminance component

Abstract

The analyses of performance versus complexity of all the available algorithms in hardware (HW) and software (SW) are necessary to the study of the effectiveness of the implementation in a SoC-based design environment. Several performance-complexity analyses have been conducted, but no standard method has been reported. In this thesis, we propose a Performance Complexity Index (PCI) to evaluate the cost-effectiveness of implementing one algorithm over the other of the same type, taking into account trade-offs in performance and complexity. Bit-rate and video quality are performance metrics, and number of instructions executed (Computational) and memory accesses (Data Transfer) per second are complexity metrics. As a demonstration, we analyze the performance and complexity of the two contending entropy coders adopted by H.264/AVC: the Context-based Adaptive Binary Arithmetic Coding (CABAC) and the Context-based Adaptive Variable Length Coding (CAVLC), in both variable and constant bit-rate implementations. Empirical test results using standard sequences show that it is more cost-effective to use CABAC for encoding when the Rate-Distortion Optimization (RDO) mode is turned off regardless of motion contents, configurations, in both variable and constant bit-rate implementations. Also, it has been found out using empirical analyses that CABAC is more cost-effective for lower motion content sequences in variable bit-rate implementation when RDO is turned on. The conclusions based on PCIs are also in total agreement with the empirical results.

CHAPTER 1 INTRODUCTION

The study of cost-effectiveness of algorithms plays an important role in SoC co-design flow. In this thesis, we introduce a measure for assessing the cost-effectiveness of an algorithm in any specific scenario. Note that even though there are various strategies and tools to measure complexities, no performance-complexity metrics have been defined. Performance of an algorithm alone is not sufficient to make a design decision. Its implication to the implementation cost is also needed to be taken into consideration. In light of that, we propose a performance-complexity metric in this thesis to facilitate assessment of the cost-effectiveness of any algorithm.

The new video coding standard Recommendation H.264 of ITU-T also known as International Standard 14496-10 or MPEG-4 part 10 Advanced Video coding (AVC) of ISO/IEC [1], [2] is the latest standard in a sequence of the video coding standards. The previous standards, namely H.261 (1990) [3], MPEG-1 Video (1993) [4], MPEG-2 Video (1994) [5], H.263 (1995, 1997) [6], MPEG-4 Visual or part 2 (1998) [7], reflect the technological progress in video compression and the adaptation of video coding to different applications and networks. Video telephony, video on CD, broadcast of TV, and networks used for video communication represent some of the applications where the previous video compression standards were used. The advancements in the field of network access technologies and the increased requirements for bandwidth savings led to the development of H.264/AVC. Evolution of new algorithms in H.264/AVC compression standard made much higher compression of video sequences possible.

In comparison to the previous video compression standards, it provides higher coding performance and better error resilience through the use of improved or new coding tools at different stages of the video coding. Multiple reference frames, 1/4 pel motion compensation, and integer transform are some of the new tools available in the new standard. H.264/AVC offers two new entropy coding schemes for coding its macroblock-level syntax elements: *Context Adaptive Binary Arithmetic Coding* (CABAC) [8] and *Context Adaptive Variable Length Coding* (CAVLC) [9]. For the first time, arithmetic coding is allowed in the compression standards. Both entropy coding schemes achieve better coding efficiency than their predecessors in the earlier standards as they employed context-conditional probability estimates. Comparatively, CABAC performs better than CAVLC in terms of coding efficiency. This is because arithmetic coding allows fractional coding of data, thus making it possible to efficiently encode symbols which exhibit a very high probability of occurrence. On the other hand, variable length codes have a fundamental minimum length limit of one. However, the higher coding efficiency of CABAC comes at the expense of increased complexity in the entropy coder. Arithmetic coding has a very high complexity in general. So as to reduce its complexity, alphabet reduction was used and only binary arithmetic coding is allowed in the new standard. Because of this, multiple passes are required to encode a single symbol using CABAC, which can be encoded by CAVLC in a single pass. This causes a complexity overhead in CABAC. This is one of the reasons why the developer team of H.264/AVC excludes CABAC from the *Baseline* profile [8].

In this work, we conduct comprehensive analyses on entropy coder tools to identify situations where CABAC is seen as more cost-effective than CAVLC at the video coder level and verify them with the proposed *Performance Complexity Index* (PCI). Our approach has a major difference over other approaches: a PCI is proposed that takes into account both performance and complexity to determine the cost-effectiveness of any algorithm over another of same type.

1.1 Research Work

In this work, we propose a performance-complexity co-analysis methodology to identify scenarios where any new algorithm is more cost-effective than the existing algorithm. As an example, we take the new algorithm and existing algorithm as CABAC and CAVLC respectively. CABAC has a higher efficiency, though at the expense of increased complexity, when compared to CAVLC. In this work, we try to determine the scenarios where CABAC is more beneficial than CAVLC. The theoretical complexity models of CABAC and CAVLC are developed. The beneficial scenarios will be determined and assessed theoretically. The theoretical models will also be used in defining a performance-complexity metric that is capable of comparing these two algorithms. Comprehensive performance and complexity analyses of CABAC and CAVLC at the video encoder/decoder levels will be conducted using software verification model. Both variable bit-rate (VBR) video encoder and constant bit-rate (CBR) video encoder will be considered. Bit-rate savings (for VBR) and changes in peak signal-to-noise ratio (for CBR) of the video luminance component (Y-PSNR) will be used as performance metrics. Computational complexity and data transfer complexity will be used as complexity metrics. Based on the empirical data, the beneficial scenarios will be identified. Finally, the performance-complexity metric defined will be used to validate both the theoretical and empirical findings. The goals of the analyses are:

(a) To present theoretical complexity models of CABAC and CAVLC
(b) To identify scenarios where the use of CABAC is more cost-effective than CAVLC

(c) To define a performance-complexity analysis methodology that can be used to compare any algorithms in any scenario taking into account both their performance and complexity for analyses.

(d) To present the computational and memory requirements of CABAC and CAVLC

1.2 Motivation

Performance of an algorithm alone is not sufficient to make a design decision. Its implication to the implementation complexity is also needed to be taken into consideration. Several performance-complexity analyses have been conducted, and no standard method has been reported. In light of that, we propose a performance-complexity analysis metric in this thesis to evaluate the cost-effectiveness of any algorithm over another of the same type, taking into account trade-offs in quality, bit-rate, computational complexity, and data transfer complexity. The need for such analysis methodologies is demonstrated using an example ï entropy coding tools of H.264/AVC video codec.

The CABAC tool is not supported in the *Baseline* profile of H.264/AVC. As such, it is commonly believed that using CABAC is computationally expensive for a video encoder. However, no work has been done on evaluating the complexity requirements of using CABAC except in [10], which gives a brief assessment of the effect of using CABAC on the video encoderôs data transfer complexity. (More details on the related works that have been carried out for H.264/AVC are given in Chapter 2.)

[10] conducted an overall cost-efficiency study of various video tools proposed in the H.264/AVC, and reported that CABAC results in up to 16 % in bit-

rate reduction, but entails an access frequency increase from 25 to 30%. However, the cost-efficiency relationship was reported by the low bit-rates, high PSNR, and comparable memory access and coding-time complexities. The complexity evaluation of CABAC was done only in one specific encoder configuration. No cost-efficiency relationship was established. Moreover, it also failed to include any complexity analyses of using CABAC at the decoder.

There are several drawbacks in conclusions obtained from evaluating the complexity increment of using CABAC over CAVLC empirically. The major limitation is the inability to compare the performance and complexity of CABAC and CAVLC across different video coder settings. The results can be misleading as such complexity figures also depend on the choices of coding tools used in the video encoder. This makes comparison of such figures across different configurations less meaningful. Analyzing the complexity and performance of CABAC from the perspective of the video encoder will be difficult for implementers who wish to achieve a cost-effective realization of the video codec, as the performance and complexity not only depend on coder settings, but also on the video content. It is also less relevant for system designers of CABAC because of their requirement to design for all the coder settings and video sequences having different properties. Rather, they will all be more interested in the complexity performance of CABAC from the perspective of the entropy coder.

As such, these provide the motivation for comprehensive co-analyses on the performance and complexity of CABAC. It also gives enough reason to define a common performance-complexity metric which could be used to compare the cost-effectiveness of any algorithm over a contending algorithm across various scenarios and video coder settings.

1.3 Thesis Contributions

The thesis contributions are as follows. I have:

(a) developed a theoretical complexity model for entropy coders of H.264/AVC video codec that can be used across multiple scenarios

(b) defined a performance-complexity methodology that can be used for comparison of algorithms taking into consideration both performance and complexity

(c) provided findings from co-evaluation of performance-complexity analyses of CABAC and CAVLC- that can assist implementer in deciding whether to use CABAC in the video encoder

(d) determined scenarios where CABAC is more beneficial at a system level, which can be used both by implementers and system level designers

(e) identified possible bottlenecks in CABAC and suggests recommendations on complexity reduction to system designer and software developers,

(f) identified when the use of CABAC hardware accelerator may not be necessarily helpful in the video encoder, and

(g) developed a set of profiler tools based on Pin [11], [12] for measuring computational and data transfer complexity of H.264/AVC that can also be used for any other video codec.

1.4 Thesis Organization

The contents in this thesis are organized as follows. In chapter 2, an overview of Context Adaptive Binary Arithmetic Coding (CABAC), a review of the complexity analysis methodologies that have been used for video multimedia system, and a literature review of existing works will be given. Chapter 3 provides the theoretical complexity models of CABAC and CAVLC, and also derives the Performance-Complexity Index (PCI), a metric to compare the performances and complexities of CABAC and CAVLC and to determine their suitability in any scenario. In Chapter 4, the performance and complexity of CABAC, benchmarked against CAVLC is given for the different video configurations so as to explore the inter-tool dependencies. Also, a performance-complexity co-evaluation is conducted to determine scenarios where CABAC is more beneficial empirically. Chapter 5 provides theoretical observations, a description of the performance-complexity analysis methodology, and uses the performance-complexity co-analysis methodology using PCI to quantitatively determine cost-effective scenarios of CABAC over CAVLC. Finally, conclusions are drawn in Chapter 6.

CHAPTER 2 BACKGROUND

In this chapter, the role of the entropy coder is discussed and an overview of CABAC is given, followed by the presentation of the different encoder controls. Lastly, a review of the complexity analysis methodologies that have been used for video multimedia system, and a literature review of existing works will be given.

2.1 Entropy Coder

H.264/AVC employs three types of compression techniques to effectively remove redundancy ï temporal, spatial, and statistical. Statistical compression, also called entropy coding, is lossless in nature. This means no information is lost after statistical compression, and all the information that was compressed can be retrieved after decompression. However, the main limitation of entropy coder (and decoder) in H.264/AVC is that the coding process cannot be parallelized. Thus they become the bottlenecks in multiprocessor systems, where all the other stages of H.264/AVC can be parallelized. So it becomes extremely important to study the performance gain obtained and the increase in complexity incurred of entropy coders.

H.264/AVC offers two entropy coding schemes ï CAVLC and CABAC. Note that previous video coding standards assumed stationary underlying statistics. So only specifically tailored but fixed VLCs were used. Context adaptation is introduced only in H.264/AVC. The entropy coder may serve up to two roles in a H.264/AVC video encoder. The primary role of the entropy coder is to generate the compressed bitstream of the video file for transmission or storage. For video encoders that optimize its mode decision using rate-distortion optimization (RDO), its entropy coder performs an additional role during the mode selection stage. The entropy coder computes the bit-rates needed by each candidate prediction mode. The computed rate information is then used to guide the mode selection.

2.2 Overview of CAVLC

This is the method used to encode residual, zig-zag ordered 4x4 (and 2x2) blocks of transform coefficients. CAVLC is designed to take advantage of several characteristics of quantized 4x4 blocks:

1. After prediction, transformation and quantization blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to compactly represent strings of zeros.
2. The highest non-zero coefficients after the zig-zag scan are often sequences of +/-1. CAVLC signals the number of high-frequency +/-1 coefficient (ñTrailing 1sò or ñT1sò) in a compact way.
3. The number of non-zero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table; the choice of look-up table depends on the number of non-zero coefficients in neighbouring blocks.
4. The level (magnitude) of non-zero coefficients tends to be higher at the start of the reordered array (near the DC coefficient) and lower towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the ñlevelò parameter depending on recently-coded level magnitudes.

CAVLC encoding of a block of transform coefficients proceeds as follows.

1. Encode the number of coefficients and trailing ones (coeff_token)

The first VLC, coeff_token, encodes both the total number of non-zero coefficients (TotalCoeffs) and the number of trailing +/-1 values (T1). TotalCoeffs can be anything from 0 (no coefficients in the 4x4 block) 1 to 16 (16 non-zero coefficients). T1 can be anything from 0 to 3; if there are more than 3 trailing +/-1s, only the last 3 are treated as ñspecial casesò and any others are coded as normal coefficients. There are 4 choices of look-up table to use for encoding coeff_token, described as Num-VLC0, Num-VLC1, Num-VLC2 and Num-FLC (3 variable-length code tables and a fixed-length code). The choice of table depends on the number of non-zero coefficients in upper and left-hand previously coded blocks Nu and NL.

2. Encode the sign of each T1

For each T1 (trailing +/-1) signalled by coeff_token, a single bit encodes the sign (0=+, 1=-). These are encoded in reverse order, starting with the highest-frequency T1.

3. Encode the levels of the remaining non-zero coefficients.

The level (sign and magnitude) of each remaining non-zero coefficient in the block is encoded in reverse order, starting with the highest frequency and working back towards the DC coefficient. The choice of VLC table to encode each level adapts depending on the magnitude of each successive coded level (context adaptive). There are 7 VLC tables to choose from, Level_VLC0 to Level_VLC6. Level_VLC0 is biased towards lower magnitudes; Level_VLC1 is biased towards slightly higher magnitudes and so on. The choice of table is adapted in the following way:

(a) Initialise the table to Level_VLC0 (unless there are more than 10 non-zero coefficients and less than 3 trailing ones, in which case start with Level_VLC1).

(b) Encode the highest-frequency non zero coefficient.

(c) If the magnitude of this coefficient is larger than a pre-defined threshold, move up to the next VLC table.

In this way, the choice of level is matched to the magnitude of the recently-encoded coefficients.

4. *Encode the total number of zeros before the last coefficient.*

TotalZeros is the sum of all zeros preceding the highest non-zero coefficient in the reordered array. This is coded with a VLC. The reason for sending a separate VLC to indicate TotalZeros is that many blocks contain a number of non-zero coefficients at the start of the array and (as will be seen later) this approach means that zero-runs at the start of the array need not be encoded.

5. *Encode each run of zeros.*

The number of zeros preceding each non-zero coefficient (run_before) is encoded in reverse order. A run_before parameter is encoded for each non-zero coefficient, starting with the highest frequency, with two exceptions:

(a) If there are no more zeros left to encode (i.e. ? [run_before] = TotalZeros), it is not necessary to encode any more run_before values.

(b) It is not necessary to encode run_before for the final (lowest frequency) non-zero coefficient.

The VLC for each run of zeros is chosen depending on (a) the number of zeros that have not yet been encoded (ZerosLeft) and (b) run_before. For example, if there are only 2 zeros left to encode, run_before can only take 3 values (0,1 or 2) and so the VLC need not be more than 2 bits long; if there are 6 zeros still to encode then run_before can take 7 values (0 to 6) and the VLC table needs to be correspondingly larger.

2.3 Arithmetic Coding and Overview of CABAC

2.3.1 Arithmetic Coding

Arithmetic coding is a type of entropy coding technique which can efficiently encode fractional codewords. This section explains the basic methodology of binary arithmetic coding and provides an insight into the cause of its efficiency. As this section is meant only to provide only an understanding, we will assume stationary probability model for simplifying the derivations.

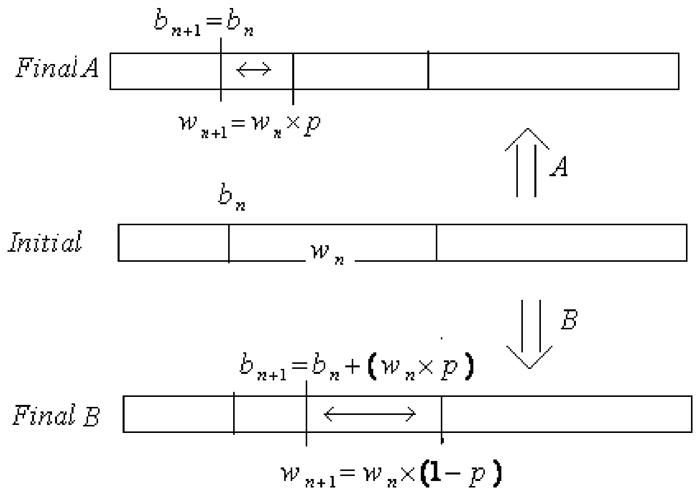


Figure 2.1: Arithmetic Coding Subdivision

Let us consider an information source that is capable of generating two symbols A and B, with probabilities *p* and *(1-p)* respectively. During the entire derivation, we will consider an interval of the form *[b, w]*, where *b* is the base of the interval and *w* is the width of the interval. Let us consider an initial interval of [0, 1]. As and when we encounter symbols generated from the source, we subdivide the interval according to the probabilities of the symbols A and B as follows.

$$[b_{n+1}, w_{n+1}] = \begin{cases} [b_n, w_n \times p] | sym = A \\ [b_n + w_n \times p, w_n \times (1-p)] | sym = B \end{cases} \tag{2-1}$$

Figure 2.1 is an attempt to shows the arithmetic coding subdivision as shown by the equation (2-1) visually. After subdividing the intervals for all the generated symbols, we can consider any of the numbers in the final interval to be the encoded message. However, for any practical purpose, the number in the interval with the least length (number of bits) is chosen to be encoded message.

Note that p and $(1-p)$ are probabilities and have values lesser than 1. So, after each subdivision of the interval, the width w becomes smaller. In the final interval $[b_N, w_N]$, the width w_N will be the contributing factor to the final length. The length of the final encoded message will be, at max, the length of the interval. If w_N is the interval value, its representation in binary will require $\log_2(w_N)$ bits, which will be the length of the final codeword.

Let the source generate a total of N symbols, out of which let M { $= p*N$ } be A, and $(N-M)$ { $(1-p)*N$ } be B. Consider the procedure used for sub-dividing the interval. The width is multiplied by the probability of the occurring symbol each time. So, the final value of w_N will be $p^M * (1-p)^{(N-M)}$. Thus, the final length will be

$$L = N \times [p \times \log_2(p) + (1-p) \times \log_2(1-p)] \tag{2-2}$$

So the number of bits per symbols is

$$l = p \times \log_2(p) + (1-p) \times \log_2(1-p) \tag{2-3}$$

Note that the above expression is also the Shannonôs limit on the minimum encoded symbol length possible. This proves that arithmetic coding can achieve the Shannonôs limit and thus explains why it is very efficient.

Also, consider a case where p is very much greater than 0.5, for instance 0.95. In this case, arithmetic coding can actually encode each symbol with number of bits per symbols approx. 0.2 (by substituting the probability values in the above equation), which means it can encode using fractional length. However, the variable length codes have a lower limit of 1 bit/symbol.

2.3.2 Overview of CABAC

Context-based Adaptive Binary Arithmetic Coding (CABAC) [8] is the more efficient of the two entropy coding schemes in H.264/AVC. It is not supported in the *Baseline* profile. The following figure shows the block diagram of CABAC encoder and decoder.

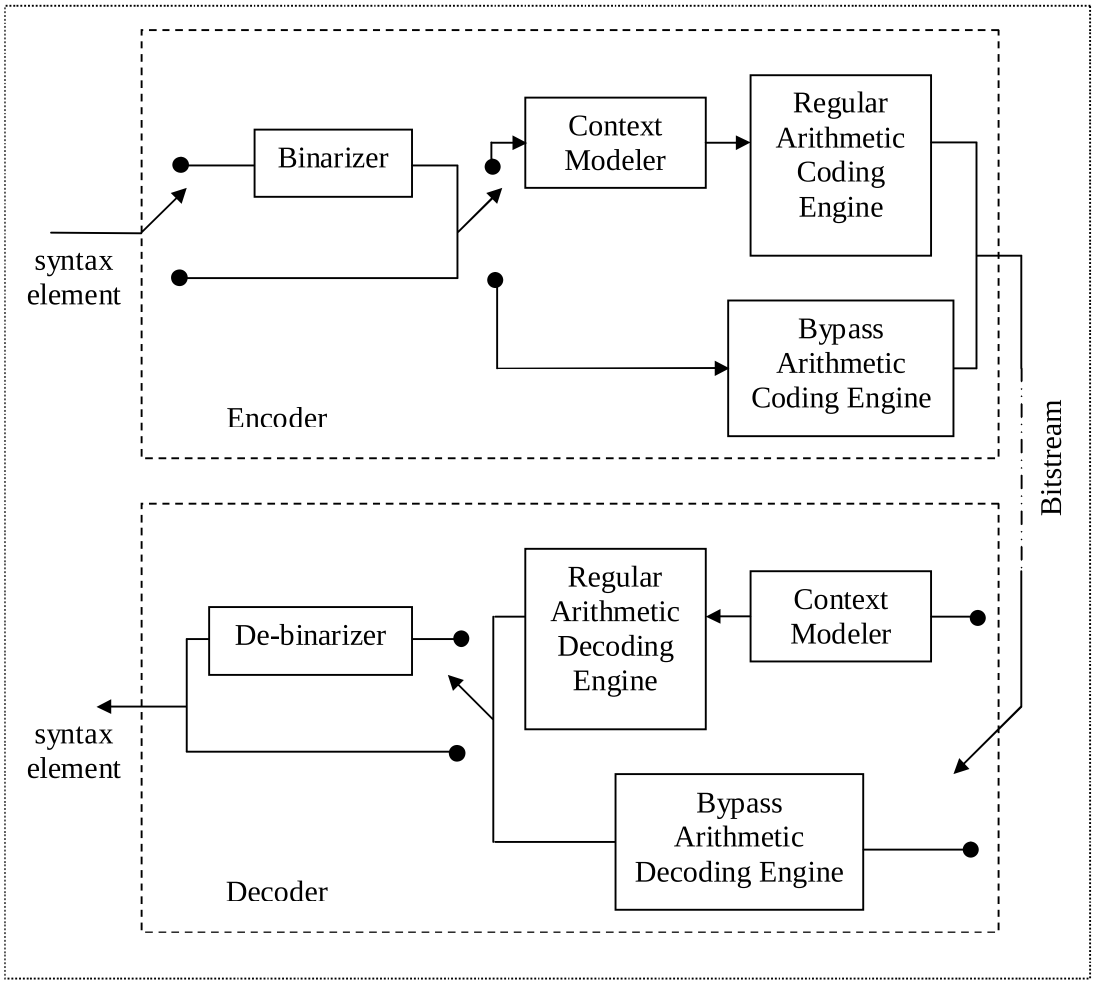

Binarizer
Context Modeler
Regular Arithmetic Coding Engine
syntax element
Bypass Arithmetic Coding Engine
Encoder
Bitstream
Regular Arithmetic Decoding Engine
De-binarizer
Context Modeler
syntax element
Bypass Arithmetic Decoding Engine
Decoder

Figure 2.2: CABAC entropy coder block diagram

The encoding/decoding process using CABAC comprises of three stages, namely binarization, context modeling and binary arithmetic coding.

2.3.3 Binarization

Arithmetic coding, in general, is extremely computationally intensive. So, H.264 supports only Binary Arithmetic Coding. Binarization block takes care of the alphabet reduction.

The binarization stage maps all non-binary syntax elements into binary codewords known as bin-string using *Unary / k^{th} order Exp-Golomb* (UEGk) binarization scheme. The Truncation Unary prefix part is context adaptive. However, on the other hand, the Exp-Golomb suffix part uses stationary context. Typically, for larger values, the EGk suffix part represents already a fairly good fit of the probability distribution.

2.3.4 Context Modeling

Note that proper probability distribution of the symbols is required to be known for efficient arithmetic coding. Thatô's where the Context Modeling stage comes into picture. Each bin in a bin string is encoded in either *normal* mode or *bypass* mode depending on the semantic of the syntax. For a bypass bin, the context modeling stage is skipped because a fixed probability model is always used. On the other hand, each normal bin selects a probability model based on its context from a specified set of probability models in the context modeling stage. In total, 398 probability models are used for all syntax elements.

There are four types of context. The type of context used by each normal bin for selecting the best probability model depends on the syntax element that is encoded. The first type of context considers the related bin values in its neighboring macroblocks or sub-blocks. The second type of context considers the values of the prior coded bins of the bin-string. These two types of contexts are only used for non-residual data syntax elements (NRDSE). The last two types of context are only used for residual data syntax elements (RDSE). One of them considers the position of the syntax element in the scanning path of the macroblock whereas the other evaluates a count of non-zero encoded levels with respect to a given threshold level.

2.3.5 Arithmetic Coding

In the binary arithmetic coding (BAC) stage, the bins are arithmetic coded. This follows the methodology described in Section 2.2. Binary arithmetic coding is based on the principle of recursive sub-division of an interval length as follows:

$$E_{LPS} = P_{LPS} \cdot E \tag{2-4}$$

$$E_{MPS} = E - E_{LPS} \tag{2-5}$$

$$L_{LPS} = L + E - E_{LPS} \tag{2-6}$$

$$L_{MPS} = L \tag{2-7}$$

where E denotes the current interval length, L denotes the current lower bound of E, P_{LPS} denotes the probability of least probable symbol (LPS) from the selected probability model. E_{LPS} and E_{MPS} denote the new lengths of the partitioned intervals corresponding to LPS and the most probable symbol (MPS). L_{LPS} and L_{MPS} denote the corresponding lower bounds of the partitioned intervals. For each bin, the current interval is first partition into two as given in equations from (2-4) to (2-7). The bin value is then encoded by selecting the new partitioned length that corresponds to the bin value (either LPS or MPS) as the new current interval. E and L are also referred as the coding states of the arithmetic coder.

In H.264/AVC, the multiplication operation of interval subdivision in Eqn. 2-4 is very computation intensive. So it is replaced by a finite state machine (FSM) with a look-up table of pre-computed intervals as follows:

$$E_{LPS} = RangeTable[\check{P}_{LPS}][\check{E}] \tag{2-8}$$

The FSM consists of 64 probability states, $\check{P}_{LPS}$ and 4 interval states, $\check{E}$. For the normal bins, the selected conditional probability model is updated with the new statistic after the bin value is encoded. Note that the 64 probability states are for the LPS, whose probability lies in the interval [0, 0.5]. So, the total probability states considered is actually 128.

2.3.6 Renormalization

To prevent underflow, H.264/AVC performs a renormalization operation when the current interval length, E falls below a specified interval length after coding a bin. This is a recursive operation which resizes the interval length through scaling till the current interval exceeds the specified interval length. The codeword is output on the fly each time bits are available after the scaling operation.

2.4 Encoder Control

The encoder control refers to the strategy used by the encoder in selecting the optimal prediction mode to encode each macroblock. This forms part of the motion estimation block of H.264/AVC. In H.264/AVC, the encoder can select from up to 11 prediction modes: 2 Intra prediction modes and 9 Inter prediction mode, including *SKIP* and *DIRECT* modes to encode a macroblock. Note that the encoder control is a non-normative part of the H.264/AVC standard. Several encoder controls have been proposed and are given below.

2.4.1 Non-RDO encoder

For a non-RDO encoder, either the sum of absolute difference (SAD) or the sum of absolute transform difference (SATD) can be used as the selection criteria. The optimal prediction mode selected to encode the macroblock corresponds to the prediction mode that minimizes the macroblock residual signal, i.e. the minimum SAD or SATD value.

2.4.2 RDO encoder

For a RDO encoder, a rate-distortion cost function is used as the selection criteria for the optimal mode and is given as

$$J = D + \lambda R \tag{2-9}$$

where J is the rate-distortion cost, D the distortion measure, λ the Lagrange multiplier, and R the bit-rate. The optimal prediction mode used to encode the macroblock corresponds to the prediction mode that yields the least rate-distortion cost. Note that to obtain the bit-rate, entropy coding has to be performed for each candidature mode. This significantly increases the amount of entropy coding performed in the video encoder.

Also, another interesting observation here is that even though distortion and rate are not linearly related, we consider them to be linearly related via a Lagrange multiplier in the above equation.

2.4.3. Fast-RDO encoder

The fast-RDO encoder employs the fast RDO algorithm proposed in [13]. Similar to the RDO encoder, it uses the rate-distortion cost function in Eqn. 2-4 as the selection criteria. However, it does not perform an ñexhaustiveò search through all candidate prediction modes. Rather, it terminates the search process once the rate-distortion cost of a candidate prediction mode lies within a threshold - a value derived from the rate-distortion cost of the co-located macroblock in the previous encoded frame. The current candidate prediction mode whose rate-distortion cost lies within the threshold is selected as the optimal prediction mode, and the remaining prediction modes are bypassed. If none of the prediction modes meets the early termination criteria, the prediction mode with the least rate-distortion cost is then selected as the optimal prediction mode.

2.5 Complexity Analysis Methodologies

In this section, a review of the known complexity analysis methodologies is given. Complexity analyses are often carried out using verification models software (in the case of video standards) such as the Verification Model (VM) and the Joint Model (JM) reference software implementations for MPEG-4 and H.264/AVC respectively. These are unoptimized reference implementations but are sufficient for analyzing the critical blocks in the algorithm for optimization and discovering the bottlenecks. On the other hand, optimized source codes are needed or preferred for complexity evaluation when performing hardware / software partitioning as in [14] or when comparing the performance-complexity between video codec as in [15].

2.5.1 Static Code Analysis

Static Code Analysis is a methodology of analyzing programs without actually executing them. Note that any program will contain lots of branching instructions. If N is the number of branching instructions in the program, the order of the number of possible paths taken by the program increases as $O(2^N)$. When a program is executed, only one of all the possible paths is taken by the program. However, any Static Code Analyzer considers all the possible paths to determine the worst and average complexities of the program. Static code analysis is one way of evaluating the computational complexity of an algorithm, a program or a system. Such analysis requires the availability of the high-level language source code such as the C codes of the Joint Model (JM) reference software of H.264/AVC. The methods based on such analysis includes counting the number of line-of-code (LOC), counting the number of arithmetic and logical operations, determining the time complexity of the algorithms, and determining the lower or upper bound running time of the program by explicit or implicit enumeration of program paths [16]. Such analyses measure the algorithmǿs efficiency but do not take into considerations the different input data statistic. In order to obtain an accurate static analysis, restricted programming style such as absence of recursion, dynamic data structure and bounded loop are needed so that the maximal time spent in any part of the program can be calculated.

2.5.2 Run-time Computational Complexity Analysis

For run-time complexity analysis, profiling data are collected when the program executes at run time on a given specific architecture. The advantage of run-time complexity analysis is that input data dependency is also included. One method of run-time computational complexity analysis is to measure the execution time of the program using ANSI C clock function [17]. An alternative is to measure the execution time of the program in terms of clock cycles using tools like Intel VTune, an automated performance analyzer or PAPI, a tool that allows access to the performance hardware counters of the processor for measuring clock cycle [18].

Function-level information can also be collected for coarse complexity evaluation using profilers such as Visual Studio Environment Profiling Tool or Gprof [19]. Such profiling tools provide information on function call frequency and the total execution time spent by each function in the program. This information allows identifying the critical functions for optimization and help partial redesign of the program to reduce the number of function calls to costly functions.

On a finer granularity, instruction level profiling can be carried out to provide the number and the type of processor instructions that were executed by the program at run-time. This can be used for performance tuning of program and to achieve more accurate complexity evaluation. However, the profiling data gathered is dependent on the hardware platform and the optimization level of the compiler. Unfortunately, there were few tools assisting this level of profiling. In [20], a simulator and profiler tool set based on SimpleScalar framework [21] was developed to measure the instruction level complexity. In our work, a set of profiler tools using Pin was developed to measure the instruction level complexity of the video codec [11], [12].

2.5.3 Data Transfer and Storage Complexity Analysis

Data transfer and storage operation is another area where complexity of the program can be evaluated. Such analyses are essential for data-dominant applications such as video multimedia applications where it has been shown that the amount of data transfer and storage operations are at least of the same order of magnitude as the amount of arithmetic operations [22]. For such application, data transfer and storage will have a dominant impact on the efficiency of the system realization.

Data transfer and storage complexity analyses have been performed for a MPEG 4 (natural) video decoder in [22] and H.264/AVC encoder/decoder in [10] using ATOMIUM [23], an automated tool. This tool measures the memory access frequency (the total number of data transfers from and to memory per second) and the peak memory usage (the maximum amount of memory that is allocated by the source code) of the running program. Such analysis allows identifying memory related hotspots in the program, and optimization of the storage bandwidth and the storage size. However, the drawback of this tool is that it uses a ñflat memory architectural modeò and does not consider other memory hierarchy such as one or more levels of caches.

2.5.4 Platform Dependent /Independent Analysis

Generally, two types of complexity analyses can be performed: platform dependent and platform independent. The complexity evaluation using automated tools like VTune and Pin are platform dependent, specifically for general purpose CISC processors such as Pentium 3 and Pentium 4.

Platform independent analysis are generally preferred compared to platform dependent analysis as the target architecture on which the system will be realized is most likely different from that used to compile and run the reference implementation. Tools such as ATOMIUM and SIT [24] are developed with such a goal: to measure the complexity of a specific implementation of an algorithm independent from the architecture that is used to run the reference implementation. Besides these tools, a complexity evaluation methodology for video applications that is platform independent is also proposed in [25]. In its methodology, the platform-independent complexity metric used is the execution frequencies of core tasks executed in the program and is combined with the platform-dependent complexity data (e.g. the execution time of each core task on different processing platforms) for deriving the system complexity on various platforms. However, this approach requires implementation cost measures for each single core task on different hardware platform to be available in the first place before the system complexity can be calculated. A similar platform-independent complexity evaluation methodology is also given in [26]. The difference lies in that for its platform-independent complexity data, it counts both the frequencies of the core tasks and the number of platform-independent operations performed by each core task. The platform-dependent data is simply a mapping table that identifies the number and types of execution subunits in each hardware platform that are capable of performing basic operations in parallel. As such, this methodology removes the needs for obtaining the implementation cost measure of each core task for the different platform but leads to a lower bound of the complexity measure, which is a few factors lower than the actual complexity.

2.6 Existing Works

In most works, the complexity analyses of H.264/AVC are performed on general-purpose processor platforms. In [17], the complexity of H.26L (designation of H.264 in the early stage of development) decoder is evaluated using two implementations and benchmark against a highly optimized H.263+ decoder. One of the implementations is a non-optimized TML-8 reference version and the other is a highly optimized version. In their work, the execution time (measured using the ANSI

C clock function) is used as the complexity metric. The complexity of CABAC which falls into the high complexity profile of H.26L was not evaluated.

Marpe *et al.* [8] reported that CABAC performs better than the baseline entropy coding method of H.264/AVC, i.e. CAVLC, with a range of acceptable video quality of about 30 to 38 dB, and an average bit-rate reduction of 9 to 14 %.

In [26], the complexity of the H.264/AVC *baseline* profile decoder is analyzed using a theoretical approach. This approach allows the computational complexity of the decoder to be derived for various hardware platforms, thereby allowing classes of candidate platforms that are suitable for the actual implementation to be identified easily. The number of computational operations is used as the complexity metric in their work. The theoretical approach is as follow: for each sub-function, its complexity is estimated using the number of basic computational operations it performs on a chosen hardware platform and its call frequency. The number of basic computational operations it performed on each hardware platform varies depending on the number of execution subunits available in each hardware platform. These execution subunits allow basic operations such as *ADD32, MUL16, OR, AND, Load* and *Store* to be performed in parallel. The draw-back of theoretical complexity analysis is that overhead operations such as loop overhead, flow control and boundary condition handling are not included. The run-time complexity of the decoder running on an Intel Pentium 3 platform is also measured using Intel VTune, an automated performance analyzer tool. Compared to the measured complexity by VTune, the estimated complexity of the H.26L decoder using the theoretical approach for the same platform is some factor lower, giving a lower-bound of the actual computational complexity of the decoder. The complexity of CABAC is not evaluated in their work as it does not fall into the baseline profile.

In [27], the performance and complexity of H.26L video encoder are given and are benchmark against the H.263+ video encoder. The complexity analysis is carried out at two levels: the application level and the kernel (or function) level. At the application level, the complexity metric used is the execution time (measured using the ANSI C clock function) whereas at the kernel level, the number of clock cycles (measured using Intel VTune) is used as the complexity metric. In [27], the authors studied the performance and complexity for a set of specific video coder settings for low bit-rate sequences. However, no performance-complexity relating metric was proposed for use across different scenarios.

In [10], the performance and complexity of H.264/AVC video encoder/decoder are reported. Unlike earlier works which focus on computational complexity, this work focused on data transfer and storage requirements. Such an approach proved to be mandatory for efficient implementation of video systems due to the data dominance of multimedia applications [28], [29]. To provide the support framework for automated analysis of H.264/AVC using the JM reference implementation, the C-in-C-out ATOMIUM Analysis environment has been developed. It consists of a set of kernels that provide functionalities for data transfer and storage analysis. In this work, all the coding tools have been used, including the use of B-frame, CABAC and multi-reference frame that were not evaluated in other works. Furthermore, the complexity analysis in this work explores the inter-dependencies between the coding tools and their impact on the trade-off between coding efficiency and complexity. This is unlike earlier works where the coding tool under evaluation is tested independently by comparing the performance and complexity of a basic configuration with the use of the evaluated tool to the same configuration without it.

In [20], the instruction level complexities of the H.264/AVC video encoder/decoder are measured using a simulator and profiler tool set based on the SimpleScalar framework. Similar to [10], the complexity analysis is carried out on a tool-by-tool basis using the JM reference implementation. However, it addressed the instruction level complexity in terms of arithmetic, logic, shift and control operations that were not covered in [10]. It also proposed a complexity-quality-bit-rate performance metric for examining the relative performance among all configurations used for the design space exploration.

Ostermann *et al.* [30] presented a good review on H.264/AVC codec, and the performance of CABAC was reportedly similar to that mentioned in [10]. Among the reports on hardware implementations of CABAC [31]-[35], Osorio *et al.* [31], [35] claimed that Rate-Distortion Optimization (RDO-on) increases CABACös load by two orders of magnitude. Nunez-Yanez *et al.* [34] did not report the additional complexity of CABAC under RDO-on, yet claimed that the combined effect of RDO-on and CABAC gave rise to an additional 20 % savings in bit-rate.

In [35], Kannangara *et al.* proposed a method to control the rate in a real-time system which also takes into account the distortion, rate, and complexity. However, the methodology proposes selections between coding a particular frame (or macroblock) and not coding it to reduce complexity. The paper, however, does not offer a selection criterion for choosing the best video coder configuration to encode the entire sequence. In [36], Tu *et al.* proposed an R-D model that could be used for making the mode decision at reduced complexity with performance comparable to that of the high complexity method proposed by H.264/AVC. However, the paper does not assess the effectiveness of CABAC or CAVLC in any specific situation.

2.7 Conclusion

In this chapter, an overview of CAVLC, arithmetic coding and the main functional blocks of CABAC, and a review of the encoder controls of the video encoders have been given. This is followed by a discussion on the known methodologies used in evaluating complexity and the existing work that have been carried out for complexity evaluation of H.264/AVC. In the next chapter, the performance of CABAC, benchmarked against CAVLC for different video encoder configurations will be presented.

CHAPTER 3 DEVELOPMENT OF THEORETICAL MODELS

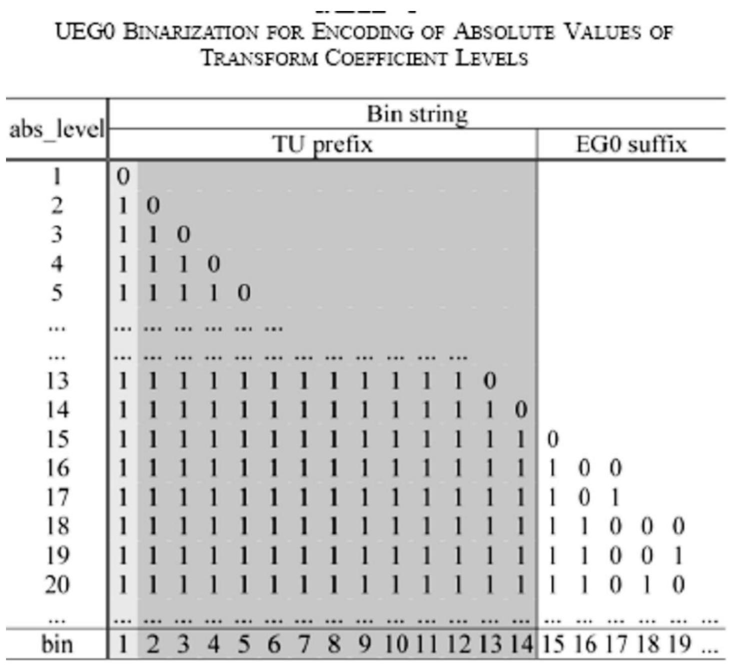
3.1 Introduction

In this chapter, the theoretical complexity model of CABAC and CAVLC are introduced. With the help of the complexity models a performance-complexity related parameter, Cost Effectiveness *(CE)*, is derived. The Performance-Complexity Index (*PCI*), a metric to compare the performances and complexities of a new algorithm in comparison to an existing algorithm and to determine their suitability in any scenario, is defined. This PCI will be used in later chapters to determine the cost-effective scenarios of using CABAC over CAVLC. In the last section, some theoretical observations are made regarding the cost-effective scenarios of CABAC over CAVLC. These observations are validated using the PCI in later chapters.

3.2 CABAC Complexity Model

The complexity of CABAC is proportional to the number of times its Context Modeler and Coding Engine are run for encoding significant coefficients, as they contribute the most to the complexity of the CABAC module. The number of times the CABAC engine is run is proportional to the length (number of bins) of the binary codewords. The binary codewords themselves depend on the value of the non-binary syntax elements. The Binarizer converts non-binary syntax elements to binary codewords. To determine the number of times the CABAC engine is run for a non-zero significant coefficient, we have to consider the Binarization process of CABAC.

The significant coefficients are binarized using Unary / 0th order Exp-Golomb (UEG0) binarization [8] as shown in the table below.



UEG0 BINARIZATION FOR ENCODING OF ABSOLUTE VALUES OF TRANSFORM COEFFICIENT LEVELS

abs_level	Bin string	
	TU prefix	EG0 suffix
1	0	
2	1 0	
3	1 1 0	
4	1 1 1 0	
5	1 1 1 1 0	
...	... ...	
...	... ...	
13	1 1 1 1 1 1 1 1 1 1 1 1 0	
14	1 1 1 1 1 1 1 1 1 1 1 1 1 0	
15	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0
16	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 0
17	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 1
18	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 0 0 0
19	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 0 0 1
20	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 0 1 0
...	... ...	... ...
bin	1 2 3 4 5 6 7 8 9 10 11 12 13 14	15 16 17 18 19 ...

Figure 3.1: Unary/0th Order Exp-Golomb Binarization

For any significant coefficient of value x, the length of its corresponding binary codeword $l(x)$, which represents the number of bins in the codeword, can be written as:

$$l(x) = \begin{cases} x \mid 1 \le x \le 14 \\ 15 + 2\log_2(x - 14) \mid x > 14 \end{cases} \tag{3-1}$$

For determining the complexity of CABAC we would be interested in the expected number of CABAC engine runs, instead of the number of runs due to any specific significant coefficient. We know that the expected number of runs of CABAC engine is equal to the expected length *E{l(x)}* of the binary codewords.

We know that

$$E\{l(x)\} = \sum_{x} p(x) \times l(x) \tag{3-2}$$

Where *p(x)* is the probability of occurrence of a significant coefficient of value *x* and *l(x)* is its corresponding length. The above equation can be written as

$$E\{l(x)\} = \sum_{x=1}^{14} p(x) \times l(x) + \sum_{x=15}^{\infty} p(x) \times l(x) \tag{3-3}$$

Also, while the probability of the Truncation Unary part of any binary codeword is context adaptive, the probability of the each bin being 1 or 0 in the 0^{th} order Exp-Golomb part of the binary codeword is always 0.5. So, for a codeword corresponding to a significant coefficient x greater than 14, the probability will be

$$\begin{aligned} & p(x \mid x > 14) = p[l(x) = 15 + 2 \times \log_2(x-14)] \\ & = p(14\, ones) \times 2^{-(1+\log_2(x-14))} \end{aligned} \tag{3-4}$$

Therefore,

$$E\{l(x)\} = \sum_{x=1}^{14} [p(x) \times l(x)] + [P(14\, ones) \times \sum_{x=15}^{\infty} (2^{-(1+\log_2(x-14))} \times (1 + 2\log_2(x-14)))] \tag{3-5}$$

Now, consider the second summation. It is possible to calculate the value of the second summation because of the following reasons:

- The probabilities are known
- The summation to infinity converges to a value. This is because the exponential decrease is much larger than logarithmic increase.

The value of the summation is

$$\sum_{15}^{\infty} 2^{-(1+2*\log(x-14))} * (1+2*\log(x-14)) = 7.35 \tag{3-6}$$

Thus,

$$E\{l(x)\} = \sum_{x=1}^{14} [p(x) \times l(x)] + (P(14\ ones) \times 21.35) \tag{3-7}$$

Now, let us define *l*¢(x) as follows

$$l'(x) = \begin{cases} x \mid 1 \le x \le 14 \\ 21.35 \mid x > 14 \end{cases} \tag{3-8}$$

Now, the equation of *E{l(x)}* can be rewritten as follows

$$E\{l'(x)\} = \sum_{x=1}^{14} p(x) \times l'(x) \tag{3-9}$$

Where *l*¢(x) is defined as above.

The Context Modeler and Coding Engine are run once for each and every bin in the binary codeword. For a non-binary significant coefficient of value *x* the complexity of the CABAC will be directly proportional to E{*l*(*x*)}. So, the complexity of CABAC module can be seen as being directly proportional to the value of *x* (for values of *x* less than 14). The complexity of CABAC module is:

$$Complexity_B \propto E_l \times M \times S \times F \times N \tag{3-10}$$

where E_l is the expected length of the significant coefficients *E{l(x)}*, *M* the number of search modes, *S* the frame size, *F* the frame rate, and *N* the number of reference frames. Even though the length of each and every significant coefficient varies, the expected length is a good measure of the number of times the CABAC module is made to run per significant coefficient, which in turn determines the complexity of the CABAC module. Note that the complexity is directly proportional to the number of modes (*M*), the frame size (*S*), the frame rate (*F*), and the number of reference frames (*N*).

To change the proportionality into an equal to in the above complexity equation, we consider the following parameters.

If p is the probability of occurrence of a non-zero significant coefficient in a frame, the complexity of CABAC module is:

$$Complexity_B = p \times (C_B + D_B) \times E_l \times M \times S \times F \times N \qquad (3\text{-}11)$$

where C_B and D_B refer to computational and data transfer complexities required per bin. Note that the above equation completely describes the complexity of CABAC.

3.3 CAVLC Complexity Model

The relationship between the value of the significant coefficient and the corresponding complexity to encode using CAVLC is much weaker.

$$Complexity_V = p \times (C_V + D_V) \times M \times S \times F \times N \qquad (3\text{-}12)$$

where C_V and D_V refer to computational and data transfer complexities required per bin. The complexity of CAVLC is independent of $E\{l(x)\}$ because CAVLC is usually encoded with the help of a lookup table, the complexity of which is independent of the value that is being looked up. Note that CAVLC is directly proportional to C_V and D_V. Also, the complexity is proportional to the number of modes (M), the frame size (S), the frame rate (F), and the number of reference frames (N), just like in the case of CABAC.

3.4 Cost-Effectiveness Model

Note that to achieve the highest coding efficiency, H.264/AVC uses a non-normative technique called *Lagrangian rate-distortion optimization* (RDO) technique to decide the coding mode [38] for an MB. In order to choose the best coding mode for an MB, H.264/AVC encoder calculates the rate-distortion (RD) cost (RDcost) of every possible mode and chooses the mode having the minimum value, and this process is repeatedly carried out for all the possible modes for a given MB. The RDCost J is defined as

$$J = D + \lambda R \tag{3-13}$$

where *D, R,* and λ are distortion, bit-rate, and lagrangian parameter, respectively. Also, we know that

$$P \propto -D \tag{3-14}$$

where *P* is PSNR. We can also observe that even though PSNR and bit-rate are not linearly related, RDCost *J* considers the two terms to be linearly related via a lagrangian parameter. Similarly, we can relate PSNR and bit-rate to complexity linearly via another lagrangian parameter η to obtain cost-effectiveness (*CE*) as follows:

$$CE_B = P - \lambda_a R_B - \eta_a (C_B + D_B) pE_l MSFN \tag{3-15}$$

$$CE_V = P - \lambda_b R_V - \eta_b (C_V + D_V) pMSFN \tag{3-16}$$

In this work, influenced by the above two equations, we propose an aggregate *Performance Complexity Index* (PCI) metric and use it to quantify the cost-effectiveness of using CABAC in each video coder setting. The PCI provides a single indicator for comparisons among different coder settings, and is defined as follows:

$$PCI = \alpha \frac{P_n}{P_e} - \beta \frac{R_n}{R_e} - \gamma \frac{C_n}{C_e} - \delta \frac{D_n}{D_e} + \varepsilon \tag{3-17}$$

where the subscripts *n* and *e* refer to the new and existing algorithms, α, β, γ, δ, and ε are coefficients obtained from linear regression plots of a set of examined video sequences.

This metric is explained further in the next section.

3.5 Performance Complexity Index (PCI)

In this work, we propose an aggregate PCI metric and use it to quantify the cost-effectiveness of using CABAC in each video coder setting. The PCI provides a single indicator for comparisons among different coder settings, and is defined as in (3-17).

Note that the PCI is a generalized relative metric that can be used for comparison of the net cost-effectiveness of any two algorithms. Y_PSNR increase and bit-rate reduction, being the measure of quality of output produced using any video coding algorithm, are seen as performance improvement indicators. Computational and data transfer complexities, which directly affect the speed of execution of any algorithm, are complexity increment indicators. The linear combination of the performance improvement indicators and the complexity increment indicators reflects the overall effect due to the changes in any algorithm.

CABAC is considered the new algorithm (*n*) and CAVLC is the existing algorithm (*e*). Though we determine the cost-effective scenarios of using CABAC in SW implementation in this thesis, PCI can also be as effectively used for HW implementations of CABAC and CAVLC. The computational and data transfer complexity not only depend on the algorithms, but also on their implementations. PCI can only be used for comparison of two algorithms in their specific implementations.

3.6 Conclusion

In this chapter, the theoretical complexity model of CABAC and CAVLC were introduced. A performance-complexity relating parameter, Cost Effectiveness *(CE)*, is derived with the help of the complexity models. The Performance-Complexity Index (*PCI*), a metric to compare the performances and complexities of any new algorithm with an existing algorithm and to determine their suitability in any scenario, is defined. This PCI will be used in later chapters to determine the cost-effective scenarios of using CABAC over CAVLC.

CHAPTER 4 PERFORMANCE COMPLEXITY CO-ANALYSIS

4.1 Introduction

The introduction of new entropy coding schemes, CAVLC and CABAC, represent major improvements in terms of coding efficiency. In previous standards, a set of fixed VLCs were used for encoding syntax elements. This was because stationary statistics was assumed. Though, this assumption is not true in practical situations. Context adaptation is introduced only in H.264/AVC.

The use of new entropy coding schemes in H.264/AVC: CABAC and CAVLC is one of the reasons for its higher coding efficiency compared to earlier video standards. Both schemes adapts to the source statistic allowing bit-rates that are closer to the source entropy to be achieved. Amongst the two schemes, CABAC is capable of introducing higher compression.

The CABAC scheme has been studied in the earlier chapters. We have reviewed the CABAC methodology, its capability to compress even up to Shannonȇs limit, and the theoretical model. CAVLC, on the other hand is an entropy coding scheme based on variable length coding (VLC) using Exp-Golomb code and a set of predefined VLC tables. Note that CAVLC has a lower limit of one bit. It has been reported that CABAC reduces the bit-rate up to 16% in [8] and a lower 10% in [9]. In our work, we will benchmark the performance of CABAC against CAVLC using different video sequences having varied properties and different combinations of coding tools.

Also, note that the increase in performance of CABAC is not without a cost. CABAC has a reduced (binary) alphabet set, thus the name Context Adaptive Binary Arithmetic Coding. CAVLC, on the other hand, has a bigger alphabet size. This means that while CABAC can encode only one bin at any instant, CAVLC can encode a whole symbol at a time. This results in higher encoding complexity of CABAC. In this chapter, we will study the benefit of using CABAC by considering the performance-complexity tradeoffs for various sequences and different video coder settings.

4.2 Performance Metric Definitions

The performance metrics used are the bit-rate savings and the peak signal-to-noise ratio of the luminance component (Y_PSNR). The assumption made here is that similar Y_PSNR values yields approximately the same subjective spatial video quality. The chrominance components (U and V) are not used as comparison metrics because the human visual system is less sensitive to chrominance components, which will have small effects on the perceived video quality.

4.3 Complexity Metric Definitions

4.3.1 Computational Complexity

The computational complexity is the number of instructions executed for one complete cycle of operation (billions of instructions per second).

4.3.2 Data Transfer Complexity

The data transfer complexity is given in terms of the number of memory accesses performed for memory read or memory write operations for one complete cycle of operation (billions of memory accesses per second).

4.4 Implementation

Performance analyses and complexity analyses of CABAC are both conducted using JM [39] reference implementation. In our work, the software version used is 14.2. The PIN tool [11], [12] was used to profile the complexity. The tools were run on a Linux platform with a 3 GHz Pentium IV processor and 1 GB of RAM.

PIN provides a set of fast *application program interfaces* (APIs) tools that analyze binary executables. Hardware events such as data cache access on the architecture in study were monitored.

The video encoder and decoder were compiled using GNU GCC compiler with -O2 optimization option. Note that this level of optimization does not include optimization for space-speed tradeoff such as loop unrolling and function in-lining.

4.5 Test Bench Definitions

A set of fourteen QCIF, CIF, and SD (720x576) sequences comprising a wide genre of video contents was used for obtaining exhaustive data which are used for making empirical analysis. The sequences are listed in Table 4-1. These sequences have been categorized based on the amount of motion content in them.

Table 4-1: Test sequences and their motion content classification

Sequence	QCIF	CIF	720x576	Motion Contents
Akiyo	X	X		Low
Mother & Daughter	X	X		Low
Container	X	X		Low
Foreman	X	X		Moderate
Walk	X	X		High
Coastguard	X	X		High
Mobile Calendar (Mobcal)			X	High
Parkrun			X	High

The categorization of the video sequences is carried out by subjective evaluation. The low-motion contents test sequences have been shaded in grey, moderate-motion content test sequences have been shaded in white and high-motion contents test sequences have been shaded in black. These denotations will be used throughout this work.

Sequences *Akiyo, Mother & Daughter* and *Container* are used to represent low-motion sequences while *Coastguard, Foreman* and *Walk* contain varying degrees of camera motion. *Mobile Calendar (Mobcal)* and *Parkrun* are high video motion content sequences with frame size 720x576. Most of these sequences have identical video content in their counterpart video format, which will be used to study the effect of picture size. All sequences comprises of 300 frames.

The configurations shown in Table 4-2 have been used for the analysis.

Table 4-2: Encoder configuration cases

	A	B
Intra 4x4	1	1
Intra 16x16	1	1
Inter modes 16x16/16x8/8x16/8x8	1	4
Sub-partition modes 8x4/4x8/4x4	0	3
Reference frame	1	5
Search Range	8	16
Hadamard	1	1
B frame	1	1
Slice per frame	1	1

Note that config. A represents lower complexity, lower performance configuration and config. B represents higher complexity higher performance configuration. This includes the use of higher number of reference frames, larger search ranges, and smaller block sizes for motion estimation. Both the configurations have Intra 4x4 and Intra 16x16 prediction modes. However, while config. A has only Inter 16x16 prediction mode, config. B has 7 inter prediction modes (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4). Note that allowing smaller block sizes mean fine searching which will ensure better performance. However, on the other hand, fine search also means more number of searches, thus higher complexity. In this work, a GOP is defined as 10 frames, with only the first frame being an Intra (I) frame. Each 300-frame sequence was encoded using a *group of pictures* (GOP) of *IBPBPBPBPB* at a frame rate of 30 fps. The RDO tool is also turned off and on in RDO-off and RDO-on mode, respectively. Also, the analysis is performed in both Variable Bit-Rate (VBR) and Constant Bit-Rate (CBR) modes.

4.6 Performance Analyses

4.6.1 Percentage bit-rate savings by CABAC in VBR Mode

The use of CABAC advocates a reduction in bit-rate needed to encode a sequence at the same video quality. Table 4-3 gives the bit-rate savings by CABAC, benchmarked against CAVLC for some configurations using both RDO-off and RDO-on video encoders in VBR mode for sequences of various frame sizes and motion contents.

Table 4-3: Percentage Bit-rate Savings Due to CABAC in VBR mode

		RDO-off encoder		RDO-on encoder	
		A	B	A	B
QCIF	Akiyo	4.58	4.46	4.54	4.30
	M&D	4.74	4.90	4.45	3.83
	Container	4.90	4.97	4.82	4.11
	Foreman	5.66	5.32	5.03	4.75
	Walk	6.92	6.71	4.95	6.05
	Coastguard	8.38	8.67	7.58	8.03
CIF	Akiyo	6.76	6.92	6.43	6.03
	M&D	7.31	7.82	7.55	6.98
	Container	6.38	6.39	5.63	5.21
	Foreman	8.06	8.15	7.36	6.77
	Walk	8.19	8.22	7.50	7.84
	Coastguard	9.93	9.49	8.95	9.10
SD	Mobcal	10.91	10.50	9.76	9.00
	Parkrun	12.43	12.96	11.73	11.06

Bit-rate savings between 4-8% for QCIF sequences, 5-8% for CIF sequences and 9-13% for SD sequences have been obtained for all configurations. The effect of CABAC on the coding performance is additive as the bit-rate savings obtained for the same sequence is consistence across the configurations. In addition, the bit-rate savings obtained from the RDO-off video encoder is much higher than that from the RDO-on video encoder for the same sequence. This implies that CABAC performs better when RDO tool is off.

Other less significant observations includes the following: bit-rate savings obtained for low-motion content sequences are generally smaller than that of high-motion content sequences. It is also observed that for identical video content, bit-rate saving are higher for larger frame sized sequences.

4.6.2 Bit-rates in various configurations in VBR Mode

For an overview, the joint performance of coding tools in improving the coding efficiency is given here. Table 4-4 summarizes the bit-rates obtained for different combinations of entropy coding schemes with config. A as well as config. B in a RDO-on encoder and RDO-off encoder.

Table 4-4: Bit-rates in various configurations in VBR Mode

		RDO-off				RDO-on			
		A		B		A		B	
		VLC	BAC	VLC	BAC	VLC	BAC	VLC	BAC
QCIF	Akiyo	76.14	72.98	72.58	69.34	78.88	75.30	74.68	71.47
	Mother & Daughter	82.12	79.05	78.51	75.45	85.01	81.23	79.12	76.09
	Container	101.25	96.29	94.73	90.02	105.07	100.01	97.72	93.70
	Foreman	196.81	187.64	176.06	166.70	211.65	201.01	180.14	171.59
	Walk	365.40	347.43	313.11	292.11	403.89	383.89	321.07	301.65
	Coastguard	250.07	231.62	227.86	208.10	284.00	262.48	249.82	229.76
CIF	Akiyo	196.16	184.87	187.40	176.31	205.75	192.51	194.25	182.54
	Mother & Daughter	223.41	209.32	215.02	202.50	232.41	214.87	217.40	202.23
	Container	375.81	351.83	358.33	335.44	408.46	385.45	386.32	366.20
	Foreman	659.19	612.67	558.03	518.13	705.06	653.16	574.78	535.86
	Walk	1093.65	1015.05	924.92	848.92	1195.74	1106.10	954.58	879.74
	Coastguard	1144.91	1042.68	1058.08	957.65	1318.68	1200.66	1145.90	1041.61
SD	mobcal	2968.61	2644.63	2526.30	2260.96	3226.62	2911.59	2717.91	2473.41
	parkrun	9342.46	8181.35	8610.23	7493.95	10732.86	9474.05	9339.57	8306.97

It can be seen from the above table that use of CABAC always achieves a lower bit-rate when compared to its counterpart, CAVLC. The use of RDO-on mode increases the bit-rate, but the PSNR of the sequences encoded using RDO-on mode is much higher. Complex configuration B always performs better than config. A.

4.6.3 Effect of CABAC on Y-PSNR in CBR mode

In this sub-section, the effect of using CABAC in improving the coding performance at constant bit-rate is studied. The performance metric used is the Y-PSNR. Table 4-5 lists the increases in Y-PSNR due to CABAC when using different video coder settings across different constant bit-rates for CIF video sequences. All Y-PSNR improvements are made with respect to the Y-PSNR values obtained for CAVLC.

Table 4-5: Δ Y-PSNR due to CABAC in different constant bit-rates

	256		512		1024	
	RDO-off	RDO-on	RDO-off	RDO-on	RDO-off	RDO-on
Akiyo	0.22	0.22	0.16	0.15	0.18	0.15
M&D	0.24	0.29	0.20	0.21	0.22	0.16
Container	0.38	0.28	0.27	0.29	0.31	0.33
Foreman	0.82	0.99	0.34	0.32	0.34	0.29
Walk	0.80	0.52	0.49	0.54	0.46	0.46
Coastguard	0.26	0.23	0.42	0.46	0.44	0.53

The results show that for lower motion-content sequences, usage of CABAC yields small increase in video quality. On the other hand, the usage of CABAC increases the video quality by up to 0.99 dB for higher motion content sequences in lower bit-rates. This indicates that CABAC is attractive as a tool for improving video quality at constant bit-rate for higher motion content sequences.

4.7 Complexity Analyses

In this section and the next, the complexity analysis of CABAC is conducted using PIN tool. The complexity metrics are computational complexity and data transfer complexity.

Analyses are carried out at the video encoder level. The additional workload required by the entropy coder when CAVLC is replaced by CABAC, is measured for different configurations in both non-RDO and RDO encoders. At the top-level video encoder, the effect of using CABAC on the overall complexity of the video encoder is observed. Besides the encoder, the complexity of the decoder is also being addressed. To achieve an exhaustive analysis of CABAC, a wide genre of video contents has been used as test sequences.

4.7.1 Effect of CABAC on the Computational Complexity in VBR encoder

In this section, the computational complexity of CABAC when used in both non-RDO encoder and RDO encoder are analyzed, and are compared with reference to CAVLC. All computational complexity measurements are expressed as percentage increase from CAVLC to CABAC.

Table 4-6: Percentage increase in computational complexity of the video coder due to CABAC in VBR mode

		RDO-off		RDO-on	
		A	B	A	B
QCIF	Akiyo	0.11	0.02	3.63	1.08
	M&D	0.21	0.04	3.48	1.04
	Container	0.10	0.02	4.48	1.29
	Foreman	0.15	0.05	4.72	1.48
	Walk	0.29	0.07	4.96	1.67
	Coastguard	0.19	0.04	5.03	1.65
CIF	Akiyo	0.00	0.03	3.03	0.93
	M&D	0.00	0.00	2.95	0.91
	Container	0.24	0.00	3.91	1.15
	Foreman	0.23	0.05	3.96	1.26
	Walk	0.18	0.00	3.94	1.37
	Coastguard	0.23	0.00	4.75	1.66
SD	Mobcal	0.10	0.02	4.38	1.39
	Parkrun	0.27	0.07	7.34	2.45

The use of CABAC requires more computation to be performed compared to CAVLC. Table 4-6 shows the percentage increase in computational complexity of the entropy coder when CABAC replaced CAVLC across different configurations.

It can be seen from the data that the computational complexity increase from CAVLC to CABAC in a RDO-off encoder is negligible (0.0-0.3%), irrespective of usage of config.A or config.B. Also, from the data, CABAC increases the computational complexity of the entropy coder by up to 7% for an RDO encoder. Also, usage of a complex setting, ie config.B, in RDO-on mode causes lesser increase in complexity.

From these observations, we can note that CABAC seems to be more effective in RDO-off mode.

4.7.2 Effect of RDO on the Computational Complexity of VBR encoder

Table 4-7 tabulates the increase in computational complexity of the video coder when RDO tool is turned on for different video coder settings.

Table 4-7: Change in Computational Complexity of VBR encoder due to RDO

		VLC	**BAC**
QCIF	Akiyo	22.33	26.64
	M&D	23.38	27.41
	Container	24.02	29.44
	Foreman	26.94	32.74
	Walk	26.22	32.09
	Coastguard	27.78	33.95
CIF	Akiyo	17.56	21.12
	M&D	18.50	22.00
	Container	21.09	25.53
	Foreman	24.12	28.74
	Walk	23.65	28.29
	Coastguard	28.10	33.88
SD	Mobcal	24.00	29.30
	Parkrun	36.18	45.78

The complexity increment factor is given by normalizing the average entropy instruction counts executed by the RDO-on encoder with that of the RDO-off encoder for the same configuration.

The use of RDO as the video encoder control significantly increases the computational complexity of the entropy coder as can be seen in the table. It can be seen that in a video coder using CABAC, the increase due to RDO on is higher than the increase in a video coder using CAVLC. This result was also observed in the previous section. This means that the use of RDO triggered a huge workload for the entropy coder and creates a bottleneck in it. Again, this observation means that CABAC is more effective in RDO-off mode.

4.7.3 Overall Computational Complexity of VBR encoder

Table 4-8 shows the computational complexities of the entropy coder for different combination of entropy coding schemes with different configurations in a RDO-off encoder and RDO-on encoder. All computational complexity measurements are expressed in billions of instructions executed per second. Results have been given with accuracy up two decimal places in order to show the finer differences among the values.

The data provides an overview of the possible variations in computational complexity of the entropy coder due to the collective use of different video coding tools in H.264/AVC for different type of sequences.

It can be seen from the table that the usage of CABAC causes an increase in complexity of the entire video encoder. Note that in RDO-on mode, the entropy coding stage is even used for motion estimation. So, in RDO-on mode, CABAC causes a much larger increase in complexity.

Table 4-8: Computational complexities of VBR encoder in different video coder settings

		RDO-off				RDO-on			
		A		B		A		B	
		VLC	BAC	VLC	BAC	VLC	BAC	VLC	BAC
QCIF	Akiyo	9.45	9.46	44.76	44.77	11.56	11.98	47.19	47.70
	M&D	9.54	9.56	45.18	45.20	11.77	12.18	48.01	48.51
	Container	9.91	9.92	45.72	45.73	12.29	12.84	48.69	49.32
	Foreman	10.16	10.17	46.62	46.65	12.89	13.50	50.20	50.95
	Walk	10.86	10.90	47.97	48.00	13.71	14.39	51.83	52.70
	Coastguard	10.26	10.28	46.53	46.55	13.11	13.77	50.29	51.12
CIF	Akiyo	39.30	39.30	185.25	185.30	46.20	47.60	193.80	195.60
	M&D	40.00	40.00	187.90	187.90	47.40	48.80	197.70	199.50
	Container	42.20	42.30	189.90	189.90	51.10	53.10	201.29	203.60
	Foreman	42.70	42.80	192.10	192.20	53.00	55.10	205.70	208.30
	Walk	44.32	44.40	195.52	195.52	54.80	56.96	209.76	212.64
	Coastguard	42.70	42.80	189.30	189.30	54.70	57.30	205.10	208.50
SD	mobcal	171.74	171.92	787.70	787.88	212.96	222.30	842.33	854.04
	parkrun	168.82	169.28	759.96	760.46	229.90	246.78	839.80	860.39

Note that CABAC causes an increase in complexity of the video coder. Also, RDO-on mode causes a huge increase in the computational complexity of the video coder, particularly for coders with CABAC. Also, some of the obvious observations are that the computational complexity increases with the frame size of the video sequence and also the increase in complexity setting from config.A to config.B.

4.7.4 Effect of CABAC on the data transfer complexity of VBR encoder

In this section, the data transfer complexities of CABAC in both RDO-on encoder and RDO-off encoder are analyzed and compared with CAVLC. All the analyses are carried in a system with 1MB cache. All data complexity measurements are expressed as the average number of memory access per second (in billions of memory accesses per sec.).

The use of CABAC requires the entropy coder to access the memory more frequently as compared to CAVLC. Table 4-9 shows the percentage increase in data transfer complexity of the entropy coder when CABAC replaced CAVLC across the different configurations.

Table 4-9: Percentage increase in data transfer complexity of the VBR encoder due to CABAC

		RDO-off		RDO-on	
		A	B	A	B
QCIF	Akiyo	0.16	0.00	4.52	1.25
	M&D	0.16	0.03	4.30	1.26
	Container	0.15	0.03	5.64	1.62
	Foreman	0.22	0.02	6.02	1.81
	Walk	0.22	0.07	6.30	2.03
	Coastguard	0.29	0.03	6.48	2.05
CIF	Akiyo	0.00	0.04	3.97	1.17
	M&D	0.37	0.00	3.54	1.07
	Container	0.00	0.00	5.11	1.40
	Foreman	0.35	0.00	5.22	1.47
	Walk	0.00	0.00	5.56	1.44
	Coastguard	0.00	0.00	6.21	1.99
SD	Mobcal	0.13	0.03	5.64	1.70
	Parkrun	0.36	0.08	10.18	3.21

It can be seen from the above table that the increase in data transfer complexity from CAVLC to CABAC in RDO-off mode is negligible (0.0-0.4%). Whereas, in RDO-on mode, the increase in complexity from CABAC to CAVLC is quite significant (1 ï 10%). This is because in RDO-on mode the entropy coding tools CABAC and CAVLC are also used in motion estimation. Note that this observation is similar to the one made in previous sections concerning computational complexity. Also, the increase in complexity is higher for config.A.

It can also be noted that for higher motion content and higher frame sizes, the increase in data transfer complexity is higher. For the sequence *parkrun*, which is a SD sequence (720x576 pixels frame) with high motion content, the increase in data transfer complexity in config.A RDO-on mode is more than 10%. In Section 4.6.2, the performance in the form of bit-rate reduction is tabulated. It can be seen in that section that for RDO-off mode, there is a higher decrease in bit-rate. So, again we observe in this section that CABAC performs better in RDO-off mode.

4.7.5 Effect of RDO on the video coder

The use of RDO has a large influence on the data transfer complexity of the entropy code. Table 4-10 gives the data transfer complexity increase of the video coder from a RDO-off mode to RDO-on mode. The complexity increment factor is given by normalizing the number of memory access performed by the RDO encoder with that of the non-RDO encoder for the same configuration.

Table 4-10: Effect of RDO on Data Transfer Complexity of VBR encoder

		VLC	BAC
QCIF	Akiyo	18.61	23.78
	M&D	19.84	24.80
	Container	20.00	26.58
	Foreman	22.75	29.85
	Walk	21.82	29.21
	Coastguard	23.13	30.72
CIF	Akiyo	14.39	18.94
	M&D	15.61	19.26
	Container	17.25	23.24
	Foreman	20.21	26.04
	Walk	20.00	26.67
	Coastguard	23.34	31.01
SD	Mobcal	19.87	26.46
	Parkrun	29.30	41.95

The use of RDO increases the data transfer complexity of the CABAC video coder significantly (14-41%). Note that the increase in data transfer complexity from RDO-off to RDO-on coder for CABAC is much higher than that for CAVLC. Like explained previously, the increase is because entropy coding is used for motion estimation too in RDO-on mode. This again suggests that CABAC is more suitable for RDO-off mode.

Another observation is that the increase in data transfer complexity is higher for video sequences with higher motion content. However, the frame size does not affect the increase in data transfer complexity as much as motion content.

4.7.6 Overall data transfer complexity of the VBR encoder

Table 4-11 shows the data transfer complexities of the entropy coder for different combination of entropy coding schemes with different configurations in a RDO-off encoder and RDO-on encoder. All data transfer complexity measurements are expressed in terms of billions of memory accesses per second. Results have been given with accuracy up two decimal places in order to show the finer differences among the values.

The data provides an overview of the possible variations in data transfer complexity of the entropy coder due to the collective use of different video coding tools in H.264/AVC for different type of sequences.

It can be seen from the table that the usage of CABAC causes an increase in complexity of the entire video encoder. Note that in RDO-on mode, the entropy coding stage is even used for motion estimation. So, in RDO-on mode, CABAC causes a much larger increase in complexity.

Table 4-11: Data transfer complexities of VBR encoder in various video coder settings

		RDO-off				RDO-on			
		A		B		A		B	
		VLC	BAC	VLC	BAC	VLC	BAC	VLC	BAC
QCIF	Akiyo	6.34	6.35	29.88	29.88	7.52	7.86	31.21	31.60
	M&D	6.40	6.41	30.16	30.17	7.67	8.00	31.77	32.17
	Container	6.65	6.66	30.52	30.53	7.98	8.43	32.19	32.71
	Foreman	6.81	6.83	31.16	31.17	8.36	8.86	33.21	33.81
	Walk	7.30	7.31	32.06	32.09	8.89	9.45	34.27	34.97
	Coastguard	6.88	6.90	31.10	31.11	8.47	9.02	33.23	33.91
CIF	Akiyo	26.40	26.40	123.75	123.70	30.20	31.40	128.40	129.90
	M&D	26.90	27.00	125.50	125.50	31.10	32.20	131.10	132.50
	Container	28.40	28.40	126.90	126.90	33.30	35.00	133.33	135.20
	Foreman	28.70	28.80	128.50	128.50	34.50	36.30	136.40	138.40
	Walk	30.00	30.00	131.00	131.00	36.00	38.00	139.00	141.00
	Coastguard	28.70	28.70	126.70	126.70	35.40	37.60	135.70	138.40
SD	mobcal	115.18	115.34	526.31	526.46	138.07	145.85	557.22	566.70
	parkrun	113.00	113.40	508.08	508.49	146.10	160.97	552.18	569.92

Note that CABAC causes an increase in complexity of the video coder. Also, RDO-on mode causes a huge increase in the computational complexity of the video coder, particularly for coders with CABAC. Also, some of the obvious observations are that the computational complexity increases with the frame size and motion content of the video sequence and also the increase in complexity setting from config.A to config.B.

4.7.7 Effect of CABAC on Computational and Data Transfer Complexities of VBR Decoder

The changes in complexity of CABAC with respect to CAVLC for video decoder are tabulated in Table 4-12.

We recognize that RDO tool does not exist for H.264/AVC decoder. However, the RDO mode in the encoder determines the number of encoded bits, which in turn influences the computational and data transfer complexities at the decoder. We analyze the complexities at the decoder as results of both RDO-off and RDO-on for both configurations A and B at the encoder. Table 4-12 presents the percentage change in computational and data transfer complexities of the CABAC-based decoder. Gray, white, and black shadings are used to differentiate among sequences of low, moderate, and high motion-content sequences, respectively.

Table 4-12: Percentage Reduction in VBR Decoderôs Complexity using CABAC

		Computational Complexity (%)				Data Transfer Complexity (%)			
		RDO-off (*)		RDO-on (*)		RDO-off (*)		RDO-on (*)	
		A	B	A	B	A	B	A	B
CIF	Akiyo	1.17	0.58	1.62	0.96	-0.42	-0.88	-0.05	-0.58
	M&D	1.11	0.24	1.52	0.90	-0.39	-1.13	-0.09	-0.61
	Container	3.07	2.48	2.84	2.15	0.29	-0.17	0.02	-0.55
	Foreman	4.16	1.69	4.88	2.57	1.01	0.79	1.41	0.19
	Walk	6.08	2.81	6.98	3.93	1.91	-0.51	2.42	0.21
	Coastguard	8.92	5.67	9.93	7.46	3.96	1.21	4.37	2.42
SD	Mobcal	6.53	4.49	6.48	4.39	2.37	0.84	1.88	0.29
	Parkrun	11.91	9.80	11.12	8.65	3.94	2.05	2.56	0.45

(*) RDO-off / RDO-on indicate the settings at the respective encoders.
(**) a negative value indicates increase in % of complexity.

The above table shows that up to 12% reduction in decoderôs computational complexity, and up to 4% reduction in data transfer complexity can be achieved with CABAC. Note that in some cases, a small increase (approximately 1%) in data transfer complexity is observed. Larger reduction is obtained for high motion-content than for low motion-content sequence. The amount of complexity reduction is also higher for configuration A compared to configuration B. Finally, the RDO tool has little effect on the complexity of the decoder.

In previous sections, we have reconfirmed that at the encoder, replacing CAVLC for CABAC requires a moderate to large increase in both computational and data transfer complexity. However, in this section, we found that at the decoder, because CABAC entropy coder results in less number of bits which leads to lower decoding complexity. As a consequence, replacing CAVLC for CABAC actually reduces both computational and data transfer complexity.

4.7.8 *Effect of CABAC on the Computational Complexity of CBR encoder*

In this section, the computational complexity of CABAC when used in both non-RDO encoder and RDO encoder are analyzed, and are compared with reference to CAVLC for video coder in Constant Bit-Rate mode. All computational complexity measurements are expressed as increase from CAVLC to CABAC in billions of instructions per second. Note that all the sequences are CIF sequences. QCIF sequences have been omitted because they show similar results. Three constant bit-rates are considered- 256, 512 and 1024 Kbps.

Table 4-13: Increase in Computation Complexity from CAVLC to CABAC (10^9/sec.) for CBR encoder

	256		512		1024	
	RDO-off	RDO-on	RDO-off	RDO-on	RDO-off	RDO-on
Akiyo	0.17	2.68	0.17	2.68	0.24	4.05
M&D	0.19	2.56	0.09	2.75	0.02	3.84
Container	0.27	2.59	0.68	2.70	0.63	4.19
Foreman	0.17	2.85	0.70	2.50	0.54	3.68
Walk	0.04	2.33	0.18	1.96	1.33	2.93
Coastguard	1.65	2.28	0.85	2.20	0.19	3.01

It can be seen from the above table that the increase in the number of instructions from CABAC to CAVLC is much higher for RDO-on mode. This is similar to the results observed in VBR mode in the previous sections. This suggests that CABAC is more beneficial in RDO-off mode even in case of CBR.

4.7.9 *Effect of RDO on the computational complexity of video coder in CBR mode*

Table 4-14 tabulates the increase in computational complexity of the video coder when RDO tool is turned on for different video coder settings and for 3 different bit-rates.

Table 4-14: Increase in Computational Complexity of CBR encoder when RDO tool is turned on (10^9/sec.)

	256		512		1024	
	VLC	BAC	VLC	BAC	VLC	BAC
Akiyo	9.90	12.40	13.36	15.87	15.90	19.72
M&D	10.38	12.75	13.79	16.45	16.51	20.33
Container	9.48	11.80	13.10	15.11	16.31	19.87
Foreman	10.84	13.52	13.41	15.21	16.33	19.46
Walk	11.04	13.33	13.43	15.21	15.96	17.56
Coastguard	11.08	11.71	11.05	12.39	14.15	16.96

The complexity increment factor is given by normalizing the average entropy instruction counts executed by the RDO-on encoder with that of the RDO-off encoder for the same configuration.

The use of RDO as the video encoder control significantly increases the computational complexity of the entropy coder as can be seen in the table. It can be seen that in a video coder using CABAC, the increase due to RDO on is higher than the increase in a video coder using CAVLC in all the 3 constant bit-rates considered. This result was also observed in the previous section. This means that the use of RDO triggered a huge workload for the entropy coder and creates a bottleneck in it. Again, this observation means that CABAC is more effective in RDO-off mode.

4.7.10 Overall computational complexity of the video coder in CBR mode

Table 4-15 shows the computational complexities of the video coder for different combination of entropy coding schemes with different configurations in a RDO-off encoder and RDO-on encoder in CBR mode. All computational complexity measurements are expressed in billions of instructions executed per second. Results have been given with accuracy up two decimal places in order to show the finer differences among the values.

The data provides an overview of the possible variations in computational complexity of the video coder due to the collective use of different video coding tools in H.264/AVC for different type of sequences.

It can be seen from the table that the usage of CABAC causes an increase in complexity of the entire video encoder. Note that in RDO-on mode, the entropy coding stage is even used for motion estimation. So, in RDO-on mode, CABAC causes a much larger increase in complexity.

Table 4-15: Computational complexities of CBR encoder in different combinations of entropy coding schemes and configurations for RDO-off and RDO-on encoders in CBR mode (10^9/sec.)

	RDO-off		RDO-on	
256	VLC	BAC	VLC	BAC
Akiyo	185.92	186.09	195.82	198.49
M&D	188.51	188.70	198.89	201.45
Container	192.21	192.48	201.69	204.28
Foreman	195.51	195.68	206.35	209.20
Walk	197.93	197.97	208.97	211.30
Coastguard	185.14	186.79	196.22	198.50
	RDO-off		RDO-on	
512	VLC	BAC	VLC	BAC
Akiyo	185.48	185.65	198.84	201.53
M&D	188.34	188.43	202.13	204.88
Container	188.24	188.92	201.33	204.03
Foreman	192.89	193.59	206.31	208.80
Walk	195.51	195.68	208.94	210.90
Coastguard	185.14	185.99	196.18	198.38
	RDO-off		RDO-on	
1024	VLC	BAC	VLC	BAC
Akiyo	180.74	180.98	196.64	200.69
M&D	186.81	186.83	203.32	207.17
Container	182.94	183.57	199.25	203.44
Foreman	189.26	189.80	205.59	209.27
Walk	190.00	191.33	205.96	208.89
Coastguard	183.44	183.63	197.59	200.59

Note that CABAC causes an increase in complexity of the video coder. Also, RDO-on mode causes an increase in the computational complexity of the video coder, particularly for coders with CABAC.

4.7.11 Effect of CABAC on the Data Transfer Complexity of video coder in CBR

In this section, the data transfer complexity of CABAC when used in both non-RDO encoder and RDO encoder are analyzed, and are compared with reference to CAVLC for video coder in Constant Bit-Rate mode. All data transfer complexity measurements are expressed as increase from CAVLC to CABAC in billions of instructions per second. Note that all the sequences are CIF sequences. QCIF sequences have been omitted because they show similar results. Three constant bit-rates are considered- 256, 512 and 1024 Kbps.

Table 4-16: Increase in Data Transfer Complexity from CAVLC to CABAC (10^9/sec.) in CBR mode

	256		512		1024	
	RDO-off	RDO-on	RDO-off	RDO-on	RDO-off	RDO-on
Akiyo	1.13	2.02	0.99	2.13	0.90	3.25
M&D	0.44	1.92	0.30	2.15	0.02	3.06
Container	0.51	1.98	0.19	2.17	1.76	3.37
Foreman	0.99	1.55	1.41	1.97	0.24	2.96
Walk	0.02	1.63	0.46	1.56	1.56	2.40
Coastguard	0.00	1.46	0.22	1.68	0.18	2.36

It can be seen from the above table that the increase in the number of instructions from CABAC to CAVLC is much higher for RDO-on mode. This is similar to the results observed in previous sections. This suggests that CABAC is more beneficial in RDO-off mode even in case of CBR.

4.7.12 Effect of RDO on the Data Transfer Complexity of video coder in CBR mode

Table 4-17 tabulates the increase in data transfer complexity of the video coder when RDO tool is turned on for different video coder settings and for 3 different bit-rates.

Table 4-17: Increase in Data Complexity of CBR encoder when RDO tool is turned on (10^9/sec.)

	256		512		1024	
	VLC	BAC	VLC	BAC	VLC	BAC
Akiyo	5.57	6.47	7.73	8.87	9.04	11.39
M&D	5.93	7.41	8.04	9.89	9.53	12.57
Container	5.24	6.72	7.47	9.45	9.26	10.88
Foreman	6.57	7.13	7.78	8.33	9.38	12.10
Walk	6.60	8.20	7.79	8.90	9.17	10.01
Coastguard	5.77	7.23	6.34	7.81	8.17	10.35

The complexity increment factor is given by normalizing the average entropy instruction counts executed by the RDO-on encoder with that of the RDO-off encoder for the same configuration.

The use of RDO as the video encoder control significantly increases the data transfer complexity of the entropy coder as can be seen in the table. It can be seen that in a video coder using CABAC, the increase due to RDO on is higher than the increase in a video coder using CAVLC in all the 3 constant bit-rates considered. This result was also observed in the previous section. This means that the use of RDO triggered a huge workload for the entropy coder and creates a bottleneck in it. Again, this observation means that CABAC is more effective in RDO-off mode.

4.7.13 Overall Data Transfer Complexity of the video coder in CBR mode

Table 4-18 shows the data transfer complexities of the video coder for different combination of entropy coding schemes with different configurations in a

RDO-off encoder and RDO-on encoder in CBR mode. All data transfer complexity measurements are expressed in billions of instructions executed per second.

The data provides an overview of the possible variations in data transfer complexity of the video coder due to the collective use of different video coding tools in H.264/AVC for different type of sequences.

It can be seen from the table that the usage of CABAC causes an increase in complexity of the entire video encoder. Note that in RDO-on mode, the entropy coding stage is even used for motion estimation. So, in RDO-on mode, CABAC causes a much larger increase in complexity.

Table 4-18: Data Transfer Complexities of video coder in different combinations of entropy coding schemes and configurations for RDO-off and RDO-on encoders in CBR mode (10^9/sec.)

	RDO-off		RDO-on	
256	VLC	BAC	VLC	BAC
Akiyo	124.22	125.35	129.80	131.82
M&D	125.91	126.35	131.84	133.76
Container	128.37	128.88	133.61	135.60
Foreman	130.65	131.64	137.22	138.77
Walk	132.39	132.41	138.99	140.61
Coastguard	124.75	124.75	130.52	131.98
	RDO-off		RDO-on	
512	VLC	BAC	VLC	BAC
Akiyo	124.01	125.00	131.74	133.87
M&D	125.88	126.18	133.91	136.06
Container	125.78	125.98	133.26	135.43
Foreman	128.97	130.38	136.75	138.72
Walk	130.77	131.23	138.56	140.12
Coastguard	123.70	123.92	130.05	131.73
	RDO-off		RDO-on	
1024	VLC	BAC	VLC	BAC
Akiyo	120.84	121.73	129.88	133.12
M&D	124.87	124.90	134.40	137.46
Container	122.23	123.99	131.49	134.86
Foreman	126.57	126.80	135.95	138.91
Walk	127.09	128.65	136.26	138.66
Coastguard	122.67	122.84	130.83	133.19

Also note that CABAC causes an increase in complexity of the video coder. Also, RDO-on mode causes an increase in the data transfer complexity of the video coder, particularly for coders with CABAC. An obvious observation is that the complexity increases with the motion content of the sequence.

4.7.14 Analysis of CABAC over CAVLC for CBR Decoders

The changes in complexity of CABAC with respect to CAVLC for video decoder are tabulated in Table 4-19.

Table 4-19: Percentage Reduction in CBR Decoderôs Complexity using CABAC

		Computational Complexity (% change)				Data Transfer Complexity (% change)			
		RDO-off (*)		RDO-on (*)		RDO-off (*)		RDO-on (*)	
		A	B	A	B	A	B	A	B
256 Kbps	Akiyo	1.48	0.86	0.96	1.67	-0.51	-2.01	-1.00	-0.20
	M&D	0.89	1.34	-0.29	1.28	-0.97	-1.76	-2.05	-0.43
	Container	1.27	1.30	1.03	1.66	-0.86	-0.92	-1.22	-0.54
	Foreman	-0.63	-2.16	-3.27	-0.97	-1.89	-3.39	-4.55	-2.00
	Walk	1.31	1.85	2.06	1.17	-0.43	0.40	0.56	-0.02
	Coastguard	-4.39	-2.48	0.87	-0.57	-6.21	-4.11	-0.55	-2.09
512 Kbps	Akiyo	5.32	3.72	5.23	4.21	1.44	-0.18	1.31	0.18
	M&D	3.80	2.23	4.35	2.47	0.44	-1.15	1.03	-1.06
	Container	5.32	4.85	4.23	4.07	1.07	0.48	0.10	-0.15
	Foreman	3.12	2.06	2.40	2.34	0.11	-0.86	-0.72	-0.72
	Walk	1.83	0.77	2.15	0.93	-0.81	-1.60	-0.53	-1.61
	Coastguard	2.93	2.00	3.89	2.78	-0.29	-1.08	0.69	-0.34
1024 Kbps	Akiyo	9.58	8.2	10.18	8.26	3.39	1.94	3.82	1.68
	M&D	8.11	5.47	7.86	6.27	2.81	0.17	2.28	0.61
	Container	9.01	8.36	9.51	8.14	2.64	1.80	3.25	1.64
	Foreman	6.37	3.94	6.43	4.65	1.27	-0.95	1.24	-0.55
	Walk	5.32	2.99	4.89	3.37	0.91	-1.14	0.45	-1.00
	Coastguard	6.46	4.38	6.70	5.17	1.32	-0.62	1.42	-0.05

(*) RDO-off / RDO-on indicate the settings at the respective encoders.
(**) a negative value indicates increase in % of complexity.

Gray, white, and black shadings are used to differentiate among sequences of low, moderate, and high motion-content sequences, respectively. Table 4-19 shows that up to 10% reduction in decoderôs computational complexity and up to 4% reduction in data transfer complexity can be achieved with CABAC. Note that in some cases, an increase in computational complexity is observed. However, the data transfer complexity seems to be higher for CABAC, especially for lower bit-rates. Larger reduction in complexities is obtained for higher bit-rates. Contrary to the video coder, in general, CABAC is found to result in lower computational complexity at its decoder.

4.8 Performance-Complexity Co-evaluation

In the previous sections, the performance and complexity of CAVLC and CABAC in different scenarios of both VBR and CBR mode have been analyzed extensively, but separately. This section deals with co-evaluation of performance and complexity together for analysis and to determine possible cost-effective scenarios of CABAC.

4.8.1 Analysis of Variable Bit-Rate Encoder Implementations

In this section, the use of CABAC in *Variable Bit-Rate* (VBR) encoder is analyzed. The benefit of using CABAC is assessed by considering the performance-complexity tradeoffs. The performance metric is the bit-rate reduction under constant video quality, and the complexity metrics comprise of computational complexity (billions of instructions per second), and data transfer complexity (billions of memory accesses per second).

Due to the overwhelming amount of data, we use scatter plots to represent changes in bit-rate and complexities. Figure 4.1 shows the scatter plot of computational complexity versus bit-rate across different coder settings for the CIF and SD sequences. Figure 4.2 shows the corresponding plot of data transfer complexity versus bit-rate. The test sequences are numbered and listed in order of increasing complexity. For each sequence, a *connecting line* is made between

CABAC implementation and its corresponding CAVLC implementation assuming the same configuration. The connecting lines help draw visual interpretation among the changes. The slope of a connecting line is an indicator for complexity increment for a given reduction in bit-rate. The steeper the connecting line, the higher the required complexity for the same amount of bit-rate reduction. Moreover, the length of a connecting line is an indicator for the relative reduction in bit-rate. The longer the connecting line, for a given slope, the larger the amount of bit-rate reduction. We will be making some observations based on these visual rules.

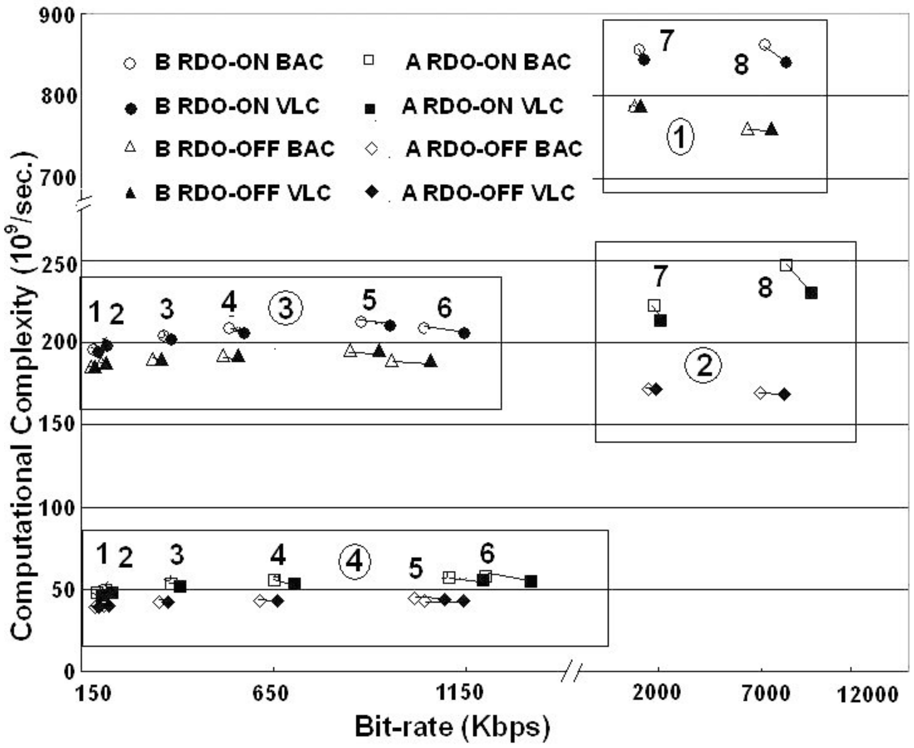


Figure 4.1: Plot of computational complexity (10^9/sec.) versus bit-rate (Kbps) of various video coder settings for CIF and SD sequences in VBR mode. (CIF: 1-Akiyo, 2-M&D, 3-Container, 4-Foreman, 5-Walk, & 6-Coastguard, SD: 7-Mobcal & 8-Parkrun)

In both Figure 4.1 and Figure 4.2, four clusters can be seen: 1) SD sequences using complex configuration B; 2) SD sequences using simple configuration A; 3) CIF sequences using configuration B; 4) CIF sequences using configuration A.

Within each cluster, the upper half is associated with RDO-on mode, and the lower half is with RDO-off mode. Figure 4.1 shows that for RDO-off mode, by replacing CAVLC with CABAC, the connecting lines have gradual slopes indicating that significant bit-rate reductions can be achieved (3 to 13%) for small increases in computational complexity (up to 0.5%) regardless of motion-content and configurations. Figure 4.1 also shows that for sequences with high motion-content, the connecting lines are longer on the linear bit-rate axis (9 to 13%). Note that for sequence 7-Mobcal, even though the connecting line seems to be short, its relative reduction is large (10 to 11%) because of the usage of larger scale on the bit-rate axis. For RDO-on mode, when using CABAC, the slopes of the connecting lines are steeper indicating higher requirement of computational complexity (1 to 7%) for similar reduction in bit-rate (3 to 12%). It can also be seen that for RDO-on mode, as the motion content increases the slope of the connecting line increases. This in turn suggests that CABAC is more beneficial for lower motion content sequences when compared to higher motion content sequences in RDO-on mode.

Figure 4.2 shows similar behaviors. For RDO-off mode, by replacing CAVLC with CABAC, the connecting lines have gradual slopes indicating significant bit-rate reductions can be achieved (3 to 13%) for little increases in data transfer complexity (up to 1%) regardless of motion-content and configurations. Figure 4.2 also shows that for sequences with high motion-content, their connecting lines are longer on the linear bit-rate axis (6 to 12%). Note that for sequence 7-Mobcal, even though the connecting line seems to be short, its relative reduction is large because of the larger scale in the bit-rate axis (9 to 10%). For RDO-on mode, when using CABAC, the slopes of the connecting lines are steeper indicating higher requirement of data transfer complexity (1 to 10%) for similar reduction in bit-rate (3 to 12%). t can also be seen that for RDO-on mode, as the motion content increases the slope of the connecting line increases. This again suggests that CABAC is more beneficial for lower motion content sequences when compared to higher motion content sequences in RDO-on mode.

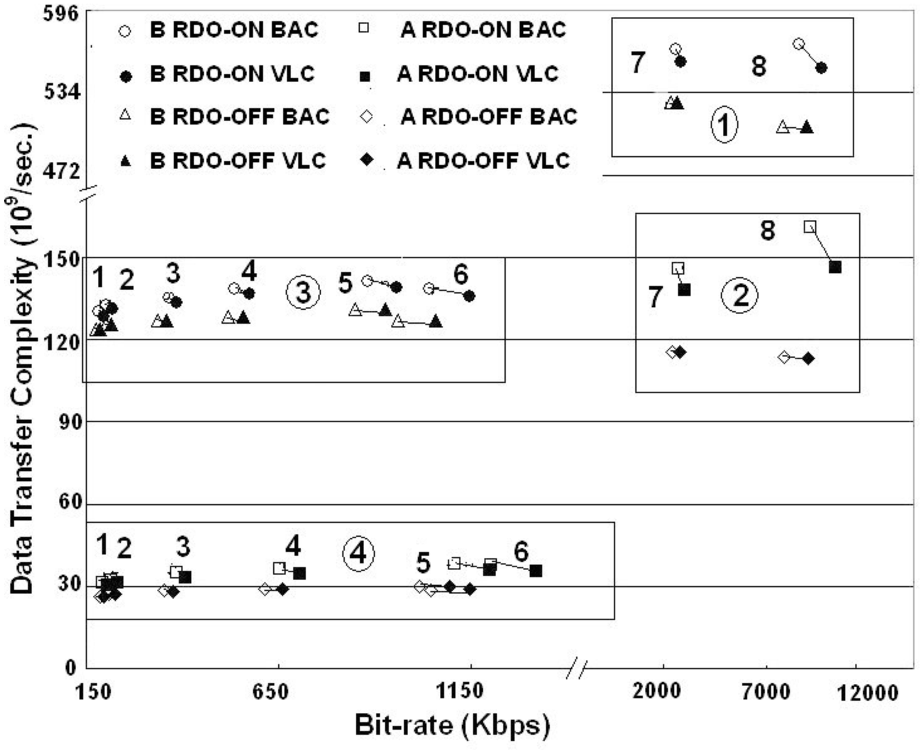


Figure 4.2: Plot of data transfer complexity (10^9/sec.) versus bit-rate (Kbps) of various video coder settings for CIF and SD sequences in VBR mode. (CIF: 1-Akiyo, 2-M&D, 3-Container, 4-Foreman, 5-Walk, & 6-Coastguard, SD: 7-Mobcal & 8-Parkrun)

Empirical data (plots not shown) were also obtained for the QCIF-sequences, and similar behaviors are observed, and the relative scales of bit-rate reduction are found to be less than those of the CIF and SD sequences.

Our analyses differ from [10] in one point. In [10], data transfer complexity (i.e. access frequency) was claimed to increase from 25-30% compared to CAVLC, whereas in our report, data transfer complexity is increased by only up to 10%. The main difference can be that the reference software used in [10] was JM version 2.1, whereas JM 14.2 was used in our study.

In this section, we have studied and identified situations where the reduction in bit-rate is perceived to be more than the increase in complexities for VBR encoders. We use the term *beneficial* hereafter to indicate a situation where the amount of bit-rate reduction is perceived to be more than the amount of complexity incurred. From the analyses, we conclude that it is beneficial to use CABAC in RDO-off mode, regardless of motion-content and configurations; and that high motion-content sequences result in larger bit-rate reduction. Also, CABAC is more beneficial for lower motion content sequences than higher motion content sequences in RDO-on mode. In the next chapter, we use a PCI to quantify if the discussed beneficial situations are in fact *cost-effective*.

4.8.2 Analysis of CABAC-Based Constant Bit-Rate Implementations

Constant bit-rate (CBR) codec has been implemented in the *asynchronous transfer mode* (ATM) networks, and is supported by the H.264/AVC standard. We analyze the benefit of using CABAC in CBR encoder in this section. The performance metric is the increase in Y_PSNR under CBR, and the complexity metrics comprise of computational complexity and data transfer complexity.

Figure 4.3 and Figure 4.4 show the scatter plots of Y_PSNR versus computational complexity and Y_PSNR versus data transfer complexity respectively, across constant bit-rates of 256, 512, and 1024 Kbps for CIF sequences. For each sequence, a link is made between CABAC-based and CAVLC-based coder assuming the same configurations. A horizontal link indicates Y_PSNR can be increased for negligible increase in complexity, while an upward diagonal link indicates Y_PSNR is increased for some increase in complexity.

The diamond, square and shadowed square points represent data associated with RDO-off mode, while the circle, triangle and shadowed triangle points RDO-on mode. Figure 4.3 shows that for RDO-off mode, by replacing CAVLC with CABAC, the long connecting lines having gradual slopes indicate that significant Y_PSNR improvements (3-12%) can be achieved for little increases in computational complexity regardless of motion-content and configurations.

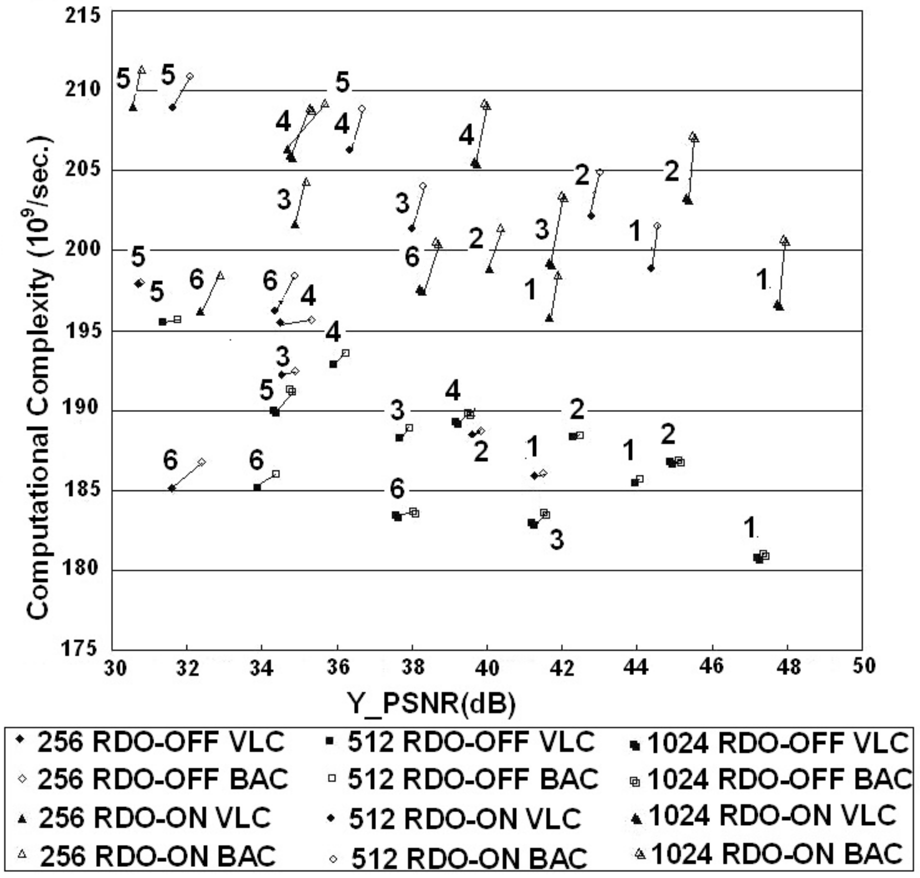


Figure 4.3: Plot of computational complexity (10^9/sec.) versus bit-rate (Kbps) of various video coder settings for CIF and SD sequences in CBR mode. (CIF: 1-Akiyo, 2-M&D, 3-Container, 4-Foreman, 5-Walk, & 6-Coastguard)

Also, it can be seen from Figure 4.3 that for RDO-on mode, by replacing CAVLC with CABAC, the long connecting lines having much steeper slopes suggest higher requirement of computational complexity (up to 6%) for the same increase in Y_PSNR (up to 12%).

Similarly, Figure 4.4 shows that by replacing CAVLC with CABAC, the short connecting lines having very steep slopes indicate that very little Y_PSNR increase can be achieved with huge increase in data transfer complexity (up to 10%) for all modes except high motion-content sequences in RDO-off mode. For high motion-content sequences in RDO-off mode, by replacing CAVLC with CABAC, significant Y_PSNR improvement can be achieved for little increases in data transfer complexity.

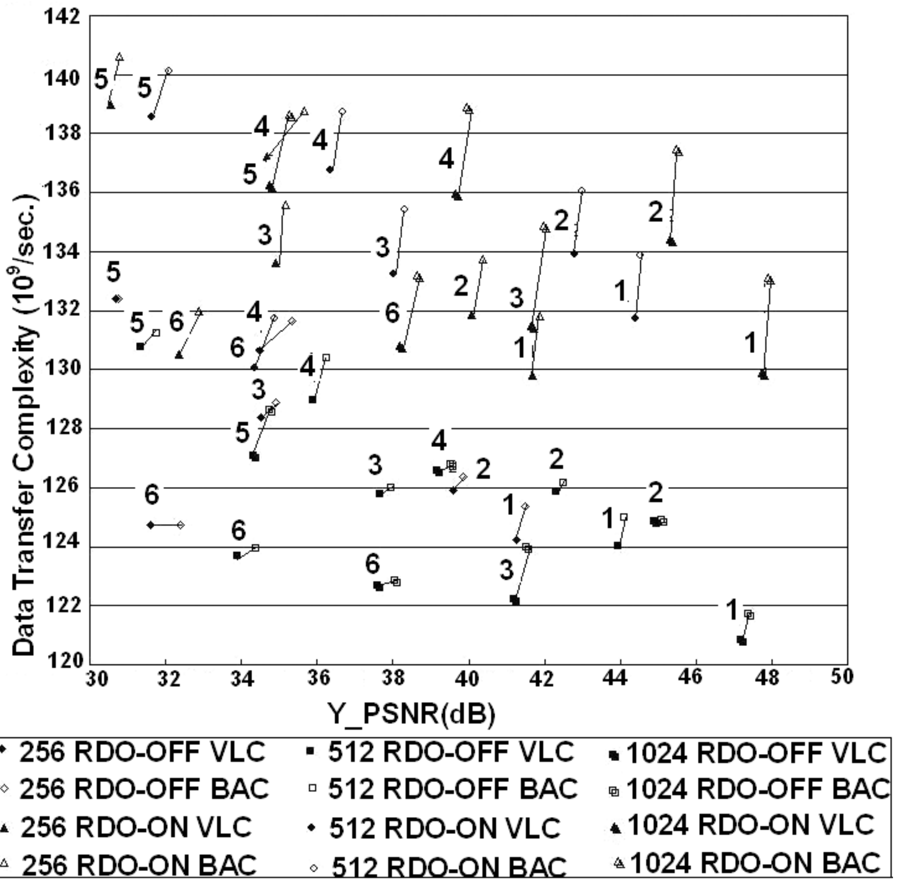


Figure 4.4: Plot of data transfer complexity (10^9/sec.) versus bit-rate (Kbps) of various video coder settings for CIF and SD sequence in CBR mode. (CIF: 1-Akiyo, 2-M&D, 3-Container, 4-Foreman, 5-Walk, & 6-Coatguard)

From the analyses, we conclude that it is beneficial to use CABAC in RDO-off mode for high motion-content sequences, regardless of configurations.

In this section, we have studied and identified situations where the increase in Y_PSNR is perceived to be more than the increase in complexities for CBR encoders, and found that CABAC is beneficial for encoding high motion-content sequences under RDO-off mode.

4.9 Conclusion

In this chapter, the performance and complexity of the video coder under various video coder settings and with video sequences of different motion contents and frame sizes have been analyzed, both individually and then together.

In VBR mode, the bit-rate savings due to CABAC is higher in RDO-off mode. Also, the increase in computational complexity due to CABAC in VBR mode when RDO tool is turned off mode is negligible, as opposed to the increase in RDO-on mode, which can be up to 8%.

The increase in data transfer complexity due to CABAC in VBR mode when RDO tool is turned on is up to 10%, as opposed to the negligible increase when RDO tool is off. The low complexity of CABAC from the perspective of the non-RDO encoder suggests the feasibility of implementing CABAC in software without any hardware assistance.

In video decoder, the effect of CABAC is just the opposite. Up to 12% reduction in decoderŵs computational complexity and up to 4% reduction in data transfer complexity can be achieved with CABAC. This is because CABAC results in less number of encoded bits, which in turn strongly influences the performance of the decoder.

In CBR mode, similar behavioral patterns can be seen. The increase in computational complexity is up to four billion instructions per second for CABAC when RDO tool is on. When RDO is off, the increase in complexity is negligible. The same behavior is seen for data transfer complexities. This means CABAC is less complex for both VBR and CBR mode and irrespective of other video coder settings as long as RDO tool is off.

As with VBR, up to 10% reduction in decoderôs computational complexity and up to 4% reduction in data transfer complexity can be achieved with CABAC in CBR mode. Contrary to the video coder, in general, CABAC is found to result in lower computational complexity at its decoder.

In the performance-complexity co-evaluation, certain conclusions were drawn based on visual interpretation of data. In VBR mode, *connecting lines* are made between CABAC implementation and its corresponding CAVLC implementation assuming the same configuration in the plots. The slope of a connecting line is an indicator for complexity increment for a given reduction in bit-rate. The steeper the connecting line, the higher the required complexity for the same amount of bit-rate reduction. In the computational complexity versus bit-rate plot, for RDO-off mode, by replacing CAVLC with CABAC, the connecting lines have gradual slopes indicating that significant bit-rate reductions can be achieved (3 to 13%) for small increases in computational complexity (up to 0.5%) regardless of motion-content and configurations. In the data transfer complexity versus bit-rate plot, for RDO-off mode, by replacing CAVLC with CABAC, the connecting lines have gradual slopes indicating significant bit-rate reductions can be achieved (3 to 13%) for little increases in data transfer complexity (up to 1%) regardless of motion-content and configurations. From the analyses, we conclude that it is beneficial to use CABAC in RDO-off mode, regardless of motion-content and configurations in VBR mode.

In the CBR mode, when considering computational complexity versus bit-rate plot, for RDO-off mode, by replacing CAVLC with CABAC, the long connecting lines having gradual slopes indicate that significant Y_PSNR improvements (3-12%) can be achieved for little increases in computational complexity regardless of motion-content and configurations. Similarly, the data transfer complexity versus bit-rate plot shows that by replacing CAVLC with CABAC, the short connecting lines having very steep slopes indicate that very little Y_PSNR increase can be achieved with huge increase in data transfer complexity (up to 10%) for all modes.

All these analyses lead to the conclusion that CABAC is more beneficial in RDO-off mode. However, the conclusions drawn in this chapter are based on empirical data. In the next chapter, we use the proposed PCI methodology to validate the conclusion obtained in this chapter.

CHAPTER 5 QUANTIFICATION OF COST-EFFECTIVENESS OF CABAC-BASED CODERS

In the previous chapter, we used empirical analysis to determine the cost-effective scenarios of using CABAC. Initially, we had analyzed performance and complexity separately. Either performance or complexity alone cannot be used for judging any algorithm. They have to be considered together. We also had a performance-complexity co-evaluation in the previous chapter. Though this methodology is comparatively better than individual analyses, it doesnøt consider all the components of complexity along with performance together.

In Chapter 3, we used a theoretical model to develop a performance-complexity metric (PCI). The PCI metric defined in that chapter relates performance, computational complexity, and data transfer complexity together to assess the overall effectiveness of CABAC in any scenario.

In this chapter, we first use a theoretical analysis to determine the cost-effective scenarios of CABAC. Then the method to be followed to obtain PCI values is described. PCI is used later in this chapter to determine the cost-effective scenarios of using CABAC. Note that the methodology can be applied for determining the cost-effective scenarios of any algorithm. PCI equations are derived and PCI values are summarized for various sequences in all the video coder configurations. Based on the obtained PCI values, certain conclusions will be drawn.

5.1 Theoretical Analysis

First, we make some theoretical observations to determine cost-effective scenarios of CABAC. Observation one is that arithmetic coding can efficiently encode fractional codewords. This has been elaborately explained in Section 2.2. For an information source that is capable of generating two symbols with probabilities p and $(1-p)$, the number of bits per symbols is

$$l = p\log_2(p) + (1-p)\log_2(1-p) \qquad (5\text{-}1)$$

Note that the above expression is also the Shannonô s limit on the minimum encoded symbol length possible. This proves that arithmetic coding can achieve the Shannonô s limit and thus explains why it is very efficient.

Also, consider a case where p is very much greater than 0.5, lets say 0.95. In this case, arithmetic coding can actually encode each symbol with number of bits per symbols approx. 0.2 (by substituting the probability values in the above equation), which means it can encode using fractional length. However, the variable length codes have a lower limit of 1 bit/symbol.

In RDO-off mode, there are lots of significant coefficients having very small values, whose probabilities exceed 0.5 many times for all sequences. For example, the probability of significant coefficients having absolute value 1 for Foreman sequence is 0.58 in RDO-on mode as compared to 0.77 in RDO-off mode. So, we can conclude that the performance of CABAC in RDO-off mode is very high.

Observation two is that the values of significant coefficients in RDO-on mode are much larger than the corresponding values in RDO-off mode, which in turn makes the expected length of the significant coefficients, E_l, much larger for RDO-on mode. Due to the nature between the complexity and E_l of significant coefficients (from Eqn. (3-11)), the complexity of the CABAC module in RDO-on mode is very much larger than the complexity of the module in RDO-off mode. However, the increase in complexity of CAVLC engine from the RDO-off to RDO-on mode is not very significant (from Eq. (3-12)).

From the above explanations, it can be seen that in RDO-off mode CABAC gives significantly better performance than CAVLC without incurring much increase in complexity. This shows that CABAC is more cost-effective in RDO-off mode.

Configuration A, being a simple configuration, results in lower performance when compared to configuration B. This in turn means that the final bit-rate will be higher for configuration A because of its lower performance. Higher bit-rate in configuration A means a much higher expected length of significant coefficients than configuration B. Because of the direct dependence of complexity of CABAC with the expected length, the complexity of CABAC in RDO-on using configuration A is much higher. Also, CABAC performs better if the expected length is shorter.

5.2 PCI Methodology

5.2.1 Method for Computing PCI in VBR Implementations

The following algorithm is proposed:

Step1: For each and every configuration, and for each and every test sequence, obtain the bit rates (Kbps) for both new algorithm (R_n) and existing algorithm (R_e). Subsequently, obtain the ratio of bit rate due to the new algorithm over that due to the existing algorithm (R_n/R_e).

Step2: For each and every configuration, and for each and every test sequence, obtain the computational complexity measures (MIPS) for both new algorithm (C_n) and existing algorithm (C_e). Subsequently, obtain the ratio of computational complexity due to the new algorithm over that due to the existing algorithm (C_n/C_e).

Step3: Draw a new plot. On the bit-rate ratio vs. computational complexity ratio plot, for each and every configuration, and for each and every test sequence, plot a point $\{R_n/R_e, C_n/C_e\}$.

Step4: On the plot obtained, draw a linear regression line for all the points. Derive the corresponding linear regression equation.

Step5: Repeat Steps 2 to 4 for each and every remaining complexity measure such as data transfer complexity, power, area, etc.

Step6: Assign a weight to each and every complexity measure (computational complexity, data transfer complexity, power, area, etc). The weights are influenced by the underlying architectures, design rules, and technologies involved. Linearly add the weighted linear regression equations.

Step7: Simplify the superimposed equation obtained in Step6. PCI_VBR is expressed in the final equation.

5.2.2 Method for Computing PCI in CBR Implementations

The following algorithm is proposed:

Step1: For each and every configuration, and for each and every test sequence, obtain the Y_PSNR (dB) for both new algorithm (P_n) and existing algorithm (P_e). Subsequently, obtain the ratio of bit rate due to the new algorithm over that due to the existing algorithm (P_n/P_e).

Step2: For each and every configuration, and for each and every test sequence, obtain the computational complexity measures (MIPS) for both new algorithm (C_n) and existing algorithm (C_e). Subsequently, obtain the ratios of computational complexity due to the new algorithm over that due to the existing algorithm (C_n/C_e).

Step3: Draw a new plot. On the bit-rate ratio vs. computational complexity plot, for each and every configuration, and for each and every test sequence, plot a point $\{P_n/P_e, C_n/C_e\}$.

Step4: On the plot obtained, draw a linear regression line for all the points. Derive the corresponding linear regression equation.

Step5: Repeat Steps 2-4 for each and every remaining complexity measure such as data transfer complexity, power, area, etc.

Step6: Assign a weight to each and every complexity measure (computational complexity, data transfer complexity, power, area, etc). The weights are influenced by the underlying architectures, design rules, and technologies involved. Linearly add the weighted linear regression equations.

Step7: Simplify the superimpose equation obtained in Step6. PCI_CBR is expressed in the final equation.

PCI_VBR and PCI_CBR are metrics that give a measure of the cost-effectiveness of an algorithm over its contending algorithm. Similar to the PCI equation derived in Section II, the PCI values that are obtained using the above 2 methodologies are dependant on both the performance and complexity parameters, thus making them an effective metric to assess the cost-effectiveness of any new algorithm.

In the next section, we will be using the above described methodologies to identify cost-effective scenarios of CABAC over CAVLC. We will also compare the results obtained using PCI values and the empirical results obtained in the previous chapter.

5.3 PCI Methodology for Analysis of CABAC and CAVLC

We now apply the above explained methodology to determine the cost-effective scenarios of CABAC. Figure 5.1 shows the plot of computational complexity ratio versus bit-rate ratio for various CIF sequences in all the video coder settings for VBR implementation. A linear regression line is drawn to obtain the relationship between computational complexity ratio and bit-rate ratio for the CIF sequences. In the figure, the cluster of points around computational complexity ratio 1.00 corresponds to RDO-off mode. The points near the linear regression line correspond to RDO-on using configuration B. As explained in Chapter 3, we take the relationship between the computational complexity ratio and bit-rate ratio to be linear, thus influencing our choice of using a linear regression plot to obtain the relationship. All the data points which lie below the regression line are more cost-effective using CABAC, as the computational complexity ratio is smaller in this region. Consequently, all the data points above the regression line indicate configurations which favor CAVLC mode.

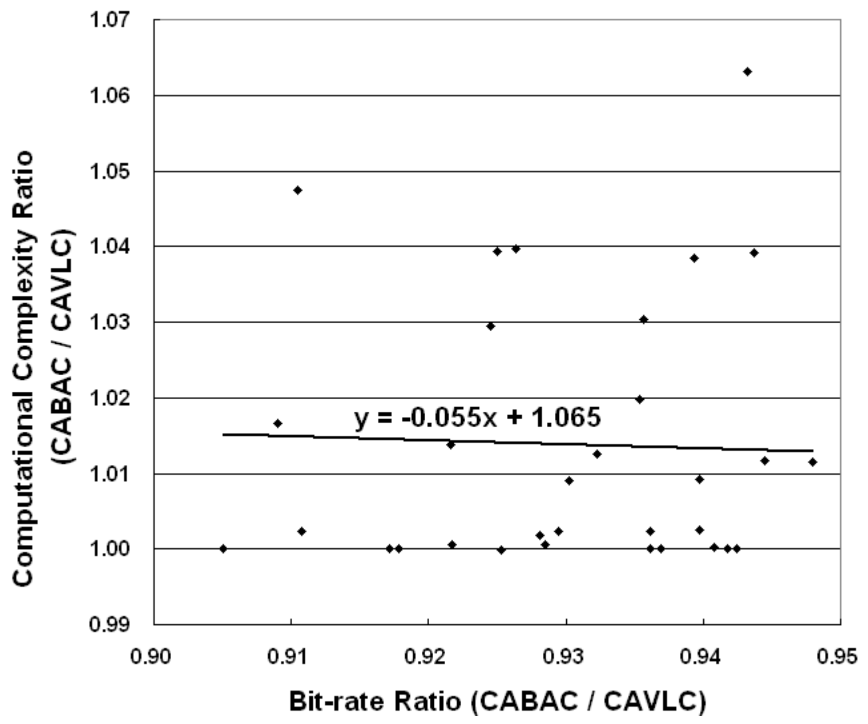


Figure 5.1: Plot of computational complexity ratio versus bit-rate ratio of various video coder settings in VBR mode for CIF sequences

From the equation of the regression line, we can conclude that CABAC is more cost-effective if the following equation is satisfied.

$$(-0.055)\frac{R_B}{R_V} - \frac{C_B}{C_{V``}} + 1.065 > 0 \qquad (5\text{-}2)$$

In the above equation *R* denotes bit-rate, *C* the computational complexity, and the subscripts *B* and *V* refer to CABAC and CAVLC respectively. Similarly, an equation can be obtained using the plot of data transfer complexity ratio versus bit-rate ratio of CIF sequences in various video coder settings for VBR implementation as shown in Figure 5.2. Again, the region above the regression line indicates configurations which favor CAVLC, and region below regression line indicates configurations which are more cost-effective using CABAC.

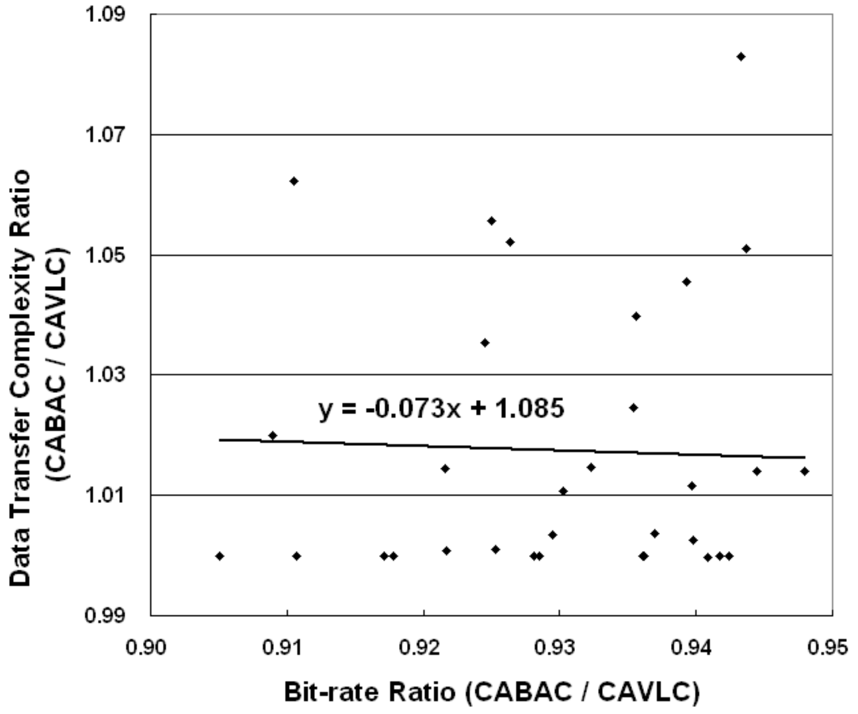


Figure 5.2: Plot of data transfer complexity ratio versus bit-rate ratio of various video coder settings in VBR mode for CIF sequences

Using the equation of the regression line, the equation to be satisfied for CABAC to be more cost-effective is obtained as follows.

$$(-0.073)\frac{R_B}{R_V} - \frac{D_B}{D_V} + 1.085 > 0 \tag{5-3}$$

where D refers to data transfer complexity. The Y_PSNR of CABAC and CAVLC are the same in these analyses.

$$\frac{P_B}{P_V} = 1 \tag{5-4}$$

where P refers to Y_PSNR. It has been reported that every year the processor speed increases by 60%, while the memory speed increases by a modest 7% [40]. Therefore, the weight associated with the linear regression line (5-2) is (100/60), while the weight associated with the linear regression line (5-3) is (100/7). Recall from (3-17) that PCI is a function of 4 ratios (Y_PSNR, Bit-rate, Computational

Complexity, and Data Transfer) and a constant. We take the weighted sum of the above three relations (5-2), (5-3), and (5-4) with weights (100/60) for (5-2) and (100/7) for (5-3) to obtain PCI in that form.

$$PCI = \frac{P_B}{P_V} - 1.135\frac{R_B}{R_V} - 1.670\frac{C_B}{C_V} - 14.285\frac{D_B}{D_V} + 17.275 \qquad (5\text{-}5)$$

It is noted that the PCI is the weighted sum of the LHS of (5-2), (5-3), and (5-4). In order for CABAC to be more cost-effective than CAVLC, PCI must be greater than the sum of RHS of the 3 equations, which is 1.

Similarly, PCI equations were obtained for QCIF, and SD sequences in VBR implementations.

Figure 5.3 shows the plot of computational complexity ratio versus PSNR ratio for CIF sequences in CBR mode at a constant bit-rate of 512 Kbps.

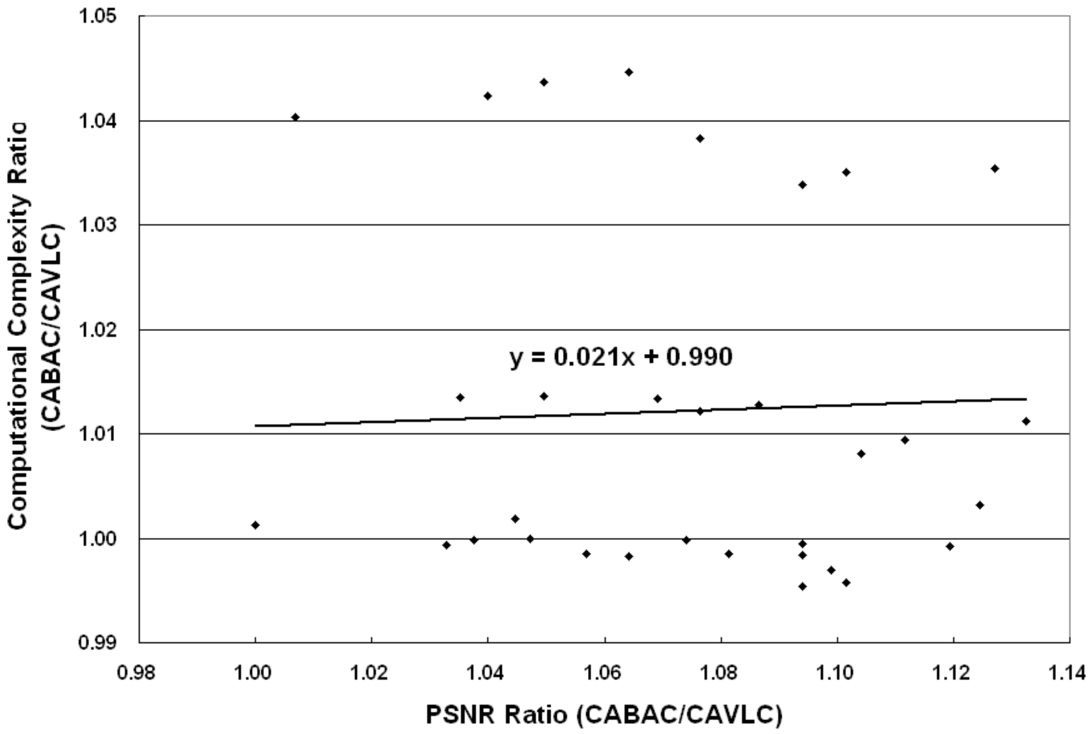


Figure 5.3: Plot of computational complexity ratio versus PSNR ratio of various video coder settings in CBR mode for CIF sequences at 512 Kbps

Note that in the above plot, the X-axis is PSNR as in CBR mode, the performance parameter in PSNR. Similarly Figure 5.4 shows the plot of data transfer complexity ratio versus PSNR ratio for CIF sequences in CBR mode at a constant bit-rate of 512 Kbps.

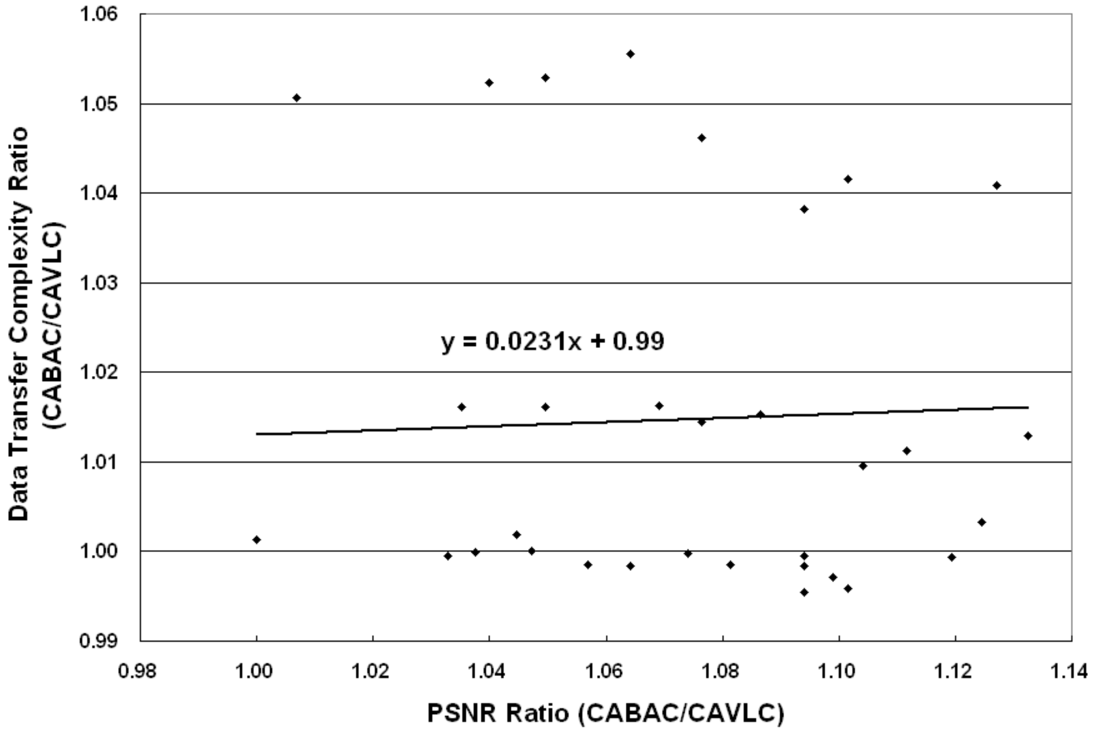


Figure 5.4: Plot of data transfer complexity ratio versus PSNR ratio of various video coder settings in CBR mode for CIF sequences at 512 Kbps

PCI equations were obtained for CBR modes of 256, 512, and 1024 Kbps and the PCI values were tabulated. The methodology used was the same as explained earlier for VBR mode CIF sequences, but the equations used were different, as shown in Figures 5.3 and 5.4.

Regression linesô goodness of fit [41] can be expressed in terms of their r^2 values. The r^2 value of a regression line is defined as

$$r^2 = 1 - \frac{SS_{reg}}{SS_{tot}} \quad (5\text{-}6)$$

where SS_{reg} is the sum of squares of distances of the data points from the regression line, and SS_{tot} is the sum of the squares of distances of the data points from the null hypothesis line. Note that the null hypothesis line is the horizontal line passing through the mean of all the values on the vertical axis. The value r^2 is a fraction between 0.0 and 1.0, and has no unit. An r^2 value of 0.0 implies that there is no linear relationship between bit-rate and complexity. When r^2 equals 1.0, all points lie exactly on a straight line, implying that if the bit-rate is known, then the corresponding complexity can also be derived. The r^2 values of the regression lines of QCIF, CIF, and SD are 0.88, 0.89 and 0.83 respectively. This means that the linear relationships established in the plots are valid. Note that we get 0.83 for SD sequences because of the fewer number of points in the plot, which in turn is due to lesser number of SD sequences (only 2- Mobile Calendar and Parkrun) as opposed to 6 QCIF or CIF sequences.

5.4 PCI Values and Inferences

All the PCI values, calculated using the obtained equations, are tabulated in Tables 5-1 and 5-2 for VBR and CBR encoder implementations respectively. Considering the VBR implementation, as can be seen, the PCI values are greater than 1 in RDO-off mode, implying that CABAC is more cost-effective in RDO-off mode. This is independent of any video coder settings. This confirms our earlier theoretical analysis and also the exhaustive empirical analyses in the previous chapter. Even in RDO-on mode, some lower motion-content sequences in configuration B for VBR mode seem to have a PCI greater than 1, suggesting that CABAC is more cost-effective in these configurations.

For VBR implementations in RDO-on mode, the PCIs corresponding to lower motion-content are higher than those of higher motion-content sequences suggesting that CABAC is more cost-effective for lower motion content sequences than for higher motion content sequences. We made this similar observation in the previous chapter.

We have conducted the empirical analyses and analyses using PCI in VBR implementations, and have shown that the latter confirms the results of the former. This validates the effectiveness of the PCI approach to identify the more cost-effective of any two contending algorithms in any scenario.

In CBR implementations, again, the PCI values are greater than 1 for all the video sequences in RDO-off mode, irrespective of the frame sizes, motion-contents or video coder settings. In RDO-on mode, higher motion-content sequences in configuration B seem to have a PCI greater than 1, indicating CABACǿs cost-effectiveness in these scenarios.

In RDO-off mode, the SD sequences seem to have a higher PCI than CIF and QCIF sequences in VBR implementation, suggesting that CABAC is more cost-effective for sequences having larger frame sizes.

Table 5-1: Comparison of PCI for VBR Encoders in Different Video Coder Settings

Configuration		RDO-off		RDO-on	
		A	B	A	B
QCIF	Akiyo	1.28	1.30	0.60	1.11
	M&D	1.27	1.29	0.63	1.10
	Container	1.29	1.31	0.43	1.05
	Foreman	1.27	1.31	0.37	1.02
	Walk	1.27	1.32	0.33	1.00
	Coastguard	1.29	1.35	0.33	1.03
CIF	Akiyo	1.25	1.26	0.64	1.07
	M&D	1.21	1.25	0.72	1.10
	Container	1.26	1.26	0.46	1.03
	Foreman	1.21	1.27	0.46	1.03
	Walk	1.27	1.28	0.41	1.05
	Coastguard	1.29	1.30	0.32	0.98
SD	Mobcal	1.39	1.40	0.52	1.12
	Parkrun	1.38	1.43	0.15	0.92

Note that these results are in line with the empirical results obtained in the earlier chapter. It was noted that bit-rate savings are more for larger framed sequences. It can also be seen from the above Table 5-1 that PCI values of SD sequences are larger in RDO-off mode.

Note that the proposed PCI methodology has been applied to entropy coding techniques CABAC and CAVLC in this chapter, and the results have been found to be the same as theoretical analysis and the empirical analysis.

Table 5-2: Comparison of PCI for CBR Encoders in Different Video Coder Settings and Bit-Rates for CIF sequences

Configuration		RDO-off		RDO-on	
		A	B	A	B
256 Kbps	Akiyo	1.17	1.20	0.49	0.97
	M&D	1.23	1.20	0.48	0.99
	Container	1.19	1.29	0.49	0.99
	Foreman	1.23	1.25	0.50	1.00
	Walk	1.18	1.26	0.41	0.99
	Coastguard	1.24	1.28	0.63	1.08
512 Kbps	Akiyo	1.23	1.22	0.40	0.96
	M&D	1.19	1.22	0.40	0.97
	Container	1.25	1.25	0.36	0.98
	Foreman	1.23	1.26	0.51	1.01
	Walk	1.21	1.27	0.61	1.05
	Coastguard	1.31	1.31	0.59	1.07
1024 Kbps	Akiyo	1.29	1.32	0.05	0.91
	M&D	1.27	1.31	0.19	0.95
	Container	1.36	1.37	0.00	0.94
	Foreman	1.33	1.36	0.30	0.99
	Walk	1.32	1.38	0.52	1.08
	Coastguard	1.37	1.41	0.43	1.10

5.5 Conclusion

In this chapter, the PCI methodology has been introduced. The methodology has been used to compare the 2 entropy coding techniques of video coding standard H.264/AVC to assess the cost-effective scenarios of the new algorithm CABAC. The results obtained using PCI is the same as the theoretical analysis and also the empirical analysis in previous chapter.

CABAC is more cost-effective in RDO-off mode. The theoretical analysis shows that CABAC offers a better performance than CAVLC in RDO-off mode because of its capability to encode small values. Also, the increase in complexity due to CABAC in RDO-off mode is much lesser when compared to RDO-on mode. This is because of the usage of the entropy coding stage for motion estimation in RDO-on mode.

For VBR implementations in RDO-on mode, CABAC is more cost-effective for lower motion content sequences than for higher motion content sequences.

The PCI values also lead to the same conclusion. The PCI values are greater than 1 for RDO-off mode irrespective of the video sequence or the video coder setting in both VBR and CBR implementations.

CHAPTER 6 CONCLUSIONS

6.1 Introduction

In this thesis work, a method using an aggregate indicator, the Performance Complexity Index (PCI), for evaluating the cost-effectiveness of trading complexity for performance for any newly proposed algorithms when compared to the existing algorithm in computing fields is introduced. This Performance Complexity Index is used to identify the cost-effective scenarios of CABAC over CAVLC. Comprehensive analyses on the performance and complexity of CABAC have been conducted, and the scenarios identified using Performance-Complexity analysis method has been verified. A summary of the findings that have not yet been reported or not highlighted in other works are given below.

6.2 Findings and Contributions

- The methodology using PCI metric introduced in this work can be used for evaluating the cost-effectiveness of any new algorithm in comparison to the existing algorithm. PCI metric was used to determine the cost-effective scenarios of using CABAC over CAVLC, which are the entropy coding algorithms used in H.264/AVC standard. The empirical analysis and the theoretical analysis led to certain conclusions about the cost-effective scenarios of CABAC. The same conclusions were later obtained using the proposed PCI methodology.

- The cost-effectiveness of CABAC largely depends on the encoder control used by the video encoder. The performance of CABAC is much higher in RDO-off mode when compared to RDO-on mode. This is because of the presence of a lot of small values in RDO-off mode, and the capability of arithmetic coding to encode efficiently values that occur very frequently. Also, the computational and data transfer complexity of CABAC is much lesser in RDO-off mode when compared to RDO-on mode. In RDO-on mode, the entropy coding stage is used during motion estimation. This increases the complexity of CABAC in RDO-on mode. So, it can be concluded that CABAC is more cost-effective in RDO-off mode.

- CABAC is more cost-effective for lower motion-content sequences than for higher motion-content sequences in RDO-on mode VBR implementation. We obtained this result from both the proposed performance-complexity analysis method and exhaustive empirical analysis.

- As opposed to the analytical results obtained at the encoder, CABAC actually reduces both the computational and data transfer complexity of the decoder. CABAC entropy coder, because of its better performance, results in lesser number of encoded bits at the output of the video encoder. This means lesser number of bits for the decoder to process. This decreases the complexity of the video decoder. The use of CABAC is always beneficial to the decoder as it results in lower computational and data transfer complexities of the decoder. (This was not reported in any work although in [10], similar result has been obtained for one of their test sequences). This leads to lower processing power, which is attractive for power-limited devices.

- In video decoder, the complexity reduction due to CABAC is much higher for video sequences with higher motion content in VBR implementations. In CBR mode, larger reduction in complexities is obtained for higher bit-rates.

- The efficiency of an encoder using CABAC in RDO-off mode suggests that no CABAC hardware accelerator is required for a video encoder using RDO-off mode. However, and encoder using RDO-on mode will require a CABAC hardware accelerator.

- H.264/AVC standard defines certain *profiles*. Profiles suggest the set of tools to be used for any specific application, based on the resource constraints. According to H.264/AVC standard, CABAC is not suitable for Baseline and Extended profiles. The finding that CABAC is cost-effective in RDO-off mode suggests that RDO-off mode should be used in Main and higher profiles.

- Both the use of CABAC and RDO improve the coding efficiency. However, in terms of coding efficiency improvements and complexity increases in the video encoder, CABAC is much more useful than RDO as it provides a substantial improvement in coding efficiency without incurring a high increases in computational and data transfer complexities of the video encoder. Furthermore, CABAC delivers consistent coding efficiency improvements regardless of the configuration used in the video encoder whereas the coding performance of RDO is dependent on the choice of coding tools used in the video encoder. It is found that the use of complex coding tools saturates the overall coding efficiency for low-motion content sequences, making the use of RDO for further bit-rate reduction less effective in such cases. However, the use of RDO has negligible impact on the decoderôs complexity. This makes the use of RDO presently more suitable for off-line encoding applications, where bandwidth is a more important issue over coding time and processing power.

- For constant bit-rate encoder, the use of CABAC in comparison to CAVLC results in improvement in video quality. However, the complexity increase is negligible when RDO-off mode is used. This indicates that CABAC is a useful tool for improving the video quality at constant bit-rate.

6.3 Future Work

- The proposed PCI metric has been used and verified for entropy coders for H.264/AVC video codec. The PCI metric has been used only with computational and data transfer complexities being considered as complexity parameters. However, as explained earlier in the thesis, the PCI metric can be extended to include any parameters such as area, power, delay etc. In the future, the PCI metrics can be defined to include those parameters to determine the cost-effective scenarios of an algorithm in a HW platform.
- The defined PCI metric can also be extended for communication systems and used to identify cost-effective scenarios of communication algorithms.
- The proposed PCI method makes a comprehensive comparison between two algorithms in any specific implementation. The method should be made more generalized by including implementation specific parameters that would free the methodology of the requirement of same implementation for the two algorithms.
- PCI method could be used for making on the fly decisions regarding the algorithm to be chosen. For instance, in case of H.264 encoder, the PCI method

could be used to make on the fly decision of choosing CABAC or CAVLC to encode the video stream. Practical implementation difficulties can arise when implementing the PCI method as part of H.264 video encoder. For real-time video streaming applications, which require the encoding to be real time, addition of the PCI decision-making block can in itself contribute to certain amount of increase in complexity. So, the PCI decision-making block should be streamlined and be made to incur very little increase in complexity.

BIBLIOGRAPHY

[1] ñDraft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC,ò in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-G050, T. Wieg, Ed., Pattaya, Thailand, Mar. 2003

[2] T. Wiegand, G.J. Sullivan, G. Bjøntegaard, and A. Luthra, ñOverview of the H.264/AVC Video Coding Standard,ò *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560ï 576, Jul. 2003

[3] ITU-T Recommendation H.261, ñVideo codec for Audiovisual Services at p X 64 kbit/s,ò Mar. 1993.

[4] ISO/IEC 11172: ñInformation technologyð coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s,ò Geneva, 1993

[5] ISO/IEC 13818ï 2: ñGeneric coding of moving pictures and associated audio informationð Part 2: Video,ò 1994, also ITU-T Recommendation H.262

[6] ITU-T Recommendation H.263, ñVideo Coding for Low bit rate Communication,ò version 1, Nov. 1995; version 2, Jan. 1998; version 3, Nov. 2000

[7] ISO/IEC 14496ï 2: ñInformation technologyð coding of audiovisual objectsð part 2: visual,ò Geneva, 2000

[8] D. Marpe, H.Schwarz and T. Wiegand, ñContext-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standardò, *IEEE Trans. Circuits Syst Video Technol.,* Vol. 13, No. 7, pp. 620-636, Jul. 2003

[9] G. Bjøntegaard and K. Lillevold, ñContext-adaptive VLC (CVLC) Coding of Coefficientsò, JVT-C028, 3rd Meeting: Fairfax, Virginia, USA, May 2002

[10] S. Saponara, K. Denolf, C. Blanch, G. Lafruit, and J. Bormans, ñPerformance and Complexity Co-evaluation of the Advanced Video Coding Standard for Cost-effective Multimedia Communicationsò, *EURASIP*, vol. 2004, pp. 220-235, Feb. 2004

[11] C.K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, and G. Lowney, ñPin: Building Customized Program Analysis Tool with Dynamic Instrumentation,ò *PLDI*, Apr. 2005.

[12] http://rogue.colorado.edu/Pin/index.html

[13] F. Pan, K. TW. Choo, and Thinh M. Le, ñFast Rate-Distortion Optimization in H.264/AVC Video Codingò, *Knowledge-Based Intelligent Information & Engineering Systems: Multimedia Compression*, Springer ï series of Lecture notes in Computer Sciences, pp. 425-432, 2005

[14] G. Stitt and F. Vahid., ñHardware/Software Partitioning of Software Binaries: A Case Study of H.264 Decode,ò *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 164-170, Nov. 2002

[15] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini and G. J. Sullivan, ñRate-Constrained Coder Control And Comparison of Video Coding Standardsò, *IEEE Trans. Circuits Video Technol.*, vol. 13, no. 7, pp. 688-702, July 2003

[16] P. Pushner and C. Koza, ñCalculating the Maximum Execution Time of Real Time Programò, *Trans. Signal Process.*, vol. 46, no.4, pp. 1027-1042, Apr 1998

[17] V. Lappalainen, A. Hallapuro and T. Hamalainen, ñComplexity of Optimized H.26L Video Decoder Implementationò, *IEEE Circuits Syst. Video Technol.,* vol. 13, pp. 717-725, Jul 2003

[18] Dongarra, J., London, K., Moore, S., Mucci, P. and Terpstra, D., "Using PAPI for Hardware Performance Monitoring on Linux Systems," *Conference on*

Linux Clusters: The HPC Revolution, Linux Clusters Institute, Urbana, Illinois, June 25-27, 2001

[19] S. Graham, P. Kessler and M. McKusick, ñGprof: A Call Graph Execution Profilerò, *Proc. Symp. Compiler Construction (SIGPLAN),* vol.17, pp.120-126, Jun 1982

[20] C. Xu, M.T. Le, T.T. Tay, ñInstruction Level Complexity Analysisò, *IMSA 2005,* pp.341-346, Aug 2005

[21] D. Burger and T.M. Austin, ñThe SimpleScalar Tool Setò, *Computer Architecture News,* pp. 13-15, Jun. 1997

[22] K. Denolf, P. Vos, J. Bormans, and I. Bolsens, ñ Cost-efficient C-level Design of an MPEG-4 Video Decoderò, *Lecture Notes in Computer Scienc,* vol. 1918, pp. 233-242, Springer-Verlag, Heidelberg, Sep 2000

[23] http://www.imec.be/design/atomium/

[24] M.Ravasi and M. Mattavelli, ñHigh-Abstraction Level Complexity Analysis And Memory Architecture Simulations of Multimedia Algorithmsò, *IEEE Circuits Syst. Video Technol.,* vol. 15, pp. 673-684, May 2005

[25] H.J. Stolberg, M. Berekovic and P. Pirsch, ñA Platform-independent Methodology for Performance Estimation of Streaming Media Applicationsò, *IEEE International Conference on Multimedia and Expo,* pp. 105-108, Lausanne, Switzerland, Aug. 2002.

[26] M. Horowitz, A. Kossentini and A. Hallapuro, ñH.264/AVC Baseline Profile Decoder Complexity Analysisò, *IEEE Circuits Syst. Video Technol.,* vol. 13, pp. 704-716, Jul. 2003.

[27] V. Lappalainen, A. Hallapuro and T. Hamalainen, ñPerformance Analysis of Low Bit-rate H.26L Video Encoder,ò *IEEE ICASSP'01,* pp. 1129-1132, May 2001

[28] F. Catthooer, *Custom Memory Management Methodology*, Kluwer Academic Publishers, 1998

[29] L. Nachtergaele, D. Moolenaar, B. Vanhoof, F. Catthoor and H. De Man, ñSystem-level Power Optimization of Video Codec on Embedded Cores: A Systematic Approach,ò *Journ. VLSI Signal Processing,* vol.18, no. 2, pp. 89-111, 1998

[30] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, P. Pereira, T. Stockhammer, and T. Wedi, ñVideo Coding with H.264/AVC: Tools, Performance, and Complexityò, *IEEE Circuits and Systems Magazine*, pp.7-28, First Quarter 2004

[31] R.R. Osorio, and J.D. Bruguera, ñArithmetic Coding Architecture for H.264/AVC CABAC Compression Systemò, *Proceedings of IEEE EUROMICRO symposium on Digital System Design*, pp. 62ï 69, 2004.

[32] H. Shojania and S. Sudharsanan, ñA High Performance CABAC Encoderò, *The 3rd International IEEE-NEWCAS Conference*, pp. 315-318, Jun. 2005.

[33] V.H.S. Ha, W.-S. Shim, and J.-W. Kim, ñReal-time MPEG-4 AVC/H.264 CABAC Entropy Coder,ò *Digest of Technical Papers*, International Conference on Consumer Electronics, pp. 255 ï 256, 2005.

[34] J.L. Nunez-Yanez, V.A. Chouliaras, D. Alfonso, and F.S. Rovati., ñHardware-Assisted Rate Distortion Optimization with Embedded CABAC Accelerator for the H.264 Advanced Video Codecò, *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 590-597, May 2006.

[35] C.S. Kannangara, I.E. Richardson, and A.J. Miller, ñComputational Complexity Management of a Real-Time H.264/AVC Encoderò, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 9, Sep. 2008.

[36] Y.-K. Tu, J.-F. Yang, and M.-T. Sun, ñRate-Distortion Modeling for Efficient H.264/AVC Encodingò, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 5, May 2007.

[37] R.R. Osorio and J.D. Bruguera, ñHigh-Throughput Architecture for H.264/AVC CABAC Compression Systemò, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 11, pp. 1376-1384, Nov. 2006.

[38] G. Sullivan, T.Wiegand, and K.-P. Lim, ñJoint Model Reference Encoding Methods and Decoding Concealment Methods,ò presented at the 9th JVT, Meeting (JVT-I049d0), San Diego, CA, Sep. 2003.

[39] http://iphome.hhi.de/suehring/tml/download

[40] D.A. Patterson, T. Anderson, and K. Yelick, ñA Case for Intelligent RAM: IRAMò, Hot chip 8, 1996.

[41] *Goodness of fit*, http://portal.wsiz.rzeszow.pl/plik.aspx?i d=5435

APPENDIX: Procedure on Installations and Configurations for Empirical Analyses

The exhaustive empirical analyses done are an integral part of this thesis. The installations and configurations done to obtain all the relevant data are described as follows:

- JM, which is H.264/AVC video codec reference software, has to be installed in the system. This involves downloading the software and using the *make* files to obtain the encoder and decoder executables.
- The video coder settings are listed in configuration file (*.cfg) which are used by the encoder executables. The configuration files have to be varied for empirical analysis across different settings.
- PIN is a tool for the dynamic instrumentation of programs. PIN does not instrument an executable statically by rewriting it, but rather adds the code dynamically while the executable is running. This makes it possible to attach PIN to an already running process. PIN has to be downloaded and installed in the system.
- PIN provides a set of APIs which can retrieve register contents, instruction type etc. However, we will have to write programs using these APIs to obtain the required data.
- Once the profiling program is written using the APIs provided by PIN, we have to compile it using the following compiler options:

```
ñg++ -g -Wl,-u,malloc -Wl,--section-start,.interp=0x05048400  -L$PIN_D/Lib/ -
L$PIN_D/ExtLib/ -lpin  -lxed -ldwarf -lelf -ldl -g -Wall -Werror -Wno-unknown-
pragmas -g -O3 -DBIGARRAY_MULTIPLIER=1 -DUSING_XED -g -fno-strict-
aliasing -I$PIN_D/Include -I$PIN_D/Include/gen -I$PIN_D/InstLib -
DTARGET_IA32 -DTARGET_LINUX”
```

- Compilation will generate the shared library file which should be used along with the PIN executable and the executable to be profiles as follows to obtain the profile data:

pin -t <shared library profiling file> -- <executable to be profiles : lencod/ldecod>

The lencod and ldecod are the JM executables responsible for H.264/AVC encoding and decoding process respectively.

- Note that the above process has to be repeated for every possible video coder setting and all the video sequences. A *bash* script (Linux Shell Script) can be used to automate this task. A line from the script that was used is as follows:

```
pin -t run_pin_complexity -- ./lencod.exe -d encoder_cif_A_cabac_no_rdo.cfg -p SourceWidth=352 -p SourceHeight=288 -p RateControlEnable=0 -p InputFile="./seq/cif/soccer_cif.yuv">"soccer_cif_A_cabac_no_rdo.dat"
```

Profiling code using PIN - a static and dynamic opcode mix profiler

```
#include "pin.H"
#include "instlib.H"
#include "portability.H"
#include <vector>
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace INSTLIB;

/* Commandline Switches */

KNOB<string> KnobOutputFile(KNOB_MODE_WRITEONCE,        "pintool",
    "o", "opcodemix.out", "specify profile file name");
KNOB<BOOL>   KnobPid(KNOB_MODE_WRITEONCE,               "pintool",
    "i", "0", "append pid to output");
KNOB<BOOL>   KnobProfilePredicated(KNOB_MODE_WRITEONCE,  "pintool",
    "p", "0", "enable accurate profiling for predicated instructions");
KNOB<BOOL>   KnobProfileStaticOnly(KNOB_MODE_WRITEONCE,  "pintool",
    "s", "0", "terminate after collection of static profile for main image");
#ifndef TARGET_WINDOWS
```

```
KNOB<BOOL>              KnobProfileDynamicOnly(KNOB_MODE_WRITEONCE,
"pintool",
    "d", "0", "Only collect dynamic profile");
#else
KNOB<BOOL>              KnobProfileDynamicOnly(KNOB_MODE_WRITEONCE,
"pintool",
    "d", "1", "Only collect dynamic profile");
#endif
KNOB<BOOL>   KnobNoSharedLibs(KNOB_MODE_WRITEONCE,      "pintool",
    "no_shared_libs", "0", "do not instrument shared libraries");


/* ===================================== */

INT32 Usage()
{
    cerr <<"This pin tool computes a static and dynamic opcode mix profile\n";
    cerr << KNOB_BASE::StringKnobSummary();
    cerr << endl;
    return -1;
}

/* ======================================= */
/* INDEX HELPERS */
/* ================================== */

const UINT32 MAX_INDEX = 4096;      // enough even for the IA-64 architecture
const UINT32 INDEX_SPECIAL =  3000;
const UINT32 MAX_MEM_SIZE = 512;
const UINT32 INDEX_TOTAL =          INDEX_SPECIAL + 0;
const UINT32 INDEX_MEM_ATOMIC =     INDEX_SPECIAL + 1;
const UINT32 INDEX_STACK_READ =     INDEX_SPECIAL + 2;
const UINT32 INDEX_STACK_WRITE =    INDEX_SPECIAL + 3;
const UINT32 INDEX_IPREL_READ =     INDEX_SPECIAL + 4;
const UINT32 INDEX_IPREL_WRITE =    INDEX_SPECIAL + 5;
const UINT32 INDEX_MEM_READ_VARIABLE =  INDEX_SPECIAL + 6;
const UINT32 INDEX_MEM_WRITE_VARIABLE =  INDEX_SPECIAL + 7;
const UINT32 INDEX_MEM_READ_SIZE =  INDEX_SPECIAL + 8;
const   UINT32   INDEX_MEM_WRITE_SIZE   =   INDEX_SPECIAL   +   8   +
MAX_MEM_SIZE;
const   UINT32   INDEX_SPECIAL_END       =     INDEX_SPECIAL   +   8   +
MAX_MEM_SIZE + MAX_MEM_SIZE;


BOOL IsMemReadIndex(UINT32 i)
{
    return (INDEX_MEM_READ_SIZE <= i && i < INDEX_MEM_READ_SIZE +
MAX_MEM_SIZE );
}
```

```
BOOL IsMemWriteIndex(UINT32 i)
{
   return (INDEX_MEM_WRITE_SIZE <= i && i < INDEX_MEM_WRITE_SIZE
+ MAX_MEM_SIZE );
}

/* ==================================== */

LOCALFUN UINT32 INS_GetIndex(INS ins)
{
   if( INS_IsPredicated(ins) )
      return MAX_INDEX + INS_Opcode(ins);
   else
      return INS_Opcode(ins);
}

/* ======================================== */

LOCALFUN  UINT32 IndexStringLength(BBL bbl, BOOL memory_acess_profile)
{
   UINT32 count = 0;
   for (INS ins = BBL_InsHead(bbl); INS_Valid(ins); ins = INS_Next(ins))
   {
      count++;
      if( memory_acess_profile )
      {
         if( INS_IsMemoryRead(ins) ) count++;   // for size
         if( INS_IsStackRead(ins) ) count++;
         if( INS_IsIpRelRead(ins) ) count++;
         if( INS_IsMemoryWrite(ins) ) count++; // for size
         if( INS_IsStackWrite(ins) ) count++;
         if( INS_IsIpRelWrite(ins) ) count++;
         if( INS_IsAtomicUpdate(ins) ) count++;
      }
   }
   return count;
}
/* ================================== */
LOCALFUN UINT32 MemsizeToIndex(UINT32 size, BOOL write)
{
   if( size == VARIABLE_MEMORY_REFERENCE_SIZE )
   {
      return write ? INDEX_MEM_WRITE_VARIABLE :
INDEX_MEM_READ_VARIABLE;
   }
   else
   {
      return (write ? INDEX_MEM_WRITE_SIZE : INDEX_MEM_READ_SIZE ) +
size;
   }
```

```
}

/* ============================== */
LOCALFUN UINT16 *INS_GenerateIndexString(INS ins, UINT16 *stats, BOOL
memory_acess_profile)
{
   *stats++ = INS_GetIndex(ins);
   if( memory_acess_profile )
   {
      if( INS_IsMemoryRead(ins) )
         *stats++ = MemsizeToIndex( INS_MemoryReadSize(ins), 0 );
      if( INS_IsMemoryWrite(ins) )
         *stats++ = MemsizeToIndex( INS_MemoryWriteSize(ins), 1 );
      if( INS_IsAtomicUpdate(ins) ) *stats++ = INDEX_MEM_ATOMIC;
      if( INS_IsStackRead(ins) ) *stats++ = INDEX_STACK_READ;
      if( INS_IsStackWrite(ins) ) *stats++ = INDEX_STACK_WRITE;
      if( INS_IsIpRelRead(ins) ) *stats++ = INDEX_IPREL_READ;
      if( INS_IsIpRelWrite(ins) ) *stats++ = INDEX_IPREL_WRITE;
   }
   return stats;
}
/* ============================== */

LOCALFUN string IndexToOpcodeString( UINT32 index )
{
   if( INDEX_SPECIAL <= index  && index < INDEX_SPECIAL_END)
   {
      if( index == INDEX_TOTAL )          return  "*total";
      else if( IsMemReadIndex(index) )
         return  "*mem-read-" + decstr( index - INDEX_MEM_READ_SIZE );
      else if( IsMemWriteIndex(index))
         return  "*mem-write-" + decstr( index - INDEX_MEM_WRITE_SIZE );
      else if( index == INDEX_MEM_READ_VARIABLE )
         return  "*mem-read-variable";
      else if( index == INDEX_MEM_WRITE_VARIABLE )
         return  "*mem-write-variable";
      else if( index == INDEX_MEM_ATOMIC )  return  "*mem-atomic";
      else if( index == INDEX_STACK_READ )  return  "*stack-read";
      else if( index == INDEX_STACK_WRITE ) return  "*stack-write";
      else if( index == INDEX_IPREL_READ )  return  "*iprel-read";
      else if( index == INDEX_IPREL_WRITE ) return  "*iprel-write";
      else
      {
         ASSERTX(0);
         return "";
      }
   }
   else
   {
      return OPCODE_StringShort(index);
```

```
    }
}

/* ==================================== */
/* =================================== */
typedef UINT64 COUNTER;

/* zero initialized */

class STATS
{
  public:
    COUNTER unpredicated[MAX_INDEX];
    COUNTER predicated[MAX_INDEX];
    COUNTER predicated_true[MAX_INDEX];
    VOID Clear()
    {
        for ( UINT32 i = 0; i < MAX_INDEX; i++)
        {
            unpredicated[i] = 0;
            predicated[i] = 0;
            predicated_true[i] = 0;
        }
    }
};
STATS GlobalStatsStatic;
STATS GlobalStatsDynamic;

class BBLSTATS
{
  public:
    COUNTER _counter;
    const UINT16 * const _stats;

  public:
    BBLSTATS(UINT16 * stats) : _counter(0), _stats(stats) {};

};

LOCALVAR vector<const BBLSTATS*> statsList;



/* ============================== */

LOCALVAR INT32 enabled = 0;
LOCALFUN VOID Handler(CONTROL_EVENT ev, VOID *val, CONTEXT * ctxt,
VOID *ip, THREADID tid)
{
    switch(ev)
    {
      case CONTROL_START:
        enabled = 1;
        break;
      case CONTROL_STOP:
        enabled = 0;
        break;
      default:
        ASSERTX(false);
    }
}

LOCALVAR CONTROL control;
/* ================================= */

VOID docount(COUNTER * counter)
{
    (*counter) += enabled;
}

/* ========================================== */

VOID Trace(TRACE trace, VOID *v)
{
    if      (       KnobNoSharedLibs.Value()                                    &&
IMG_Type(SEC_Img(RTN_Sec(TRACE_Rtn(trace))))                                     ==
MG_TYPE_SHAREDLIB)
        return;

    const BOOL accurate_handling_of_predicates = KnobProfilePredicated.Value();

    for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl))
    {
        const INS head = BBL_InsHead(bbl);
        if (! INS_Valid(head)) continue;
        // Summarize the stats for the bbl in a 0 terminated list
        // This is done at instrumentation time
        const UINT32 n = IndexStringLength(bbl, 1);
        UINT16 *const stats = new UINT16[ n + 1];
        UINT16 *const stats_end = stats + (n + 1);
        UINT16 *curr = stats;
        for (INS ins = head; INS_Valid(ins); ins = INS_Next(ins))
        {
            // Count the number of times a predicated instruction is actually executed
            // this is expensive and hence disabled by default
            if( INS_IsPredicated(ins) && accurate_handling_of_predicates )
            {
                INS_InsertPredicatedCall(ins,IPOINT_BEFORE,
AFUNPTR(docount),IARG_PTR,
&(GlobalStatsDynamic.predicated_true[INS_Opcode(ins)]),
                              IARG_END);
            }
            curr = INS_GenerateIndexString(ins,curr,1);
        }
        // string terminator
        *curr++ = 0;
        ASSERTX( curr == stats_end );
        // Insert instrumentation to count the number of times the bbl is executed
        BBLSTATS * bblstats = new BBLSTATS(stats);
        INS_InsertCall(head, IPOINT_BEFORE, AFUNPTR(docount), IARG_PTR,
&(bblstats->_counter), IARG_END);
        // Remember the counter and stats so we can compute a summary at the end
        statsList.push_back(bblstats);
    }
}

/* ============================== */
VOID DumpStats(ofstream& out, STATS& stats, BOOL predicated_true,  const
string& title)
{
    out <<"#\n# " << title << "\n#\n#   opcode      count-unpredicated      count-
predicated";
    if( predicated_true )
        out << "    count-predicated-true";
    out << "\n#\n";

    for ( UINT32 i = 0; i < INDEX_TOTAL; i++)
    {
        stats.unpredicated[INDEX_TOTAL] += stats.unpredicated[i];
        stats.predicated[INDEX_TOTAL] += stats.predicated[i];
        stats.predicated_true[INDEX_TOTAL] += stats.predicated_true[i];
    }

    for ( UINT32 i = 0; i < MAX_INDEX; i++)
    {
        if( stats.unpredicated[i] == 0 &&
            stats.predicated[i] == 0 ) continue;

        out << setw(4) << i << " " <<  ljstr(IndexToOpcodeString(i),15) << " " <<
            setw(16) << stats.unpredicated[i] << " " <<
            setw(16) << stats.predicated[i];
        if( predicated_true ) out << " " << setw(16) << stats.predicated_true[i];
        out << endl;
    }
}



/* ================================== */
static std::ofstream* out = 0;
```

```
VOID Fini(int, VOID * v)
{
   // static counts
   DumpStats(*out, GlobalStatsStatic, false, "$static-counts");
   *out << endl;
   // dynamic Counts
   statsList.push_back(0); // add terminator marker
   for (vector<const BBLSTATS*>::iterator bi = statsList.begin(); bi !=
statsList.end(); bi++)
   {
      const BBLSTATS *b = (*bi);
      if ( b == 0 ) continue;
      for (const UINT16 * stats = b->_stats; *stats; stats++)
      {
         GlobalStatsDynamic.unpredicated[*stats] += b->_counter;
      }
   }
   DumpStats(*out, GlobalStatsDynamic, KnobProfilePredicated, "$dynamic-
counts");
   *out << "# $eof" <<  endl;
   out->close();
}

/* ===================================== */

VOID Image(IMG img, VOID * v)
{
   for (SEC sec = IMG_SecHead(img); SEC_Valid(sec); sec = SEC_Next(sec))
   {
      for (RTN rtn = SEC_RtnHead(sec); RTN_Valid(rtn); rtn = RTN_Next(rtn))
      {
         // Prepare for processing of RTN, an  RTN is not broken up into BBLs,
         // it is merely a sequence of INSs
         RTN_Open(rtn);

         for (INS ins = RTN_InsHead(rtn); INS_Valid(ins); ins = INS_Next(ins))
         {
            UINT16 array[128];
            UINT16 *end  = INS_GenerateIndexString(ins,array,1);

            if( INS_IsPredicated(ins) )
            {
               for( UINT16 *start= array; start < end; start++)
GlobalStatsStatic.predicated[ *start ]++;
            }
            else
            {
               for( UINT16 *start= array; start < end; start++)
GlobalStatsStatic.unpredicated[ *start ]++;
```

```
                }
            }

            // to preserve space, release data associated with RTN after we have processed
            RTN_Close(rtn);
        }
    }
    if( KnobProfileStaticOnly.Value() )
    {
        Fini(0,0);
        exit(0);
    }
}
/* ========================================= */
int main(int argc, CHAR *argv[])
{
    PIN_InitSymbols();
    if( PIN_Init(argc,argv) )
    {
        return Usage();
    }
    control.CheckKnobs(Handler, 0);
    string filename =  KnobOutputFile.Value();
    if( KnobPid )
    {
        filename += "." + decstr( getpid_portable() );
    }
    out = new std::ofstream(filename.c_str());
    TRACE_AddInstrumentFunction(Trace, 0);
    PIN_AddFiniFunction(Fini, 0);
    if( !KnobProfileDynamicOnly.Value() )
        IMG_AddInstrumentFunction(Image, 0);
    // Never returns
    PIN_StartProgram();
    return 0;
}

/* End-Of-Code*/
```