Bitplane Image Coding with Parallel Coefficient Processing

Francesc Aulí-Llinàs, *Senior Member, IEEE*, Pablo Enfedaque, Juan C. Moure, and Victor Sanchez, *Member, IEEE*

***Abstract*—Image coding systems have been traditionally tailored for Multiple Instruction, Multiple Data (MIMD) computing. In general, they partition the (transformed) image in codeblocks that can be coded in the cores of MIMD-based processors. Each core executes a sequential flow of instructions to process the coefficients in the codeblock, independently and asynchronously from the others cores. Bitplane coding is a common strategy to code such data. Most of its mechanisms require sequential processing of the coefficients. The last years have seen the upraising of processing accelerators with enhanced computational performance and power efficiency whose architecture is mainly based on the Single Instruction, Multiple Data (SIMD) principle. SIMD computing refers to the execution of the same instruction to multiple data in a lockstep synchronous way. Unfortunately, current bitplane coding strategies can not fully profit from such processors due to inherently sequential coding task. This paper presents bitplane image coding with parallel coefficient processing (BPC-PaCo), a coding method that can process many coefficients within a codeblock in parallel and synchronously. To this end, the scanning order, the context formation, the probability model, and the arithmetic coder of the coding engine have been re-formulated. Experimental results suggest that the penalization in coding performance of BPC-PaCo with respect to traditional strategies is almost negligible.**

***Index Terms*—Bitplane image coding, Single Instruction Multiple Data (SIMD), JPEG2000.**

I. Introduction

OVER the past 20 years, the computational complexity of image coding systems has been increased notably. Codecs of the early nineties were based on computationally simple techniques like the discrete cosine transform (DCT) and Huffman coding [1]. Since then, techniques have been sophisticated to provide higher compression efficiency and enhanced features. Currently, image compression standards such as JPEG2000 [2] or HEVC intra-coding [3] employ complex algorithms that transform and scan the image multiple times. This escalation in computational complexity continues in each new generation of coding systems.

In general, modern coding schemes tackle the computational complexity by means of fragmenting the image in sets of (transformed) samples, called codeblocks, that do not hold (or hold in a well-orderly way) dependencies among them. Each codeblock [4], or group of codeblocks [5], can be coded independently from the others employing the innermost algorithms of the codec. These algorithms scan the samples repetitively, producing symbols that are fed to an entropy coder. Key in such a system is the context formation and the probability model, which determine probability estimates employed by the entropy coder. Commonly, the samples are visited in a sequential order so that the probability model can adaptively adjust the estimates as more data are coded. In many image coding systems [6]–[10], these algorithms employ bitplane coding strategies and context-adaptive arithmetic coders.

Modern Central Processing Units (CPUs) are mainly based on the Multiple Instruction, Multiple Data (MIMD) principle. They have multiple cores, each able to execute a flow of instructions independently and asynchronously from the others. CPUs handle well the computational complexity of image coding systems. The tasks of the image codec are straightforwardly mapped to the CPU: each codeblock is assigned to a core that runs a bitplane coding engine. This parallel processing of codeblocks is called *macro*scopic parallelism [4].

*Micro*scopic parallelism refers to parallel strategies of data coding within a codeblock. There are few such strategies due to the difficulty to unlock the data dependencies that arise when the coefficients are processed in a sequential fashion. Also, because most codecs are tailored for their execution in CPUs, so parallelization in the bitplane coding stage is not appealing. It has not been until recent years that microscopic parallelism has become attractive due to the upraising of accelerators, which are processors mainly based on the Single Instruction, Multiple Data (SIMD) principle. The main idea behind SIMD computing is to execute a flow of instructions to multiple data in parallel and synchronously. This architectural principle permits to increase the number of instructions simultaneously executed by an order of magnitude while lowering the power consumption. Nowadays, the Graphics Processing Units (GPUs) are the main representatives of such processors.

The fine level of parallelism required for SIMD computing can only be achieved in image coding systems via microscopic parallel strategies. Even so, the current trend is to implement *already developed* coding schemes for their execution in GPUs. Without aiming to be exhaustive, GPU implementations of JPEG2000 are found in [11]–[14] and there exist commercial products like [15] as well. The JPEG XR standard is implemented in [16], and video coding standards are studied in [17], [18]. Other coding schemes such as EBCOT and

Francesc Aulí-Llinàs and Pablo Enfedaque are with the Department of Information and Communications Engineering, Universitat Autònoma de Barcelona, Spain (phone: +34 935811861; fax: +34 935813443; e-mail: {fauli | pablo}@deic.uab.cat). Juan C. Moure is with the Department of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Spain (e-mail: juancarlos.moure@uab.cat). Victor Sanchez is with the Department of Computer Science, The University of Warwick, United Kingdom (e-mail: vsanchez@dcs.warwick.ac.uk). This work has been partially supported by the Spanish Government (MINECO), by FEDER, and by the Catalan Government, under Grants RYC-2010-05671, UAB-472-02-2/2012, TIN2012-38102-C03-03, TIN2011-28689-C02-1, and 2014SGR-691.

wavelet lower trees are also implemented in GPUs in [19] and [20], respectively. Such implementations reduce the execution time of CPU-based implementations. Nonetheless, none of them can fully exploit the resources of the GPU due to the aforementioned sequential coefficient processing.

This paper introduces bitplane image coding with parallel coefficient processing (BPC-PaCo), a wavelet-based coding strategy tailored for SIMD computing. To this end, a new scanning order, context formation, probability model, and arithmetic coder are devised. All the proposed mechanisms permit the processing of the samples in parallel or sequentially, allowing efficient implementations for both SIMD and MIMD computing. The coding performance achieved by the proposed method is similar to that of JPEG2000. This paper describes the employed techniques and assesses their performance from an image coding perspective. Future work will describe its implementation in a GPU appraising the computational throughput. This paper extends our preliminary work [21] with a sequential version of the algorithm, more experimental data, and a revised and more descriptive text.

The remainder of the paper is structured as follows. Section II provides preliminary concepts. Section III describes the proposed bitplane coding strategy. Section IV assesses its coding performance through experimental results carried out for four different corpora of images. The last section concludes with a brief summary.

II. Preliminaries

The proposed bitplane coding strategy can be employed in any wavelet-based compression scheme. We adopt the framework of JPEG2000 due to its excellent coding performance and advanced features. A conventional JPEG2000 implementation is structured in three main coding stages [4]: data transformation, data coding, and codestream re-organization. The first stage applies the wavelet transform and quantizes wavelet coefficients. This represents approximately 15$\sim$20% of the overall coding task and does not pose a challenge for its implementation in SIMD architectures [22]–[28]. After data transformation, the image is partitioned in small sets of wavelet coefficients, the so-called codeblocks. Data coding is carried out in each codeblock independently. It represents approximately 70$\sim$75% of the coding task. The routines employed in this stage are based on bitplane coding and context-adaptive arithmetic coding. The last stage re-organizes the final codestream in quality layers that include segments of the bitstreams produced for each codeblock in the previous stage. Commonly, the codestream re-organization is carried out employing rate-distortion optimization techniques [29]–[31], representing less than 10% of the coding task.

Bitplane coding strategies work as follows. Let $[b_{M-1}, b_{M-2}, ..., b_1, b_0]$, $b_i \in \{0, 1\}$ be the binary representation of an integer v which represents the magnitude of the index obtained by quantizing wavelet coefficient ω, with M being a sufficient number of bits to represent all coefficients. The collection of bits b_j from all coefficients is called a bitplane. Bits are coded from the most significant bitplane $M-1$ to the least significant bitplane 0. The first non-zero bit of the binary representation of v is denoted by b_s and is referred to as the significant bit. The sign of the coefficient is denoted by $d \in \{+, -\}$ and is coded immediately after b_s, so that the decoder can begin approximating ω as soon as possible. The bits $b_r,\ r < s$ are referred to as refinement bits.

JPEG2000 codes each bitplane employing three coding passes [4] called significance propagation pass (SPP), magnitude refinement pass (MRP), and cleanup pass (CP). The SPP and CP perform significance coding. They visit those coefficients that did not become significant in previous bitplanes, coding whether they become significant in the current bitplane or not. The difference between them is that the SPP visits coefficients that are more likely to become significant. The MRP refines the magnitude of coefficients that became significant in previous bitplanes. The order of the coding passes in each bitplane is SPP, MRP, and CP except for the most significant bitplane, in which only the CP is applied. This three coding pass scheme is convenient for rate-distortion optimization purposes [10].

With regard to SIMD architectures, it is worth knowing that they execute vector instructions. The vector unit (i.e., the hardware component that executes the vector instructions) is composed of T replicated lanes, each producing a different data output element. A vector instruction is processed by simultaneously executing the same operation in all the lanes of the unit. GPUs adopt a convenient programming model that simplifies the details of SIMD computing. In GPUs, the lanes of a vector unit are abstracted as individual threads that execute a flow of instructions in a lockstep synchronous way. If the execution flow diverges (due to conditionals), the divergent paths are executed sequentially one after another. In general, divergent paths are to be minimized.

III. Proposed bitplane coding strategy

A parallel bitplane coding strategy must be deterministic, i.e., the parallel execution must unambiguously correspond to an equivalent sequential execution. The codestream generated or processed by both the parallel and sequential versions of the algorithm must be the same. Three mechanisms of the bitplane coder have been re-formulated keeping in mind this purpose: the scanning order, the context formation and its probability model, and the arithmetic coder.

A. Scanning order

Scanning orders visit coefficients employing a pre-defined sequence. Typical sequences are row by row or column by column [5], in zig zag [32], using stripes of 4 rows that are scanned from left to right [2], or via quadtree strategies [8]. Regardless of the scanning sequence, all methods visit coefficients in a consecutive fashion, which prevents parallelism while executing a coding pass. The only way to achieve microscopic parallelism in current bitplane coding engines is to execute coding passes in parallel. JPEG2000, for instance, provides the RESET, RESTART, and CAUSAL coding variations to achieve it. The main problem of coding pass parallelism is that in order to code a coefficient in the current pass, some information of its neighbors coded in previous passes may be needed. This is addressed by

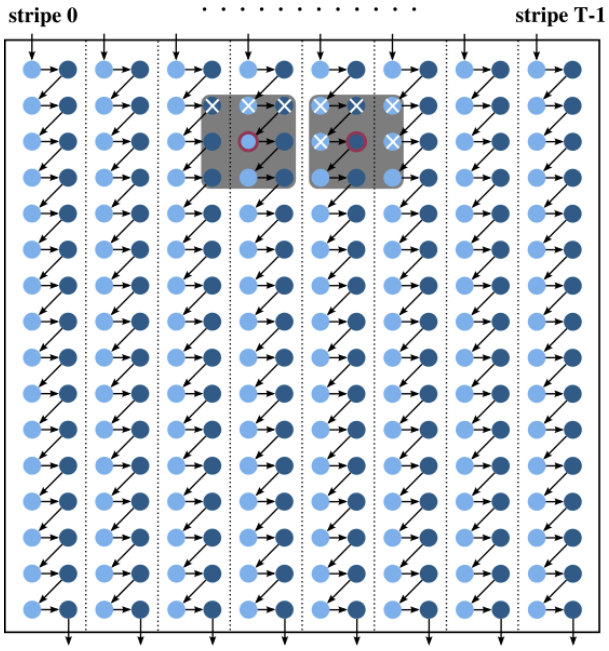
delaying the beginning of the execution of each coding pass some coefficients with respect to its immediately previous pass [4], [33]. Such an elaborate strategy is not suitable for SIMD computing since each coding pass carries out different operations, which generates divergence among threads.

The proposed method achieves microscopic parallelism by means of coding T coefficients in parallel during the execution of a coding pass. Sets of T threads perform the same operation to different coefficients, so vector instructions can be naturally mapped to process each codeblock. Fig. 1(a) depicts the scanning order employed. The light- and dark-blue dots in the figure represent the coefficients within a codeblock. The coefficients are organized in vertical stripes that contain two columns. Each stripe is processed by a thread. Coefficients are scanned from the top to the bottom row, and from the left to the right coefficient. All coefficients in the same position of the stripes are processed at the same time.

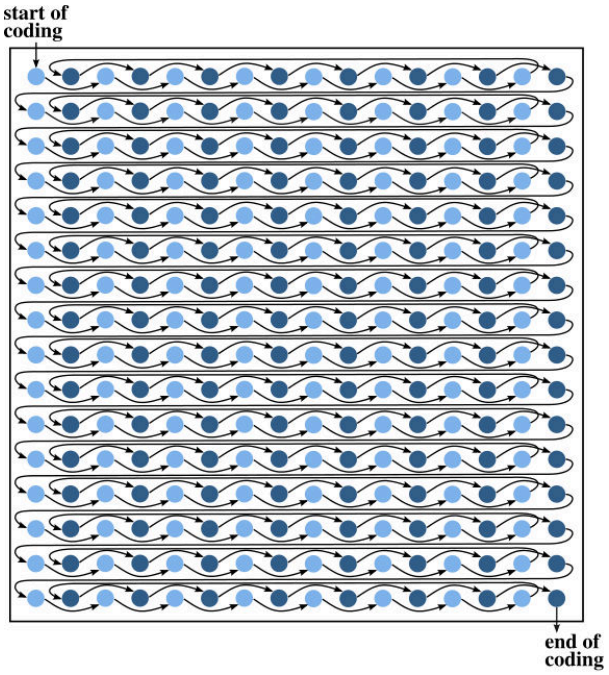
The scanning order of Fig. 1(a) is highly efficient for context formation purposes. Let us explain further. As seen in the following section, the context of a coefficient is determined via its eight adjacent neighbors. All information coded in previous passes is available when forming the context since such information has been already transmitted to the decoder. Also, information coded in the current coding pass that belongs to those neighbors visited before the current coefficient can also be employed. This information is valuable since it helps to predict with higher precision the symbols coded. The higher the Average number of already Visited Neighbors in the current coding Pass (AVNP), the better the coding performance. The AVNP is computed without considering those coefficients in the border of the codeblock. Fig. 1(a) depicts in gray the eight adjacent neighbors of two coefficients, one in the left and the other in the right column of a stripe. The coefficient for which the context is formed is depicted with a red circle. The neighbors that were already visited in the current coding pass are depicted with a white cross. The coefficients in the left column (depicted in light blue) have 3 already visited neighbors, whereas the coefficients in the right column have 5. So the AVNP achieved by the proposed scanning order is 4. JPEG2000 and other coding systems employing sequential scanning orders also achieve an AVNP of 4.

The sequential version of the proposed scanning order is depicted in Fig. 1(b). As seen in the figure, all light-blue coefficients of a row are visited first from left to right, followed by the dark-blue coefficients. The same routine is carried out in each row, from the top to the bottom of the codeblock. Since the parallel operation is synchronous and deterministic, the context formation resulting from the parallel and sequential version of this scanning order is identical. This scanning order does not use fast-coding primitives such as the run mode of JPEG2000 since they do not provide significant coding gains when employed with the proposed probability model [10].

The scanning order of BPC-PaCo is employed with the same coding passes as those defined in JPEG2000. Though other schemes may be utilized, the three-coding pass strategy of JPEG2000 is adopted herein due to its high coding efficiency [10]. The number of significant bitplanes coded for each codeblock are signaled in the headers of the codestream.



(a)



(b)

Fig. 1: Illustration of the proposed scanning order for (a) parallel and (b) sequential processing.

B. Context formation and probability model

The contexts employed for significance coding use the significance state of the eight adjacent neighbors of coefficient ω. The neighbors of ω are denoted by ω^k, with $k \in \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow\}$ referring to the neighbor in the top, top-right, right,... position, respectively. The magnitude of the quantization index of these neighbors is denoted by υ^k. The

significance state of υ^k in bitplane j is denoted by $\Phi(\upsilon^k, j)$. It is 1 when its significance bit (i.e., b_s) has already been coded. Clearly, this definition includes all neighbors that became significant in bitplanes higher than the current, i.e., $\Phi(\upsilon^k, j) = 1$ if $s > j$. It also includes the neighbors that become significant in the current bitplane –and that are already visited in the current coding pass–, i.e., $\Phi(\upsilon^k, j) = 1$ if $s = j$ and υ^k is already visited. Otherwise, $\Phi(\upsilon^k, j) = 0$.

The contexts employed for significance coding are denoted by $\phi_{sig}(\cdot)$. They are computed as the sum of the significance state of the eight adjacent neighbors of ω, more precisely, the context of υ at bitplane j is computed as

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

Therefore, $\phi_{sig}(\cdot) \in \{0, ..., 8\}$. Although other works in the literature [7], [9], [34] determine the context depending on the position of the significance neighbors, the analysis in [35] shows that simple context formation approaches like (1) also achieve competitive coding performance. This approach is employed herein due to its computational simplicity.

The contexts employed for sign coding are similar to those of JPEG2000 since they obtain high efficiency. Sign contexts employ the sign of the neighbors in the vertical and horizontal positions. Let $\chi(\omega^k, j)$ represent the sign of ω^k when coding bitplane j. $\chi(\omega^k, j)$ is 0 if the coefficient is not significant, otherwise is 1 and -1 for positive and negative coefficients, respectively. Then, $\chi^V = \chi(\omega^{\uparrow}, j) + \chi(\omega^{\downarrow}, j)$ and $\chi^H = \chi(\omega^{\leftarrow}, j) + \chi(\omega^{\rightarrow}, j)$. Context $\phi_{sign}(\omega, j)$ is computed according to

$$\phi_{sign}(\omega, j) = \begin{cases} 0 & \text{if } (\chi^V > 0 \text{ and } \chi^H > 0) \text{ or} \\ & \quad (\chi^V < 0 \text{ and } \chi^H < 0) \\ 1 & \text{if } \chi^V = 0 \text{ and } \chi^H \neq 0 \\ 2 & \text{if } \chi^V \neq 0 \text{ and } \chi^H = 0 \\ 3 & \text{otherwise} \end{cases} . \tag{2}$$

Contexts for refinement coding should be based on computationally intensive techniques such as the local average, or otherwise use only one context for all refinement bits, as suggested in [35]. Herein, the latter approach is used for computational simplicity, so $\phi_{ref}(\upsilon, j) = 0$.

The contexts are employed together with the probability model to determine the probability estimate that is fed to the arithmetic coder. Conventional probability models adaptively adjust the probability estimates of the symbols as more data are coded. Such models are convenient since they are computationally simple, achieve high compression efficiency, and avoid a pre-processing step to collect statistics of the data. Compression standards such as JBIG [36], JPEG2000 [2], and HEVC [3] employ them. Unfortunately, context-adaptive models cannot be employed herein. To do so, the probability adaptation should be carried out for all data of the codeblock, which is not possible due to the parallel processing of coefficients. Such models achieve poor performance when coding short sequences [33], so to use them independently for each stripe is not effective.

The proposed bitplane coder employs a stationary probability model that uses a fixed probability for each context and bitplane. As shown in [33], this model is based on the empirical evidence that the probabilities employed to code all symbols with a context are mostly regular in the same bitplane. The probability estimates are precomputed off-line and stored in a lookup table (LUT) that is known by the encoder and the decoder, so there is no need to transmit it. The LUT contains one probability estimate per context and bitplane for each wavelet subband. It is accessed as $\mathcal{P}_u[j][\phi_{\{sig|sign|ref\}}(\cdot)]$, providing the probability of the symbol coded. u denotes the wavelet subband. Note that such a probability model does not need the adaptive probability tables employed in context-adaptive arithmetic coders such as the MQ.

The probability estimates needed to populate the LUTs are determined as follows. Let $F_u(\upsilon \mid \phi_{sig}(\upsilon, j))$ denote the probability mass function (pmf) of the quantization indices at bitplane j given their significance context. This pmf is computed for each wavelet subband using the data from all images in a training set. Its support is $[0, ..., 2^{j+1} - 1]$ since it contains quantization indices that were not significant in bitplanes greater than j. The probability estimates used to populate the LUTs are generated by integrating the pmfs to obtain the probabilities of emitting 0 or 1 in the corresponding contexts. Let us denote the probability that b_j is 0 during significance coding by $P_{sig}(b_j = 0 \mid \phi_{sig}(\upsilon, j))$. This probability is determined from the corresponding pmf according to

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

The probability estimates for refinement and sign coding are derived similarly. The LUT is different for each image type since the probability model exploits the fact that the data produced after transforming images of the same type (e.g., natural, medical, etc.) with the same wavelet filter-bank are statistically similar [35], [37], [38]. A more in-depth study on this stationary probability model can be found in [33].

C. Arithmetic coder

The symbol and its probability estimate are fed to an arithmetic coder. Conventional arithmetic coding works as follows. The coder begins by segmenting the interval of real numbers $[0, 1)$ into two subintervals. The size of the subintervals is chosen according to the probability estimate of the symbol. The first symbol is coded by selecting its

corresponding subinterval. Then, this procedure is repeated within the selected subintervals for the following symbols. The transmission of any number within the range of the final subinterval guarantees that the reverse procedure decodes the original message losslessly. The number transmitted is generally referred to as codeword.

Most arithmetic coders employed for image compression produce variable-to-variable length codes. This is, a variable number of input symbols are coded with a codeword of a priori unknown length. In JPEG2000, for instance, all data of a codeblock is coded with a single –and commonly very long– codeword. Practical realizations of arithmetic coders operate with hardware registers of 16 or 32 bits, so the generation of the codeword is carried out progressively. Roughly described, this is done as follows. Let $[L, R)$ denote the current interval of the coder, with L and R being the fractional part of the left and right boundaries of the interval stored in hardware registers. Assume that the leftmost bits of the binary representations of L and R are not equal in the current interval. When a new symbol is coded, this interval is further reduced to $[L', R')$. If the leftmost bits of L' and R' are then equal, all following segmentations of the interval will also start with those same bit(s) since $L \leq L' \leq \ldots \leq R' \leq R$. This permits to dispatch the leftmost bits of L' and R' that are identical and to shift the remaining bits of the registers to the left. This procedure is called renormalization.

Two aspects of conventional arithmetic coding prevent its use in the proposed bitplane coding strategy. The first is the generation of a single codeword. The scanning order described above utilizes T threads that code data in parallel. Forcing them to produce a single codeword would require to code their output in a sequential order, ruining the parallelism. The second aspect is the computational complexity of current arithmetic coders. Part of this complexity is due to the renormalization procedure, which requires conditionals and repositioning operations as explained before.

These aspects are addressed herein by means of a new technique that employs multiple arithmetic coders that work in parallel and generate fixed-length codewords that are optimally positioned in the bitstream. As previously described, each thread codes all data of a stripe. The coefficients coded by a thread are visited in a sequential order, so an arithmetic coder can be *individually* employed to code all symbols emitted for a stripe. Instead of using conventional arithmetic coding, we employ an arithmetic coder that generates codewords of fixed length [39]–[43]. Variable-to-fixed length arithmetic coding avoids renormalization, reducing the complexity of the coder [43]. It uses an integer interval with a pre-defined range, say $[0, 2^W - 1]$ with W being the length of the codeword (in bits). The division of the interval is carried out in a similar way as with conventional arithmetic coding until its size is less than 2. Then, the number within the last interval is dispatched to the bitstream and a new interval is set (see below).

The codewords produced in each stripe are sorted generating a *single* quality-embedded bitstream for all stripes that can be truncated at any point so that the quality of the recovered image is maximized. Such a bitstream is similar to that produced by conventional image codecs, so it can be employed

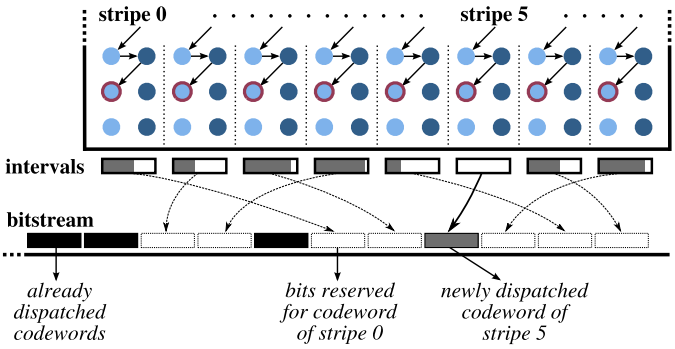


Fig. 2: Illustration of the sorting technique employed to situate the codewords in the bitstream when encoding.

in the same framework of rate-distortion optimization defined in JPEG2000 to construct layers of quality and/or different progression orders [4], [30]. In the encoder, the bitstream is constructed as follows. Each time that a thread initializes its interval (because is the beginning of coding or because the interval is exhausted and a new symbol needs to be coded), W bits are reserved at the end of the bitstream. This space is reserved –but it is not filled– at this instant because the interval of the thread has just been initialized, so the codeword is still not available. After coding some symbols (possibly from different coding passes), the interval of this thread is exhausted, so its codeword is put in the reserved space. Fig. 2 illustrates an example of this sorting technique. All stripes in the figure have its own space in the bitstream, which was reserved when needed. The coefficients depicted with a red circle are those currently visited. When the thread processing the fifth stripe emits its symbol, it exhausts its interval, so the codeword is put in the space that was reserved for this thread. Note that this thread does *not* reserve a new space at the end of the bitstream at this instant but it will do it when coding a new symbol. Evidently, if two or more threads need to reserve space at the same instant, some priority must be employed. In order to provide determinism, stripes on the left have higher priority. When the coding of the codeblock data finishes, the arithmetic coders put their codewords in the bitstream, without needing a byte flush operation.

As previously stated, the order in which the codewords are sorted minimizes the distortion at any truncation point. This can be seen from the perspective of the decoder. All the threads need a non-exhausted interval to decode the data of their corresponding stripes. The first thread that –while decoding– exhausts its interval stops the whole decoding procedure for that codeblock since all threads are synchronized. The codewords are sorted so that, at any instant of the decoding, the thread that exhausts its interval and needs to decode a new symbol can found its immediately next codeword at the immediately next position of the bitstream. In other words, any thread of the decoder only needs to read the next W bits of the bitstream when its interval is exhausted *and* a new symbol is to be decoded. This decodes the maximum amount of data for any given segment of the bitstream, thus the distortion of the reconstructed coefficients is minimized.

The proposed arithmetic coding technique slightly penalizes

the coding performance with respect to an implementation that produces a single codeword. This is because either if the bitstream is truncated for rate-distortion optimization purposes, or if it is fully transmitted, the last codeword that is read for each stripe may contain some bits that are not really needed to decode the data of the corresponding coding pass. Since the proposed strategy utilizes T stripes, these excess bits may not be negligible. The penalization in coding performance decreases as more data are coded in each stripe. We found that the coding of two columns is a good tradeoff between coding performance and parallelism. Evidently, the implementation of the proposed method in hardware architectures such as FPGAs requires an arithmetic coder per stripe. Replication is a common strategy to obtain high performance codecs [44].

D. Algorithm

The encoding procedures of BPC-PaCo are embodied in Algorithm 1. One procedure per coding pass is specified. These procedures detail the operations carried out for a stripe. The "ACencode" procedure describes the operations of the arithmetic coder. The scanning order is specified in the first two lines of the "SPP", "MRP", and "CP" procedures. The (quantized) coefficient visited is denoted by $(\upsilon_{y,x})$ $\omega_{y,x}$, with y, x indicating its row and column within the codeblock, respectively. The SPP and CP check whether the visited coefficient is significant in previous bitplanes or not. If not, they code bit b_j of the quantized coefficient. The SPP only visits coefficients that have at least one significant neighbor (i.e., those that have $\phi_{sig}(\upsilon_{y,x}, j) \neq 0$), whereas the CP visits all non-significant coefficients that were not coded by the SPP. The MRP codes the bit b_j of all coefficients that became significant in previous bitplanes.

The "ACencode" procedure codes all symbols emitted. The interval of stripe t is stored in registers $L[t]$ and $S[t]$, which are the left boundary and the size minus one of the interval, respectively. Since the length of the codewords is W, both $L[t]$ and $S[t]$ are integers in the range $[0, 2^W - 1]$. The codeword is dispatched to the bitstream in lines 13-15 of this procedure when the interval is exhausted. Note that when $S[t] = 0$, $L[t]$ represents the final number within the interval or, in other words, the emitted codeword. If a new symbol is coded and $S[t] = 0$, the procedure reserves W bits and sets $L[t] \leftarrow 0$ and $S[t] \leftarrow 2^W - 1$ (see lines 1-5).

The interval division is carried out in lines 6-12. When the symbol is 0 or $-$, the lower subinterval is kept, so the interval size is reduced to

$$S[t] \leftarrow (S[t] \cdot p) \gg \widehat{\mathcal{P}} \ , \tag{4}$$

and $L[t]$ is left unmodified. $\gg$ above denotes a bit shift to the right. p is the probability of the symbol to be $0/+$ expressed in the range $[0, 2^{\widehat{\mathcal{P}}} - 1]$, determined according to

$$p = \lfloor P_{sig}(b_j = 0 \mid \phi_{sig}(\upsilon, j)) \cdot 2^{\widehat{\mathcal{P}}} \rfloor \tag{5}$$

for significance coding, and equivalently for refinement and sign coding. $\lfloor \cdot \rfloor$ denotes the floor operation. As seen in Algorithm 1, p is the value that is stored in the LUTs, so (5) is computed off-line. $\widehat{\mathcal{P}}$ is the number of bits employed to express the symbol's probability. The result of the multiplication in (4) (i.e., $(S[t] \cdot p)$) must not cause arithmetic overflow in the hardware registers, so $W + \widehat{\mathcal{P}} \leq 64$ in modern architectures. Experimental evidence indicates that $16 \leq W \leq 32$ and $\widehat{\mathcal{P}} \geq 7$ achieve competitive performance. In our implementation $W = 16$ and $\widehat{\mathcal{P}} = 7$.

The coding of $1/+$ keeps the upper subinterval, so

$$\begin{aligned} L[t] &\leftarrow L[t] + ((S[t] \cdot p) \gg \widehat{\mathcal{P}}) + 1 \ , \text{ and} \\ S[t] &\leftarrow S[t] - ((S[t] \cdot p) \gg \widehat{\mathcal{P}}) - 1 \ . \end{aligned} \tag{6}$$

Algorithm 1 BPC-PaCo encoding procedures
Initialization: $S[t] \leftarrow 0 \ \ \forall \ \ 0 \leq t < T$

SPP (u subband, j bitplane, t stripe)
1: **for** $y \in [0, \text{numRows} - 1]$ **do**
2: **for** $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
3: **if** $\upsilon_{y,x}$ is not significant **AND** $\phi_{sig}(\upsilon_{y,x}, j) \neq 0$ **then**
4: ACencode(b_j, $\mathcal{P}_u[j][\phi_{sig}(\upsilon_{y,x}, j)]$, t)
5: **if** $b_j = 1$ **then**
6: ACencode(d, $\mathcal{P}_u[j][\phi_{sign}(\omega_{y,x}, j)]$, t)
7: **end if**
8: **end if**
9: **end for**
10: **end for**

MRP (u subband, j bitplane, t stripe)
1: **for** $y \in [0, \text{numRows} - 1]$ **do**
2: **for** $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
3: **if** $\upsilon_{y,x}$ is significant in $j' > j$ **then**
4: ACencode(b_j, $\mathcal{P}_u[j][\phi_{ref}(\upsilon_{y,x}, j)]$, t)
5: **end if**
6: **end for**
7: **end for**

CP (u subband, j bitplane, t stripe)
1: **for** $y \in [0, \text{numRows} - 1]$ **do**
2: **for** $x \in [t \cdot 2, t \cdot 2 + 1]$ **do**
3: **if** $\upsilon_{y,x}$ is not significant **AND** not coded in SPP **then**
4: ACencode(b_j, $\mathcal{P}_u[j][\phi_{sig}(\upsilon_{y,x}, j)]$, t)
5: **if** $b_j = 1$ **then**
6: ACencode(d, $\mathcal{P}_u[j][\phi_{sign}(\omega_{y,x})]$, t)
7: **end if**
8: **end if**
9: **end for**
10: **end for**

ACencode (c symbol, p probability, t stripe)
1: **if** $S[t] = 0$ **then**
2: Reserve the next W bits of the bitstream
3: $L[t] \leftarrow 0$
4: $S[t] \leftarrow 2^W - 1$
5: **end if**
6: **if** $c = 0$ **OR** $c = -$ **then**
7: $S[t] \leftarrow (S[t] \cdot p) \gg \widehat{\mathcal{P}}$
8: **else**
9: $f \leftarrow ((S[t] \cdot p) \gg \widehat{\mathcal{P}}) + 1$
10: $L[t] \leftarrow L[t] + f$
11: $S[t] \leftarrow S[t] - f$
12: **end if**
13: **if** $S[t] = 0$ **then**
14: Put $L[t]$ in reserved space of the bitstream
15: **end if**

Algorithm 2 BPC-PaCo relevant decoding procedures
Initialization: $S[t] \leftarrow 0 \quad \forall \quad 0 \leq t < T$

ACdecode (p probability, t stripe)
1: **if** $S[t] = 0$ **then**
2: $I[t] \leftarrow$ read the next W bits of the bitstream
3: $S[t] \leftarrow 2^W - 1$
4: $L[t] \leftarrow 0$
5: **end if**
6: $f \leftarrow ((S[t] \cdot p) \gg \widehat{\mathcal{P}}) + 1$
7: $g \leftarrow L[t] + f$
8: **if** $I[t] \geq g$ **then**
9: $c \leftarrow 1$ OR $+$
10: $S[t] \leftarrow S[t] - f$
11: $L[t] \leftarrow g$
12: **else**
13: $c \leftarrow 0$ OR $-$
14: $S[t] \leftarrow f - 1$
15: **end if**
16: **return** c

The interval division is carried out via integer multiplications and bit shifts because these are the fastest operations in hardware architectures. Also, because floating point arithmetic should be avoided to prevent incompatibilities with different architectures. An alternative to (4), (6) is the use of LUTs that contain the result of these operations with relative precision, similarly as how it is done in [1], [4], [45]–[48]. Our implementation employs the above operations since they are faster than any other alternative tested.

The decoding procedures of the SPP, MRP, and CP are similar to those of the encoder, so they are not detailed. Algorithm 2describes the decoding procedure of the arithmetic coder. In this procedure, $I[t]$ is the codeword read from the bitstream for stripe t. The procedure is similar to that of the encoder. An extended description of the arithmetic coder employed in Algorithms 1 and 2 can be found in [43].

The sequential version of BPC-PaCo carries out the same instructions detailed above except that the two loops in lines 1 and 2 of the coding passes are replaced by loops that implement the scanning order depicted in Fig. 1(b). The call to "ACencode" or "ACdecode" replaces t by $x/2$, so that each stripe employs a different interval. Also, sign coding is computed slightly different. In the parallel version, it is carried out just after emitting bit b_j. In the sequential version, the sign can *not* be emitted just after b_j since that would produce a different bitstream from that obtained by the parallel algorithm. When the coefficients are coded sequentially, sign coding for the odd (even) coefficients must be carried out just before starting the significance coding of the even (odd) coefficients of the same (next) row. This is necessary to ensure that the codewords are sorted in the bitstream identically in both versions of the algorithm.

The replacement of the original algorithms of JPEG2000 by the proposed bitplane coding strategy does not sacrifice any feature of the coding system. The formation of quality layers, the use of different progression orders, the region of interest coding, or the scalability of the system is unaffected by the use of the proposed strategy.

IV. Experimental Results

Four corpora of images are employed to assess the performance of BPC-PaCo. The first consists of the eight natural images of the ISO 12640-1 corpus (2048×2560, gray scale, 8 bits per sample (bps)). The second is composed of four aerial images provided by the Cartographic Institute of Catalonia, covering vegetation and urban areas (7200×5000, gray scale, 8 bps). The third corpus has three xRay angiography images from the medical community (512×512 with 15 components, 12 bps). The last corpus contains three AVIRIS (Airbone Visible/Infrared Imaging Spectrometer) hyperspectral images provided by NASA (512×512 with 224 components, 16 bps). BPC-PaCo is implemented in the framework of JPEG2000 by replacing the bitplane coding engine and the arithmetic coder of a conventional JPEG2000 codec. The resulting codestream is not compliant with JPEG2000, though it does not undermine any feature of the standard. Our implementation BOI [49] is employed in these experiments. Except when indicated, the coding parameters for all tests are: 5 levels of wavelet transform, codeblocks of 64×64, single quality layer, and no precincts. The 9/7 and the 5/3 wavelet transforms are employed for lossy and lossless regimes, respectively. BPC-PaCo employs the same rate-distortion optimization techniques as those of JPEG2000, which select the coding passes of each codeblock included in the final codestream.

The first test evaluates the coding performance achieved by BPC-PaCo as compared to that of JPEG2000. Fig. 3 depicts the results achieved for the four corpora. The results are reported as the peak signal to noise ratio (PSNR) difference achieved between BPC-PaCo and JPEG2000. The performance of JPEG2000 is depicted as the horizontal straight line in the figures. Results below this line indicate that BPC-PaCo achieves lower PSNR than that of JPEG2000. To avoid cluttering the figure, results for only four of the eight natural images are reported in Fig. 3(a), though similar plots are achieved for the remaining The results of Fig. 3 indicate that, for natural images, the proposed method achieves PSNR values between 0.2 to 1 dB below those of JPEG2000. As it is explained in the previous section and analyzed below, this penalization is mainly due to the use of multiple arithmetic coders. The results achieved by BPC-PaCo for aerial images are between 0.2 to 0.4 dB below those of JPEG2000 at low and medium bitrates, and from 0 to 0.6 dB above those of JPEG2000 at high bitrates. For the corpus of xRay and AVIRIS images, the results are similar to those obtained for aerial images.

For comparison purposes, Fig. 3(a) and 3(b) also report the results when the RESET, RESTART, and CAUSAL coding variations of JPEG2000 are in use when coding the first image of the natural and aerial corpus (i.e., "Portrait" and "forest1"). The results are reported with the plot with dots. We recall that these coding variations are employed to enable coding pass parallelism in JPEG2000 (see Section III-A). When they are in use, the coding performance difference between BPC-PaCo and JPEG2000 is reduced between 0.2 to 0.5 dB.

Table I reports the results achieved when coding all images in lossless mode. The third column of the table reports the bitrate achieved by JPEG2000, in bps. The fourth column

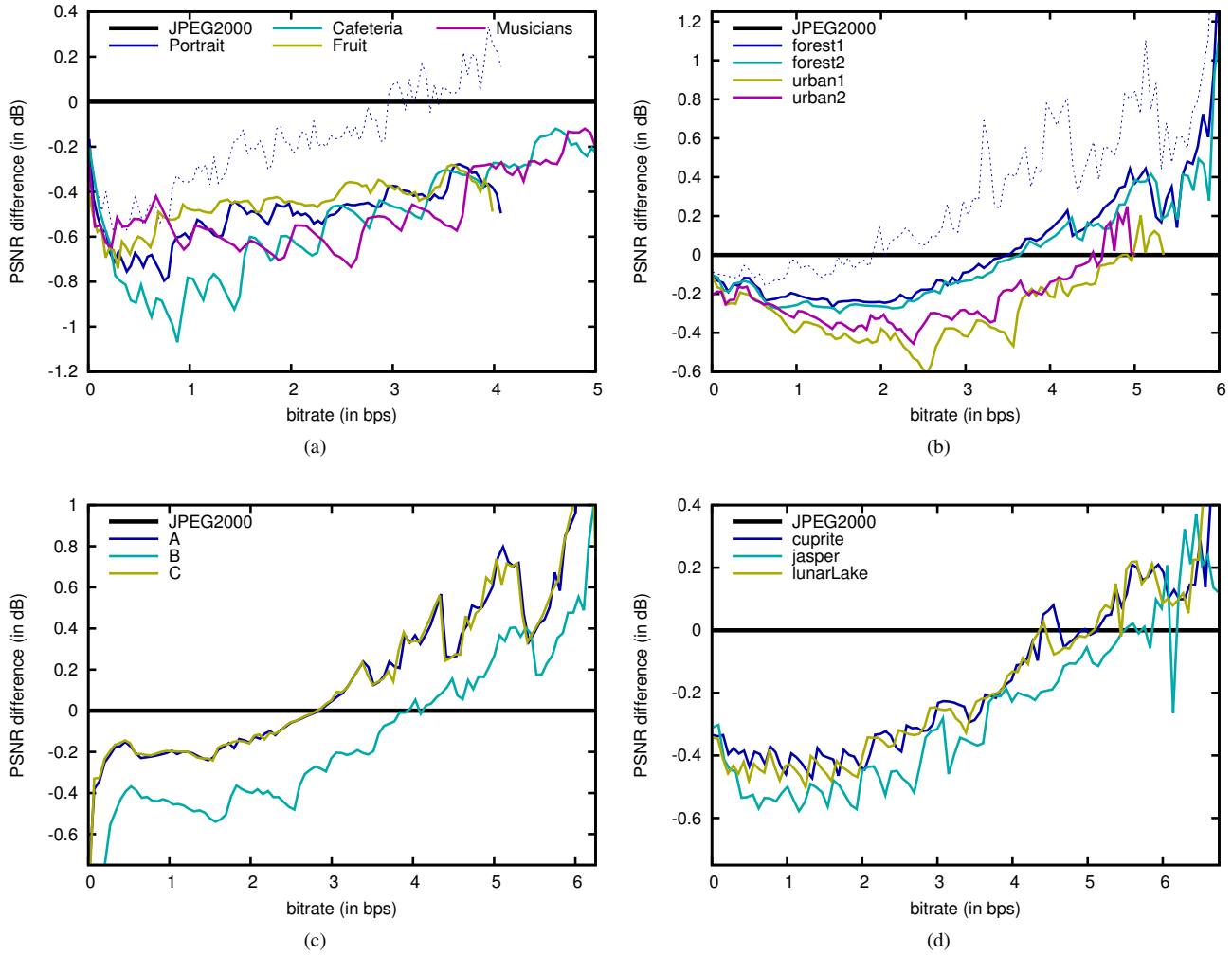


Fig. 3: Evaluation of the lossy coding performance achieved by BPC-PaCo compared to that of JPEG2000. Each subfigure reports the performance achieved for images from a specific corpus: (a) natural, (b) aerial, (c) xRay, and (d) AVIRIS.

reports the bitrate difference between the proposed method and JPEG2000. Again, BPC-PaCo achieves slightly lower and higher compression efficiency than that of JPEG2000 for the corpus of natural images and for the remaining corpora, respectively. On average, BPC-PaCo increases the length of the codestream negligibly.

The aim of the next test is to appraise three key mechanisms of the proposed bitplane coding strategy. To this end, three modifications are carried out to BPC-PaCo. The first replaces its arithmetic coder and utilizes the MQ coder of JPEG2000. The MQ coder employs context-adaptive mechanisms and produces a single codeword for all data coded in a code-block. The second modification compels the arithmetic coder of BPC-PaCo to employ a single codeword for all stripes. Evidently, these two modifications prevent parallelism. Their sole purpose is to appraise the coding efficiency of these two mechanisms. The third modification removes the context formation approach and employs one context for significance coding, one for refinement coding, and one for sign coding. Fig. 4 reports the results obtained for one image of each corpus when these modifications are in use. For comparison purposes, the figure also reports the performance achieved by the original BPC-PaCo. When the MQ coder is employed, the coding performance achieved by BPC-PaCo is almost the same as that of JPEG2000 for all images. This indicates that the scanning order and the context formation employed in BPC-PaCo do not penalize coding performance significantly. Clearly, the use of multiple arithmetic coders producing multiple codewords is the technique mainly responsible for the penalization in compression efficiency. This can also be seen in Fig. 4 via the second modification of BPC-PaCo, which employs a single codeword for all stripes. When this modification is in use, the coding performance of BPC-PaCo is enhanced from 0.25 to 0.5 dB, achieving higher PSNR than that of JPEG2000 for all corpora except the natural. The third modification shows that the proposed context formation approach enhances the coding performance of the proposed method significantly (more than 3 dB in some cases). The results of these modifications are also reported in Table I for the lossless regime. Similar results are achieved for both lossy and lossless regimes.

The last test evaluates an interesting feature of the proposed method. Vector instructions are commonly composed of 32

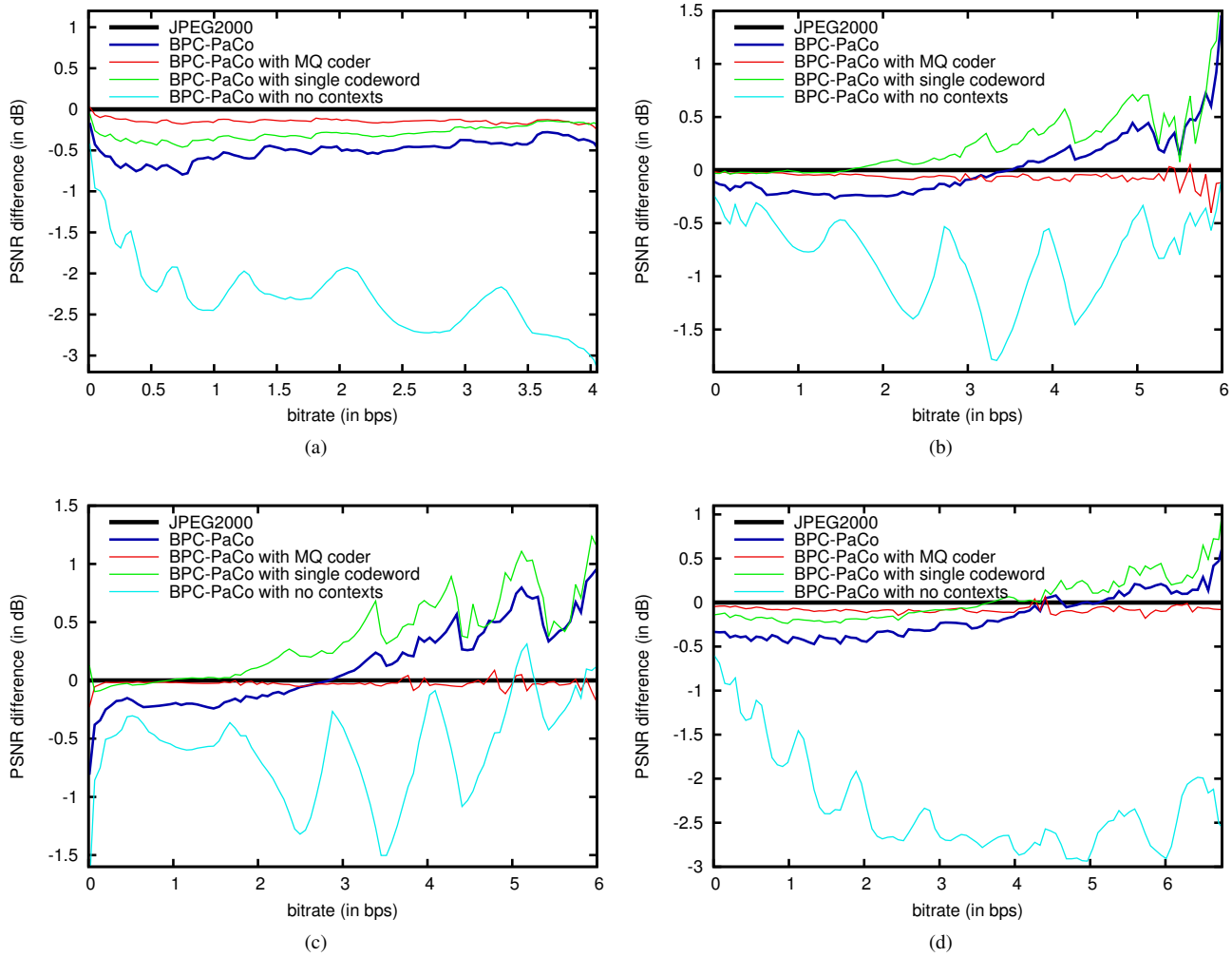


Fig. 4: Evaluation of the lossy coding performance achieved by BPC-PaCo when three modifications are employed. Each subfigure reports the performance achieved for one image of a specific corpus: (a) natural image "Portrait", (b) aerial image "forest1", (c) xRay image "A", and (d) AVIRIS image "cuprite".

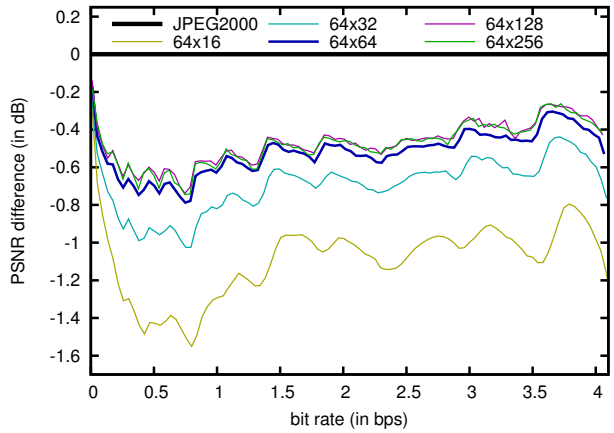


Fig. 5: Evaluation of the lossy coding performance achieved by BPC-PaCo and JPEG2000 when using different sizes of codeblock. Results are reported for the "Portrait" image of the ISO 12640-1 corpus.

lanes.[1] Each thread codes a stripe containing two columns, so the use of codeblocks with 64 columns is convenient. The number of rows of the codeblock, on the other hand, strongly influences the coding performance achieved. This is because the more data coded in a stripe, the fewer the excess bits stored in its codewords. This is illustrated in Fig. 5 for the natural image "Portrait". Results for codeblocks of 64 columns and a variable number of rows are reported (both JPEG2000 and BPC-PaCo use the same variable codeblock size). The results suggest that the more rows the codeblock has, the better the coding performance. In general, codeblocks of 64×64 already achieve competitive performance while exposing a large degree of parallelism. Results hold for the other images of the corpus and the other corpora.

V. Conclusions

The computational complexity of modern image coding systems can *not* be efficiently tackled with SIMD computing.

[1] All Nvidia GPUs, for instance, currently implement vector instructions of 32 lanes.

TABLE I: Evaluation of the lossless coding performance achieved by BPC-PaCo and JPEG2000. Results are reported in bps. The three rightmost columns report the results achieved when variations in BPC-PaCo are employed.

				BPC-PaCo with		
	image	JP2	BPC-PaCo	MQ	single cwd. AC	no ctx.
ISO 12640-1	"Portrait"	4.38	+0.09	+0.02	+0.06	+0.45
	"Cafeteria"	5.28	+0.08	+0.04	+0.05	+0.49
	"Fruit"	4.29	+0.17	+0.01	+0.14	+0.47
	"Wine"	4.57	+0.16	+0.02	+0.13	+0.43
	"Bicycle"	4.37	+0.20	+0.04	+0.16	+0.57
	"Orchid"	3.58	+0.24	+0.01	+0.21	+0.59
	"Musicians"	5.56	+0.11	+0.02	+0.07	+0.47
	"Candle"	5.65	+0.08	+0.04	+0.04	+0.58
aerial	"forest1"	6.20	-0.04	+0.01	-0.08	+0.12
	"forest2"	6.28	-0.05	+0.01	-0.09	+0.13
	"urban1"	5.54	+0.01	+0.02	-0.03	+0.21
	"urban2"	5.20	+0.03	+0.01	0.00	+0.29
xRay	"A"	6.37	-0.07	0.00	-0.12	0.00
	"B"	6.48	-0.03	0.00	-0.11	+0.02
	"C"	6.35	-0.06	0.00	-0.11	+0.02
AVIRIS	"cuprite"	7.00	-0.03	+0.01	-0.07	+0.41
	"jasper"	7.66	-0.04	+0.02	-0.08	+0.48
	"lunarLake"	6.91	-0.02	+0.01	-0.05	+0.46
average		*5.65*	*+0.05*	*+0.02*	*+0.01*	*+0.34*

The main difficulty is that the innermost algorithms of current coding systems process the samples in a sequential fashion. This paper presents a bitplane coding strategy tailored to the kind of parallelism required in SIMD computing. Its main insight is to employ vector instructions that process T coefficients of a codeblock in parallel and synchronously. To achieve this coefficient-level parallelism, some aspects of the bitplane coder are modified. First, the scanning order is devised to allow parallel coefficient processing without penalizing the formation of contexts. Second, the context formation approach is implemented via low-complexity techniques. Third, the probability estimates of the emitted symbols employs a stationary probability model that does not need adaptive mechanisms. And fourth, entropy coding is carried out by means of multiple arithmetic coders generating fixed-length codewords that are optimally sorted in the bitstream. The proposed bitplane coding strategy with parallel coefficient processing provides a very fine level of parallelism that permits its efficient implementation for both SIMD and MIMD computing. Experimental results indicate that the coding performance of the proposed method is highly competitive, similar to that achieved by the JPEG2000 standard. Future work implements the proposed method in a GPU. Results of computational throughput are not included in this paper because the proposed GPU implementation requires a detailed description. Nonetheless, preliminary results indicate speedups of 15 or more with respect to the best CPU and GPU implementations of JPEG2000.

REFERENCES

[1] W. Pennebaker and J. Mitchell, *JPEG still image data compression standard*. New York: Van Nostrand Reinhold, 1993.

[2] *Information technology - JPEG 2000 image coding system - Part 1: Core coding system*, ISO/IEC Std. 15 444-1, Dec. 2000.

[3] *High Efficiency Video Coding Standard*, International Telecommunication Union Std. H.265, 2013.

[4] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image compression fundamentals, standards and practice*. Norwell, Massachusetts 02061 USA: Kluwer Academic Publishers, 2002.

[5] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[6] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243–250, Jun. 1996.

[7] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000.

[8] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 11, pp. 1219–1235, Nov. 2004.

[9] N. Mehrseresht and D. Taubman, "A flexible structure for fully scalable motion-compensated 3-D DWT with emphasis on the impact of spatial scalability," *IEEE Trans. Image Process.*, vol. 15, no. 3, pp. 740–753, Mar. 2006.

[10] F. Auli-Llinas and M. W. Marcellin, "Scanning order strategies for bitplane image coding," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 1920–1933, Apr. 2012.

[11] S. Datla and N. S. Gidijala, "Parallelizing motion JPEG 2000 with CUDA," in *Proc. IEEE International Conference on Computer and Electrical Engineering*, Dec. 2009, pp. 630–634.

[12] R. Le, I. R. Bahar, and J. L. Mundy, "A novel parallel tier-1 coder for JPEG2000 using GPUs," in *Proc. IEEE Symposium on Application Specific Processors*, Jun. 2011, pp. 129–136.

[13] M. Ciznicki, K. Kurowski, and A. Plaza, "Graphics processing unit implementation of JPEG2000 for hyperspectral image compression," *SPIE Journal of Applied Remote Sensing*, vol. 6, pp. 1–14, Jan. 2012.

[14] M. Ciznicki, M. Kierzynka, P. Kopta, K. Kurowski, and P. Gepnerb, "Benchmarking JPEG 2000 implementations on modern CPU and GPU architectures," *ELSEVIER Journal of Computational Science*, vol. 5, no. 2, pp. 90–98, Mar. 2014.

[15] Comprimato. (2014, Apr.) Comprimato JPEG2000@GPU. [Online]. Available: http://www.comprimato.com

[16] B. Pieters, J. D. Cock, C. Hollemeersch, J. Wielandt, P. Lambert, and R. V. de Walle, "Ultra high definition video decoding with motion JPEG XR using the GPU," in *Proc. IEEE International Conference on Image Processing*, Sep. 2011, pp. 377–380.

[17] N.-M. Cheung, O. C. Au, M.-C. Kung, P. H. Wong, and C. H. Liu, "Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 11, pp. 1692–1703, Nov. 2009.

[18] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung, "Video coding on multicore graphics processors," *IEEE Signal Process. Mag.*, vol. 27, no. 2, pp. 79–89, Mar. 2010.

[19] J. Matela, V. Rusnak, and P. Holub, "Efficient JPEG2000 EBCOT context modeling for massively parallel architectures," in *Proc. IEEE Data Compression Conference*, Mar. 2011, pp. 423–432.

[20] V. Galiano, O. Lopez-Granado, M. Malumbres, L. A. Drummond, and H. Migallon, "GPU-based 3D lower tree wavelet video encoder," *EURASIP Journal on Advances in Signal Processing*, vol. 1, pp. 1–13, 2013.

[21] F. Auli-Llinas, P. Enfedaque, J. C. Moure, I. Blanes, and V. Sanchez, "Strategy of microscopic parallelism for bitplane image coding," in *Proc. IEEE Data Compression Conference*, Apr. 2015, pp. 163–172.

[22] T.-T. Wong, C.-S. Leung, P.-A. Heng, and J. Wang, "Discrete wavelet transform on consumer-level graphics hardware," *IEEE Trans. Multimedia*, vol. 9, no. 3, pp. 668–673, Apr. 2007.

[23] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, and F. Tirado, "Parallel implementation of the 2D discrete wavelet transform on graphics processing units: Filter bank versus lifting," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 3, pp. 299–310, Mar. 2008.

[24] J. Matela, "GPU-Based DWT acceleration for JPEG200," in *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, Jan. 2009, pp. 136–143.

[25] J. Franco, G. Bernabe, J. Fernandez, and M. E. Acacio, "A parallel implementation of the 2D wavelet transform using CUDA," in *Proc. IEEE International Conference on Parallel, Distributed and Network-based Processing*, Feb. 2009, pp. 111–118.

[26] W. J. van der Laan, A. C. Jalba, and J. B. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 132–146, Jan. 2011.

[27] V. Galiano, O. Lopez, M. P. Malumbres, and H. Migallon, "Parallel strategies for 2D discrete wavelet transform in shared memory systems and GPUs," *SPRINGER The Journal of Supercomputing*, vol. 64, no. 1, pp. 4–16, Apr. 2013.

[28] P. Enfedaque, F. Auli-Llinas, and J. C. Moure, "Implementation of the DWT in a GPU through a register-based strategy," *IEEE Trans. Parallel Distrib. Syst.*, 2015, in Press.

[29] F. Auli-Llinas and J. Serra-Sagrista, "Low complexity JPEG2000 rate control through reverse subband scanning order and coding passes concatenation," *IEEE Signal Process. Lett.*, vol. 14, no. 4, pp. 251–254, Apr. 2007.

[30] ——, "JPEG2000 quality scalability without quality layers," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 7, pp. 923–936, Jul. 2008.

[31] F. Auli-Llinas, J. Bartrina-Rapesta, and J. Serra-Sagrista, "Self-conducted allocation strategy of quality layers for JPEG2000," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, pp. 1–7, 2008, article ID 728794.

[32] *Digital compression and coding for continuous-tone still images*, ISO/IEC Std. 10 918-1, 1992.

[33] F. Auli-Llinas and M. W. Marcellin, "Stationary probability model for microscopic parallelism in JPEG2000," *IEEE Trans. Multimedia*, vol. 16, no. 4, pp. 960–970, Jun. 2014.

[34] A. J. R. Neves and A. J. Pinho, "Lossless compression of microarray images using image-dependent finite-context models," *IEEE Trans. Med. Imag.*, vol. 28, no. 2, pp. 194–201, Feb. 2009.

[35] F. Auli-Llinas, "Stationary probability model for bitplane image coding through local average of wavelet coefficients," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2153–2165, Aug. 2011.

[36] *Information technology - Lossy/lossless coding of bi-level images*, ISO/IEC Std. 14 492, 2001.

[37] R. W. Buccigrossi and E. P. Simoncelli, "Image compression via joint statistical characterization in the wavelet domain," *IEEE Trans. Image Process.*, vol. 8, no. 12, pp. 1688–1701, Dec. 1999.

[38] F. Auli-Llinas, M. W. Marcellin, J. Serra-Sagrista, and J. Bartrina-Rapesta, "Lossy-to-lossless 3D image coding through prior coefficient lookup tables," *ELSEVIER Information Sciences*, vol. 239, no. 1, pp. 266–282, Aug. 2013.

[39] D.-Y. Chan, J.-F. Yang, and S.-Y. Chen, "Efficient connected-index finite-length arithmetic codes," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 5, pp. 581–593, May 2001.

[40] M. D. Reavy and C. G. Boncelet, "An algorithm for compression of bilevel images," *IEEE Trans. Image Process.*, vol. 10, no. 5, pp. 669–676, May 2001.

[41] H. Chen, "Joint error detection and vf arithmetic coding," in *Proc. IEEE International Conference on Communications*, Jun. 2001, pp. 2763–2767.

[42] Y. Xie, W. Wolf, and H. Lekatsas, "Code compression using variable-to-fixed coding based on arithmetic coding," in *Proc. IEEE Data Compression Conference*, Mar. 2003, pp. 382–391.

[43] F. Auli-Llinas, "Context-adaptive binary arithmetic coding with fixed-length codewords," *IEEE Trans. Multimedia*, 2015, in Press.

[44] K. Sarawadekar and S. Banerjee, "An efficient pass-parallel architecture for embedded block coder in JPEG 2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 6, pp. 825–836, Jun. 2011.

[45] P. Howard and J. S. Vitter, "Design and analysis of fast text compression based on quasi-arithmetic coding," in *Proc. IEEE Data Compression Conference*, Mar. 1992, pp. 98–107.

[46] W. D. Wither, "The ELS-coder: a rapid entropy coder," in *Proc. IEEE Data Compression Conference*, Mar. 1997, pp. 475–475.

[47] L. Bottou, P. G. Howard, and Y. Bengio, "The Z-Coder adaptive binary coder," in *Proc. IEEE Data Compression Conference*, Mar. 1998, pp. 1–10.

[48] M. Slattery and J. Mitchell, "The Qx-coder," *IBM Journal of Research and Development*, vol. 42, no. 6, pp. 767–784, Nov. 1998.

[49] F. Auli-Llinas. (2014, Nov.) BOI codec. [Online]. Available: http://www.deic.uab.cat/~francesc/software/boi