H.264/AVC Baseline Profile Decoder Complexity Analysis

Michael Horowitz, Anthony Joch, Faouzi Kossentini, *Senior Member, IEEE*, and Antti Hallapuro

***Abstract*—In this paper, we study and analyze the computational complexity of a software-based H.264/AVC baseline profile decoder. Our analysis is based on determining the number of basic computational operations required by a decoder to perform the key decoding subfunctions. The frequency of use of each of the required decoding subfunctions is empirically derived using bitstreams generated from two different encoders for a variety of content, resolutions and bit rates. Using the measured frequencies, estimates of the decoder time complexity for various hardware platforms can be determined. A detailed example is provided to assist readers in deriving their own time complexity estimates. We compare the resulting estimates to numbers measured for an optimized decoder on the Pentium 3 hardware platform. We then use those numbers to evaluate the dependence of the time complexity of each of the major decoder subfunctions on encoder characteristics, content, resolution and bit rate. Finally, we compare an H.264/AVC-compliant baseline decoder to a decoder that is compliant with the H.263 standard, which is currently dominant in interactive video applications. Both "C" only decoder implementations were compared on a Pentium 3 hardware platform. Our results indicate that an H.264/AVC baseline decoder is approximately 2.5 times more time complex than an H.263 baseline decoder.**

***Index Terms*—Complexity, decoder, H.264/AVC, H.26L, JVT.**

I. Introduction

THE ITU-T/ISO/IEC draft H.264/AVC video coding standard [1] promises improved coding efficiency over existing video coding standards [2], [3]. Although its coding algorithms are based on the same block-based motion compensation and transform-based spatial coding framework of prior video coding standards, H.264/AVC provides higher coding efficiency through added features and functionality. However, such features and functionality also entail additional complexity in encoding and decoding. The computational complexity of the coding algorithms directly affects the cost effectiveness of the development of a commercially viable H.264/AVC-based video solution.

In this paper, we study and analyze the computational complexity of the H.264/AVC baseline profile decoder, which we describe in Section II. Unlike encoder implementations, which can vary by orders of magnitude in terms of computational complexity, decoder implementations usually entail the same level of complexity. This is because the behavior of the decoder is fully specified by the standard that it supports and the characteristics (e.g., bit rate, resolution) of the coded content that it must process. Therefore, it will be simpler and arguably more useful to study and analyze the computational complexity of the H.264/AVC baseline decoder.

To estimate the computational complexity of an H.264/AVC baseline decoder implementation, it is important to understand its two major components: time complexity and space (or storage) complexity. Time complexity is measured by the approximate number of operations required to execute a specific implementation of an algorithm. A clever implementation of a given algorithm may have significantly lower time complexity than a less clever one for a given hardware platform despite the fact that the two implementations produce identical results. Storage complexity is measured by the approximate amount of memory required to implement an algorithm. It is a function of the algorithm, though it may be affected somewhat by implementation design specifics. The cost of hardware for a given application is a function of both time and storage complexity as both affect the size of the underlying circuits. In this work, the computational complexity of the H.264/AVC baseline decoder is studied and analyzed in the context of a software implementation on either a general purpose or a DSP/media processor.

In a software implementation, the time complexity of an algorithm's implementation in conjunction with its memory bandwidth requirements determine the time that is required to execute the algorithm on a specific hardware platform. The memory bandwidth of an algorithm's implementation is loosely defined as the bandwidth that is required to move data from memory to the processor and back in the course of execution. A viable software architecture will strike a balance between the time spent operating on data and the time spent loading/storing the same data. A typical H.264/AVC baseline video decoder has large memory bandwidth and computational resource requirements, making this balancing act all the more difficult. Consider an example implementation of an algorithm in which a simple set of operations require a total of 20 machine cycles to process a set of data on a particular hardware platform. Furthermore, assume that loading of that data from memory requires 50 cycles. It is clear that, in this example, the implementation is memory I/O bound. In other words, we would spend more time loading data than performing the operations. In this case, an estimate of the number of operations will not provide an accurate estimate of the overall execution time. Optimization techniques such as

Manuscript received April 24, 2002; revised May 9, 2003.

M. Horowitz is with Polycom Inc., Austin, TX 78746 USA (e-mail: mhorowitz@austin.polycom.com).

A. Joch is with UB Video Inc., Vancouver, BC V6B 2R9, Canada (e-mail: anthony@ubvideo.com).

F. Kossentini is with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: faouzi@ece.ubc.ca).

A. Hallapuro is with Nokia Research Center, Tampere, FIN-33721, Finland (e-mail: antti.hallapuro@nokia.com).

Digital Object Identifier 10.1109/TCSVT.2003.814967

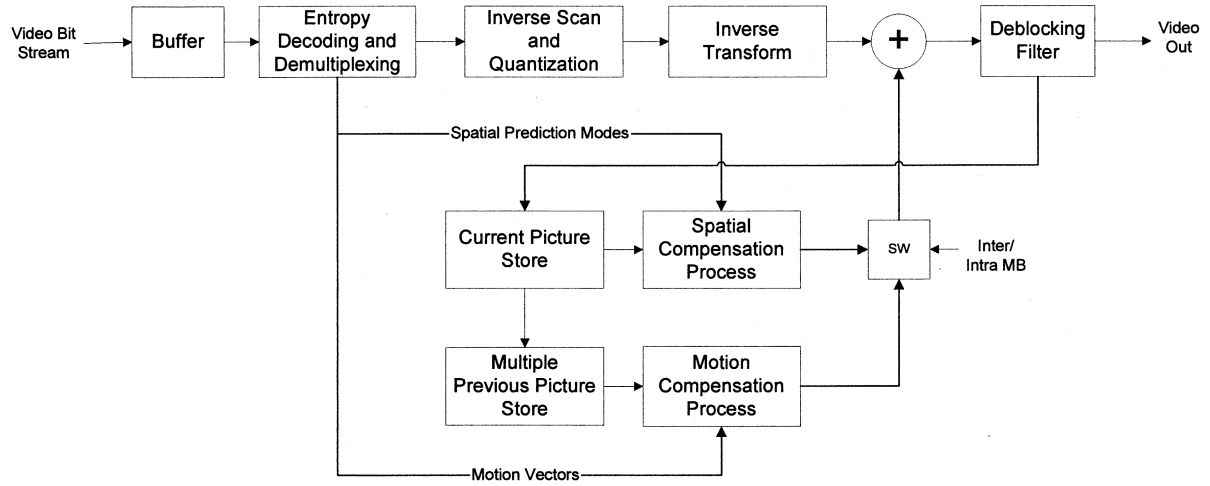


Fig. 1. Block diagram of an H.264/AVC decoder.

pre-loading data to a data cache, an intermediate memory location that provides very fast data access to the processor, can alleviate the memory I/O bottleneck in this example. When a good balance between memory bandwidth and operations is achieved, the time complexity of an algorithm provides an accurate estimate of the execution time for a given implementation of an algorithm on a specific platform.

In a software implementation, some storage complexity can be traded for time complexity. However, a minimum amount of storage is often required for an efficient software implementation. Knowledge of such minimum storage requirements for an H.264/AVC baseline decoder is important, particularly in the case of DSP/media processors where memory cost is a large component of the overall system cost. The storage requirements of an H.264/AVC baseline decoder are presented in Section III. In the remaining sections of the paper, we focus on the evaluation of the time complexity of an H.264/AVC baseline decoder software implementation. First, we develop and validate a methodology for estimating decoder time complexity. The methodology allows the reader, using data presented in this paper (and elsewhere), to derive the time complexity estimates for a variety of operating conditions and processors. Second, we study the relationship between decoder time complexity and encoder characteristics, source content, resolution and bit rate.

We take a systematic approach to quantifying the time complexity of an H.264/AVC baseline decoder. The basis of our method is to determine the number of basic operations required by the decoder to perform each of the key decoder subfunctions. The number of times that these subfunctions must be performed is measured using bitstreams generated from two different encoders (that are compliant with the latest draft of the standard at the time our experiments were performed [4]) for different content, resolutions and bit rates. By mapping these computational requirements to the processing capabilities of several hardware platforms, an estimate of the decoder's time complexity can be determined for each platform. Our methodology is described in Section IV. In Section V, we compare our time complexity estimates to empirical results generated for an optimized decoder implementation on the Pentium 3 (P3) platform. In Section VI, we discuss (using empirical results) the behavior of the H.264/AVC baseline decoder as a function of content, resolution and bit rate. Section VII presents results that quantify the difference in time complexity between a decoder compliant with the baseline profile of the emerging H.264/AVC standard, and a decoder compliant with the H.263 standard,[1] which is currently dominant in video conferencing applications. Conclusions are presented in Section VIII.

II. H.264/AVC Decoder: Overview

As with all popular video coding standards, the operation of a compliant H.264/AVC decoder is fully specified by the standard. The decoding process consists of interpreting the coded symbols of a compliant bitstream and processing this data according to the standard specification in order to generate a reconstructed video sequence. H.264/AVC defines a hybrid block-based video codec, similar to earlier standards such as ISO/IEC MPEG-2, ITU-T H.261, and H.263. Such codecs combine inter-picture prediction to exploit temporal redundancy with transform-based coding of the prediction errors to exploit the spatial redundancy within this signal. Thus, the decoding process consists generally of two primary paths: the generation of the predicted video blocks and the decoding of the coded residual blocks. The sample values of the output blocks resulting from these two paths are summed and clipped to the valid range for pixel data to form the reconstructed blocks. While it is based on the same hybrid-coding framework, the H.264/AVC standard features a number of significant components that distinguish it from its predecessors. This section provides a description of the data flow and processes of an H.264/AVC decoder. For a detailed description of the standard, see [5].

A generalized block diagram of an H.264/AVC decoder is given in Fig. 1. The H.264/AVC decoder shares several common features with all other hybrid video decoders. This includes the

[1]Although more recent (e.g., MPEG-4 simple profile) and/or more successful (e.g., MPEG-2 Part 2 or H.262) visual coding standards exist, the H.263 standard is here compared to H.264/AVC baseline, which also targets real-time video communication applications such as video conferencing and video telephony.

buffering of the input bitstream, entropy decoding, motion-compensated prediction, inverse scanning, quantization and transformation, and the summation of the prediction and the transform-coded residual video blocks.

The incoming video bitstream is stored in a buffer at the input to the decoder. The first stage in the decoding process includes the parsing and decoding of the entropy coded bitstream symbols that are stored in the buffer. In H.264/AVC, two different entropy coding modes are supported at this time,[2] : a simple universal variable length coding (UVLC) method, which makes use of a single code-table for all syntax elements; and a more complex and compression efficient context-adaptive binary arithmetic coding (CABAC) method. The complexity of the CABAC decoding method is derived from the fact that it is a bit-serial operation, requiring several operations to be performed sequentially in order to decode each bit of data. In addition, CABAC decoding requires continuous updates to a large set of context models throughout the decoding process, and the arithmetic decoding of symbols. The UVLC method can be implemented using a relatively small number of operations, requiring the parsing of each symbol and table lookups. Since the analysis presented in this work focuses on the baseline profile of H.264/AVC (which supports two other profiles, namely Main Profile and Extended Profile), only the UVLC method is addressed.

The various syntax elements of the bitstream are demultiplexed for use in different processes within the decoder. High-level syntax elements include temporal information for each frame, slice coding type, and frame dimensions. H.264/AVC coding, as with earlier standards, is based primarily on macroblocks consisting of one 16×16 luminance-sample block and two 8×8 chrominance sample blocks. On the macroblock level, syntax elements include the coding mode of the macroblock, information required for forming the prediction, such as motion vectors and spatial prediction modes, and the coded information of the residual (difference) blocks, such as the coded block pattern (CBP) and quantized transform coefficients.

Depending on the coding mode of each macroblock, the predicted macroblock can be generated either temporally (inter-coding) or spatially (intra-coding). The prediction for an inter-coded macroblock is determined by the set of motion vectors that are associated with that macroblock. The motion vectors indicate the position within the set of previously decoded frames from which each block of pixels will be predicted. Each inter-coded macroblock can be partitioned in one of seven ways, with luminance block sizes ranging from 16×16 samples to 4×4 samples. Also, a special *skip* mode exists in which no motion vectors (or coded residual blocks) are transmitted. Thus, 0–16 motion vectors can be transmitted for each inter-coded macroblock. Finally, note that additional predictive modes are supported when B-pictures are employed, and since we focus on the baseline profile of H.264/AVC, only I- and P-pictures are considered in this work.

[2]An entropy coding mode, content-adaptive variable-length coding (CAVLC) introduced in [6] was recently incorporated into the draft standard to encode the quantization indices of transform coefficients, subsequent to the analyzes prepared for this paper. Our latest empirical experiments suggest that CAVLC decoding is roughly two times more time complex than UVLC decoding.

A	d	*b*	d
e	h	f	h
b	g	*c*	g
e	h	f	i

Fig. 2. Spatial relationships of the integer position A, half-sample positions *b* and *c*, and quarter-sample positions, d, e, f, g, h and i.

Motion vectors are coded differentially using either median or directional prediction, depending on the partitioning mode that is used for the macroblock. For each motion vector, a predicted block must be computed by the decoder and then arranged with other blocks to form the predicted macroblock. Motion vectors for the luminance component are specified with quarter-sample accuracy. Interpolation of the reference video frames is necessary to generate the predicted macroblock using subsample accurate motion vectors. The complexity of the required interpolation filter varies depending on the phase specified by the motion vector. To generate a predicted macroblock using a motion vector that indicates a half-sample position, an interpolation filter that is based on a 6-tap windowed sinc function is employed. In the case of prediction using a motion vector that indicates a quarter-sample position, filtering consists of averaging two integer- or half-sample position values. Fig. 2 shows the spatial relationship of the integer, half- and quarter-sample positions. A bilinear filter is used to interpolate the chrominance frames when subsampled motion vectors are used to predict the underlying chrominance blocks.

H.264/AVC also supports the use of multiple reference frames for prediction. Selection of a particular reference frame is made on a macroblock basis. This feature can improve both coding efficiency and error resilience. However, this feature also requires that the decoder buffer store several previously decoded and reconstructed frames, rather than just the most recent reconstructed frame, increasing substantially the decoder's memory requirements.

For intra-coding of macroblocks, two different modes are supported. In the 4×4 intra-coding mode, each 4×4 luminance block within a macroblock can use a different prediction mode. There are nine possible modes: DC and eight directional prediction modes. The time complexity of generating the prediction for each mode varies, with the DC, vertical, and horizontal modes being the least complex, and the diagonal modes being the most complex. With the 16×16 intra-coding mode, which is generally used for coding smooth areas, there are four modes available: DC, vertical, horizontal, and planar, with the latter being the most complex. The prediction of intra-coded blocks is always based on neighboring pixel values that have already been decoded and reconstructed.

The decoding of a residual (difference) macroblock requires that a number of inverse transforms be performed, along with associated inverse scanning and quantization operations. The

TABLE I
STORAGE REQUIREMENTS, IN BYTES, FOR THE H.264/AVC BASELINE PROFILE DECODER

Buffer name	**Formula**	**QCIF** w=176, h=144,n=1	**CIF** w=352,h=288,n=1
Reconstruction frame	1.5*w*h	38016	152064
Reference frames	n*1.5*w*h	38016	152064
Slice map	w/16*h/16*4	396	1584
Reference indices	w/16*h/16*4	396	1584
Motion vectors	w/16*h/16*64	6336	25344
Intra-prediction modes	w/16*h/16*16	1584	6336
CBP values	w/16*h/16*4	396	1584
MB types	w/16*h/16	99	396
QP values	w/16*h/16	99	396
CAVLC coefficient counts	1.5*w/16*h/16*16	2376	9504
MB temp data	~2048	2048	2048
Constants	~2048	2048	2048
TOTAL	(n+1)*1.5*w*h + w/16*h/16*118 + 4096	91810	354952

encoding of a difference macroblock is based primarily on the transformation of 4×4 blocks of both the luminance and chrominance samples, although in some circumstances, a second-level transform must be performed on the DC coefficients of a group of 4×4 blocks. More specifically, a special 2×2 transform is applied to the 4 DC coefficients of the blocks of the chrominance samples in a macroblock. For macroblocks that are being coded in the 16×16 intra-coding mode, an additional scan and transform are applied to the DC values of each of the 16 4×4 luminance blocks of a macroblock.

The inverse transforms that are required for each macroblock are determined based on the coding mode and the CBP of the macroblock. The input data are the run-level codes (for the UVLC entropy decoder) that are parsed by the entropy decoder. These are ordered based on the run values through the inverse scanning process and then the levels, which represent quantized transform coefficients, are inverse quantized via multiplication by a scaling factor. Finally, the integer-specified inverse transform is performed on the inverse quantized coefficients. The inverse transformed result for each macroblock is added to the predicted macroblock and stored in the reconstructed frame buffer.

In the final stage of the decoding process, an H.264/AVC decoder must apply the normative deblocking filtering process, which reduces blocking artifacts that are introduced by the coding process at block boundaries. Since the standard specifies that the filter be applied within the motion compensation loop, any compliant decoder must perform this filtering exactly. Since it is inside the motion compensation loop, the deblocking filter is often referred to as a "loop filter". The filtering is based on the 4×4 block edges of both luminance and chrominance components. The type of filter used, the length of the filter, and its strength are dependent on several coding parameters as well as picture content on both sides of each edge. A stronger filter is used if either side of the edge is a macroblock boundary where one or both sides of the edge are intra-coded. The length of the filtering is also determined by the sample values over the edge, which determine the so-called "activity parameters". These parameters determine whether 0, 1, or 2 samples on either side of the edge are modified by the standard filter.[3]

III. H.264/AVC BASELINE DECODER: STORAGE REQUIREMENTS

The storage required by an H.264/AVC baseline decoder can be divided into four memory classes.

1) *Memory that is needed for the whole frame:* This memory includes reconstructed frame memory and reference frame memory. Assuming a 4:2:0 picture format with 8 bits per color component, one pixel requires 1.5 bytes of storage.
2) *Memory that is needed for one line of macroblocks:* This memory includes tables that are used for loop filtering and intra prediction. The loop filter and spatial prediction processes require values from macroblocks above them so the whole line of values has to be stored.
3) *Memory that is needed for a macroblock:* This memory includes temporary buffers that are used for storing transform coefficients, prediction values and pixel values. It is possible to reduce this memory to less than 1 kbyte by copying prediction and reconstruction values directly to the frame buffer.
4) *Memory that is needed for constant data:* This includes variable-length decoding (VLD) tables, intra-prediction probability tables and a few other small constant tables.

Table I presents a summary of the storage requirements of an H.264/AVC baseline decoder. In the table, w is the width of the

[3]The specification of the loop filtering has been modified slightly subsequent to the draft standard upon which this paper is based. The main effect of the changes is that the possibility of parallelism in the filtering operations has been improved, which has reduced the complexity of the filtering by 25%–50% for DSP/media processor implementations. For the P3 processor, upon which our empirical analysis is based, the change in the complexity has been found to be much smaller, although the complexity here has also been reduced somewhat due to increased possibility of SIMD processing and fewer operations required for filtering.

picture, h is the height of the picture and n is the number of reference frames. As expected, frame buffers dominate the storage requirements, particularly for high-resolution video (95% for QCIF, 98% for CIF). It is also clear from the table that less than 1 Mbyte is required when CIF resolution and three reference frames are used. Missing from the table is the memory required for the bit buffer that holds bits received from the bitstream. The size of the bit buffer depends on the implementation. In low-end systems, the size of the bit buffer might be only a few hundred bytes, while in high-end systems it could be as big as the number of bits needed for storing one or more uncoded video frames.

IV. H.264/AVC Baseline Decoder Time Complexity: Theoretical Analysis

As mentioned in Section II, H.264/AVC contains a number of optional coding features that can have a significant effect on decoder time complexity. The features used in our analysis represent the current understanding of the features of the baseline profile of H.264/AVC at the time our experiments were performed. This profile would most likely be used in the first generation of H.264/AVC-based video products, particularly those that require real-time encoding. This profile's feature set includes I- and P-pictures only, UVLC entropy coding, and up to five previous reference frames for motion-compensated prediction.

To develop both the theoretical and experimental results, we have generated bitstreams using two H.264/AVC baseline encoders with differing characteristics. The use of two different encoders helps to establish the amount of variation that might be experienced in decoder complexity as a function of the encoder's characteristics. The first encoder is the public JM 1.9 encoder, which performs an exhaustive search of all possible motion vectors in a ± 32 search range around the median predicted value and coding modes in order to achieve near-optimal rate-distortion performance. The second encoder is a real-time H.264/AVC encoder developed by UB Video Inc. This encoder differs from the JM encoder in that it implements suboptimal motion search and mode decision algorithms in order to reduce the encoder's time complexity and thereby enable real-time encoding. The UB Video encoder is highly optimized algorithmically, enabling it to achieve objective and subjective performance levels close to those of the JM encoder with significantly reduced time complexity [7].

To generate the bitstreams, we have used a set of three QCIF- and three CIF-resolution video sequences. The QCIF video sequences are Container and Foreman at 10 Hz and Silent Voice at 15 Hz. The CIF video sequences are Paris at 15 Hz, and Foreman and Mobile & Calendar at 30 Hz. Each video sequence was encoded by each of the two encoders at three different bit rates. The selected rates represent typical video data rates used in video conferencing systems. Neither of the two encoders employed a rate-control method, and as such the target rates were approximated by choosing, in each case, a constant quantization parameter that resulted in an average bit rate that is nearest to the target rate.

Estimating time complexity of a video decoder is a challenging and time-consuming task. Besides its dependence on factors such as the source video content, resolution and bit rate, this task depends on the intended use of the estimate. For hardware design, often a worst-case estimate is warranted, other times an average-case estimate is of interest, still other times the worst-likely-case estimate is considered. Next, we describe a methodology and provide tools in the form of data tables that together will be used to derive H.264/AVC baseline video decoder time complexity estimates over a variety of operational scenarios.

A. Table Descriptions

In this subsection, we describe the contents of the data tables that lie at the heart of the time complexity analysis methodology. As detailed later, the decoder subfunction, operation count, and execution subunit tables may be used to estimate the time complexity of the H.264/AVC baseline video decoder.

1) Decoder Subfunction Tables: The decoder subfunction tables contain data that describe the frequency of use of the decoder subfunctions (e.g., loop filter). Although we have generated several tables containing frequencies as a function of content, resolution, bit rate, and encoder type, we present a subset in Table II. Specifically, Table II contains frequencies corresponding to the CIF video sequences Mobile and Calendar encoded using the JM encoder with a fixed quantization parameter (QP) of 21, and Foreman encoded with a QP of 28, respectively.[4]

2) Operation Count Table: The operation count table contains a detailed description of the number of fundamental operations required to perform the key decoder subfunctions. Each type of operation in the table is listed in a separate column. To better understand how the operation count table was constructed, consider that in a given decoder implementation, there are two type of operations categories. One category includes implementation dependent operations that are strongly dependent on the hardware platform and varies widely with implementation. For example, on some hardware platforms, function calls are computationally expensive. Consequently, a good software implementation may replace frequently used calls with the source code of the referenced functions using a procedure called function in-lining. On other platforms, this strategy may be inappropriate. Other examples are operations related to loop overhead, arithmetic related to computing memory load, and store locations and special-case codes for boundary conditions. The other category includes those fundamental operations that every H.264/AVC compliant video decoder must perform. Examples of these fundamental operations include the computation of loop filter output values and 6-tap half-sample and bilinear quarter-sample interpolation values. The operation count table accounts for the operations in this second category only.

The operation count table[5] is quite large. For clarity, in Table III, we present the number of required basic operations for the three most compute intensive subfunctions of the decoder: 1) inverse transform and reconstruction (including inverse quantization and secondary DC transforms); 2) interpolation; and 3) loop filtering. These subfunctions account for 71% of the decoder's time complexity in the UB Video

[4]The complete set of decoder subfunction tables may be found [Online] at http://spmg.ece.ubc.ca/h264complexity.

[5]This can be obtained in its entirety [Online] at http://spmg.ece.ubc.ca/h264complexity.

TABLE II
SUBFUNCTION FREQUENCY OF USE (DATA SPECIFIED AS AVERAGE NUMBER OF SUBFUNCTION EXECUTIONS PER SECOND)

	Foreman, QP = 28	**Mobile, QP = 21**
Skipped macroblocks	4322.8	276.7
Inter-coded macroblocks	7222.3	11545.7
Intra-coded macroblocks	334.9	57.6
Inverse Quantization		
DC Luma Coefficients	69.8	33.3
AC Luma Coefficients	2785.7	67915.7
DC Chroma Coefficients	586.5	3631.5
AC Chroma Coefficients	33.2	7443.1
Transform and Reconstruction		
4x4 Luma Transforms	2591.8	28266.6
4x4 Chroma Transforms	1918	13899.1
4x4 Luma DC Transforms	37.0	7.2
2x2 Chroma DC Transforms	826.6	6972.0
Interpolation		
"b" 4x4 blocks	30669.4	37481.2
"c" 4x4 blocks	8713.8	6134.2
"d" + "e" 4x4 blocks	24141	59797.4
"f" + "g" 4x4 blocks	8945.4	22023.6
"h" 4x4 blocks	13634.2	13634.2
"i" 4x4 blocks	6307.2	6307.2
Loop Filter		
Edges with Non-zero Strength	343608.4	746964.8
Filter Type Decision (L+C)	322985.2	332858.8
Strong Luma Filter	5770.7	269.1
Strong Chroma Filter	3882	493.5
Standard Filter (P0 + Q0)	313332.5	332096.2
Filter P1 or Q1	304341	160411.8

encoder. Note that there is a near one-to-one correspondence between the rows in the operation count table and the rows in the subfunction tables to facilitate the process of combining data to generate a time complexity estimate.

3) Execution Subunit Table: The execution subunit table (Table IV) contains data that identifies the execution subunits capable of performing each of the basic operations found in the operation count table for four hardware platforms. It is apparent from the data in the table that all platforms have several execution subunits operating in parallel to enhance efficiency. The execution subunit table may be easily extended, allowing time complexity estimates to be derived for other hardware platforms.

B. Analysis Methodology

Here, we describe a method of using the tables described above to derive lower-bound time complexity estimates for H.264/AVC baseline decoder implementations on a variety of hardware platforms. The method consists of two basic steps. In the first step, we compute the number of cycles required to execute a particular subfunction on a chosen hardware platform. This intermediate result may be derived by mapping the subfunction operations onto the hardware, applying single-instruction multiple data (SIMD) parallelism and distributing the required operations as efficiently as possible among the execution subunits on the hardware. In the second step, the cycle count estimate is derived by multiplying the intermediate result from the first step by the frequency with which the subfunction was used. The subfunction frequency data is available in the subfunction tables (see Table II for an example).

We illustrate the time complexity estimation methodology in detail with an example in which the number of cycles required to execute the 4×4 inverse transform and reconstruction is estimated for the TriMedia TM-1300 for the CIF video sequence Mobile and Calendar encoded at 30 frames/s with a QP of 21. There are two cases to consider when analyzing the 4×4 inverse transform and reconstruction. The first case is when both the inverse transform and reconstruction are required, and the second is the reconstruction only case where no inverse transform is necessary (i.e., no nonzero coefficients were decoded). An estimate for the number of cycles required in each case is derived below. Finally, the estimates are then combined with the subfunction frequency data to yield the time complexity estimate.

TABLE III
SUMMARY OF THE SUBFUNCTION OPERATION COUNT TABLE

	Add8	Add16	Mult8	Mult16	MAC8	MAC16	Branch	Shift	Load	Store	AND	OR	? :
Inverse Quantization													
DC Luma Coefficients	0	Y_{DC}	0	$2Y_{DC}$	0	0	0	$2Y_{DC}$	0	Y_{DC}	0	0	0
AC Luma Coefficients	0	0	0	Y_{AC}	0	0	0	0	Y_{AC}	Y_{AC}	0	0	0
DC Chroma Coefficients	0	Cr_{DC}	0	Cr_{DC}	0	0	0	$2Cr_{DC}$	0	Cr_{DC}	0	0	0
AC Chroma Coefficients	0	0	0	Cr_{AC}	0	0	0	0	Cr_{AC}	Cr_{AC}	0	0	0
4x4 Inverse Transform + Reconstruction	0	96	0	0	0	0	0	32	48	16	0	0	0
4x4 Null Inv. Transform (reconstruct only)	0	0	0	0	0	0	0	0	16	16	0	0	0
4x4 Luma DC Transform	0	64	0	0	0	0	0	0	16	0	0	0	0
2x2 Chroma DC Transform (u & v)	0	16	0	0	0	0	0	0	8	0	0	0	0
Luma Interpolation (ops per sample)													
"b" (MAC support)	1	2	0	0	1	0	0	1	7	1	0	0	0
"b"	0	6	0	2	0	0	0	1	7	1	0	0	0
"c" (MAC support)	2	5	0	0	2	3	0	1	20	3	0	0	0
"c"	0	17	0	6	0	0	0	1	20	3	0	0	0
"d", "e" (MAC support)	2	2	0	0	1	0	0	2	9	2	0	0	0
"d", "e"	1	6	0	2	0	0	0	2	9	2	0	0	0
"f", "g" (MAC support)	3	6	0	0	2	3	0	3	24	5	0	0	0
"f", "g"	1	17	0	6	0	0	0	3	24	5	0	0	0
"h" (MAC support)	1	2	0	0	0	12	0	3	16	3	0	0	0
"h"	1	12	0	4	0	0	0	3	16	3	0	0	0
"i" - "funny position[5]"	4	0	0	0	0	0	0	1	2	2	0	0	0
Chroma Interpolation													
One Chroma Sample (MAC support)	0	1	0	0	0	2	0	1	4	1	0	0	0
One Chroma Sample	0	4	0	4	0	0	0	1	4	1	0	0	0
Deblocking Filter (ops per block edge)													
Strength Measure (inter-coded blocks)	0	5	0	0	0	0	1	8	11	1	4	3	7
Strength Measure (intra-coded blocks)	0	0	0	0	0	0	0	0	0	2	0	0	0
Edges With Non-Zero Strength	7	0	0	0	0	0	1.25	0.25	5.25	0	2.5	0	3
Filtered Edges													
Filter Type Decision (luma+chroma)	2	0	0	0	0	0	1	0	2	0	3	0	2
Strong Luma Filter (MAC support)	4	10	0	0	6	0	0	6	2	6	0	0	0
Strong Luma Filter	28	0	2	0	0	0	0	12	2	6	0	0	0
Strong Chroma Filter (MAC support)	0	8	0	0	4	0	0	4	2	4	0	0	0
Strong Chroma Filter	20	0	0	0	0	0	0	8	2	4	0	0	0
Standard Filter p0 + q0	8	0	0	0	0	0	0	2	2	2	0	0	2
Standard Filter p1, q1 (each)	3	0	0	0	0	0	0	2	0	1	0	0	2

[5] The method for computing the quarter-sample value at the funny position was changed in the draft standard subsequent to the analyses performed in this paper. The new value requires the same time complexity to compute as the quarter-sample values at the positions labeled "h".

TABLE IV
A MAPPING OF EXECUTION SUBUNITS TO ARITHMETIC OPERATIONS FOR SELECTED HARDWARE PLATFORMS

	Add8	Add16	Add32	Mult8	Mult16	MAC8	MAC16	Branch	Shift	Load	Store	OR	AND	? :
TriMedia	1,3	1,3	1,2,3,4,5	2,3	2,3	2,3	2,3	2,3,4	1,2	4,5	4,5	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5
Pentium III	1,2	1,2	1,2	N/A	1	N/A	1	2	2	3	4	1,2	1,2	1,2
TI C64x	1,2	1,2,5,6,7,8	1,2,5,6,7,8	3,4	3,4	N/A	N/A	5,6	3,4,5,6	7,8	7,8	1,2,5,6,7,8	1,2,5,6,7,8	1,2
Equator BSP-15[6]	2,4	2,4	1,2,3,4	2,4	2,4	2,4	2,4	1,3	2,4	1,3	1,3	1,2,3,4	1,2,3,4	1,2,3,4

[6] The BSP-15 has four functional units, an I-ALU and IFG-ALU for cluster 0 and I-ALU and IFG-ALU for cluster 1. In Table IV, the functional units are numbered 1, 2, 3, and 4, respectively.

Case 1: 4 × 4 Inverse Transform and Reconstruction: The operations used in the 4 × 4 inverse transform and reconstruction are mapped onto the TriMedia TM1300 hardware architecture using data extracted from the operation count (Table III) and execution subunit (Table IV) tables as shown in Table V. Next, the parallelism provided by the SIMD feature of the architecture

TABLE V
MAPPING OF 4×4 INVERSE TRANSFORM OPERATIONS ONTO THE TRIMEDIA TM-1300

Operation	Operation Count	Sub-units Capable of Performing Operation
Add	96	1, 2, 3, 4, 5
Shift	32	1, 2
Store	16	4, 5
Load	48	4, 5

is taken into account. For the inverse transform subfunction, no SIMD parallelism is exploited.[6] Therefore, the cycle count is estimated by distributing the required operations among the execution subunits capable of performing the operations so as to minimize the total number of cycles. More specifically, the minimum number of cycles C is computed as follows:

$$\begin{aligned} C = \max\Bigg[& \frac{(A+S+ST+L)}{5}, \frac{(A+S+ST)}{5} \\ & \frac{(A+S+L)}{5}, \frac{(A+L+ST)}{5}, \frac{(A+ST+L)}{4} \\ & \frac{(A+S)}{5}, \frac{(A+ST)}{5}, \frac{(A+L)}{5}, \frac{(S+ST)}{4} \\ & \frac{(S+L)}{4}, \frac{(ST+L)}{2}, \frac{A}{5}, \frac{S}{2}, \frac{ST}{2}, \frac{L}{2}\Bigg] \\ = & \frac{(A+S+ST+L)}{5} = 38.4 \text{ cycles} \end{aligned} \tag{1}$$

where A, S, ST, and L represent the number of adds, shifts, stores, and loads, respectively. Rounding to the next higher integer, the 4×4 inverse transform and reconstruction requires 39 cycles on the TriMedia TM-1300 architecture. For completeness, we show all operation combinations in (1).

Case 2: 4 × 4 Reconstruction Only: As for Case 1, the operations are mapped onto the hardware as shown in Table VI. Also in this case, SIMD parallelism is not applied. Therefore, the minimum number of cycles C is computed as follows:

$$C = \max\left[\frac{(ST+L)}{2}, \frac{ST}{2}, \frac{L}{2}\right] = \frac{(ST+L)}{2} = 16 \text{ cycles}.$$

Now that an estimate for the number of cycles required to execute the subfunction has been computed, the overall computational cost is derived by combining the cycle estimate with the subfunction frequency data provided in the subfunction table. More specifically, from Table II we see that an average of 42 165.7 (28266.6 luminance and 13 899.1 chrominance) 4×4 inverse transforms are required per second to decode Mobile and Calendar encoded by the JM encoder with a QP of 21. Using the cycle estimate derived in the first step, the computational cost due to the 4×4 inverse transform and reconstruction is $42\,165.7 * 39 = 1\,644\,462.3$ cycles/s. Likewise, noting that, on average, 242 954.3 blocks/s require reconstruction only, it follows that the computational burden for reconstruction only is $242\,954.3 * 16 = 3\,887\,268.8$ cycles/s. The value, 242 954.3 is derived using the fact that there are 396 macroblocks/frame $*$ 24 4×4 blocks/macroblock $*$ 30 frames/s $=$ 285 120 4×4 blocks/s. A transform was applied to 42 165.7 of the 4×4 blocks, consequently, 242 954.3 required reconstruction only. The computational cost estimate is computed by summing the cycle estimates for the two cases. That is, the estimated cost of the 4×4 inverse transform and reconstruction for Mobile at a QP of 21 is $1\,644\,462.3 + 3\,887\,268.8 = 5\,531\,731.1$ TM-1300 cycles/s.

[6]The TriMedia TM-1300 data bus is 32 bits wide. Consequently, it is possible to perform two 16-bit loads or four 8-bit stores per operation, thereby reducing the total operation count. Sometimes data must be reformatted using pack and merge instructions to exploit the SIMD parallelization of loads and stores other times reformatting is not necessary. Since parallelization of loads and stores is beneficial only sometimes, we have opted not to try to exploit this kind of parallelism to simplify our methodology.

TABLE VI
MAPPING OF 4×4 RECONSTRUCTION ONLY OPERATIONS ONTO THE TRIMEDIA TM-1300

Operation	Operation Count	Sub-units Capable of Performing Operation
Store	16	4, 5
Load	16	4, 5

Finally, we note that it is possible to compute alternative lower-bound time complexity estimates using variations on the method described above. For example, to derive a benchmark for the time complexity, it is necessary to identify the most computationally complex decoding mode (e.g., intra-coding) for a particular hardware platform using the operation count and execution subunit tables. Once identified, a lower bound for a benchmark time complexity estimate may be derived under the assumption that every macroblock is to be decoded in that mode.

V. H.264/AVC BASELINE DECODER TIME COMPLEXITY: COMPARATIVE ANALYSIS

In Section IV, two factors lead to the lower bound estimates, or more precisely, the under estimation of the time complexity. First, recall that the operation count table contains data for only those operations that are fundamental to the decoding process. Overhead operations such as loop overhead, flow control, and boundary condition handling are not included. Second, we assume that the software is designed so that the overhead due to instruction cache misses is negligible, hardware register counts are not exceeded and operation latency is hidden. Therefore, it is usual practice to multiply an estimate derived using similar assumptions by a factor of 2–4 (depending on platform and algorithm) to achieve a realistic time complexity target for highly optimized platform specific code. For optimized code that is not platform specific (i.e., "C" only code), factors of 3 to 5 are more typical. In fact, in this section we will show through two examples and through an analysis summary for the Pentium 3 (P3) that the theoretical estimates are approximately two to six times lower than the experimental results. Furthermore, our results show that the specific factor within this range for each subfunction depends mainly on the characteristics of the subfunction, such as the regularity of operations, amount of overhead, and memory-bandwidth requirements, and the use of platform-specific optimization (i.e., MMX code on the P3).

We will now revisit the example used in Section IV, except that we now develop estimates for the Pentium 3 platform. More

specifically, we will estimate the number of cycles required to execute the 4×4 inverse transformation and reconstruction for the P3 for the video sequence Mobile and Calendar encoded with a QP of 21. The same two cases are considered.

Case 1: 4 × 4 Inverse Transform and Reconstruction: The operations used in the 4×4 inverse transform and reconstruction are mapped onto the P3, as shown in Table VII. Next, the parallelism provided by the fact that the P3 has several execution units is taken into account. Therefore, the minimum cycle count C is computed as follows:

$$C = \max\left[\frac{(A+S+2*ST+L)}{3}, \frac{(A+S)}{2}, \frac{A}{2}, S, ST, L\right] = \frac{(A+S+2*ST+L)}{3} = 69.3 \text{ cycles}$$

where A, S, ST, and L represent the number of adds, shifts, stores, and loads, respectively. Therefore, the cost of the 4×4 inverse transform and reconstruction is 69.3 cycles on the P3.

Case 2: 4 × 4 Reconstruction Only: As in the first case, the operations are mapped onto the Pentium hardware as shown in Table VIII. Assuming SIMD parallelism is not exploited, for reasons described in an earlier example, C can be computed as follows:

$$C = \max\left[\frac{(2*ST+L)}{3}, ST, L\right] = 16 \text{ cycles.}$$

Next, from the subfunction table, we see that 42 165.7 total 4×4 transforms/s are required on average to decode Mobile and Calendar encoded with a QP of 21. Using the cycle estimate derived above, the computational cost due to the 4×4 inverse transform and reconstruction is $42\,165.7 * 69.3 = 2\,922\,083.01$ cycles/s. Similarly, noting that (30 frames/s $*$ 396 macroblocks per frame $*$ 24 4×4 blocks per macroblock—42 165.7) $=$ 242 954.3 4×4 luminance and chrominance blocks/s require reconstruction only to decode Mobile and Calendar encoded with a QP of 21, it follows that the computational cost for reconstruction only is $242\,954.3 * 16 = 3\,887\,268.8$ cycles/s. Therefore, the estimated cost of the 4×4 inverse transform and reconstruction for Mobile with a QP of 21 is $2\,922\,083.01 + 3\,887\,268.8 = 6\,809\,351.81$ Pentium 3 cycles/s.

If we continue with this example to also include other components of the reconstruction process, such as inverse quantization of coefficients and the second-level transforms that apply to 16×16 intra-predicted macroblocks and chrominance macroblocks, we arrive at a theoretical result of 6 974 588 Pentium-3 cycles/s for the entire reconstruction process for Mobile and Calendar at QP of 21.

To see how the above result compares with empirical data, we generated profiling results with Intel's VTune Performance Analyzer. This software analysis tool allows a user to collect run-time data indicating the number of cycles consumed by each section or function of a piece of software. For our analysis on a 600-MHz P3, we first note that 600 million cycles/s $*$ (30 frames/s)/(65.6 frames/s) $= 274.39$ million cycles/s are needed to decode 1 s of Mobile and Calendar with QP $= 21$. The VTune analysis shows that 10.29% of the decoder computations are used for transforms and reconstruction for this sequence,

TABLE VII
MAPPING OF 4 × 4 INVERSE TRANSFORM OPERATIONS ONTO THE P3

Operation	Operation Count	Sub-units Capable of Performing Operation
Add	96	1, 2
Shift	32	2
Store	16	4
Load	48	3

TABLE VIII
MAPPING OF 4 × 4 RECONSTRUCTION ONLY OPERATIONS ONTO THE P3

Operation	Operation Count	Sub-units Capable of Performing Operation
Store	16	4
Load	16	3

and 10.29% of 274.39 million cycles/s gives 28.23 million cycles/s for this process. Therefore, the ratio of measured to theoretical performance results is (28.23 million)/(6 974 588) $=$ 4.05.

Following the approach described above, we arrive at a theoretical estimate of 4 815 346 P3 cycles/s to decode Foreman encoded with a QP of 28. The VTune results show that inverse transforms and reconstruction account for 10.6% of the decoder time complexity, or 19.5 million cycles/s. This is a factor of 4.05 larger than the theoretically derived result.

The above example indicates that, as expected, the theoretical estimates are approximately 4:1 lower than the experimental results. On average, across all sources and data rates, we found the theoretical estimates to be 3.6 times lower than the measured values, as shown in Table IX. The same table, which displays the (theoretical/experimental) estimate ratios for several sequences, resolutions, and bit rates, also demonstrates that our theoretical analysis can be used effectively to predict accurately the time complexity of a H.264/AVC baseline decoder implementation on a specific general-purpose platform.

From Table IX, we can also observe that for each subfunction, the ratio of theoretical to measured complexity is remarkably similar from one sequence to another despite significant differences in source format, content and data rates. However, these ratios do vary significantly between the different subfunctions. These variations can be explained by the differing characteristics of the subfunctions in terms of the amount of complex logic and overhead incurred in the implementation of each subfunction, as well as the level of optimization in the implementation of each subfunction (e.g., the use of processor-specific code).

Notably, the luminance interpolation subfunction yields the lowest ratio of theoretical to measured cycles (that is, our estimate is most accurate on this subfunction), with an average of 1.8:1 across the test set. This subfunction requires minimal logic and overhead. Also, the implementation of this subfunction includes extensive processor-specific optimization, since the simple functions are more easily implemented using MMX code.

On the other hand, the subfunction with the highest ratio is the chrominance interpolation, with an average of 5.8:1. There

TABLE IX
EXPERIMENTAL (VTUNE)/THEORETICAL ESTIMATE RATIOS FOR THE P3

	Container (QCIF, QP=15)	Foreman (QCIF, QP=24)	Silent (QCIF, QP=22)	Paris (CIF, QP=17)	Foreman (CIF, QP=28)	Mobile (CIF, QP=21)
Loop filtering	2.57	2.36	2.71	2.85	2.58	2.84
Reconstruction	3.90	3.79	3.94	4.57	4.05	4.05
Luma interpolation	2.25	1.45	1.87	2.14	1.62	1.72
Chroma interpolation	6.38	4.94	6.12	6.96	5.42	5.81

are a few factors that contribute to a larger amount of overhead and more difficult optimization of this subfunction compared to the luminance interpolation. One important difference is that for chrominance interpolation of each block of data, filter-tap coefficients are computed linearly based on the phase of the interpolation, rather than using fixed filter-tap values for each phase, as is done for luminance. Also, the size of each block of samples that are interpolated at once is smaller by a factor of 4 (two in each direction) for chrominance, again, leading to a larger percentage of overhead. Finally, our implementation of the chrominance interpolation subfunction does not include any processor-specific optimization. These differences lead to a much larger number of cycles being consumed by this subfunction than our theoretical analysis indicates.

VI. H.264/AVC BASELINE DECODER TIME COMPLEXITY: EXPERIMENTAL ANALYSIS

Analysis of the run-time profiling data of decoder subfunctions generated by VTune over the test set used in our experiments leads to a number of interesting observations regarding the time complexity of an H.264/AVC baseline video decoder. In particular, the distribution of time complexity amongst decoder subfunctions and the effect that certain key factors, such as the source content, resolution, bit rate and encoder characteristics can have on the complexity of each subfunction and the decoder as a whole will be discussed. Our experimental analysis shows that the time complexity of the decoder and its major subfunctions are strongly dependent upon the average bit rate of the coded bitstream. Moreover, our results illustrate that the characteristics of the source content can also have a significant effect on decoder complexity, although such characteristics are more difficult to quantify.

However, one significant observation from our experiments is that the optimality of the motion estimation and mode decision processes in the source encoder do not have a significant impact on decoder complexity. This result is based on performance measurements from the two encoders that were used in this study—the rate-distortion optimized public reference software and UB Video's computationally optimized implementation. Both encoders, which used the same picture structure (i.e., IPPP…), produced similar rate-distortion performance, with the reference software providing approximately 0.5 dB improved PSNR. Of course, it is possible that another compliant encoder could produce bitstreams with more widely varying characteristics than those produced by the two encoders used in our experiments, and thus produce larger differences in decoder complexity than those observed here. However, we believe that the JM and UB Video encoders produce bitstreams that provide a reasonable representation of those that would be produced in future products by off-line and real-time H.264/AVC baseline video encoders, respectively. Further, certain encoder-specific decisions, such as frequent intra-coded macroblock refresh, would certainly have an impact on the complexity of specific decoder subfunctions, and consequently the overall decoder complexity, but the effects of such decisions were not tested in our experiments.

For each of the most computationally intensive subfunctions of the decoder, we present an analysis of the factors that affect their complexity, as measured in our experiments. Since the choice of encoder did not have a significant impact on the decoder complexity as measured in our experiments, this dimension will be excluded from the discussion. All of the analysis is based on VTune profile data generated using the UB Video codec.

One of the most important pieces of information in the complexity analysis of a system is the distribution of time complexity amongst its major subsystems. To this end, we have generated results that have been averaged over all sequences in the test set. Loop filtering (33%) and interpolation (25%) are the largest components, followed by bitstream parsing and entropy decoding (13%), and inverse transforms and reconstruction (13%). In the following subsections, we discuss the factors that can affect the complexity of each of these important subfunctions, as observed in our experiments.

A. Loop Filtering

The operation of the loop filter, which is one of the most complex parts of the decoder, can be separated into two subfunctions: the computation of the "boundary strength" parameter, Bs, for each 4-sample edge segments and the content-dependent filtering process. The purpose of the boundary strength computation is to determine whether a block artifact may have been introduced over an edge, and thus determine the strength of the filter to be used on the edge segment, based on the parameters used in coding its bounding blocks. These parameters include the prediction mode (inter/intra), existence of nonzero residual transform coefficients, and differences in motion vectors across the boundary. Since the computation of the edge strength is generally performed in the same way for every macroblock, the time required for this operation remains relatively constant per macroblock over various types of content and bit rates. However, at lower bit rates, and for nonactive content, our experimental results show that the complexity of this operation can be reduced slightly in some implementations. This reduction can occur because the strength computation can be simplified in the presence of skipped macroblocks, which occur more frequently in these cases. The boundary strength computation can also be

simplified for intra-coded macroblocks, however, the amount of intra-coding is limited in our experiments, and so this is not a significant factor.

Unlike the strength computation, the complexity of the filtering process varies significantly with content and bit rate. The most important factor in this variability is the percentage of edges that might be filtered due to the boundary strength determined for that edge. Edges of blocks that are copied directly from a previous frame without having a difference in motion vectors or nonzero residual data are not filtered ($\mathrm{Bs} = 0$), since new blocking artifacts are not produced by such coding modes. Therefore, nonactive sequences and those coded at low bit rates that generally contain a high percentage of skipped and zero-valued residual blocks contain fewer edges that require filtering. We observed in our experiments that the number of blocks with nonzero boundary strength and, consequently, the cycles consumed by the filtering operations, increase as the bit rate is increased. The increase is primarily due to a larger number of blocks containing nonzero coefficients. Note however that at very high bit rates, when the quantization parameter falls below a particular threshold value that was not surpassed in any of our experiments, the de-blocking filter is turned off regardless of boundary strength.

For all edges with $\mathrm{Bs} > 0$, a local activity check is performed before performing each line-based filtering operation. The purpose of the check is to prevent over-filtering of natural edges. Depending on the architecture and implementation, the percentage of the activity checks that are passed may also affect the complexity of the filtering process. Specifically, for hardware and software implementations for highly parallel and deeply pipelined architectures such as DSP/media processors, no computational savings will be observed if the activity checks result in frequent early exits from filtering. This is because efficient implementations for such platforms would likely perform all of the filtering operations and then conditionally store the filtered output based on the result of the content-dependent checks. However, on architectures such as the Pentium processor used in our experiments, the cost of branching is not so high as to outweigh the cost of the filtering operations that can be skipped, so the complexity of the filter varies both with the number of edges with $\mathrm{Bs} > 0$, and with the number of edges that pass the activity checks. Thus, content that is high in spatial detail and contrast, which results in many early exits at the activity checks, was observed in our experiments to require fewer filtering operations, and therefore fewer cycles in the decoder's loop filter function.

It should be mentioned that, while our experiments and the above discussion address average time complexity measured over a number of different sequences, worst-case complexity is generally the key design parameter used when constructing a decoder. Since real-time decode and display capability is a requirement, the decoder must posses the ability to perform all operations required for decoding the most complex compliant video bitstream within a fixed frame interval. Thus, in the case of the de-blocking filter, an implementer would have to ensure that the decoding hardware can handle, in a fixed amount of time, a worst-case bitstream representing a video frame in which all edges have nonzero block strengths (as they would in an intra-coded frame) and the content activity checks are always passed (as might happen in a smooth frame).

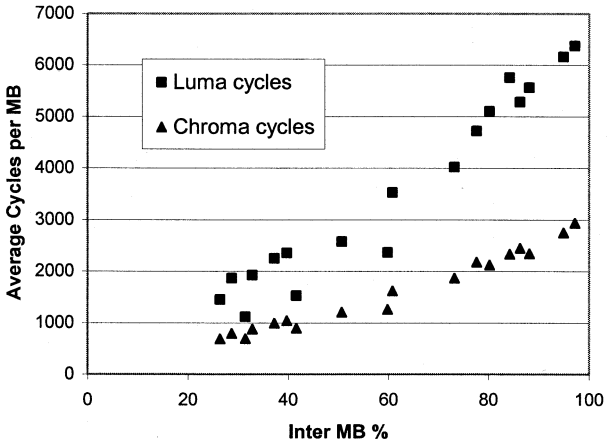


Fig. 3. Interpolation complexity as a function of inter-macroblock frequency.

B. Interpolation

Interpolation of reference samples to generate a motion-compensated prediction is generally performed for each macroblock that is inter-coded (i.e., not skipped or intra-coded). Thus, we expect the average time required by the decoder for interpolation to be a function of the number of inter-coded macroblocks. Our experimental results meet this expectation, as illustrated in Fig. 3, which shows plots of the percentage of inter-coded macroblocks as a function of the average number of cycles per macroblock spent performing interpolation of luminance and chrominance samples for each case in the test set. The relationship is approximately linear. Inter-coded macroblocks are used most frequently when the content exhibits temporal redundancy that fits the H.264/AVC motion model, making intra-coding or skipping less suitable as determined in our experiments with the fixed quantization parameter.

We can also observe from Fig. 3 that the time complexity of the chrominance interpolation is just less than half that of the luminance interpolation. This approximate relationship is expected, since there are half as many chrominance samples as there are luminance sample in the input data. While fewer basic operations are required per sample for the chrominance interpolation, there is also a larger proportion of setup and looping overhead for chrominance, since the basis for interpolation is subsampled by a factor of 2 in each direction. Our results show that these two factors roughly balance each other out in our implementation, resulting in approximately the same computational time per sample for luminance as for chrominance.

C. Inverse Quantization, Transforms, and Reconstruction

The complexity of the inverse transform and reconstruction is directly affected by the number of blocks and macroblocks that contain nonzero coefficients. The macroblocks that are skipped or inter-coded with no coefficients require only simple and regular copying of data from the reference frame or predicted macroblock to the current frame. However, inverse transforms, addition of the coded residual and clipping must be performed to reconstruct macroblocks with nonzero coefficients. Thus, the

TABLE X
DECODING SPEED RATIOS FOR H.263 BASELINE AND H.264 BASELINE MEASURED ON THE 600-MHZ PENTIUM 3

Sequence	H.263 Baseline			H.264/AVC Baseline			
	QP	Bit Rate (Kbps)	Decoding Speed (fps)	QP	Bit Rate (Kbps)	Decoding Speed (fps)	H.263/H.264 Speed Ratio
Container, QCIF, 10 Hz	6	61	710	11	65	300	2.37
	9	30	900	15	32	390	2.31
Foreman, QCIF, 10 Hz	13	62	580	18	66	230	2.52
	22	34	670	24	32	280	2.39
Silent, QCIF, 15 Hz	9	61	750	16	64	330	2.27
	15	32	880	22	29	390	2.26
Paris, CIF, 15 Hz	5	672	140	11	677	67	2.09
	10	298	183	17	313	82	2.23
	21	111	220	24	116	100	2.20
Foreman, CIF, 30 Hz	9	652	149	14	680	56	2.66
	14	342	192	19	326	67	2.87
	31	149	225	28	120	85	2.65
Mobile & Calendar, CIF 30 Hz	12	1842	96	16	1723	44	2.18
	23	699	130	21	704	55	2.36
	31	443	155	26	307	65	2.38
AVERAGE							**2.4**

number of inverse transforms is a key factor in determining the complexity of this decoder subfunction. Again, the bit rate is the most important factor that determines the number of nonzero blocks that require inverse transforms, with larger numbers occurring at higher bit rates.

D. Spatial Macroblock Prediction

The time complexity of spatial macroblock prediction (intra-prediction) is not a large factor in our tests, due to the relatively small percentage of intra-coded macroblocks in our test set. Intra-coding is infrequent because the generated bitstreams only used a single intra-coded picture per sequence, there was no forced intra-coding refresh, and there were no scene changes in the source content. However, although it is not addressed by our results, the complexity of intra-prediction can still be a major factor in certain video applications. The amount of intra-coding and its effect on complexity are largely dependent on the application (i.e., how much intra-coding it requires, either for the purpose of random access or error resilience). Of course, the content and the bit rate also affect the amount of intra-coding in a bitstream.

E. Bitstream Parsing and Entropy Decoding

Bitstream parsing and entropy decoding time varies directly with the bit rate, as one would expect. More time is spent with coefficients at higher bit rates, since they occupy a larger percentage of the bitstream, while more time is spent with motion vectors at the lower bit rates.

VII. TIME COMPLEXITY: H.264/AVC BASELINE VERSUS H.263 BASELINE

The baseline profile of ITU-T Recommendation H.263 is the most widely implemented standard for use in real-time interactive video communications. Since the decoder cost is a critical factor in the cost effectiveness of deploying an H.264/AVC-based video solution, a comparison of the decoding performance of these two standards is valuable. As discussed previously in this paper, the storage requirements of an H.264/AVC baseline decoder are dominated by frame buffers, the number of which is determined by the number of reference frames that are supported. In this section, we describe an experiment that we have performed to estimate the time complexity of decoding an H.264/AVC baseline compliant bitstream compared to decoding a bitstream that is compliant with the H.263 baseline standard.

Our comparison is based on two similarly optimized "C" only decoder implementations running on a 600-MHz P3 PC. The encoders also used similar motion estimation and mode decision processes while generating bitstreams compliant with the respective standards. The H.264/AVC baseline encoder used the same feature set as in our previous experiments, as described in Section IV. Several QCIF- and CIF-resolution sequences were encoded by both encoders at a range of bit rates, using different constant quantization parameters to control the bit rate. The decoding speed was measured for each coded sequence and bit rate. These results, along with the speed ratio between the two decoders for each sequence and matching bit rate, are presented in Table X.

For all nine coded bitstreams, the ratio of decoding speed (excluding overhead such as display and file access time) between H.263 baseline and H.264/AVC baseline lies in the range from 2.1 to 2.9, with an average of 2.4. Therefore, our conclusion is that the time complexity of H.264/AVC baseline decoding is two to three times that of H.263 baseline decoding for a given sequence encoded at a given bit rate. We also note that at the same bit rate, H.264/AVC baseline has a significant quality advantage, and if the goal in an application was to achieve similar visual quality, the H.264/AVC baseline bit rate could be 35%–50% lower than that of the H.263 baseline 0. Since lowering the bit rate reduces the decoding complexity, as discussed in the previous section, the ratio of complexity between H.263 and H.264/AVC would also be reduced in this scenario.

VIII. Conclusions

In this paper, we study the computational complexity of the H.264/AVC baseline decoder using both theoretical and experimental methods. We derive our theoretical estimates using bitstreams generated from two different encoders for different content, resolution and bit rates. Together with hardware-specific cycle counts for each basic operation (e.g., add, shift) as well as measured numbers of computational operations for each subfunction (e.g., interpolation), the resulting frequencies can be used to estimate the H.264/AVC baseline decoder time complexity for various hardware platforms. The resulting estimates are on average 3.6 times lower than measured values, and this is expected since we do not account for loop overhead, flow control, and boundary-condition handling, as well as instruction and data cache misses and operation latency in our theoretical analysis. We also study the behavior of the H.264/AVC baseline decoder as a function of encoder characteristics, content, resolution, and bit rate. Moreover, we demonstrate, through the use of similarly optimized decoders on the P3 platform, that the H.264/AVC baseline decoder is two to three times higher in terms of time complexity than an H.263 baseline decoder. Finally, we note that, at the time of the writing of this paper, H.264/AVC was still an evolving standard, and the final baseline decoder may differ from the decoder analyzed in this paper.

References

[1] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 \ ISO/IEC 14 496-10 AVC) Joint Video Team (JVT), Mar. 2003, Doc. JVT-G050.

[2] A. Joch, F. Kossentini, H. Schwarz, T. Wiegand, and G. Sullivan, "Performance comparison of video coding standards using lagrangian coder control," in *Proc. Int. Conf. Image Processing (ICIP)*, vol. 2, Rochester, NY, Oct. 2002, pp. 501–504.

[3] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 688–703, July 2003.

[4] *Working Draft Number 2, Revision 2 (WD-2) Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, T. Wiegand, Ed., Feb. 2002, JVT-B118r2.

[5] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, July 2003.

[6] G. Bjøntegaard and K. Lillevold, "Context-adaptive VLC (CVLC) coding of coefficients," in 3rd Meeting of the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Fairfax, VA, May 6–10, 2002, Doc. JVT-C028rl.

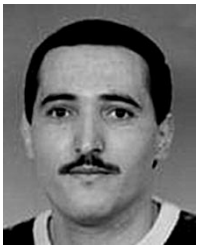
[7] A. Joch, J. In, and F. Kossentini, "Demonstration of FCD-conformant baseline real-time codec," in 5th Meeting of the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Geneva, Switzerland, Oct. 9–17, 2002, Doc. JVT-E136r1.

Michael Horowitz received the A.B. degree (with distinction) in physics from Cornell University, Ithaca, NY, in 1986 and the Ph.D. degree in electrical engineering from The University of Michigan, Ann Arbor, in 1998.

In 1998, he joined Polycom, Austin, TX, where he invents and develops video algorithms for video conferencing products. His recent work includes perceptual video coding, video error concealment for transmission of compressed video over lossy channels and the development of a low-latency H.264/MPEG-4 AVC baseline profile codec for video conferencing. He has served as ad-hoc group chair for H.264 complexity reduction in both the ITU-T Video Coding Experts Group and the ITU-T/ISO/IEC Joint Video Team and regularly contributes to those standards committees.

Anthony Joch received the B.Eng. degree in computer engineering from McMaster University, Hamilton, ON, Canada, in 1999, and the M.A.Sc. degree in electrical engineering from the University of British Columbia, Vancouver, BC, Canada, in 2002.

In 2000, he joined UB Video Inc., Vancouver, BC, where he is currently a Senior Engineer involved in the development of software codecs for the H.264/MPEG-4 AVC standard. His research interests include reduced-complexity algorithms for video encoding, video pre- and post-processing, and multimedia systems. He has been an active contributor to the H.264/MPEG-4 AVC standardization effort, particularly in the area of deblocking filtering and as a co-chair of the *ad-hoc* group for bitstream exchange.



Faouzi Kossentini (S'90–M'95–SM'98) received the B.S., M.S., and Ph.D. degrees from the Georgia Institute of Technology, Atlanta, in 1989, 1990, and 1994, respectively.

He is presently the President and Chief Executive Officer of UB Video, Vancouver, BC, Canada. He is also an Associate Professor in the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada, performing research in the areas of signal processing, communications, and multimedia. He has co-authored over 150 journal papers, conference papers, and book chapters. He has also participated in numerous international ISO and ITU-T activities involving the standardization of coded representation of audiovisual information.

Dr. Kossentini was an Associate Editor for the IEEE Transactions on Image Processing and the IEEE Transactions on Multimedia and a Vice General Chair for ICIP-2000.

Antti Hallapuro was born in Vimpeli, Finland, in 1975. He is currently working toward the M.Sc. degree at Tampere University of Technology, Tampere, Finland.

He joined Nokia Research Center in 1998. At Nokia, he works in the Advanced Video Coding group, where his main concern is real-time implementation of video coding algorithms on various processor architectures. He has participated in the H.264/MPEG-4 AVC video codec standardization effort since 2001. He is author and co-author of several technical contributions submitted to the standardization group.