14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

FAST RENORMALIZATION FOR H.264/MPEG4-AVC ARITHMETIC CODING

Detlev Marpe, Heiner Kirchhoffer, and Gunnar Marten

Fraunhofer Institute for Telecommunications – Heinrich Hertz Institute, Image Processing Department
Einsteinufer 37, D-10587 Berlin, Germany
phone: +493031002-619, fax: +49303927200, email: [marpe|kirchhoffer|marten]@hhi.fraunhofer.de
web: iphome.hhi.de

ABSTRACT

We propose a fast, standard-compliant realization of the computationally expensive renormalization part of the binary arithmetic coder in H.264/MPEG4-AVC. Our technique allows to replace time-consuming, bitwise-operating input and output procedures as well as bitwise carry-over handling in a conventional implementation with corresponding operations in units of multiple bits. Experimental results demonstrate that the proposed method enables a considerable speed-up of both arithmetic encoding and decoding in the range of 24–53% of their corresponding run time.

1. INTRODUCTION

In the course of the standardization of H.264/MPEG4-AVC [1], a novel design of a family of table-based adaptive binary arithmetic coders has been developed [2]. This so-called M coder design [3] involves the innovative features of a table-based interval subdivision in conjunction with a fast and accurate table-based probability estimator as well as a fast bypass coding mode. The computationally critical operation of interval subdivision is approximated by using a pre-quantization of the range of possible interval width values induced by renormalization. For each quantized interval width and for each representative probability value, the corresponding product value is pre-calculated and stored with suitable precision into a 2-D lookup table. Probability estimation is performed by employing a finite-state machine with tabulated transition rules. For approximately uniform distributed sub-sources, an optional bypass of the probability estimator is employed, which results in an additional speed-up [3][4].

A member of the M-coder family has been adopted as normative element of the H.264/MPEG4-AVC context-based adaptive binary arithmetic coding (CABAC) scheme [4]. This specific M-coder incarnation in H.264/MPEG4-AVC provides virtually the same coding efficiency as a conventional multiplication- and division-based implementation of binary arithmetic coding at significantly higher throughput rates, corresponding to speed-up factors in the range of 1.5–2.0 [2]. Compared to other well-established low-complexity binary arithmetic coding techniques like that of the Q coder [5] and its derivatives of QM and MQ coder [6], the M coder achieves an increase in throughput of up to 18%, depending on the implementation platform. At the same time, average bit rate savings of 2–4% can be obtained by the M coder relative to the MQ coder, when measured in the native H.264/MPEG4-AVC CABAC environment [3].

Despite these remarkable properties, arithmetic coding in H.264/MPEG4-AVC still poses some severe problems for real-time applications. Due to its sequential nature of processing, the computational requirements for real-time software-based parsing and arithmetic decoding of high-definition video at, e.g., bit rates of 8–12 Mbits/sec, may yet exceed the capabilities of today's generic CPUs. In the view of these challenges, it is obvious that any progress in substantially reducing the implementation costs of the binary coding engine of H.264/MPEG4-AVC will be extremely beneficial.

One of the major bottlenecks in any arithmetic encoding and decoding process is given by the renormalization procedure. Renormalization in the M coder is required whenever the new interval range R after interval subdivision does no longer stay within its legal range. Each time a renormalization operation must be carried out one or more bits can be output at the encoder or, equivalently, have to be read by the decoder. This process, as it is currently specified in the standard [1], is performed bit-by-bit, and it is controlled by some conditional branches to check each time if further renormalization loops are required. Both conditional branching and bitwise processing, however, constitute considerable obstacles to a sufficiently high throughput.

As a solution to this problem, we propose a fast renormalization policy for the M coder with the following main characteristics:

- The loop in the renormalization part is completely removed and conditional branching is omitted as far as possible.
- The internal register representing the code interval base L in the encoder or alternatively, the register for the offset V in the decoder is implemented with a higher accuracy in order to allow writing/reading of multiple code bits at a time.
- The carry-over handling in the encoder is substantially simplified in a way that the demand for storage and computational resources can be greatly reduced.
- A virtual floating point is maintained for the registers L and V to always guarantee the required precision of the corresponding variables relative to the code interval width R.
- All proposed changes as applied to the H.264/MPEG4-AVC version of the M coder are fully standard compliant.

2. BACKGROUND AND PROBLEM STATEMENT

In the following, we first present a brief review of the basic principles of binary arithmetic coding (BAC) with a particular focus on implementation-related aspects. In binary arithmetic coding, it is convenient to discriminate between the two symbols of the binary alphabet not by using their actual symbol values "0" and "1" but rather by referring to their estimated probability values. By distinguishing between the *least probable symbol* (LPS) and the *most probable symbol* (MPS) and by keeping track of the symbol value of the MPS (valMPS) as well as the probability p_{LPS} of the LPS, a simple parameterization of the underlying probability model of a given binary alphabet is achieved.

Based on those settings, BAC is performed by subdividing an initially given interval represented by its lower bound (base) L and its width (range) R into two disjoint subintervals: one interval of width $R_{\text{LPS}} = R \cdot p_{\text{LPS}}$, which is associated with the LPS, and the dual interval of width $R_{\text{MPS}} = R - R_{\text{LPS}}$ which is assigned to the MPS. Depending on the binary value to encode, either identified as LPS or MPS, the corresponding subinterval is then chosen as the new coding interval. By recursively applying this interval-subdivision scheme to each element a_j of a given sequence $\mathbf{a} = (a_1, a_2, \ldots, a_N)$ of binary symbols to encode, BAC finally determines a value $c_{\mathbf{a}}$ in the subinterval $[L^{(N)}, L^{(N)} + R^{(N)})$ that results after the N^{th} interval subdivision process. The (minimum) binary representation of $c_{\mathbf{a}}$ is the arithmetic codeword of the input sequence **a**.

To ensure that registers with a precision of b bits are sufficient to represent the variables $R^{(j)}$ and $L^{(j)}$ for all j, a renormalization operation is required, whenever $R^{(j)}$ falls below a certain limit after one or more interval subdivision process(es). Furthermore, by renormalizing $R^{(j)}$ and $L^{(j)}$ accordingly, leading bits of the arithmetic codeword $c_{\mathbf{a}}$ can be output as soon as they are unambiguously identified.

Fig. 1 shows pseudocode of a BAC implementation including the renormalization part (lines no. 8−19). This implementation is based on [7], which serves as the common technical ground for all the variants discussed in the rest of the paper. As can be seen from Fig. 1, this BAC scheme uses the convention to place the LPS-related subinterval on top of the subinterval assigned to the MPS by modifying the interval base according to $L^{(j)} = L^{(j-1)} + R_{\text{MPS}}$. Renormalization is triggered whenever the constraint $R \geq 2^{b-2}$ is violated. As shown in Fig. 1, the corresponding renormalization part contains a loop with a stepwise doubling of the register values R and L, whereby in each step a single bit is emitted. However, in cases where a future carry bit may affect the value of the current bit, a counter (Fig. 1: `BitsOutstanding`) is incremented and the actual output of that bit is delayed until the carry can be resolved.

The most critical drawback in terms of computational complexity, when using a straightforward BAC implementation like that shown in Fig. 1, is given by the multiplication operation in line no. 1 of the BAC routine. Actually, if the probability estimation involves a simple estimator based on

```
// interval subdivision
1:  R_LPS = R × p_LPS
2:  R_MPS = R - R_LPS
3:  if (value == valMPS)
4:     R = R_MPS
5:  else
6:     L = L + R_MPS
7:     R = R_LPS

// renormalization
8:  while (R < 2^(b-2))
9:     if (L >= 2^(b-1))
10:       put_one_bit_plus_outstanding(1)
11:       L = L - 2^(b-1)
12:    else
13:       if (L < 2^(b-2))
14:          put_one_bit_plus_outstanding(0)
15:       else
16:          L = L - 2^(b-2)
17:          BitsOutstanding++
18:    R = R << 1
19:    L = L << 1
```

Fig. 1 – Binary arithmetic encoding.

scaled cumulative frequency counts of symbols, this operation may even involve an integer division [7]. As a consequence, most of the research on fast binary arithmetic coding has been devoted to the problem of employing a suitable low-complexity operation as an approximation of the operation(s) required to perform the interval subdivision. The most prominent representatives of that kind of BAC schemes are given by the Q, QM and MQ coder as part of JPEG, JBIG, JPEG-LS, and JPEG2000 image coding standards [5][6].

Recently, a new design of a family of multiplication-free binary arithmetic coders has been proposed [2][3]. Its main innovative features are given by a table-based interval subdivision coupled with probability estimation based on a finite-state machine (FSM) as well as a fast bypass coding mode. This so-called *modulo* (*M*) *coder* family of BAC schemes offers a parameterizable trade-off between coding efficiency and memory requirements for the underlying lookup tables. Actually, the M-coder design can be considered as a generalization of the Q-coder family[1], since the latter can be derived from a specific M-coder incarnation belonging to the simplest choice of parameter (see below).

Another, more elaborate choice of a member of the M-coder family has been adopted by the ITU-T and ISO/IEC as a normative part of the H.264/MPEG4-AVC video coding standard [1]. It offers a good trade-off between complexity (in terms of throughput) and compression performance, as experimentally verified in [3]. In the following section, we briefly summarize some basic facts of the M coder.

2.1 Brief review of the M-coder design principles

The basic idea of the low-complexity M-coder approach of interval subdivision is to quantize the admissible domain $\mathcal{D} = [2^{b-2}, 2^{b-1})$ for the range register R induced by renormalization into a small number of K different cells. To further simplify matters, we assume a uniform quantization of $\mathcal{D}$ to be applied, resulting in a set of representative equi-

[1] This is strictly true only with regard to the way the interval subdivision is approximated.

spaced range values $\mathcal{Q} = \{Q_0, Q_1, \ldots, Q_{K-1}\}$, where K is further constrained to be a power of 2, i.e., $K = 2^{\kappa}$ for a given integer $\kappa \geq 0$. By a suitable discretisation of the range of LPS-related probability values $p_{\mathrm{LPS}} \in (0, 0.5]$, a representative set $\mathcal{P} = \{p_0, p_1, \ldots, p_{M-1}\}$ of probabilities can be constructed together with a set of corresponding transition rules for FSM-based probability estimation. Both discrete sets $\mathcal{P}$ and $\mathcal{Q}$ together enable an approximation of the multiplication operation $p_{\mathrm{LPS}} \times R$ for interval subdivision by means of a 2-D table *RTAB* that contains all $M \times K$ pre-calculated product values $\{p_m \times Q_k \mid 0 \leq m < M; 0 \leq k < K\}$ in a suitably chosen integer precision. The entries of the *RTAB* table can be easily addressed by using the (probability) state index m and the quantization cell index k related to the given value of R. Computation of k is easily carried out by a concatenation of a bit-shift and a bit-masking operation applied to R, where the latter can be interpreted as a *modulo operation* using the operand $K = 2^{\kappa}$, hence the naming of the proposed family of coders:

$$k = (R >> (b - 2 - \kappa))\ \&\ (2^{\kappa} - 1). \tag{1}$$

Please note that for a specific realization of the M coder, κ and b are fixed, and therefore both operands on the right hand side of (1) are given as fixed values. By choosing a value of $\kappa = 0$, the 2-D table *RTAB* degenerates to a linear table, where for all possible values of R only one single representative value is used for the approximation of $p_{\mathrm{m}} \times R$. This case is equivalent to the subinterval division operation performed in the Q coder and its corresponding derivatives.

However, for clarity of presentation and without loss of generality, we will restrict ourselves in the following to the specific case of an H.264/MPEG4-AVC compliant M coder corresponding to the choice of $\kappa = 2$ and the specification of a lookup table *RTAB* with 64×4 entries [1]. As a further simplification, we will neglect the table lookup operations required to adapt the probability state m during each encoding/decoding cycle. For more details, especially on the latter aspect, please refer to [3][4].

2.2 Discussion of renormalization

In terms of implementation costs, the renormalization part of the M coder still suffers from bit-by-bit input/output and – as far as the encoder side concerns – also from bitwise carry-over handling. The related computationally critical parts in an encoder implementation can be mainly attributed to the bitwise operating renormalization loop and the conditional branching inside this loop as shown in Fig. 1.

Although from a decoder perspective, the problem appears to be slightly alleviated when comparing the renormalization parts of Fig. 1 and Fig. 2, there is still a considerable computational overhead involved in a conventional M-decoder implementation due to its sequential reading of bits from the bitstream (as exemplified in line no. 11 of Fig. 2).

The following section presents an alternative but still standard-compliant realization of renormalization by enabling the processing of multiple bits at a time both for the output at the encoder and the input at the decoder.

```
// interval subdivision
1:  R_LPS = RTAB[m][(R >> 6) & 3]
2:  R_MPS = R - R_LPS
3:  if (V < R_MPS)
4:      R = R_MPS
5:  else
6:      V = V - R_MPS
7:      R = R_LPS

// renormalization
8:  while (R < 2^8)
9:      R = R << 1
10:     V = V << 1
11:     V = V | read_one_bit()
```

Fig. 2 – Implementation of an H.264/MPEG4-AVC compliant M decoder w/o probability estimation (using a fixed probability state *m*).

3. FAST STANDARD-COMPLIANT RENORMALIZATION

3.1 Determination of renormalization cycles

The first natural step toward a simplification of renormalization consists in unrolling the while loop (line no. 8 of both Fig. 1 and Fig. 2). It is quite obvious that for avoiding multiple checks of the while condition, it is sufficient to determine in advance the bit index of the most significant bit (MSB) in the R register relative to the loop guard with its MSB placed at bit index equal to 8 (for the specific M coder under consideration). Since the value of R is doubled or left-shifted for each loop cycle, the numerical difference between 8 and the current bit index of the MSB of R is equal to the number of cycles the renormalization loop has to be executed.

Many hardware architectures allow to determine the MSB bit index within a single instruction like, e.g., the Bit Scan Reverse (BSR) instruction on Intel's x86 architecture [8]. However, in cases where the implementation of the M coder has to be more generic or platform-independent, the use of such low-level machine dependent instructions may be prohibited. For those use cases or simply for cases where no specific instructions for MSB index detection are available, we propose an alternative way of determining the number of renormalization cycles.

To this end, we first discriminate between the MPS and LPS case. In case of encoding/decoding an MPS event, the value of $R_{\mathrm{MPS}} = R - R_{\mathrm{LPS}}$ can be bounded from below as follows. Let us assume that according to equation (1), a specific value of k with $0 \leq k < 4$ is derived from R. Then, the estimation $R \geq (4 + k) \times 2^6$ holds and from the specification of the *RTAB* table in [1], we can deduce the following upper bounds for R_{LPS}, depending on the value of k:

$$R_{\mathrm{LPS}} = RTAB[m][k] \leq \begin{cases} 128, & \text{if } k = 0 \\ 176 + (k-1) \times 2^5, & \text{else} \end{cases}, \tag{2}$$

for all m with $0 \leq m < 64$. Combining both estimations, we can conclude that $R_{\mathrm{MPS}} \geq 128$ always holds and therefore, at most one renormalization cycle is required for the MPS. This is equivalent to what one would expect from an exact implementation of subinterval division as outlined in Sec. 2,

```
 1:  R_LPS = RTAB[m][(R >> 6) & 3]
 2:  R_MPS = R - R_LPS
 3:  if (V < (R_MPS << BitsLeft))
 4:     R = R_MPS
 5:     Rnorm = (R_MPS >> 8) XOR 1
 6:  else
 7:     V = V - (R_MPS << BitsLeft)
 8:     R = R_LPS
 9:     Rnorm = RnormTAB[R_LPS >> 3]
10:  R = R << Rnorm
11:  BitsLeft = BitsLeft - Rnorm
12:  if (BitsLeft <= 0)
13:     V = (V << M_BITS) | read_bits(M_BITS)
14:     BitsLeft = BitsLeft + M_BITS
```

Fig. 3 – Proposed fast renormalization in an H.264/MPEG4-AVC compliant M decoder (w/o probability estimation).

because from the definition we have $p_{\text{MPS}} \geq 0.5 \geq p_{\text{LPS}}$.[2] Thus, for the MPS, a simple bit test is sufficient to compute the number of renormalization cycles, denoted by *Rnorm*:

$$Rnorm = (R_{\text{MPS}} >> 8) \text{ XOR } 1,$$

where XOR denotes the logical exclusive-or operation.

In the LPS case, we can directly deduce *Rnorm* from the tabulate R_{LPS} values of *RTAB*. A straightforward method, for instance, would be to put the corresponding *Rnorm* values in a complementary 2D-table with 64 × 4 entries. However, by observing that the entries of *RTAB* imply a strict lower bound for R_{LPS}, a much smaller and hence more practical lookup table can be derived as follows.

First, from the definition of the underlying FSM of the M coder [1][3][4], it is clear that the values of $p_m \in \mathcal{P}$ are given in decreasing order with increasing value of the probability state *m*. This, in turn, implies that the minimum value of *RTAB*[*m*][*k*] is given for the maximum value of *m*. The probability state with $m = 63$, however, corresponds to an autonomous, non-adaptive state within the H.264/MPEG4-AVC-based M-coder realization, and it is only used for encoding/decoding of terminating syntax elements, for which often a separate encoding/decoding routine is utilized [1][4]. Since *RTAB*[63][*k*] = 2 for all values of *k*, the corresponding *Rnorm* value can be determined to be equal to 7. For all regular states of the FSM corresponding to $m < 63$, we have $R_{\text{LPS}} = RTAB[m][k] \geq 6$. Based on this lower bound for all states with $m < 63$, we can aggregate the *Renorm* values that correspond to R_{LPS} values strictly less than 8, because for those values exactly 6 renormalization cycles have to be performed. Consequently, we can discard the 3 least significant bits of R_{LPS} and use the remaining 5 MSBs for indexing a table *RnormTAB* which is constructed to indicate the unique number of renormalization cycles for each value of R_{LPS} (with the exception of those related to $m = 63$):

$$Rnorm = RnormTAB[R_{\text{LPS}} >> 3].$$

Note that *RnormTAB* is a table that requires not more than 31 entries, each with a precision of 3 bits only. This follows from the upper bound on the *Renorm* value (equal to 6 for the table index equal to 0) as well as from the overall upper bound on R_{LPS} given by a value of 240 according to (2). As a result, we can detect the MSB of R_{LPS} and hence the corresponding *Rnorm* value with a comparably low cost of one bit shift and one table lookup operation, where the table size is smaller than 1/8 the size of the table needed for the naive approach, as mentioned above.

Having predetermined values of renormalization cycles *Rnorm* available allows to in-/output *Rnorm* bits at once. However, instead of forcing the decoder/encoder to carry out the corresponding in-/output operation each time *Rnorm* has a non-vanishing value, we propose to bring forward/postpone the actual in-/output until a fixed amount *M_BITS* of multiple bits have been exhausted/accumulated. *M_BITS* can take any positive value, though in a practical scenario we may choose its value to represent a multiple of 8. In the following, we first present a realization of this rather simple idea applied to an H.264/MPEG4-AVC-compliant *M decoder*.

3.2 Multiple-bit input at the decoder

As already pointed out above, the main idea is to decouple the in-/output of bits from the actual renormalization operation. In the decoder, this is accomplished by inputting *M_BITS* bits in advance into the *V* register. Since the *V* (and *R*) register in an ordinary M decoder requires a minimum precision of 9 bits, we have to first enlarge the *V* register to maintain a precision of 9 + *M_BITS*. In addition, we have to introduce an auxiliary register *BitsLeft* which serves two purposes. First, it indicates the number of available bits in the *V* register and, secondly, it keeps track of a virtual floating point for balancing the precision of *R* and *V*, i.e., it indicates the amount of bit shifts to the left that have to be applied to *R* before combining or comparing it with *V*.

Fig. 3 shows the pseudocode of an M decoder that includes the aforementioned ideas. Note that the actual renormalization of *R* (line no. 10 of Fig. 3) is performed after each decoding cycle. However, any input of bits is performed in chunks of *M_BITS* and only after the cumulative amount of *Rnorm* values exceeds the value of *M_BITS*. That implies in particular that the corresponding lines no. 12-14 of Fig. 3 are executed only for each chunk of *M_BITS* input bits.

3.3 Multiple-bit output and carry over at the encoder

The same ideas as described in the previous section can be applied to the *M encoder* as well in a rather straightforward manner. Since the interval subdivision is performed in a way that encoder and decoder are perfectly synchronized, the corresponding changes (in terms of code instructions) between Fig. 2 and Fig. 3 can be transferred in an analogue fashion to Fig. 1.

However, one substantial difference in the renormalization part of the encoder is given by the handling of potential carry-over events, as already pointed out in Sec. 2. Since the output of leading bits (MSBs) from the *L* register is performed in chunks of *M_BITS* as well, we need to monitor only the case where all *M_BITS* output bits have a value of "1" by incrementing a corresponding "outstanding" event counter. This fact together with the buffering of the previous *M_BITS* output bits before actually writing them to the bit-

[2] Note, however, that due to the approximations involved, it is not always guaranteed that $R_{\text{MPS}} \geq R_{\text{LPS}}$ for the M coder.

```
encode()
  R_LPS = RTAB[m][(R >> 6) & 3]
  R_MPS = R - R_LPS
  if (value == valMPS)
     R = R_MPS
     Rnorm = (R_MPS >> 8) XOR 1
  else
     L = L + (R_MPS << BitsLeft)
     if (L >= 2^(10 + M_BITS))
        L = L - 2^(10 + M_BITS)
        propagate_carry()
     R = R_LPS
     Rnorm = RnormTAB[R_LPS >> 3]
  R = R << Rnorm
  BitsLeft = BitsLeft - Rnorm
  if (BitsLeft <= 0)
     output()
```

```
output()
  out = (L >> 10) & (2^M_BITS - 1)
  if (out == (2^M_BITS - 1))
     ChunksOutstanding++
  else
     put_chunk_bits_plus_outstanding(out)
  L = L << M_BITS
  BitsLeft = BitsLeft + M_BITS

propagate_carry()
  Buffer = Buffer + 1
  while (ChunksOutstanding > 0)
     put_chunk_bits(0)
     ChunksOutstanding--
```

Fig. 4 – Left: M encoder implementation including the proposed fast renormalization part. **Right:** Auxiliary routines for output of chunks of *M_BITS* bits and for carry propagation. Note that the minimum precision of the register *L* is 11 + *M_BITS,* whereas that for *R* is still 10 bits.

stream enables a significantly simplified carry-over processing. Fig. 4 shows an encoder implementation equipped with the proposed multiple-bit output and carry-over handling.

4. EXPERIMENTAL RESULTS

For an experimental evaluation of our proposed fast renormalization scheme, we first implemented the decoder related changes into our own run-time optimized H.264/MPEG4-AVC High profile (HP) decoder. In addition to what has been discussed in the previous section, we also adapted the initialization, the termination as well as the bypass part of the M decoder according to the modified renormalization strategy. For that implementation, a value of 16 was chosen for *M_BITS*. We generated H.264/MPEG4-AVC HP-conforming bitstreams for four 1080p test sequences (each with 50 frames) by using intra-only coding with fixed QPs of 20 and 24. These settings were chosen because for a typical HDTV broadcast scenario, the generated bit-rates can be regarded as a kind of upper bound for HP@L4 [1].

For the purpose of performing reliable run-time measurements, we disabled the decoding process and measured the parsing process only. The run-time measurements for parsing the test bitstreams were performed by using our H.264/MPEG4-AVC HP decoder, comparing a conventional M decoder implementation with our proposed fast renormalization scheme. The corresponding experiments were performed on a Pentium 4, 3.6 GHz machine with Linux OS, where the code was generated using gcc, version 4.03.

As a result of those experiments, we obtained an average reduction in measured run time for the M decoder part in the range of 23.5–26.9%. Note that the proportion of the actual arithmetic decoding process relative to the whole parsing process was about 35%.

In another set of experiments, we compared run time for two different *encoder* versions – one conventional implementation and another version using the proposed fast renormalization. These experiments were carried out by using a JPEG still image coding implementation [9], where the corresponding QM coder was replaced by our two versions of the M coder. The reason for using JPEG instead of H.264/MPEG4-AVC for these experiments is given by the fact that in a typical H.264/MPEG4-AVC encoder, the proportion of run time of the arithmetic encoding part is usually too small to deliver statistically reliable results. As an outcome of our JPEG-based encoding experiments, we obtained average run-time improvements for the M encoder of 47.3–52.7%, where parts of these relatively large gains can be attributed to the largely improved carry-over handling.

5. CONCLUSIONS

We have introduced a fast, standard-compliant renormalization method for the binary arithmetic encoder and decoder in H.264/MPEG4-AVC. By replacing the conventionally bit-wise performed operations with byte-wise or word-wise processing, a considerably increased throughput can be achieved. We presented experimental results for demonstrating the benefits of the novel renormalization technique, especially for the purpose of software-based decoding of HD video.

REFERENCES

[1] ITU-T and ISO/IEC JTC 1, "Advanced Video Coding for Generic Audiovisual Services", ITU-T Recommendation H.264 & ISO/IEC 14496-10 (MPEG4-AVC), Version 4, July, 2005.

[2] D. Marpe, G. Heising, G. Blättermann, and T. Wiegand, "Fast arithmetic coding for CABAC," *ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6*, Doc. JVT-C061, Fairfax, USA, March 2002.

[3] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," *Proc. IEEE ICIP 2003*, Barcelona, Spain, Sept. 2003.

[4] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 620-636, July 2003.

[5] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, "An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder", *IBM J. Res. Dev*., vol. 32, pp. 717-726, 1988.

[6] D. Taubman and M. W. Marcellin, *JPEG2000 Image Compression: Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2002.

[7] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited", *ACM Trans. on Information Systems*, 16(3), pp. 256-294, July 1998.

[8] IA-32 Intel Architecture Software Developers's Manual, Vol. 2: Instruction Set Reference, 2002.

[9] Independent JPEG Group online: http://www.ijg.org.