[1]FAST AND EFFICIENT SPATIAL SCALABLE IMAGE COMPRESSION USING WAVELET LOWER TREES

J. Oliver, *Student Member, IEEE,* M. P. Malumbres, *Member, IEEE*

Department of Computer Engineering (DISCA)
Technical University of Valencia
Camino de Vera 17, 46017 Valencia, Spain
Phone: +34 96 387 97 06, Fax: +34 96 387 75 79
E-mail: {joliver, mperez}@disca.upv.es

ABSTRACT

In this paper, a new image compression algorithm is proposed based on the efficient construction of wavelet coefficient lower trees. This lower-tree wavelet (LTW) encoder presents state-of-the-art compression performance, while its temporal complexity is lower than the one presented in other wavelet coders, like SPIHT and JPEG 2000. This fast execution is achieved by means of a simple two-pass coding and one-pass decoding algorithm. On the other hand, its computation does not need additional lists or complex data structures, so there is not memory overhead. A formal description of the algorithm is provided, so that an implementation can be performed straightforward. The results show that our codec works faster than SPIHT and JPEG2000 (up to eight times faster) with better performance in terms of rate-distortion metric.

1. INTRODUCTION

During the last decade, several image compression schemes emerged in order to overcome the known limitations of block-based algorithms that use the Discrete Cosine Transform (DCT), the most widely used compression technique at that moment.

Some of these alternative proposals were based on more complex techniques, like Vector Quantization and Fractal image coding, while others simply proposed the use of a different and more suitable mathematical transform, the Discrete Wavelet Transform (DWT). At that time, there was a generalized idea: more efficient images coders would only be achieved by means of sophisticated techniques with high temporal complexity, or by the combination of some of them. The Embedded Zero-Tree Wavelet coder (EZW) can be considered as the first Wavelet image coder that broke that trend. Since then, many wavelet coders were proposed and finally, the DWT was included in the JPEG 2000 standard due to its compression efficiency and other features (scalability, etc.).

All the wavelet-based image coders, and generally all the transform-based coders, consist in two fundamental stages. During the first one, the image is transformed from spatial domain to another one, in the case of wavelet transform a combined spatial-frequency domain called wavelet domain. In the second pass the coefficients resulting

[1] This work was supported by the *Generalitat Valenciana* research grant CTIDIB/2002/19

from the domain transform are encoded in an efficient way, in order to achieve high rate/distortion performance and other features.

The wavelet transform can be implemented as a regular filter-bank convolution, however several strategies have been proposed in order to speed-up the computation time of this transform, along with the reduction of the total amount of memory needed to compute it. In this way, a line-based processing is proposed in [1], while an alternative wavelet transform method, called lifting scheme, is proposed in [2]. This lifting transform provides in-place calculation of the wavelet coefficients by overwriting the input sample values, and thus it minimizes the number of memory accesses.

However, the wavelet coefficient coding pass is not usually improved in terms of temporal complexity and memory requirements. When designing and developing a new wavelet image encoder, the most important factor to be optimized uses to be the rate/distortion (R/D) performance, while other features like embedded bit-stream, SNR scalability, spatial scalability and error resilience may be considered.

In this paper we propose an algorithm aiming to achieve equal or higher rate/distortion performance than state-of-the-art image coders, while the required execution time is minimized. Moreover, due to in-place processing of the wavelet coefficients, there is not memory overhead (it only needs memory to store the source image or piece of image to be coded). Additionally, our algorithm is naturally spatial scalable and it is possible to perform SNR scalability and certain level of error resilience.

This paper is organized as follows. In Section 2 we will focus on some important wavelet image coders that have been previously proposed, focusing on their computational complexity and R/D performance. Section 3 shows a fast and simple algorithm that will be used as a starting point for the proposed algorithm, which it is explained in detail in section 4. Finally, in section 5 we compare our proposal with other wavelet image coders using real implementations, and not simulation results.

2. PREVIOUS WAVELET IMAGE CODERS AND THEIR PERFORMANCE

One of the first efficient wavelet image coders reported in the literature is the EZW [3]. It is based on the construction of coefficient-trees and successive-approximations, that can be implemented as bit-plane processing. Due to its successive-approximation nature, it is SNR scalable, although at the expensive of sacrificing spatial scalability. SPIHT [4] is an advanced version of this algorithm, where coefficient-trees are processed in a more efficient way. Both algorithms need the computation of coefficient-trees and perform different iterations focusing on a different bit plane in each iteration, what usually requires high computational complexity.

Other compression algorithms perform bit plane processing along with other techniques like run-length coding. In this way, the stack-run encoder [5] achieves spatial scalability, while the embedded run-length coder described in [6] is SNR scalable. Authors state that these proposals are faster due to the lack of computation needed to define coefficient-trees, but it also leads them to poorer R/D performance. Moreover, bit plane processing is still present, limiting the speed-up of the encoder. Notice that in software implementations, several iterations accessing to the same coefficients reduce the computer cache performance.

Recently, the JPEG 2000 standard was released [7]. The proposed algorithm does not use coefficient-trees, but it performs bit-plane processing with three passes per bit plane. In this way, it achieves finer rate control, along with a new R/D optimization algorithm. It also uses large number of contexts in order to overcome the inconvenience of not using coefficient-trees. JPEG 2000 attains both spatial and SNR scalability by means of postprocessing the encoded image, at the cost of higher computational complexity.

3. A SIMPLE FAST MULTIRESOLUTION IMAGE CODING ALGORITHM

For the most part, digital images are represented with a set of pixel values. The encoder proposed in this paper can be applied to a set of coefficients C resulting from a dyadic decomposition $\Omega(\cdot)$, in order that $C=\Omega(P)$. The most commonly used dyadic decomposition in image compression is the hierarchical wavelet subband transform [8], therefore an element $c_{i,j} \in C$ is called transform coefficient. In a wavelet transform, we call LH_1, HL_1 and HH_1 to the subbands resulting from the first level of the image decomposition, corresponding to horizontal, vertical and diagonal frequencies. The rest of image transform is performed by recursive wavelet decomposition on the remaining low frequency subband, until a desired decomposition level (N) is achieved (LL_N is the remaining low frequency subband)

As we have seen in section 2, one of the main drawbacks in previous wavelet image encoders is their high temporal complexity. That is mainly due to the bit plane processing, that is performed along different iterations, using a threshold that focuses on a different bit plane in each iteration. In this way, it is easy to achieve an embedded bit-stream with progressive coding, since more bit planes add more SNR resolution to the recovered image.

Although embedded bit-stream is a nice feature in an image coder, it is not always needed and other alternatives, like spatial scalability, may be more valuable according to the final purpose. In this section, we are going to propose a very simple algorithm that is able to encode the wavelet coefficients without performing one loop scan per bit plane. Instead of it, only one scan of the transform coefficients will be needed.

In this algorithm, the quantization process is performed by two strategies: one coarser and another finer. The finer one consists on applying a scalar uniform quantization to the coefficients, and it is performed just after the DWT is applied. On the other hand, the coarser one is based on removing bit planes from the least significant part of the coefficients, and it is performed while our algorithm is applied. At this moment we define *rplanes* as the number of less significant bits to be removed.

At encoder initialization, the maximum number of bits needed to represent the higher coefficient (*maxplane*) is calculated. This value and the *rplanes* parameter are output to the decoder. Afterwards, we initialize an adaptive arithmetic encoder that will be used to encode the number of bits required by the coefficients. We will only output those coefficients ($c_{i,j} \in C$) that require more than *rplanes* bits to be coded (those $c_{i,j}<2^{rplanes}$), thus only *maxplane*-*rplanes* symbols are needed to represent this information. An extra symbol is required, so-called *LOWER* symbol, to indicate those coefficients that are lower than the established threshold ($2^{rplanes}$). Notice that we say that $c_{i,j}$ is a significant coefficient when it is different to zero after discarding the least significant *rplanes* bits, in other words, if $c_{i,j} \geq 2^{rplanes}$.

In the next stage, the wavelet coefficients are encoded as follows. For each subband, from the Nth level to the first one, all the coefficients are scanned in Morton order (i.e., in medium-sized blocks). For each coefficient in that subband, if it is significant, a symbol indicating the number of bits required to represent that coefficient is arithmetically encoded. As coefficients in the same subband have similar magnitude, and due to the order we have established to scan the coefficients, the adaptive arithmetic encoder is able to represent very efficiently this information. However, we do not have enough information to reconstruct correctly the coefficient; we still need to encode its significant bits and sign.

On the other hand, if a coefficient is not significant, we should transmit a *LOWER* symbol so that the decoder can determine that it has been absolutely quantized, and thus it does not have associated information (neither coefficient bits nor sign).

Notice that when encoding the bits of a significant coefficient, the first *rplanes* bits and the most significant bit are not coded, the decoder can deduce the most significant bit through the arithmetic symbol that indicates the number of bits required to encode this coefficient. Moreover, in order to speed up the execution time of the algorithm, the bits and sign of significant coefficients are raw encoded, what results in very small lost in R/D performance.

The proposed encoding algorithm, *algorithm I*, is defined as follows.

```
(E1) INITIALIZATION
  **output** $rplanes$
  **output** $maxplane = \max_{\forall c_{i,j} \in C} \left\{ \left\lceil \log_2\left(\left|c_{i,j}\right|\right) \right\rceil \right\}$
(E2) OUTPUT THE COEFFICIENTS. Scan the subbands in the established order.
  For each $c_{i,j}$ in a subband
    $nbits_{i,j} = \left\lceil \log_2\left(\left|c_{i,j}\right|\right) \right\rceil$
    **if** $nbits_{i,j} > rplanes$
        **arithmetic_output** $nbits_{i,j}$
        **output** $\mathrm{bit}_{nbits_{(i,j)}-1}\left(\left|c_{i,j}\right|\right) \ldots \mathrm{bit}_{rplane+1}\left(\left|c_{i,j}\right|\right)$
        **output** $\mathrm{sign}(c_{i,j})$
    **else**
        **arithmetic_output** $LOWER$
```

Note: $\mathrm{bit}_n(c)$ is a function that returns the n^{th} bit of c.

This algorithm is resolution scalable due to the selected scanning order and the nature of the wavelet transform. In this way, the first subband that the decoder attains is the LL_N, which is a low-resolution scaled version of the original image. Then, the decoder progressively receives the remaining subbands, from lower frequency subbands to higher ones, which are used as a complement to the low-resolution image to recursively double its size, what is know as Mallat decomposition [9].

Notice that spatial and SNR scalability are closely related features. Spatial resolution allow us to have different resolution images from the same image. Through interpolation techniques, all these images could be resized to the original size, so that the larger an image is, the closer to the original it gets. Therefore, in some way this algorithm could also be used for SNR scalability purposes.

The robustness of the proposed algorithm lies in the low dependency among the encoded information. This dependency is only present in consecutive arithmetic encoded symbols, and thus the use of synchronism marks would increase the error resilience at the cost of slightly decreasing the R/D performance. However, the lack of dependency that yields to higher robustness also causes lower compression performance in the encoding process. The correlation among coefficients and subbands can be exploited in a more efficient way, as we can see in section 4, which takes this algorithm as a starting point to define a more efficient encoder, with similar characteristics to those presented in this section.

4. TWO-PASS EFFICIENT BLOCK CODING USING LOWER TREES

Tree based wavelet image encoders are proved to efficiently store the transform coefficients, achieving great performance results. In these algorithms, two stages can be established. The first one consists on encoding the significance map, i.e., the location and amount of bits required to represent those coefficients that will be encoded (significant coefficients). In the second stage, significant transform coefficients are encoded, i.e. their sign and magnitude bits, depending on the desired target bit rate.

Previous tree based wavelet image coders performs bit plane processing in both stages, resulting in high temporal complexity. The algorithm presented in this section is an extension of the one-pass algorithm that we have described in section 3. In this algorithm, a new tree based structure is introduced in order to reduce the data redundancy among subbands. This structure is called lower-tree.

Like in the rest of tree encoding techniques, coefficients from C=$\Omega(P)$ can be logically arranged as a tree. In these trees, the offspring for every coefficient $c_{a,b}$ can be computed as follows:

$$if\ c_{a,b} \in LL_N \wedge a \in \{2k_1\} \wedge b \in \{2k_2\} : k_1, k_2 \in Z$$
$$offspring(c_{a,b}) = \{\phi\}$$
$$if\ c_{a,b} \in LL_N \wedge a \in \{2k_1+1\} \wedge b \in \{2k_2\} : k_1, k_2 \in Z,\ w = width(LL_N),\ h = height(LL_N)$$
$$offspring(c_{a,b}) = \{c_{a+w,b},\ c_{a+w+1,b},\ c_{a+w,b+1},\ c_{a+w+1,b+1}\}$$
$$if\ c_{a,b} \in LL_N \wedge a \in \{2k_1\} \wedge b \in \{2k_2+1\} : k_1, k_2 \in Z,\ w = width(LL_N),\ h = height(LL_N)$$
$$offspring(c_{a,b}) = \{c_{a,b+h},\ c_{a+1,b+h},\ c_{a,b+h+1},\ c_{a+1,b+h+1}\}$$
$$if\ c_{a,b} \in LL_N \wedge a \in \{2k_1+1\} \wedge b \in \{2k_2+1\} : k_1, k_2 \in Z,\ w = width(LL_N),\ h = height(LL_N)$$
$$offspring(c_{a,b}) = \{c_{a+w,b+h},\ c_{a+w+1,b+h},\ c_{a+w,b+h+1},\ c_{a+w+1,b+h+1}\}$$
$$if\ c_{a,b} \in HL_1 \vee c_{a,b} \in LH_1 \vee c_{a,b} \in HH_1$$
$$offspring(c_{a,b}) = \{\phi\}$$
$$if\ c_{a,b} \notin LL_N \wedge c_{a,b} \notin HL_1 \wedge c_{a,b} \notin LH_1 \wedge c_{a,b} \notin HH_1$$
$$offspring(c_{a,b}) = \{c_{2a,2b},\ c_{2a+1,2b},\ c_{2a,2b+1},\ c_{2a+1,2b+1}\}$$

In this way, if a coefficient has offspring, the direct descendents always form a 2x2 block of coefficients, and the rest of descendant levels can be attained by successively calculating the offspring of these direct descendents.

The quantization used in this new proposed algorithm is the same as in the *algorithm I*, so that a coefficient is called lower-tree root if this coefficient and all its descendants are lower than $2^{rplanes}$. The set formed by all these coefficients form a lower-tree.

In this new proposal, we use the label *LOWER* to point out that a coefficient is the root of a lower-tree. The rest of coefficients in the lower-tree are labeled as *LOWER_COMPONENT*. On the other hand, if a coefficient is lower than $2^{rplanes}$ but it does not belong to a lower-tree, it is considered as *ISOLATED_LOWER*. Labels are applied by overwriting the coefficient value by an integer value associated to the corresponding label, which must be outside the possible range of the wavelet coefficients (typically by reusing the values in the quantized range [0.. $2^{rplanes}$])

Once we have defined the basic concepts to understand the algorithm, we are ready to define the coding process. It is a two-pass algorithm. During the first pass, the wavelet coefficients are properly labeled according to their significance and all the lower-trees are formed. This coding pass is new with respect to the *algorithm I*. In the second image pass, the coefficient values are coded in a similar way than in the E2 stage of *algorithm I*, taking into account the new symbols introduced in the *algorithm II*. Notice that both significance map and significant values are encoded in this pass in only one iteration.

The encoder algorithm, *algorithm II*, is described as follows.

```
(E1) INITIALIZATION
  output rplanes
  output $maxplane = \max_{\forall c_{i,j} \in C} \left\{ \left\lceil \log_2 \left( \left| c_{i,j} \right| \right) \right\rceil \right\}$

(E2) CALCULATE SYMBOLS (first image pass)
  Scan the first level subbands (HH$_1$, LH$_1$ and HL$_1$) in 2x2 blocks.
  For each block $B_n$
    if $c_{i,j} < 2^{rplanes} \quad \forall c_{i,j} \in B_n$
      set $c_{i,j}$ =LOWER_COMPONENT $\forall c_{i,j} \in B_n$
    else
      for each $c_{i,j} \in B_n$
          if $c_{i,j} < 2^{rplanes}$
              set $c_{i,j}$ =LOWER
  Scan the rest of level subbands (from level 2 to N) in 2x2 blocks.
  For each block $B_n$
    if $c_{i,j} < 2^{rplanes} \wedge$ descendant($c_{i,j}$)=LOWER_COMPONENT $\forall c_{i,j} \in B_n$
      set $c_{i,j}$ =LOWER_COMPONENT $\forall c_{i,j} \in B_n$
    else
      for each $c_{i,j} \in B_n$
          if $c_{i,j} < 2^{rplanes} \wedge$ descendant($c_{i,j}$)=LOWER_COMPONENT
```

```
                set $c_{i,j}$ =$LOWER$
            **if** $c_{i,j} < 2^{rplanes} \wedge$ descendant($c_{i,j}$)$\neq$ $LOWER\_COMPONENT$
                set $c_{i,j}$ =$ISOLATED\_LOWER$
```

(E3) OUTPUT THE COEFFICIENTS (*second image pass*)

```
  Scan the subbands in an established order.
  **For each** $c_{i,j}$ in a subband
    **if** $c_{i,j} \neq LOWER\_COMPONENT$
        **if** $c_{i,j}$ =$LOWER$
           **arithmetic_output** $LOWER$
        **else** if $c_{i,j}$ =$ISOLATED\_LOWER$
           **arithmetic_output** $ISOLATED\_LOWER$
        **else**
           $nbits_{i,j} = \lceil \log_2 (|c_{i,j}|) \rceil$
           **if** descendant($c_{i,j}$)$\neq$ $LOWER\_COMPONENT$
              **arithmetic_output** $nbits_{i,j}$
           **else**
              **arithmetic_output** $nbits_{i,j}^{LOWER}$
           **output** $\text{bit}_{nbits_{(i,j)}-1}(|c_{i,j}|)\ldots\text{bit}_{rplane+1}(|c_{i,j}|)$
           **output** sign($c_{i,j}$)
```

Note: $\text{bit}_n(c)$ is a function that returns the nth bit of c.

If we analyze the first image pass we can observe that labeling of lower-trees is performed in a recursive way, building the lower-trees from leaves to the root. In the first level subbands, coefficients are scanned in 2x2 blocks and, if all the four coefficients are insignificant (i.e., lower than $2^{rplanes}$), they are considered to be part of the same lower-tree, being labeled as *LOWER_COMPONENT*. Later on, if a 2x2 block in an upper level has four insignificant coefficients, and all their offspring have been labeled as *LOWER_COMPONENT*, the coefficients in the block are also labeled as *LOWER_COMPONENT*, creating a new, larger lower-tree.

However, when any coefficient in the block is significant, the lower-tree cannot continue growing. In that case, for any insignificant coefficient, if its descendants are *LOWER_COMPONENT*, it is the root of that lower-tree and thus it is labeled as *LOWER* symbol, otherwise it is labeled as *ISOLATED_LOWER*.

In the second pass, all the subbands are explored from the Nth level to the first one, and all their coefficients are scanned in medium-sized blocks (to take profit from data locality). For each coefficient in a subband, if it is a lower-tree root or an isolated lower, the corresponding *LOWER* or *ISOLATED_LOWER* symbol is output. On the other hand, if the coefficient has been labeled as *LOWER_COMPONENT* no output is needed because this coefficient is already represented by another symbol.

A significant coefficient is coded in a similar way as in *Algorithm I*. A symbol indicating the number of bits required to represent that coefficient is arithmetically coded, and the coefficient bits and sign are raw coded. However, two types of numeric symbols are used according to the coefficient offspring. A regular numeric symbol ($nbits_{i,j}$) simply shows the number of bits needed to encode a coefficient, but a special *lower* numeric symbol ($nbits_{i,j}^{LOWER}$) not only indicates the number of bits of the coefficient, but also the fact that all its descendants are labeled as *LOWER_COMPONENT*, and thus they belong to a not yet codified lower-tree. In this way, this type of symbol is able to represent efficiently some special lower-trees, in which the root coefficient is significant and the rest of coefficients are insignificant. Notice that the number of symbols needed to represent both set of numeric symbols is $2 \times (maxplane - rplanes)$, therefore the arithmetic encoder must be initialized to handle at least this amount of symbols along with two additional symbols: the *LOWER* and *ISOLATED_LOWER* symbols (a *LOWER_COMPONENT* coefficient is never encoded)

On the other hand, notice that the decoder algorithm may perform the reverse process in only one pass, without the need of calculating the symbols for the coefficients, which are directly input from the encoder side.

Further adjustments may improve the compression performance of the algorithm, as for example, simple context coding based on the left and the upper coefficients, and fitting the *maxplane* parameter in every subband level, and hence fitting the number of symbols needed in the arithmetic encoder.

5. COMPARISON WITH OTHER WAVELET CODERS USING REAL IMPLEMENTATIONS

We have implemented this Lower-Tree Wavelet (LTW) encoder and decoder algorithms in order to test its performance. It has been implemented using standard C++ language, and all the simulation tests have been performed on a regular Personal Computer. The reader can easily perform new tests using the win32 version of LTW available at *http://www.disca.upv.es/joliver/LTW/LTW.zip*.

In order to compare our algorithm with other wavelet encoders, the standard Lena (monochrome, 8bpp, 512x512) and café (monochrome, 8bpp, 2560x2048) images have been selected. On the one hand, the widely used Lena image (from the USC) allow us to compare it with practically all the published algorithms, because results are commonly expressed using this image. On the other hand, the café image, less blurred and more complex than Lena, has been obtained from the JPEG2000 test set and represents a typical picture that has been taken using a high quality 5-Megapixel digital camera.

Table 1 provides PSNR performance when using the Lena image. We can see as LTW outperforms the other coders, including the recently released JPEG 2000 standard, which results have been attained using JASPER [10], an official implementation included in the ISO/IEC 15444-5 standard.

Results for the café image are shown in table 2. Notice that only SPIHT and JASPER have been compared to LTW because the compiled versions of the rest of coders have not been released. In this case, the PSNR rate is still higher for our algorithm, although JASPER practically performs equally to LTW.

According to tables 1 and 2, we can deduce that LTW works better with low frequency images, due to its capability of grouping low amplitude coefficients in lower-tree structures.

codec\ rate	EZW	SPIHT	Stack Run	Embedded Run-length	JASPER/ JPEG2000	LTW
2	n/a	45.07	n/a	n/a	44.62	45.46
1	39.55	40.41	n/a	40.28	40.31	40.50
0.5	36.28	37.21	36.79	37.09	37.22	37.35
0.25	33.17	34.11	33.63	34.01	34.04	34.31
0.125	30.23	31.10	n/a	n/a	30.84	31.27

Table 1: PSNR (dB) with different bit rates and coders using Lena (512x512)

codec\ rate	SPIHT	JASPER/ JPEG2000	LTW
2	38.91	39.09	39.11
1	31.74	32.04	32.03
0.5	26.49	26.80	26.85
0.25	23.03	23.12	23.24
0.125	20.67	20.74	20.76

Table 2: PSNR (dB) with different bit rates and coders using Café (2560x2048)

One of the main advantages of the LTW algorithm is its lower temporal complexity. Table 3 shows as our algorithm greatly outperforms SPIHT and JASPER in terms of execution time. Our encoder is from 2.5 to 8.5 times faster than JASPER, while LTW decoder executes from 1.5 to 2.5 faster than JASPER decoder, depending on the bit-rate and the image size. In the case of SPIHT, our encoder is from 1.25 to 2.5 times faster, and the decoding process is from 2.5 to 7.5 times faster. Notice that in these tables we only have evaluated the coding or decoding process, since the used wavelet transform is the same in all the cases (the popular Daubechies 9/7 bi-orthogonal wavelet filter).

The computation of the symbols pass is only performed on the encoder side, and its temporal complexity is the same at any rate. That is what makes the LTW asymmetric at low bit rates.

Lena coding *Lena decoding*

codec\ rate	SPIHT	JASPER / JPEG 2000	LTW		SPIHT	JASPER / JPEG 2000	LTW
2	210.4	278.5	92.4		217.0	108.8	85.0
1	119.4	256.1	62.4		132.7	72.3	47.1
0.5	72.3	238.2	46.7		90.7	51.4	27.1
0.25	48.7	223.4	38.9		69.9	38.1	17.4
0.125	36.8	211.3	34.7		59.7	31.1	12.3

café coding *café decoding*

codec\ rate	SPIHT	JASPER / JPEG 2000	LTW		SPIHT	JASPER / JPEG 2000	LTW
2	4368.7	7393.1	1663.5		5375.4	2373.0	1505.6
1	2400.1	6907.6	1203.7		3514.0	1475.2	911.6
0.5	1399.3	6543.9	938.6		2619.7	991.8	569.5
0.25	889.8	6246.2	782.8		2193.0	763.6	366.6
0.125	626.5	6058.2	696.5		1969.9	635.3	253.4

Table 3: Execution time comparison for Lena and café (time in Million of CPU cycles)

6. CONCLUSIONS

In this paper, we have presented a new wavelet image encoder based on the construction and efficient coding of wavelet lower-trees (LTW). Its compression performance is within the state-of-the-art, outperforming the typically used algorithms (SPIHT is improved in 0.2-0.4 dB, and JPEG 2000 with Lena in 0.35 dB as mean value).

However, we have shown that the main contribution of this algorithm is its lower temporal complexity. According to the image size and the bit-rate, it is able to encode an image up to 8.5 times faster than JASPER and 2.5 times faster than SPIHT.

Due to its lower temporal complexity and the lack of memory overhead, we think that the LTW is a good candidate for real-time interactive multimedia communications, allowing implementation both in hardware and in software.

7. REFERENCES

[1] C. Chrysafis, A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Transactions on Image Processing*, vol. 9, pp. 378-389, March 2000.

[2] W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.*, 3, pp. 186-200, 1996.

[3] J.M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445-3462, Dec. 1993.

[4] A. Said, A. Pearlman. "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on circuits and systems for video technology*, vol. 6, nº 3, June 1996.

[5] M.J. Tsai, J. Villasenor, F. Chen. "Stack-run image coding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol 6, pp. 519-521, Oct. 1996

[6] W. Berghorn, T. Boskamp, M. Lang, H. Peitgen. "Fast variable run-length coding for embedded progressive wavelet-based image compression," *IEEE Trans Image Processing*, vol 10. no 12. pp. 1781-1790, Dec 2001

[7] ISO/IEC 15444-1: "JPEG2000 image coding system," 2000.

[8] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies. "Image coding using wavelet transform," *IEEE Trans Image Processing*, vol 1. no 2. pp. 205-220, 1992

[9] S. Mallat. "A theory for multiresolution signal decomposition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol 11. pp. 669-718, July 1989

[10] M. Adams. "Jasper Software Reference Manual (Version 1.600.0," *ISO/IEC JTC 1/SC 29/WG 1 N 2415*, Oct. 2002