Signal Processing: Image Communication 112 (2023) 116914

Contents lists available at ScienceDirect

Signal Processing: Image Communication

journal homepage: www.elsevier.com/locate/image

Probability models for highly parallel image coding architecture

Francesc Aulí-Llinàs *, Joan Bartrina-Rapesta, Miguel Hernández-Cabronero

Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain

ARTICLE INFO

Keywords:
Image coding
Parallel computing
Entropy coding
Probability models

ABSTRACT

A key aspect of image coding systems is the probability model employed to code the data. The more precise the probability estimates inferred by the model, the higher the coding efficiency achieved. In general, probability models adjust the estimates after coding every new symbol. The main difficulty to apply such a strategy to a highly parallel coding engine is that many symbols are coded simultaneously, so the probability adaptation requires a different approach. The strategy employed in previous works utilizes stationary estimates collected a priori from a training set. Its main drawback is that statistics are dependent of the image type, so different images require different training sets. This work introduces two probability models for a highly parallel architecture that, similarly to conventional systems, adapt probabilities while coding data. One of the proposed models estimates probabilities through a finite state machine, while the other employs the statistics of already coded symbols via a sliding window. Experimental results indicate that the latter approach improves the performance achieved by the other models, including that of JPEG2000 and High Throughput JPEG2000, at medium and high rates with only a slight increase in computational complexity.

1. Introduction

The core of most compression systems holds an entropy coder that converts pre-processed data into a more compact representation. Key to achieve compression is to exploit the probabilities of the coded symbols. Symbols with a higher probability can be represented more compactly, with the limit imposed by the Shannon's entropy. In general, two aspects affect the efficiency of the entropy coder: the coding scheme and the probability model. There are many different coding schemes, ranging from Huffman coding [1], arithmetic coding [2], or the recently introduced asymmetric numeral systems [3], among others. Each coding scheme imposes a computational burden and an efficiency limit. The probability model, on the other hand, extracts relationships among the coded data in order to estimate the probabilities of the new symbols fed to the coder. In general, this is achieved employing contextual information, since previous symbols with the same context tend to have similar probabilities.

In the field of image and video coding, the first compression standards such as JPEG (ISO/IEC 10918-1) and MPEG-2 (ITU H.262) employed, among others, entropy coding schemes based on Huffman. Variants of such entropy coders and their probability models were proposed to improve coding efficiency and reduce computational costs [4–6]. The next generation of standards, including JPEG2000 (ISO/IEC 15444-1), AVC (ITU H.264), and HEVC (ITU H.265) adopted low-complexity, context-adaptive binary arithmetic coders [7–9]. They were the topic of many works in the literature too [10–14]. The latest standards such as VVC (ITU H.266) and High Throughput (HT) JPEG2000 utilize arithmetic coding [15] and variable length coding [16], respectively. These standards also provide parallelization opportunities that can be adopted in implementations tailored for CPUs or GPUs, since this is the current trend to increase the throughput of computationally intensive applications.

In the same vein as these latest standards, Bitplane Image Coding with Parallel Coefficient Processing (BPC-PaCo) [17] is a wavelet-based image coding engine with features similar to those of the JPEG2000 standard. Its main difference is the bitplane and entropy coding engine, which provides more opportunities to exploit fine-grain parallelism in highly parallel architectures such as those of GPUs. Contrarily to most image codecs that process image coefficients one by one, BPC-PaCo codes 32 coefficients in parallel. The probability model employed by the original BPC-PaCo [17] is based on stationary probabilities generated with a training set of images [12]. This model was adopted due to its low computational complexity, since probability estimates are obtained by simply accessing a lookup table (LUT). The main drawback is that it needs to be trained with a set of images similar to that coded, which may not be feasible in some scenarios, and that the LUT must be (possibly transmitted and) specified in the encoder and the decoder. To use adaptive probabilities like most conventional image coding systems poses a challenge in the highly parallelized engine of BPC-PaCo because probability estimates need to be synchronized and shared among all threads of execution.

* Correspondence to: Escola Enginyeria, UAB, 08193 Bellaterra, Spain.
E-mail addresses: francesc.auli@uab.cat (F. Aulí-Llinàs), joan.bartrina@uab.cat (J. Bartrina-Rapesta), miguel.hernandez@uab.cat (M. Hernández-Cabronero).

https://doi.org/10.1016/j.image.2022.116914
Received 12 July 2022; Received in revised form 25 November 2022; Accepted 7 December 2022
Available online 30 December 2022

This paper approaches this challenge with the study of two probability models that use adaptive probabilities. This study is carried out mostly from the point of view of coding performance. The first model employs a finite state machine that predicts the probability of future symbols. This strategy is similar to that employed in JPEG2000 and other standards such as HEVC and VVC. The second model is based on a sliding window mechanism that adjusts probability estimates through statistics of already coded symbols. This latter approach achieves high performance regardless of the image type, improving the results achieved by the remaining models and also by JPEG2000.

The rest of the paper is structured as follows. Section 2 reviews BPC-PaCo. Section 3 describes the original probability model employed in BPC-PaCo and introduces the two new models proposed in this paper. Section 4 provides extensive experimental results with four different corpora, assessing coding performance and computational complexity. The final section concludes with a brief summary and some remarks.

2. Bitplane coding with Parallel Coefficient Processing

BPC-PaCo is a coding engine tailored for wavelet-based image coding systems. The first stage of such codecs commonly entails one or various transforms, including the discrete wavelet transform, to remove spatial, spectral, and/or temporal redundancy. The resulting wavelet coefficients are quantized and coded via bitplane and entropy coding, which is the stage that accounts for more than 80% of the computational workload [18]. There are many bitplane coding engines in the literature, e.g., SPIHT [19], SPECK [20], EBCOT [21], JPEG2000's tier-1 [22], or BPC-PaCo [17]. To allow coarse-grain parallelism, the wavelet coefficients are typically partitioned in small sets, referred to as codeblocks, that are coded independently, producing a bitstream for each. These bitstreams are possibly (truncated and) re-organized to fit a target rate, or to form layers of quality, in the last stage of the coding pipeline.

The main idea behind bitplane coding is to code all data within the codeblock bit by bit, beginning from the highest magnitude bit of all quantized coefficients and finishing with the lowest. If $[b_{M-1}, b_{M-2}, \cdots, b_1, b_0]$ with $b_i \in \{0,1\}$ being the binary representation of the absolute value of υ obtained when wavelet coefficient ω is quantized, bitplane j is defined as the set of bits b_j from all coefficients. The coding engine codes all bits from bitplane $M-1$ to bitplane 0, assuming that all indices υ are lower than 2^M.

The coefficients are defined as significant or non-significant. A coefficient becomes significant at bitplane s when the first non-zero bit of its binary representation is found, more precisely, when $b_s = 1$ with $b_{s'} = 0, M > s' > s$. b_s is called the significant bit for that coefficient, and $b_r, 0 \leq r < s$ are called refinement bits. Commonly, the coefficients are scanned multiple times in each bitplane, first coding the bits of non-significant coefficients in previous bitplanes because these bits reduce most the image distortion [23]. Then, a second coding pass emits the refinement bits of the remaining coefficients. JPEG2000 further refines this procedure employing three coding passes. The first is called Significance Propagation Pass (SPP) and codes the bits of non-significant coefficients that are neighbors of already significant coefficients. The second is the magnitude refinement pass (MRP) and codes refinement bits. The last is the Cleanup Pass (CP) and codes the bits of the remaining coefficients. Once a coefficient becomes significant, its sign $d \in \{+,-\}$ is immediately coded to allow the decoder approximate ω as soon as possible.

The order in which the coefficients are scanned is a key difference between JPEG2000 and BPC-PaCo. Fig. 1 illustrates both orders for a codeblock of 16 × 16 coefficients. In JPEG2000, a single thread consecutively scans all the coefficients within the codeblock, producing an embedded bitstream in which each bit can only be decoded after the previous. BPC-PaCo employs one execution thread per each pair of columns, coding 32 coefficients simultaneously (in codeblocks of 64 columns). The scanning is repeated three times in each bitplane, once per coding pass. The parallel processing of BPC-PaCo is tailored for the highly parallel architecture of GPUs, which typically maps parallel execution flows to vector instructions. In the following, thread is referred to such an execution flow.

Each bit emitted by the bitplane coder is fed to the entropy coder accompanied by its context. The context is an important aspect of the probability model because the entropy coder obtains the probability estimate of the incoming bit depending on its context. A comprehensive study of context formation [24] concludes that simple strategies obtain high performance. BPC-PaCo embraces this simplicity to reduce computational costs too. Contexts are defined as follows: let the eight adjacent neighbors of υ be denoted by $\upsilon^k, 1 \leq k \leq 8$. Its significance state in bitplane j is referred to as $\Phi(\upsilon^k, j)$, being 1 when υ^k has been coded in the previous significance pass (i.e., if there is an $s > j$ such that $b_s(\upsilon^k) = 1$), or when it is coded in the current bitplane -and the coefficient is already visited in the current coding pass. Otherwise $\Phi(\upsilon^k, j) = 0$. The context employed for significance coding in SPP and CP in bitplane j is defined as the sum of the significance states of the adjacent neighbors of υ, more precisely

$$\phi_{sig}(\upsilon, j) = \sum_{k=1\ldots8} \Phi(\upsilon^k, j)\ , \tag{1}$$

so $\phi_{sig}(\upsilon, j) \in [0, 8]$. We note that using the significance state of these neighbors does not create thread dependencies because all threads advance their execution at the same pace, i.e., in a lockstep synchronous way.

The context for sign coding employs the sign of the vertical and horizontal adjacent neighbors of coefficient ω. Let $\chi(\omega^k, j)$ be 0, 1 or −1 at bitplane j when the coefficient is still not significant, significant and positive, and significant and negative, respectively. χ^V and χ^H are defined as the sum of $\chi(\omega^k, j)$, respectively for the two vertical and horizontal adjacent neighbors of ω. The context for sign coding is defined as

$$\phi_{sign}(\omega, j) = \begin{cases} 9 & \text{if } (\chi^V > 0 \text{ and } \chi^H > 0) \text{ or} \\ & \quad (\chi^V < 0 \text{ and } \chi^H < 0) \\ 10 & \text{if } \chi^V = 0 \text{ and } \chi^H \neq 0 \\ 11 & \text{if } \chi^V \neq 0 \text{ and } \chi^H = 0 \\ 12 & \text{otherwise} \end{cases} . \tag{2}$$

A single context is employed for refinement bits since more complex approaches do not obtain significant coding gains, so $\phi_{ref}(\upsilon, j) = 13$. In total 14 different contexts are employed, 9 for significance, 4 for sign, and 1 for refinement coding.

Let $P_{sig}(b_j = 0 \mid \phi_{sig}(\upsilon, j))$ denote the probability estimate for one context of significance coding. Conventional models devised for single-threaded execution, like JPEG2000, typically set $P_{sig}(\cdot) = 0.5$ at the beginning of coding and then adjust this probability depending on every new bit coded. This procedure cannot be reproduced in BPC-PaCo due to the parallel processing of coefficients.

3. Probability models

3.1. Stationary

The original model of BPC-PaCo is based on stationary probabilities. Instead of adjusting probabilities adaptively depending on the incoming data, this model uses fixed estimates that depend on the wavelet subband, bitplane, and context. The model is thoroughly studied in [12]. The main idea is to collect statistics of images captured with the same sensor in order to build a LUT that both the encoder and decoder share. The data produced by the same wavelet filter bank for images of the same type are statistically similar [24–26], so the LUT can be employed to estimate the symbols' probability of other images.

First, the probability mass function (pmf) of the quantization indices is computed with all the images in the training set. The pmf for subband u at bitplane j is referred to as $F_u(\upsilon \mid \phi_{sig}(\upsilon, j))$ for the contexts of

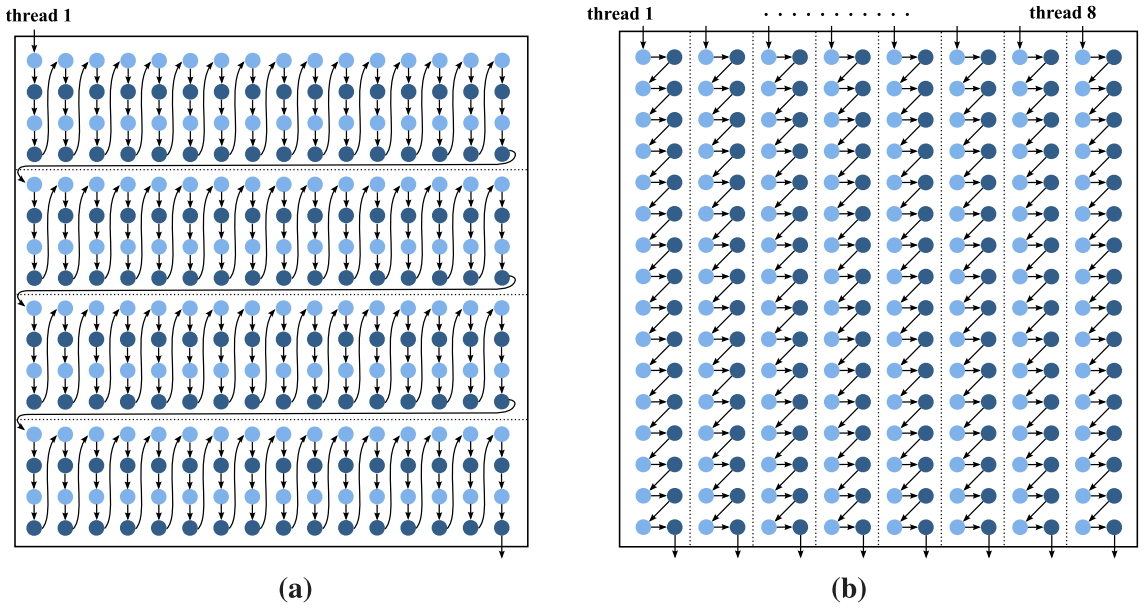

Fig. 1. Scanning orders employed by (a) the original tier-1 coding stage of JPEG2000 standard, and (b) the highly parallel engine BPC-PaCo.

significance coding. The support of this pmf is $[0, \dots, 2^{j+1}-1]$ because it considers coefficients $\upsilon \in [0, 2^{j+1})$. Then, the probability that $b_j = 0$ is determined as

$$P_{sig}(b_j = 0 \mid \phi_{sig}(\upsilon, j)) = \frac{\sum_{\upsilon=0}^{2^j-1} F_u(\upsilon \mid \phi_{sig}(\upsilon, j))}{\sum_{\upsilon=0}^{2^{j+1}-1} F_u(\upsilon \mid \phi_{sig}(\upsilon, j))}. \tag{3}$$

These probabilities populate the LUT for significance coding, which is accessed as $\mathcal{P}_u[j][\phi_{sig}(\cdot)]$. The LUTs for refinement and sign coding are derived similarly. Through these LUTs, the coding procedure computes the coefficient's context and then accesses the corresponding LUT to obtain the estimate. No further operations are required since the LUTs are not updated during the coding process, so the computational complexity of such a strategy is very low.

3.2. Finite state machine

Adapting the probability estimates with a finite state machine is a widely employed technique in image and video coding [8,22,27]. Each state holds a probability estimate for the most probable symbol (MPS), in the range $[0.5, 1)$, and has two transitions to other states. One of the transitions is employed when coding the MPS, and the other when coding the least probable symbol (LPS). Each context points at one state. They are commonly initialized at the state with the lowest probability (i.e., 0.5). Every time a new symbol is coded, a transition to a new state with a higher (lower) probability in the case of an MPS (LPS) is carried out. In general, the state machine contains a series of states that increase the probability estimate very rapidly at the beginning of coding while MPSs are found, and then the probabilities are more finely adjusted as more data are processed.[1]

Such a state machine works well when transitions among states are carried out for every new symbol coded, which is not feasible in BPC-PaCo. The threads processing coefficients in parallel cannot share information about the symbols that are currently coding because that would entail causal relations among them, disrupting the parallel processing. However, once the coefficients are processed, the coding results can be employed to update the probability estimate of each context. Fig. 2 illustrates the state machine used in our approach. It has 64 states depicted with circles. Transitions of MPSs and LPSs are respectively depicted in blue and red arrows. The coding of an MPS increases the probability estimate linearly from 0.5 to 0.992, whereas coding an LPS decreases the estimate exponentially. Similar strategies are employed in other image codecs (such as the MQ coder of JPEG2000 [12]). It has been devised empirically with a large dataset to achieve high coding efficiency regardless of the image type.

Algorithm 1 describes the three coding passes carried out by BPC-PaCo when encoding. Decoding is the inverse procedure, so it is not detailed herein. These coding passes are called consecutively in each bitplane (not shown in the algorithm). The algorithm is described from the point of view of a single execution flow (i.e., a thread) that codes the data of two adjacent columns. Nonetheless, note that the execution in the GPU runs all threads of the codeblock in parallel, advancing their execution synchronously. The procedures in Algorithm 1 receive the bitplane and thread as parameters. The bitplane is employed to emit the corresponding bit and to compute the context, whereas the thread is employed to compute the processed columns. The two first lines in each coding pass embody the scanning order. Then, *SPP(·)* and *CP(·)* check whether the coefficient has to be coded in that pass or not. Quantization indices and coefficients are respectively referred to as $\upsilon_{y,x}$ and $\omega_{y,x}$ following the same notation as above but denoting the position within the codeblock. If the coefficient needs to be coded, *ACencode(·)* is called. If the coefficient is significant, sign coding also calls this procedure with its corresponding context (line 6 in both *SPP(·)* and *CP(·)*). Refinement coding is similar but without sign coding. *ACencode(·)* is also executed in parallel and synchronously for all threads. If a thread does not call this procedure, it remains idle until the remaining threads finish its processing. In all coding passes, *ACupdate(·)* is called just after the coefficient is coded (also in parallel and synchronously for all threads). The parameter given to this procedure is thread t, although it is received as context c in Algorithm 2. Whereas Algorithm 1 and *ACencode(·)* in Algorithm 2 use one thread per each pair of columns, *ACupdate(·)* uses the same threads to update the probabilities of all contexts in parallel. As there are only 14 contexts, threads $t \geq 15$ do not contribute to the updating operation and are idle during this step.

Algorithm 2 describes the operations related to the probability model that are carried out by the arithmetic coder. The first line in *ACencode(·)* sets probability p, employed to code the interval. In this

[1] We refer the reader to [12, Figure 2] for an illustration of the state machine employed in the MQ coder of JPEG2000.

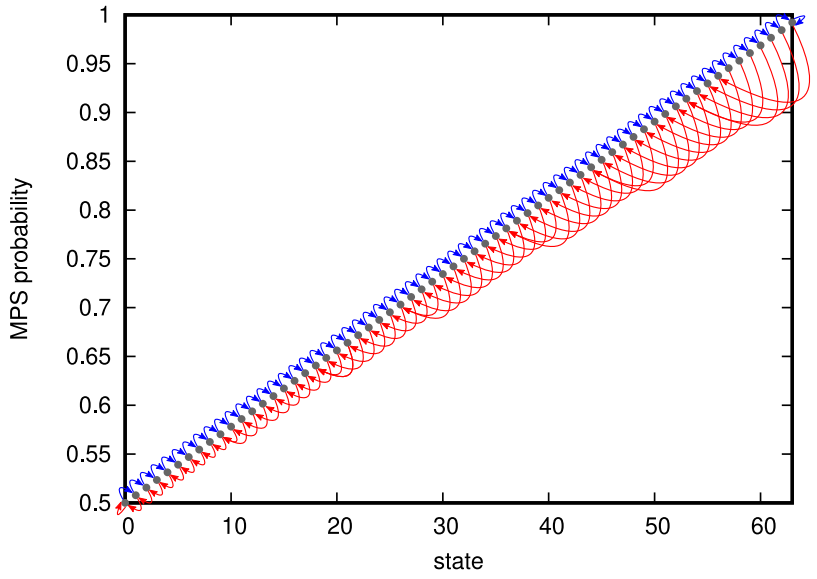

Fig. 2. Finite state machine employed to estimate probabilities of future symbols. Each state is depicted with a gray circle. The estimated probability of the MPS symbol associated with each state is depicted in the vertical axis. Blue and red lines indicate transitions when coding an MPS and LPS, respectively.

Algorithm 1 BPC-PaCo with adaptive prob. models

SPP(j bitplane, t thread)
for $y \in [0, \text{numRows} - 1]$ **do**
  for $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
    if $v_{y,x}$ is not significant **AND** $\phi_{sig}(v_{y,x}, j) \neq 0$ **then**
      ACencode(b_j, $\phi_{sig}(v_{y,x}, j)$, t)
      if $b_j = 1$ **then**
        ACencode(d, $\phi_{sign}(\omega_{y,x}, j)$, t)
      end if
    end if
    ACupdate(t)
  end for
end for

MRP(j bitplane, t thread)
for $y \in [0, \text{numRows} - 1]$ **do**
  for $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
    if $v_{y,x}$ is significant in $j' > j$ **then**
      ACencode(b_j, $\phi_{ref}(v_{y,x}, j)$, t)
    end if
    ACupdate(t)
  end for
end for

CP(j bitplane, t thread)
for $y \in [0, \text{numRows} - 1]$ **do**
  for $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
    if $v_{y,x}$ is not significant **AND** not coded in SPP **then**
      ACencode(b_j, $\phi_{sig}(v_{y,x}, j)$, t)
      if $b_j = 1$ **then**
        ACencode(d, $\phi_{sign}(\omega_{y,x})$, t)
      end if
    end if
    ACupdate(t)
  end for
end for

Algorithm 2 Arithmetic coder (relevant operations)

ACencode(s symbol, c context, t thread)
$p \leftarrow \begin{cases} P'[S[c]] \ / * \textit{ finite state machine } * / \\ P''[c] \ / * \textit{ sliding window } * / \end{cases}$
code(s, p, t) / * *interval coding – see [17, Algorithm 1]* * /
if $s = 0$ **then**
  $\mathcal{T}_0[c] \leftarrow \mathcal{T}_0[c] + 1$
end if
$\mathcal{T}[c] \leftarrow \mathcal{T}[c] + 1$

ACupdate(c context) / * *finite state machine* * /
if $c < 15$ **AND** $\mathcal{T}[c] > 0$ **then**
  $S[c] \leftarrow \text{LPStransition}(\mathcal{T}[c] - \mathcal{T}_0[c])$
  $S[c] \leftarrow \text{MPStransition}(\mathcal{T}_0[c])$
  $\mathcal{T}[c] \leftarrow 0$
  $\mathcal{T}_0[c] \leftarrow 0$
end if

ACupdate(c context) / * *sliding window* * /
if $c < 15$ **AND** $\mathcal{T}[c] > 0$ **then**
  $P''[c] \leftarrow \mathcal{T}_0[c]/\mathcal{T}[c]$
  if $\mathcal{W}_0[c] = -1$ **AND** $\mathcal{T}[c] \geq W$ **then**
    $\mathcal{W}_0[c] \leftarrow \mathcal{T}_0[c]$
    $\mathcal{W}[c] \leftarrow \mathcal{T}[c]$
  end if
  if $\mathcal{T}[c] \geq 2W$ **then**
    $\mathcal{T}[c] \leftarrow \mathcal{T}[c] - \mathcal{W}[c]$
    $\mathcal{W}[c] \leftarrow \mathcal{T}[c]$
    $\mathcal{T}_0[c] \leftarrow \mathcal{T}_0[c] - \mathcal{W}_0[c]$
    $\mathcal{W}_0[c] \leftarrow \mathcal{T}_0[c]$
  end if
end if

model, $S[c]$ denotes the state of the corresponding context, whereas P' is the probability associated with that state. Once p is set, operations to code the interval with that probability are carried out via *code*(s, p, t), which uses an independent arithmetic coder for each thread. This procedure is not included in Algorithm 2 for simplicity, but can be consulted in [17, Algorithm 1]. The remaining lines in *ACencode(·)* update the number of 0s and the total number of symbols coded for this context, which are stored in $\mathcal{T}_0[c]$ and $\mathcal{T}[c]$, respectively. These operations can be implemented as atomic increments in a GPU, so that even if two or more threads use the same context in the same step, the result remains correct.

The procedure *ACupdate(·)* employed in this model (second in Algorithm 2) updates the state of the context. If one or more symbols have been coded with that context, the state is decreased by the number of LPSs coded (i.e., $\mathcal{T}[c] - \mathcal{T}_0[c]$) and then increased by the number of MPSs coded (i.e., $\mathcal{T}_0[c]$). Afterwards, $\mathcal{T}_0[c]$ and $\mathcal{T}[c]$ are reset to zero.

3.3. Sliding window

A variable-size sliding window [28,29] is employed by this last model of probabilities. The main idea is to use statistics of the last coded symbols, enclosed in this window, to estimate the probability of the incoming. This strategy assumes that the new symbols have a similar distribution to the last ones. The window has between W and $2W$ symbols, approximately, except at the beginning of coding. With some abuse of notation, let the number of 0s and total number of symbols coded within this window be denoted by $\mathcal{T}_0[c]$ and $\mathcal{T}[c]$, respectively. When $\mathcal{T}[c] \geq 2W$, the window size is reduced subtracting the statistics of the first W symbols, approximately, which are stored in $\mathcal{W}_0[c]$ and $\mathcal{W}[c]$ for the 0s and the total, respectively.

Like the previous model, probability estimates are updated after processing 32 coefficients in parallel, so the procedures detailed in Algorithm 1 hold for this approach too. Algorithm 2 details the operations required to compute the probability and update the window. Procedure *ACencode(·)* carries out the same operations for both models except the first, which in this case uses $P''[c]$ to set the probability p corresponding to that context. $P''[c]$ is updated after checking that at least one symbol has been coded with this context in the first operation of *ACupdate(·)* (third procedure in Algorithm 2). This updating is expressed as a division, though in practice it can be implemented with bit shifts and additions depending on the interval partitioning procedure employed. We also note that the result of the division may never reach 1. At the beginning of coding, all array positions in $P''[c]$ are initialized to 0.5 except for the first, which is set to 0.9. This first context is initialized to 0.9 because it is employed to code the significance bits of coefficients that do not have any significant neighbor, so they are generally 0 too. This increases slightly the efficiency of the coder. The conditional in line 3 checks if the number of symbols coded in the window exceeds W for the first time (via $\mathcal{W}_0[c]$ that is initialized to -1). If so, sets variables $\mathcal{W}_0[c]$ and $\mathcal{W}[c]$. Every time the window contains $2W$ symbols or more, it is half emptied as detailed in lines 7–12. Experimental evidence indicates that $W = 256$ obtains high coding efficiency.

4. Experimental results

The experimental tests below utilize 4 corpora with different types of images to consider a wide variety of scenarios. The first corpus is the ISO 12640-1, which has eight color images of 2048 × 2560 and 8 bits per sample (bps). The images in the second corpus are captured with an aerial sensor covering vegetation and urban areas, are gray scale, 8 bps and have a size of 7200 × 5000. The third corpus belongs to the medical field. It consists of three X-ray angiography images with a size of 512 × 512 with 15 components and 12 bps. The last corpus consists of three AVIRIS (Airbone Visible/Infrared Imaging Spectrometer) hyperspectral images of 512 × 512 with 224 components and 16 bps that are provided by NASA. This corpora is the same to that employed in the original BPC-PaCo paper [17] to allow comparison. All codecs except HT JPEG2000, are implemented in our Java framework [30]. Results for HT JPEG2000 are obtained with Kakadu [31]. Codeblocks of 64 × 64 and 5 levels of irreversible (reversible) wavelet transformation are employed for the lossy (lossless) regime.[2] Training is neither needed by (HT) JPEG2000 nor by BPC-PaCo using either of the two models proposed in this work, since probabilities are adaptively adjusted while coding new data. The training set required by the stationary model includes all images of each corpus except the one that is evaluated.

First, lossy coding performance is assessed. The stationary, state machine, and sliding window models are compared against JPEG2000 and HT JPEG2000. Each image is coded at 100 equally spaced rates. Quality is evaluated in terms of peak signal to noise ratio (PSNR). Fig. 3 reports the results achieved by one image of each corpus. Similar results hold for the others. Results are reported as the PSNR difference between that obtained by the evaluated method and that of JPEG2000. The horizontal straight line in each plot is the performance of JPEG2000, while the remaining represent the evaluated methods. Results below (above) this horizontal line indicate lower (higher) performance than that of JPEG2000. The results of Fig. 3 suggest that the state machine model, which is similar to that employed by JPEG2000, obtains low performance when applied to parallel processing. Differences vary depending on the rate and image type. This seems to indicate that the transitions among states, which are necessarily performed only once for every 32 coefficients coded, diminishes the precision of the probability estimates. Contrarily, the sliding window model achieves the highest performance at medium and high rates in three of the four types of images evaluated. At these rates, this model obtains gains in the order of 0.5 dB with respect to JPEG2000, while at low rates it is in general less than 0.2 dB inferior to the standard, suggesting that very precise probability estimates are obtained when enough data are within the window (i.e., from medium to high rates). Compared to HT JPEG2000, the sliding window model obtains similar performance at low rates but much higher at medium and high rates. Similar results are obtained for codeblocks of smaller size (not shown in the figure), with the performance achieved by the two adaptive approaches being slightly more penalized than that achieved by the stationary model because fewer data are coded and so the probabilities are adjusted with less precision. The abrupt variations in coding performance seen in Fig. 3 for some of the images are due to the statistical behavior of the data coded in each coding pass, which is handled differently depending on the employed probability model.

The evaluation of lossy coding performance is completed with the test reported in Fig. 4, which compares the above-mentioned methods against JPEG XS (ISO/IEC 21122). JPEG XS is a wavelet-based image compression standard tailored for very low complexity and high throughput. Results are obtained with the JPEG XS Reference Software using profile "High444.12." As seen in the figure, the performance achieved by this standard is lower than that obtained by the proposed

Table 1
Evaluation of lossless coding performance.

				BPC-PaCo		
		JP2	*HT JP2*	*Station-ary*	*State machine*	*Sliding window*
ISO 12640-1	Portrait	**3.81**	4.0	3.94	3.94	3.82
	Cafeteria	**4.69**	4.97	4.79	4.80	**4.69**
	Fruit	**3.96**	4.22	4.15	4.06	**3.96**
	Wine	**3.94**	4.20	4.12	4.05	**3.94**
	Bicycle	**3.90**	4.18	4.09	4.04	3.93
	Orchid	**3.45**	3.69	3.68	3.56	3.46
	Music.	5.34	5.63	5.52	5.42	**5.29**
	Candle	**4.75**	5.03	4.87	4.86	**4.75**
	Average	*4.23*	*+0.27*	*+0.16*	*+0.11*	*+0.00*
aerial	forest1	6.20	6.51	6.16	6.25	**6.10**
	forest2	6.28	6.59	6.23	6.33	**6.18**
	urban1	5.54	5.88	5.55	5.68	**5.50**
	urban2	5.20	5.52	5.23	5.33	**5.16**
	Average	*5.80*	*+0.32*	*−0.01*	*+0.09*	*−0.07*
X-ray	A	6.37	6.68	6.30	6.40	**6.27**
	B	6.48	6.79	6.45	6.52	**6.37**
	C	6.35	6.66	6.29	6.38	**6.25**
	Average	*6.40*	*+0.31*	*+0.05*	*+0.03*	*−0.11*
AVIRIS	cuprite	7.01	7.32	6.98	7.09	**6.90**
	jasper	7.66	7.98	7.61	7.74	**7.52**
	lunarLake	6.91	7.22	6.89	6.98	**6.81**
	Average	*7.19*	*+0.31*	*−0.03*	*+0.08*	*−0.12*

[2] The proposed methods do not employ the bit stuffing technique deployed in JPEG2000 that avoids coding bytes with a $0xFF$ value. This does not significantly affect the rate–distortion comparisons since this technique increments negligibly the length of the codestream.

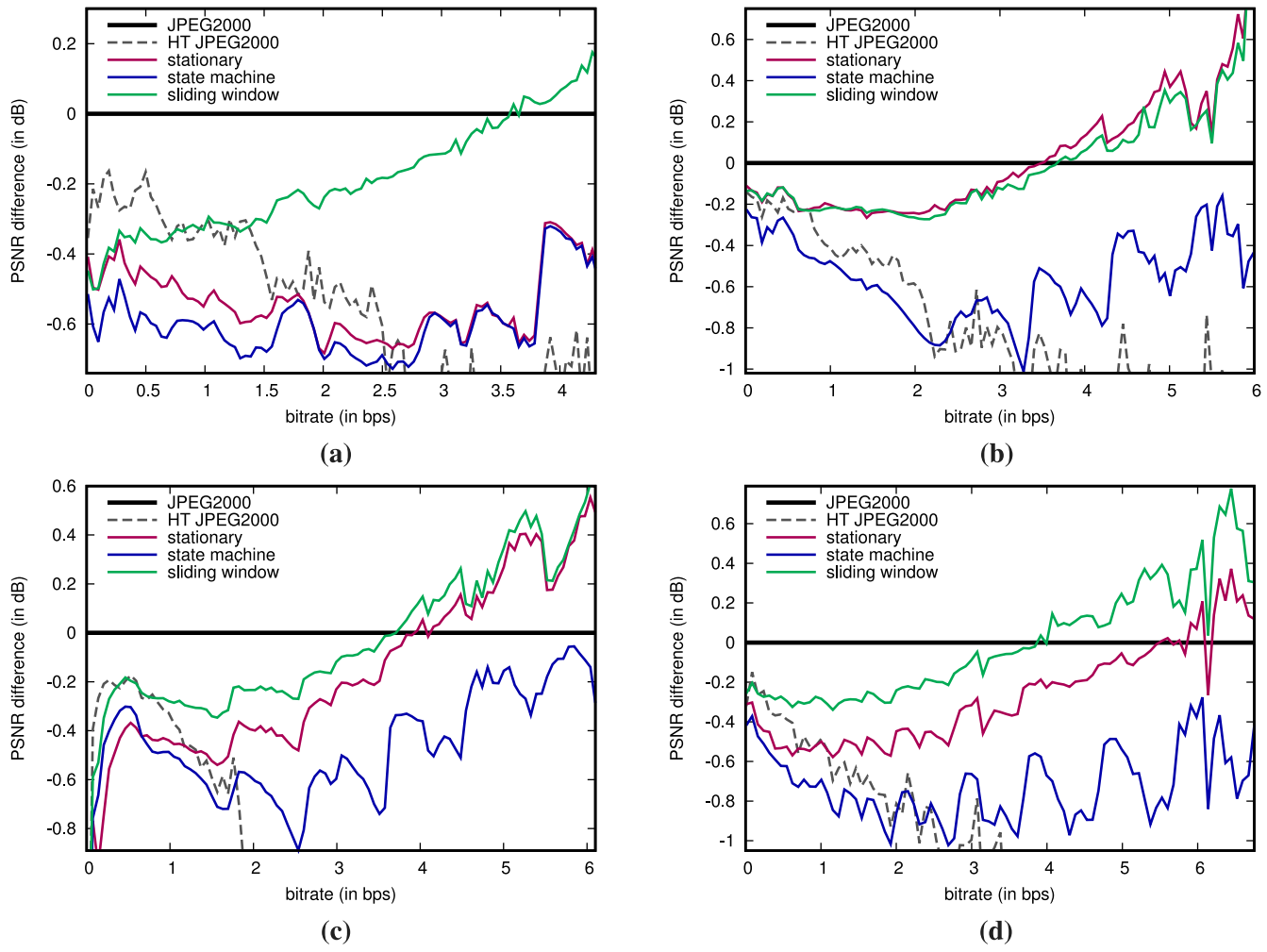

Fig. 3. Evaluation of lossy coding performance for images of different corpora: (a) "Musicians" of the ISO12640-1 corpus, (b) "forest2" of the aerial corpus, (c) "B" of the X-ray corpus, and (d) "jasper" of the AVIRIS corpus.

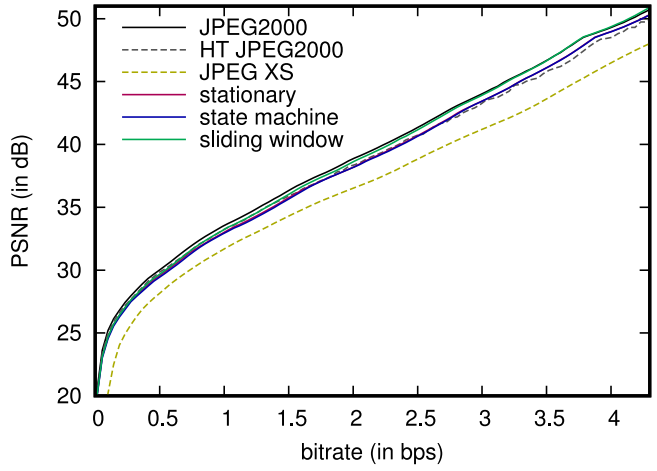

Fig. 4. Evaluation of lossy coding performance, including JPEG XS, for the "Musicians" image of the ISO12640-1 corpus.

methods, in part due to the lack of arithmetic coding. Similar results are obtained for the other images.

Second, lossless coding performance is appraised. Table 1 reports the results for all images. They are expressed in bps of the compressed file except for the average of each corpus. These rows report the average difference between the method evaluated and JPEG2000. The best results are depicted in bold. The sliding window model obtains the lowest rate for almost all images, achieving improvements about 0.1 bps with respect to JPEG2000. Again, these results suggest that this model is highly efficient at high rates, when more data are available. This is also seen through the gains obtained for the X-ray and AVIRIS images, which are slightly higher than those achieved for the natural and aerial images because X-ray and AVIRIS images have higher bit-depths. The state machine model achieves inferior performance to that of JPEG2000, similar to that achieved by the original stationary model. HT JPEG2000 obtains lower performance, suggesting that it is more indicated for lossy regimes.

Third, computational complexity is analyzed. It is approximated as the execution time spent by the bitplane and entropy coder when executed with a single thread.[3] This provides an idea of the complexity burden imposed by each model as compared to the stationary, which is the simplest computationally. Our experience indicates that this burden is similar to that achieved with GPU implementations. This comparison only considers the complexity of the probability models proposed for BPC-PaCo to evaluate its complexity differences, leaving apart JPEG2000 and HT JPEG2000 since they utilize different scanning orders and entropy coders. A workstation with an i7-6700K CPU at 4.00 GHz and 32 GB of DDR4 RAM is employed to carry out these

[3] The lowest execution time of 10 execution runs is employed, since that corresponds with the execution that is less disrupted.

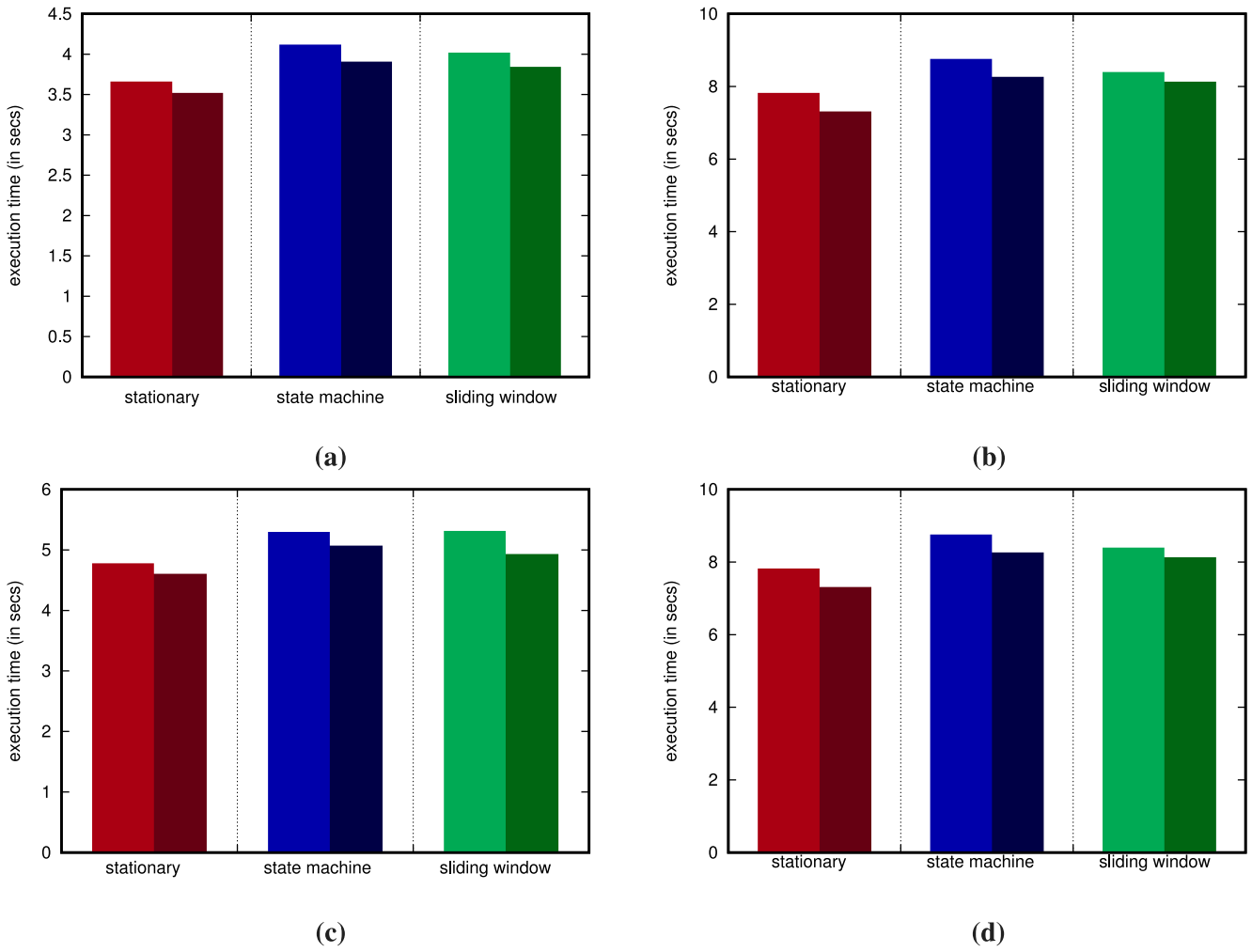

Fig. 5. Evaluation of computational complexity for different corpora: (a) ISO12640-1, (b) aerial, (c) X-ray, and (d) AVIRIS.

tests. Fig. 5 shows the average results achieved in lossy regime for each corpus. The left and right column depicted for each method represent encoding and decoding time, respectively. Results hold for lossless regimes as well. As seen in the figure, the complexity increase of the sliding window model is slightly lower than that of the state machine, requiring an approximate increase of 10% with respect to the stationary model. In general, encoding takes slightly more time than decoding in all tests due to operations required by the rate–distortion optimization process.

5. Conclusions

This work evaluates two probability models for a wavelet-based, highly-parallel image coding architecture. The main novelty of these models is that they adaptively adjust the probabilities while coding data instead of using training as required in previous work. The first proposed model utilizes a finite state machine to adaptively adjust probabilities. It is inspired in the same mechanism employed in conventional image and video codecs. Experimental results indicate regular coding efficiency, suggesting that this model is not suitable for fine-grain parallelism. The second proposed model employs a sliding window, a completely different mechanism that determines probability estimates based on statistical data from past symbols. Experimental results suggest that the sliding window is well suited for fine-grain parallelism, achieving higher compression efficiency than that of the original BPC-PaCo and, at medium to high rates, than that of JPEG2000. This model obtains gains of approximately 0.1 bps in lossless coding as compared to JPEG2000. In terms of computational complexity, experimental tests indicate a moderate increase of 10%, approximately. Although this work is based on our previous algorithm BPC-PaCo, conclusions may also be extended to other parallel codecs. Future work will implement the sliding window model in a modern GPU to assess its complexity and performance when executed in a highly parallel processor.

CRediT authorship contribution statement

Francesc Aulí-Llinàs: Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Writing – review & editing. **Joan Bartrina-Rapesta:** Methodology, Investigation, Writing – review & editing. **Miguel Hernández-Cabronero:** Methodology, Investigation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Two of the image corpora employed (ISO 12640-1 and NASA AVIRIS) are publicly available, and the other two corpora (aerial and X-ray) are provided by institutions without permission to share them.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science, Innovation and Universities (MICIU) and by the European Regional Development Fund (FEDER) under Grants RTI2018-095287-B-I00 and PID2021-125258OB-I00, by the Catalan Government under Grants 2018-BP-00008 and 2017SGR-463, and by the Horizon 2020 under the Marie Skłodowska-Curie grant agreement #801370.

References

[1] D. Huffman, A method for the construction of minimum redundancy codes, Proc. IRE 40 (1952) 1098–1101.
[2] J. Rissanen, Generalized Kraft inequality and arithmetic coding, IBM J. Res. Dev. 20 (3) (1976) 198–203.
[3] J. Duda, K. Tahboub, N.J. Gadgil, E.J. Delp, The use of asymmetric numeral systems as an accurate replacement for huffman coding, in: Proc. IEEE Picture Coding Symposium, 2015, pp. 65–69.
[4] G. Lakhani, Modified JPEG huffman coding, IEEE Trans. Image Process. 12 (2) (2003) 159–169.
[5] J.-S. Lee, J.-H. Jeong, T.-G. Chang, An efficient method of huffman decoding for MPEG-2 AAC and its performance analysis, IEEE Trans. Speech Audio Process. 13 (6) (2005) 1206–1209.
[6] G. Lakhani, Modifying JPEG binary arithmetic codec for exploiting inter/intra-block and DCT coefficient sign redundancies, IEEE Trans. Image Process. 22 (4) (2013) 1326–1339.
[7] A. Skodras, C. Christopoulos, T. Ebrahimi, The JPEG2000 still image compression standard, IEEE Signal Process. Mag. 18 (5) (2001) 36–58.
[8] D. Marpe, H. Schwarz, T. Wiegand, Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard, IEEE Trans. Circuits Syst. Video Technol. 13 (7) (2003) 620–636.
[9] G.J. Sullivan, J.-R. Ohm, W.-J. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1649–1668.
[10] M. Dyer, D. Taubman, S. Nooshabadi, A.K. Gupta, Concurrency techniques for arithmetic coding in JPEG2000, IEEE Trans. Circuits Syst. I 53 (6) (2006) 1203–1212.
[11] M. Rhu, I.-C. Park, Optimization of arithmetic coding for JPEG2000, IEEE Trans. Circuits Syst. Video Technol. 20 (3) (2010) 446–451.
[12] F. Auli-Llinas, M.W. Marcellin, Stationary probability model for microscopic parallelism in JPEG2000, IEEE Trans. Multimedia 16 (4) (2014) 960–970.
[13] D. Zhou, J. Zhou, W. Fei, S. Goto, Ultra-high-throughput VLSI architecture of H.265/HEVC CABAC encoder for UHDTV applications, IEEE Trans. Circuits Syst. Video Technol. 25 (3) (2015) 497–507.
[14] Y. Zhang, C. Lu, A highly parallel hardware architecture of table-based CABAC bit rate estimator in an HEVC intra encoder, IEEE Trans. Circuits Syst. Video Technol. 29 (5) (2019) 1544–1558.
[15] D. Karwowski, Precise probability estimation of symbols in VVC CABAC entropy encoder, IEEE Access 9 (2021) 65361–65368.
[16] D. Taubman, A. Naman, R. Mathew, High throughput block coding in the HTJ2K compression standard, in: Proc. IEEE International Conference on Image Processing, 2019, pp. 1079–1083.
[17] F. Auli-Llinas, P. Enfedaque, J.C. Moure, V. Sanchez, Bitplane image coding with parallel coefficient processing, IEEE Trans. Image Process. 25 (1) (2016) 209–219.
[18] C. de Cea-Dominguez, J.C. Moure, J. Bartrina-Rapesta, F. Auli-Llinas, GPU-oriented architecture for an end-to-end image/video codec based on JPEG2000, IEEE Access 8 (2020) 68474–68487.
[19] A. Said, W.A. Pearlman, A. new, fast, And efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circuits Syst. Video Technol. 6 (3) (1996) 243–250.
[20] W.A. Pearlman, A. Islam, N. Nagaraj, A. Said, Efficient, Low-complexity image coding with a set-partitioning embedded block coder, IEEE Trans. Circuits Syst. Video Technol. 14 (11) (2004) 1219–1235.
[21] D. Taubman, High performance scalable image compression with EBCOT, IEEE Trans. Image Process. 9 (7) (2000) 1158–1170.
[22] D.S. Taubman, M.W. Marcellin, JPEG2000 Image Compression Fundamentals, Standards and Practice, Kluwer Academic Publishers, Norwell, Massachusetts 02061 USA, 2002.
[23] F. Auli-Llinas, M.W. Marcellin, Scanning order strategies for bitplane image coding, IEEE Trans. Image Process. 21 (4) (2012) 1920–1933.
[24] F. Auli-Llinas, Stationary probability model for bitplane image coding through local average of wavelet coefficients, IEEE Trans. Image Process. 20 (8) (2011) 2153–2165.
[25] R.W. Buccigrossi, E.P. Simoncelli, Image compression via joint statistical characterization in the wavelet domain, IEEE Trans. Image Process. 8 (12) (1999) 1688–1701.
[26] F. Auli-Llinas, M.W. Marcellin, J. Serra-Sagrista, J. Bartrina-Rapesta, Lossy-to-lossless 3D image coding through prior coefficient lookup tables, ELSEVIER Inf. Sci. 239 (1) (2013) 266–282.
[27] J.L. Mitchell, W.B. Pennebaker, Software implementations of the Q-Coder, IBM J. Res. Dev. 32 (1988) 753–774.
[28] E. Belyaev, A. Turlikov, K. Egiazarian, M. Gabbouj, An efficient adaptive binary arithmetic coder with low memory requirement, IEEE J. Sel. Top. Signal Process. 7 (6) (2013) 1053–1061.
[29] F. Auli-Llinas, Context-adaptive binary arithmetic coding with fixed-length codewords, IEEE Trans. Multimedia 17 (8) (2015) 1385–1390.
[30] F. Auli-Llinas, BOI codec, 2022, URL https://deic.uab.cat/~francesc/software/boi.
[31] D. Taubman, Kakadu software, 2022, URL http://www.kakadusoftware.com.