de Cea Dominguez, Carlos; Moure, Juan C.; Bartrina Rapesta, Joan; [et al.]. «Complexity scalable bitplane image coding with parallel coefficient processing». IEEE Signal Processing Letters, Vol. 27 (June 2020), p. 840-844. DOI 10.1109/LSP.2020.2990307

Complexity Scalable Bitplane Image Coding with Parallel Coefficient Processing

Carlos de Cea-Dominguez, Juan C. Moure, Joan Bartrina-Rapesta, and Francesc Aulí-Llinàs

***Abstract*—Very fast image and video codecs are a pursued goal both in the academia and the industry. This paper presents a complexity scalable and parallel bitplane coding engine for wavelet-based image codecs. The proposed method processes the coefficients in parallel, suiting hardware architectures based on vector instructions. Our previous work is extended with a mechanism that provides complexity scalability to the system. Such a feature allows the coder to regulate the throughput achieved at the expense of slightly penalizing compression efficiency. Experimental results suggests that, when using the fastest speed, the method almost doubles the throughput of our previous engine while penalizing compression efficiency by about 10%.**

***Index Terms*—High-throughput image coding, JPEG2000.**

I. Introduction

THE pursuit of faster image and video coding systems began shortly after the development of the first codecs and compression standards. Traditional image coding systems, such as SPIHT [1] or EBCOT [2], have been revisited many times introducing modifications that accelerate their coding process and/or alleviate computational resources [3]–[8]. Also, many hardware architectures of such systems are optimized to reduce execution time and meet the real-time requirements of some environments [9]–[12]. These works focus on the improvement, or efficient implementation, of the most demanding tasks of the codec, without modifying the techniques of the original system. In general, such techniques code the data via a single-thread procedure. This strategy together with the soaring of the processor's clock speed for more than three decades, enhanced the codecs' throughput significantly. Since 2005 the increase in the clock's speed slowed and processors began augmenting their processing power via parallel architectures.

The transition from single- to multi-thread algorithms in the image coding field began with the advent of multi-core Central Processing Units (CPUs) in the 2000s [13]–[16]. The first multi-thread codecs partitioned the image in multiple pieces (referred to as codeblocks onward) that can be processed independently. In international standards such as JPEG2000 (ISO/IEC 15444) or HEVC (ISO/IEC 23008-2), for instance, the coding system provides multiple opportunities for such coarse-grain parallelism. However, the core algorithms do not allow fine-grain parallelism since they are envisaged from a single-thread perspective, resulting in a causal relationship among samples that makes parallel processing difficult.

This drawback was inconsequential while highly parallel computing was not widely available. This changed in the last years, when CPUs began including vector instructions to exploit fine-grain parallelism and, more importantly, when parallel architectures and platforms like CUDA were introduced allowing massive parallelism in commodity Graphics Processing Units (GPUs). Algorithms of other fields that were well suited to fine-grain parallelism were rapidly adapted in GPUs, achieving 20× speedups or more [17]. When implemented in GPUs, image and video coding systems did not achieve such speedups due to the sequential techniques employed.

Aware of this fact, the Joint Photographics Experts Group launched a call for proposals in 2017 [18] to introduce a new part to the JPEG2000 standard that defines a new tier-2 coding variant that offers high throughput [19]. This part is called HTJ2K (ISO/IEC 15444-15). It is devised to benefit from the modern instruction sets like AVX2, NEON, and BMI2 included in new CPUs, and also from the GPU's highly parallel architecture [20]. It is about 10× faster than the standard when executed in a CPU, though it penalizes coding performance in approximately 10%. Also, it sacrifices quality scalability, which is a valued feature of the standard that allows transmitting the image progressively by quality.

In a similar line, in 2014 we started a research whose goal is a JPEG2000-like codec that provides opportunities for fine-grain parallelism in all the stages of the coding process [21]–[25]. The proposed codec was recently evaluated using a commodity GPU. Experimental results suggest that it achieves 10× speedups compared to an implementation of JPEG2000 that is executed in a workstation with 4 CPUs [26]. The adaptation of the bitplane coding engine was the most demanding task since it requires the modification of the original techniques of JPEG2000, losing compliance. The proposed bitplane coding engine with parallel coefficient processing (BPC-PaCo) uses vector instructions of 32 lanes (or, equivalently, 32 CUDA threads) to process 32 coefficients within a codeblock in parallel. BPC-PaCo sacrifices coding efficiency as compared to JPEG2000 by about 2% but maintains all its features.

This paper introduces a mechanism that provides a new feature to BPC-PaCo: complexity scalability. The proposed mechanism allows trading computational complexity for compression efficiency. The underlying motivation is that some environments may be willing to sacrifice coding performance in exchange of throughput. Complexity scalable BPC-PaCo (CS BPC-PaCo) allows tuning the codec to accelerate more or less the coding process. Evidently, the higher the throughput achieved, the more affected are the compression efficiency and quality scalability of the system. Experimental results indicates

Carlos de Cea-Dominguez, Joan Bartrina-Rapesta and Francesc Aulí-Llinàs are with the Dep. of Information and Communications Engineering and Juan C. Moure is with the Dep. of Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Spain (phone: +34 935811861; fax: +34 935813443; e-mail: carlos.decea@uab.cat). This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under Grants TIN2017-84553-C2-1-R and RTI2018-095287-B-I00 (MINECO/FEDER, UE), and by the Catalan Government under Grants 2017SGR-463 and 2017SGR-313.

that speedups of almost 2× are achieved compared to BPC-PaCo while penalizing performance by about 10%.

The rest of the paper is structured as follows. Section II reviews BPC-PaCo and Section III describes the proposed complexity scalable mechanism. Experimental results are presented in Section IV. The last section summarizes this work.

II. Review of BPC-PaCo

BPC-PaCo utilizes a traditional bitplane coding strategy that codes the wavelet coefficients from the most significant bitplane $M-1$ to the least, with M being a sufficient number of bits to represent all coefficients within a codeblock. A bitplane is the collection of bits b_j from all coefficients, with $[b_{M-1}, b_{M-2}, ..., b_1, b_0]$, $b_i \in \{0,1\}$ denoting the binary representation of an integer υ that represents the magnitude of the index obtained by quantizing wavelet coefficient ω. The first non-zero bit of the binary representation of υ 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. Although two or three coding passes may be employed [24], [25], the two coding pass version is employed herein as baseline since it achieves higher throughput. The first is called significance coding. It processes the bits of non-significant coefficients, i.e., those coefficients whose $s \leq j$ or, more precisely, whose significance state $\Phi(\upsilon, j) = 0$. The second pass is called refinement coding and processes the bits of the remaining coefficients (i.e., those whose $\Phi(\upsilon, j) = 1$).

The main difference between BPC-PaCo and other bitplane coding engines is that BPC-PaCo codes multiple coefficients in parallel. The scanning order is organized in stripes of two columns. The stripes are processed by threads that advance their execution synchronously, all coding the coefficient in the same position of their corresponding stripe. This is the key to achieve fine-grain parallelism, since a single vector instruction is executed to code T coefficients of the codeblock at the same clock cycle. In general, the codeblock contains 64×64 coefficients, so $T = 32$. Evidently, this strategy must be accompanied with parallel techniques for context formation, probability estimation, and entropy coding.

For significance coding, the context of υ at bitplane j is determined considering its eight adjacent neighbors, denoted by υ^k, via $\phi_{sig}(\upsilon, j) = \sum_k \Phi(\upsilon^k, j)$. The context for sign coding, denoted by $\phi_{sign}(\omega, j)$, employs a similar strategy, whereas the refinement pass employs a single context since little gain is achieved with more complex models [27], so $\phi_{ref}(\upsilon, j) = 0$. Through the context, the probability estimate of the encoded bit is extracted from a lookup table (LUT) known by encoder and decoder [21]. The LUT for significance coding is accessed as $\mathcal{P}_u[j][\phi_{sig}(\cdot)]$, with u denoting the wavelet subband. This LUT contains the probability that b_j is 0, which is determined according to

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

where $F_u(\upsilon \mid \phi_{sig}(\upsilon, j))$ is the probability mass function (pmf) of the quantization indices at bitplane j given their context. Its support is $[0, ..., 2^{j+1} - 1]$ since it contains quantization indices that were not significant in bitplanes greater than j. Probabilities for sign and refinement coding are derived similarly. Their respective LUTs are denoted by $\mathcal{P}'_u$ and $\mathcal{P}''_u$.

Entropy coding is carried out through multiple arithmetic coders that produce fixed-length codewords [22] as data are coded. Each thread employs one such a coder. The dispatching of the codewords in the quality embedded bitstream generated for the codeblock requires cooperation among threads. It is optimally constructed so that the bitstream can be truncated at the end of coding passes yielding minimum distortion (see Section III.C and III.D in [24]).

III. Complexity scalable BPC-PaCo

A distinct feature of bitplane coding engines, including BPC-PaCo, is that they code the coefficients in multiple passes per bitplane. This strategy is aimed to code first those data that mostly decrease the image distortion. At the decoder, the wavelet coefficients are progressively reconstructed, allowing a fine refinement of the estimates of the incoming data. These estimates are key to achieve compression. They are commonly embodied in the context formation and probability model. Another advantage of using multiple passes per bitplane is that the bitstream contains multiple truncation points, one at the end of each coding pass. They are key to achieve quality scalability since they are employed by the rate-distortion optimization method to minimize the distortion at a target rate(s). Unfortunately, more coding passes entail more computational complexity. Each pass scans all coefficients of the codeblock despite coding the bits for only some of them. This is repeated in each coding pass, so a coefficient is accessed as many times as coding passes are executed.

The main idea behind Complexity Scalable BPC-PaCo (CS BPC-PaCo) is to reduce the computational complexity of the coding engine by reducing the number of times that each coefficient is visited. To do so while minimizing the impact on compression efficiency and quality scalability, bitplanes $[M-1, N]$ are coded as defined in BPC-PaCo. From bitplane $N-1$ to the lowest, each coefficient is coded with a fast mode that uses a single pass. Differently from conventional bitplane coding strategies, this single pass carries out inter-bitplane coding since it transmits the information of multiple bitplanes at once. Through N, the granularity of the quality scalability, the compression efficiency, and the computational complexity of the algorithm are controlled. When N is low, more bitplanes are coded with two coding passes, producing many truncation points that can be employed by rate-distortion optimization procedures. Also, coefficients are reconstructed progressively, allowing fine estimates. Evidently, low Ns do not reduce computational complexity significantly. When N is high, more bitplanes are coded in fast mode, reducing computational complexity though producing fewer truncation points and penalizing compression efficiency due to rougher estimates. This mechanism provides complexity scalability to the codec, since it can be employed to favor the application's throughput or the compression efficiency/quality scalability.

The same coding techniques of [24], with the modifications described below, are valid in the fast mode to remove the data dependency when coding coefficients in parallel.

Algorithm 1 describes the proposed coding engine from a thread (or a single lane of a vector instruction) perspective. From line 1 to 14, it employs the same procedure as that of BPC-PaCo (see Section III.D in [24]). The only difference is that the loop in line 1 codes bitplanes $[M-1, N]$ instead of $[M-1, 0]$. The position of the coefficient within the codeblock is denoted by y and x for the row and column, respectively. The fast mode is embodied in lines 15 to 29. It encodes bits $[N-1, 0]$ at once for each coefficient. The significance context is computed in line 17 before start coding and it is employed until b_s is found. This context does not change from bitplane $N-1$ to 0 since no more information of the adjacent neighbors is available once the fast mode begins. This also needs to be considered in the probability model, so the LUT employed in the fast mode for significance coding in bitplanes $j' = [N-1, 0]$ is populated according to

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

Probabilities for sign coding are determined accordingly. The LUT for refinement is unchanged due to the use of a single context.

Algorithm 1 Complexity Scalable BPC-PaCo (encoder)
Parameters: u subband, t stripe, M bitplanes to code, N bitplanes in fast mode

```
1: for $j \in [M-1, N]$ do
2:     for $y \in [0, \text{numRows}-1]$ do
3:         for $x \in [t \cdot 2, t \cdot 2 + 1]$ do
4:             if $\Phi(\upsilon_{y,x}, j+1) = 0$ then
5:                 ACencode($b_j$, $\mathcal{P}_u[j][\phi_{sig}(\upsilon_{y,x}, j)]$, $t$)
6:                 if $b_j = 1$ then
7:                     ACencode($d$, $\mathcal{P}'_u[j][\phi_{sign}(\omega_{y,x}, j)]$, $t$)
8:                 end if
9:             else
10:                ACencode($b_j$, $\mathcal{P}''_u[j][0]$, $t$)
11:            end if
12:        end for
13:    end for
14: end for
15: for $y \in [0, \text{numRows}-1]$ do
16:     for $x \in [t \cdot 2, t \cdot 2 + 1]$ do
17:         $c \leftarrow \phi_{sig}(\upsilon_{y,x}, N-1)$
18:         for $j \in [N-1, 0]$ do
19:             if $\Phi(\upsilon_{y,x}, j+1) = 0$ then
20:                 ACencode($b_j$, $\mathcal{P}_u[j][c]$, $t$)
21:                 if $b_j = 1$ then
22:                     ACencode($d$, $\mathcal{P}'_u[j][\phi_{sign}(\omega_{y,x}, N-1)]$, $t$)
23:                 end if
24:             else
25:                 ACencode($b_j$, $\mathcal{P}''_u[j][0]$, $t$)
26:             end if
27:         end for
28:     end for
29: end for
```

The selection of N is key to control the computational complexity of the engine. A straightforward approach is to apply the same N to all codeblocks. Our experience indicates that this may penalize quality scalability significantly because at the lowest N bitplanes there is only one truncation point available for each codeblock. When the bitstreams segments of higher bitplanes are already selected, the rate-distortion optimization method can only include the whole segment of some codeblocks, completely discarding some others. At low rates, this may cause that none information is transmitted for some areas of the image, producing an image with blank areas or with no color information. The quality scalability of the system is less affected when N is chosen depending on the codeblock's data and the wavelet subband. Let us explain further. As indicated in [28], the highest bitplanes of a codeblock contain the information that mostly decreases the distortion. In terms of rate-distortion optimization, this means that is more valuable the data coded in bitplane $j = 4$ for a codeblock with $M = 5$ than for another with $M = 6$, for example. Therefore, our strategy selects N depending on M. The wavelet subband is also considered. The first decomposition levels (i.e., the largest wavelet subbands) contain most codeblocks, whereas the latest contain much fewer, so the use of the fast mode in the codeblocks of the smallest resolution subbands barely affects the throughput achieved. However, these codeblocks contain the rougher details of the image, important for its reconstruction. Our strategy selects N in each codeblock according to

$$N = \min\left(M, \left\lfloor M \cdot \frac{K}{\mathcal{L}_u} \right\rfloor\right) . \tag{3}$$

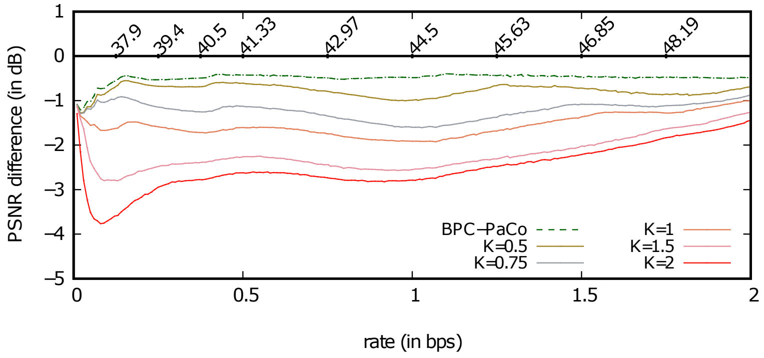
K is the input parameter of our implementation that controls the computational complexity of the codec. Larger Ks achieve larger Ns, so more bitplanes are coded in fast mode, rising the codec's throughput. $\mathcal{L}_u$ is the L$_2$norm of the synthesis basis vectors of the subband's filter-bank (it is assumed equal energy gain factor in all subbands). The higher the decomposition level, the more decreases the K, resulting in lower Ns in the smallest resolution levels. Through this strategy, the fast mode is applied at different bitplanes depending on the data and subband of the codeblock, providing more variability to the rate-distortion optimization method.

IV. Experimental results

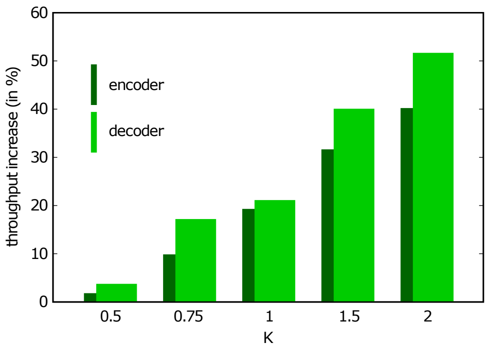
The ISO 12640-1 corpus is employed (8 color images, 2560×2048, and 8 bits per sample (bps)). The results report the performance achieved by JPEG2000, BPC-PaCo, and the proposed CS BPC-PaCo. The same Java framework BOI [29] is used for all codecs, using the same rate-distortion optimization method. Results for throughput are computed when the coding engine is executed with a single thread. This gives an approximation of the computational complexity of the algorithm. 5 levels of wavelet decomposition and codeblocks of 64×64 are employed. These coding parameters are selected as the most commonly used. A smaller codeblock size alleviates the penalization in coding efficiency of the proposed method,

TABLE I: Evaluation of the proposed method for lossless and lossy compression. All results are reported in bps except for the speedup, which is the percentage of CS BPC-PaCo with respect to BPC-PaCo (on average for the encoder and decoder).

	LOSSLESS COMPRESSION								LOSSY COMPRESSION							
		BPC-	*CS BPC-PaCo*							*BPC-*	*CS BPC-PaCo*					
	JP2	*PaCo*	$K = 0.5$		$K = 1.5$		$K = \infty$		*JP2*	*PaCo*	$K = 1$		$K = 2$		$K = \infty$	
Portrait	3.80	+0.21	+0.22	7%	+0.39	56%	+0.43	76%	2.60	+0.10	+0.16	21%	+0.27	51%	+0.31	68%
Cafe.	4.68	+0.13	+0.16	10%	+0.47	59%	+0.52	74%	3.61	+0.08	+0.23	40%	+0.43	63%	+0.50	81%
Fruit	3.96	+0.20	+0.22	5%	+0.40	52%	+0.43	70%	2.73	+0.11	+0.18	24%	+0.28	48%	+0.32	62%
Wine	3.94	+0.20	+0.21	8%	+0.37	55%	+0.41	72%	2.71	+0.08	+0.14	27%	+0.22	50%	+0.27	70%
Bicycle	3.90	+0.20	+0.22	7%	+0.41	54%	+0.46	69%	2.67	+0.11	+0.20	29%	+0.32	55%	+0.36	70%
Orchid	3.44	+0.26	+0.27	7%	+0.39	48%	+0.43	71%	2.15	+0.12	+0.17	20%	+0.25	46%	+0.28	62%
Music.	5.34	+0.20	+0.29	8%	+0.78	56%	+0.84	68%	4.40	+0.11	+0.36	42%	+0.61	67%	+0.66	79%
Candle	4.74	+0.15	+0.20	11%	+0.53	56%	+0.60	73%	3.69	+0.08	+0.26	40%	+0.47	66%	+0.52	76%
average	4.22	+0.20	+0.22	8%	+0.47	54%	+0.51	72%	3.07	+0.09	+0.21	30%	+0.36	56%	+0.40	71%



(a)



(b)

Fig. 1: Evaluation of (a) lossy coding performance and (b) throughput achieved for the "Orchid" image when using different Ks. In (a), the horizontal straight plot depicts the performance achieved by JPEG2000, whereas the other plots depict the performance achieved by the proposed method with different Ks, or by BPC-PaCo.

whereas fewer wavelet levels degrades coding performance significantly at low rates.

Table I (left) reports the results for lossless compression. They use $K = \{0.5, 1.5, \infty\}$. $K = \infty$ achieves the fastest speed since all bitplanes are coded with the fast mode. The results of this table suggest that CS BPC-PaCo can accelerate the coding process of the original engine by 72% whereas the penalization in coding performance is, with the fastest speed, 13% and 10% as compared to JPEG2000 and BPC-PaCo, respectively.

Table I (right) reports lossy compression results when $K = \{1, 2, \infty\}$. This test evaluates the rate and throughput increase achieved when all bitplanes are coded, achieving a quality above 50 dB. The results suggest that when $K = 1$ the engine's throughput is increased by 30% whereas the rate by 4% with respect to BPC-PaCo. For $K = 2$ ($K = \infty$), throughput and rate are respectively increased by 56% (71%) and 9% (10%). Note that, in terms of percentage, the throughput is more increased than the decrease in compression efficiency.

Fig. 1(a) evaluates the quality scalability achieved by the proposed method, for the "Orchid" image. Results hold for the others. The figure reports the coding performance achieved at 200 rates equivalently distributed between 0.01 and 2 bps, in terms of Peak Signal to Noise Ratio (PSNR) difference between JPEG2000 and BPC-PaCo or CS BPC-PaCo when using different Ks. The results are obtained when all bitplanes are coded and then the rate-distortion optimization method constructs the final file, or quality layers, at the target rates. For $K \in (0, 1)$ the losses in coding performance are below 2 dB for the whole rate range. Larger Ks result in higher losses, especially for low rates. This penalization in compression efficiency is caused by both the lack of enough truncation points and the poorer efficiency of the arithmetic coder due to rougher estimates of the coefficients. Fig. 1(b) reports the throughput's increase when using the same Ks as before. When $K = 1$ the throughput is increased in 20% while slightly affecting the coding efficiency (see Fig. 1(a)). Although larger Ks penalize more the image quality, the throughput's enhancement is significant, achieving more than 40% when $K = 2$.

V. Conclusions

Many efforts have been done to increase the throughput of image and video codecs. Common approaches are to implement them in hardware or to simplify their algorithms, which sometimes sacrifices some features. This paper presents a fast bitplane coding engine that, in addition to the features of the JPEG2000 standard, provides complexity scalability. To do so, it uses a coding engine that processes the coefficients in parallel and, when indicated, changes the conventional coding of bitplanes to a fast mode that codes all bits of the coefficients at once. Experimental results indicate that the proposed method can effectively regulate the codec's throughput. When using the minimum complexity, the throughput and rate are increased by about 70% and 13%, respectively, whereas the maximum complexity increases throughput and rate by about 10% and 2%, respectively, on average for the employed corpus and for lossy and lossless compression.

References

[1] 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.

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

[3] 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.

[4] G. Xie and H. Shen, “Highly scalable, low-complexity image coding using zeroblocks of wavelet coefficients,” *IEEE Trans. Image Process.*, vol. 15, no. 6, pp. 762–770, Jun. 2005.

[5] M. Dyer, D. Taubman, S. Nooshabadi, and A. K. Gupta, “Concurrency techniques for arithmetic coding in JPEG2000,” *IEEE Trans. Circuits Syst. I*, vol. 53, no. 6, pp. 1203–1212, Jun. 2006.

[6] M. Rhu and I.-C. Park, “Optimization of arithmetic coding for JPEG2000,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 3, pp. 446–451, Mar. 2010.

[7] 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.

[8] X. Song, Q. Huang, S. Chang, J. He, and H. Wang, “Three-dimensional separate descendant-based SPIHT algorithm for fast compression of high-resolution medical image sequences,” vol. 11, no. 1, pp. 80–87, Jan. 2017.

[9] A. K. Gupta, S. Nooshabadi, D. Taubman, and M. Dyer, “Realizing low-cost high-throughput general-purpose block encoder for JPEG2000,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 7, pp. 843–858, Jul. 2006.

[10] K. Mei, N. Zheng, C. Huang, Y. Liu, and Q. Zeng, “VLSI design of a high-speed and area-efficient JPEG2000 encoder,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 8, pp. 1065–1078, Aug. 2007.

[11] M. Dyer, S. Nooshabadi, and D. Taubman, “Design and analysis of system on a chip encoder for JPEG2000,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 2, pp. 215–225, Feb. 2009.

[12] S. Kim, D. Lee, J.-S. Kim, , and H.-J. Lee, “A high-throughput hardware design of a one-dimensional SPIHT algorithm,” *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 392–404, Mar. 2016.

[13] H.-C. Fang, Y.-W. Chang, T.-C. Wang, C.-J. Lian, and L.-G. Chen, “Parallel embedded block coding architecture for JPEG 2000,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 1086–1097, Sep. 2005.

[14] Y. Li and M. Bayoumi, “A three-level parallel high-speed low-power architecture for EBCOT of JPEG 2000,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 9, pp. 1153–1163, Sep. 2006.

[15] 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.

[16] Y. Jin and H.-J. Lee, “A block-based pass-parallel SPIHT algorithm,” vol. 22, no. 7, pp. 1064–1075, Jul. 2012.

[17] M. S. Nobile, P. Cazzaniga, A. Tangherloni, and D. Besozzi, “Graphics processing units in bioinformatics, computational biology and systems biology,” *Briefings in Bioinformatics*, vol. 18, no. 5, pp. 870–885, Sep. 2017.

[18] *High Throughput JPEG 2000 (HTJ2K): Call for Proposals*, ISO/IEC Std., 2017, document ISO/IEC JTC 1/SC29/WG1 N76037.

[19] D. Taubman, A. Naman, and R. Mathew, “High throughput block coding in the HTJ2K compression standard,” in *Proc. IEEE International Conference on Image Processing*, Sep. 2019, pp. 1079–1083.

[20] A. Naman and D. Taubman, “Decoding high-throughput JPEG2000 (HTJ2K) on a GPU,” in *Proc. IEEE International Conference on Image Processing*, Sep. 2019, pp. 1084–1088.

[21] 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.

[22] F. Auli-Llinas, “Context-adaptive binary arithmetic coding with fixed-length codewords,” *IEEE Trans. Multimedia*, vol. 17, no. 8, pp. 1385–1390, Aug. 2015.

[23] 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.*, vol. 26, no. 12, pp. 3394–3406, Dec. 2015.

[24] F. Auli-Llinas, P. Enfedaque, J. C. Moure, and V. Sanchez, “Bitplane image coding with parallel coefficient processing,” *IEEE Trans. Image Process.*, vol. 25, no. 1, pp. 209–219, Jan. 2016.

[25] P. Enfedaque, F. Auli-Llinas, and J. C. Moure, “GPU implementation of bitplane coding with parallel coefficient processing for high performance image compression,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2272–2284, Aug. 2017.

[26] C. de Cea-Dominguez, J. C. Moure, J. Bartrina-Rapesta, and F. Auli-Llinas, “GPU architecture for wavelet-based video coding acceleration,” in *Parallel Computing: Technology Trends*, vol. 36, Apr. 2020, pp. 83–92, IOSPress Series in Advances in Parallel Computing.

[27] 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.

[28] F. Auli-Llinas and J. Serra-Sagrista, “JPEG2000 quality scalability without quality layers,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 7, pp. 923–936, Jul. 2008.

[29] F. Auli-Llinas. (2019, Nov.) BOI codec. [Online]. Available: http://www.deic.uab.cat/~francesc/software/boi